

Diplomarbeit

Tapyre

Entwicklung eines KI-integrierten Produktivitätstools mit Plugin-System

Eingereicht von

Christian Vorhofer
Raphael Ladinig

Eingereicht bei

Höhere Technische Bundeslehr- und Versuchsanstalt
Anichstraße

Abteilung für Wirtschaftsingenieure/Betriebsinformatik

Betreuer

GREINÖCKER Albert, Mag. Dr. DI

Innsbruck, April 2026

Abgabevermerk:

Betreuer/in:

Datum:

Kurzfassung /Abstract

Eine Kurzfassung ist in deutscher sowie ein Abstract in englischer Sprache mit je maximal einer A4-Seite zu erstellen. Die Beschreibung sollte wesentliche Aspekte des Projektes in technischer Hinsicht beschreiben. Die Zielgruppe der Kurzbeschreibung sind auch Nicht-Techniker! Viele Leser lesen oft nur diese Seite.

Beispiel für ein Abstract (DE und EN)

Die vorliegende Diplomarbeit beschäftigt sich mit verschiedenen Fragen des Lernens Erwachsener – mit dem Ziel, Lernkulturen zu beschreiben, die die Umsetzung des Konzeptes des Lebensbegleitenden Lernens (LBL) unterstützen. Die Lernfähigkeit Erwachsener und die unterschiedlichen Motive, die Erwachsene zum Lernen veranlassen, bilden den Ausgangspunkt dieser Arbeit. Die anschließende Auseinandersetzung mit Selbstgesteuertem Lernen, sowie den daraus resultierenden neuen Rollenzuschreibungen und Aufgaben, die sich bei dieser Form des Lernens für Lernende, Lehrende und Institutionen der Erwachsenenbildung ergeben, soll eine erste Möglichkeit aufzeigen, die zur Umsetzung dieses Konzeptes des LBL beiträgt. Darüber hinaus wird im Zusammenhang mit selbstgesteuerten Lernprozessen Erwachsener die Rolle der Informations- und Kommunikationstechnologien im Rahmen des LBL näher erläutert, denn die Eröffnung neuer Wege zur ort- und zeitunabhängiger Kommunikation und Kooperation der Lernenden untereinander sowie zwischen Lernenden und Lernberatern gewinnt immer mehr an Bedeutung. Abschließend wird das Thema der Sichtbarmachung, Bewertung und Anerkennung des informellen und nicht-formalen Lernens aufgegriffen und deren Beitrag zum LBL erörtert. Diese Arbeit soll einerseits einen Beitrag zur besseren Verbreitung der verschiedenen Lernkulturen

leisten und andererseits einen Reflexionsprozess bei Erwachsenen, die sich lebensbegleitend weiterbilden, in Gang setzen und sie somit dabei unterstützen, eine für sie geeignete Lernkultur zu finden.

This thesis deals with the various questions concerning learning for adults – with the aim to describe learning cultures which support the concept of live-long learning (LLL). The learning ability of adults and the various motives which lead to adults learning are the starting point of this thesis. The following analysis on self-directed learning as well as the resulting new attribution of roles and tasks which arise for learners, trainers and institutions in adult education, shall demonstrate first possibilities to contribute to the implementation of the concept of LLL. In addition, the role of information and communication technologies in the framework of LLL will be closer described in context of self-directed learning processes of adults as the opening of new forms of communication and co-operation independent of location and time between learners as well as between learners and tutors gains more importance. Finally the topic of visualisation, validation and recognition of informal and non-formal learning and their contribution to LLL is discussed.

Gliederung des Abstract in **Thema, Ausgangspunkt, Kurzbeschreibung, Zielsetzung**.

Projektergebnis Allgemeine Beschreibung, was vom Projektziel umgesetzt wurde, in einigen kurzen Sätzen. Optional Hinweise auf Erweiterungen. Gut machen sich in diesem Kapitel auch Bilder vom Gerät (HW) bzw. Screenshots (SW). Liste aller im Pflichtenheft aufgeführten Anforderungen, die nur teilweise oder gar nicht umgesetzt wurden (mit Begründungen).

Erklärung der Eigenständigkeit der Arbeit

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe. Meine Arbeit darf öffentlich zugänglich gemacht werden, wenn kein Sperrvermerk vorliegt.

Ort, Datum

Verfasser 1

Ort, Datum

Verfasser 1

Inhaltsverzeichnis

| | |
|--|-----------|
| Abstract | ii |
| 1 Einführung in Neuronale Netzwerke | 1 |
| 1.1 Künstliche Neuronen | 1 |
| 1.2 Feed-Forward Neural Networks (FNN) | 2 |
| 1.3 Rekurrente Neuronale Netze (RNN, LSTM) | 3 |
| 1.4 Die Transformer-Architektur | 4 |
| 1.5 Bedeutung von Transformern für LLMs und Embeddings | 5 |
| 2 Einführung in Natural Language Processing (NLP) | 7 |
| 2.1 Klassische NLP-Ansätze | 7 |
| 2.2 Einführung in Embeddings | 8 |
| 2.3 Word Embeddings: Word2Vec und GloVe | 9 |
| 2.4 Kontextualisierte Embeddings | 9 |
| 2.5 Embeddings mit der Transformer-Architektur | 10 |
| 2.6 Relevanz für Tapyre Paper Search | 10 |
| 3 Einführung in Agentic AI | 11 |
| 3.1 ReAct: Reasoning + Acting | 11 |
| 3.2 Tool Usage | 12 |
| 3.3 Model Context Protocol (MCP) | 12 |
| 3.4 Agent-to-Agent Kommunikation | 13 |
| 3.5 RAG: Retrieval-Augmented Generation | 13 |
| 3.6 Multi-Agent Systems | 14 |
| 4 Grundkonzepte der verwendeten Technologien | 15 |
| 4.1 Docker und Containerisierung | 15 |
| 4.2 MySQL als relationale Datenbank | 16 |

| | | |
|----------|--|-----------|
| 4.3 | Qdrant und Approximate Nearest Neighbor Search | 17 |
| 4.3.1 | Speicherstruktur | 17 |
| 4.3.2 | Approximate Nearest Neighbor (ANN) | 17 |
| 4.3.3 | Ähnlichkeitsmaße | 18 |
| 4.4 | Flask und REST-APIs | 18 |
| 4.5 | PyTorch und GPU-Beschleunigung | 19 |
| 4.6 | Zusammenfassung | 20 |
| 5 | Entwicklung von Tapyre als Agentic-AI-System | 21 |
| 5.1 | Abstraktion der LLM-Schnittstelle | 22 |
| 5.2 | Der Agent und der ReAct-Loop | 23 |
| 5.3 | Plugins als Tools: Lose Kopplung durch Interfaces | 24 |
| 5.4 | Dynamisches Laden der Plugins | 25 |
| 5.5 | Beispiel: AppPlugin zur Steuerung lokaler Anwendungen . | 26 |
| 5.6 | Zusammenspiel von Agent, Plugins und ReAct-Loop | 28 |
| 6 | Architektur und Implementierung von Tapyre Paper Search | 29 |
| 6.1 | Abstraktion der Datenquellen | 30 |
| 6.2 | arXiv als konkrete Datenquelle | 31 |
| 6.2.1 | Initialisierung und Persistenz des Fortschritts | 31 |
| 6.2.2 | Iteratives Abrufen der Dokumente | 33 |
| 6.2.3 | ID-Inkrementierung und Bereichslogik | 35 |
| 6.2.4 | PDF-Download mit Rate-Limit und Retry-Strategie . . | 35 |
| 6.3 | PDF-Verarbeitung und Textextraktion | 36 |
| 6.4 | Abstraktion der Embedding-Erzeugung | 39 |
| 6.5 | Specter2 als semantisches Embedding-Modell | 39 |
| 6.6 | Abstraktion der Datenhaltung | 41 |
| 6.7 | MySQL für strukturierte Metadaten | 41 |
| 6.8 | Qdrant als Vektordatenbank | 44 |
| 6.9 | Pipeline zur Orchestrierung des Gesamtprozesses . . | 47 |
| 6.10 | Zusammenspiel der Komponenten | 49 |
| 6.11 | Analyse der Performance-Charakteristika | 50 |
| 6.11.1 | Limitierungen bei der semantischen Suche | 50 |
| 6.11.2 | Anfängliche Implementierungsineffizienzen und Opti- mierungen | 51 |
| 6.11.3 | Periodische Verarbeitungseinbrüche durch geplante Unterbrechungen | 52 |

| | |
|---|-----------|
| 6.11.4 Kumulative Verarbeitung und Gesamtstabilität des Systems | 52 |
| 6.11.5 Zusammenfassende Einordnung | 53 |
| Literaturverzeichnis | 63 |

Christian Vorhofer
Raphael Ladinig

1 Einführung in Neuronale Netzwerke

1.1 Künstliche Neuronen

Künstliche Neuronen bilden die Grundbausteine moderner neuronaler Netze und orientieren sich konzeptionell am Funktionsprinzip biologischer Nervenzellen. Ein künstliches Neuron erhält mehrere Eingangswerte x_1, x_2, \dots, x_n , die jeweils mit Gewichten w_1, w_2, \dots, w_n multipliziert werden. Zusammen mit einem Bias-Term b entsteht die gewichtete Summe

$$z = \sum_{i=1}^n w_i x_i + b.$$

Um dem Modell die Fähigkeit zu geben, nichtlineare Zusammenhänge zu lernen, wird auf diese Summe eine Aktivierungsfunktion angewendet. Typische Aktivierungsfunktionen sind die Sigmoid-Funktion, die Tanh-Funktion oder im modernen Deep Learning vor allem die *Rectified Linear Unit* (ReLU). Das resultierende Ausgabe-Signal des Neurons lautet somit

$$y = \sigma(z).$$

Durch die Verschachtelung vieler solcher Neuronen in mehreren Schichten (sogenannten Layers) können sehr komplexe Funktionen modelliert werden. Das „Wissen“ eines neuronalen Netzes ist dabei in den Gewichten und Bias-Werten gespeichert, die während des Trainingsprozesses mithilfe von Optimierungsverfahren wie dem Gradientenabstieg angepasst werden.

Die Fähigkeit eines einzelnen Neurons, lineare Entscheidungsgrenzen zu modellieren, wurde bereits früh durch das Perzepron-Modell demonstriert. Erst durch die Kombination vieler Neuronen in tieferen Netzen wurde es möglich, hochkomplexe Muster wie Bildmerkmale oder sprachliche Zusammenhänge effizient zu verarbeiten. Künstliche Neuronen stellen somit die Grundlage aller modernen Deep-Learning-Architekturen dar, aus denen später fortgeschrittene Modelle wie Convolutional Neural Networks (CNNs), Rekurrente Neuronale Netze (RNNs) und Transformer hervorgegangen sind (vgl. [Goodfellow et al. \(2016\)](#)).

Für diese Arbeit sind künstliche Neuronen besonders relevant, da sie die elementare Recheneinheit der verwendeten Transformer- und Embedding-Modelle darstellen. Alle später beschriebenen Verfahren zur semantischen Repräsentation wissenschaftlicher Texte basieren letztlich auf der Kombination und Optimierung dieser einfachen Bausteine.

1.2 Feed-Forward Neural Networks (FNN)

Ein Feed-Forward Neural Network (FNN) ist die einfachste Form eines neuronalen Netzes und bildet die Grundlage vieler moderner Deep-Learning-Modelle. Der Name beschreibt die zentrale Eigenschaft dieser Architektur: Informationen fließen ausschließlich in eine Richtung, nämlich vom Eingang (*Input Layer*) über eine oder mehrere verdeckte Schichten (*Hidden Layers*) zum Ausgang (*Output Layer*). Rückkopplungen oder Schleifen sind nicht vorhanden.

Ein FNN besteht aus mehreren künstlichen Neuronen, die schichtweise miteinander verbunden sind. Jedes Neuron berechnet aus seinen Eingaben eine gewichtete Summe und wendet anschließend eine Aktivierungsfunktion wie ReLU, Sigmoid oder Tanh an. Dadurch ist das Netzwerk in der Lage, auch komplexe und nichtlineare Zusammenhänge zu modellieren.

Das Training eines FNNs erfolgt mittels des Verfahrens der *Backpropagation*. Dabei wird zunächst der Fehler zwischen der vorhergesagten Ausgabe des Netzes und dem tatsächlichen Zielwert berechnet. Anschließend wird dieser Fehler schrittweise durch das Netzwerk zurückpropagiert, um die Gewichte

so anzupassen, dass der Fehler in zukünftigen Durchläufen minimiert wird (vgl. [Goodfellow et al. \(2016\)](#)).

Obwohl FNNs im Vergleich zu neueren Architekturen wie CNNs, RNNs oder Transformern relativ einfach aufgebaut sind, bilden sie das Fundament des Deep Learning. Viele moderne Modelle – einschließlich Transformer – lassen sich als Weiterentwicklungen dieses grundlegenden Prinzips verstehen, bei denen zusätzliche Mechanismen wie Attention oder spezielle Schichttypen eingeführt wurden (vgl. [Goodfellow et al. \(2016\)](#)).

In dieser Arbeit dienen FNNs vor allem als konzeptionelle Grundlage, um den Übergang von einfachen neuronalen Netzen zu komplexeren Architekturen wie Transformern nachvollziehbar zu machen.

1.3 Rekurrente Neuronale Netze (RNN, LSTM)

Rekurrente Neuronale Netze (RNNs) wurden entwickelt, um sequenzielle Daten wie Texte, Sprache, Musik oder Zeitreihen zu verarbeiten. Im Gegensatz zu Feed-Forward- oder Convolutional-Netzen besitzen RNNs Rückkopplungen, sodass Informationen aus vorherigen Zeitschritten in die Verarbeitung aktueller Eingaben einfließen können. Dadurch verfügen RNNs über eine Art internes „Gedächtnis“.

Ein einfaches RNN kombiniert zu jedem Zeitschritt den aktuellen Eingabewert mit dem vorherigen internen Zustand. Dieses Verfahren eignet sich gut für kurze Sequenzen, stößt jedoch bei längeren Abhängigkeiten an seine Grenzen. Ursache hierfür ist das sogenannte *Vanishing Gradient Problem*, bei dem Gradienten während des Trainings stark abnehmen und relevante Informationen verloren gehen.

Zur Lösung dieses Problems wurden Long Short-Term Memory Netze (LSTMs) entwickelt. LSTMs verfügen über spezielle Schaltelemente, sogenannte *Gates*, die steuern, welche Informationen gespeichert, weitergegeben oder verworfen werden. Dadurch sind LSTMs in der Lage, relevante Informationen über längere Zeiträume hinweg zu behalten und stabiler zu trainieren als klassische RNNs (vgl. [Hochreiter & Schmidhuber \(1997\)](#)).

LSTMs wurden über viele Jahre erfolgreich in der Sprachverarbeitung eingesetzt, etwa für maschinelle Übersetzung oder Textklassifikation. In modernen NLP-Systemen werden sie jedoch zunehmend durch Transformer-Modelle ersetzt, da diese effizienter trainierbar sind und besser mit sehr langen Texten umgehen können. Für diese Arbeit sind RNNs und LSTMs daher insbesondere im historischen Kontext relevant (vgl. [Hochreiter & Schmidhuber \(1997\)](#)).

1.4 Die Transformer-Architektur

Transformer-Modelle wurden im Jahr 2017 mit dem Paper *Attention Is All You Need* vorgestellt und haben die natürliche Sprachverarbeitung grundlegend verändert (vgl. [Ashish Vaswani \(2017\)](#)). Im Gegensatz zu RNNs und LSTMs verarbeiten Transformer die Eingabe nicht sequenziell, sondern betrachten alle Token eines Textes gleichzeitig. Dies ermöglicht eine effiziente Parallelisierung auf modernen GPUs und verbessert den Umgang mit langen Texten erheblich.

Das zentrale Konzept der Transformer-Architektur ist die sogenannte Self-Attention. Dieser Mechanismus erlaubt es dem Modell, zu bewerten, welche Wörter eines Satzes für die Bedeutung eines anderen Wortes besonders relevant sind. Dadurch können auch weit entfernte Abhängigkeiten innerhalb eines Textes effektiv modelliert werden (vgl. [Ashish Vaswani \(2017\)](#)).

Ein weiterer wichtiger Bestandteil ist die Multi-Head Attention. Dabei werden mehrere Attention-Mechanismen parallel eingesetzt, die jeweils unterschiedliche Arten von Beziehungen erfassen können, etwa syntaktische, semantische oder thematische Zusammenhänge. Die Ergebnisse dieser Attention-Köpfe werden anschließend zusammengeführt (vgl. [Ashish Vaswani \(2017\)](#)).

Da Transformer keine inhärente Reihenfolge der Eingabe besitzen, werden sogenannte Positional Encodings verwendet. Diese kodieren die Position eines Tokens im Text und werden den Eingaberepräsentationen hinzugefügt, sodass das Modell die Struktur der Sequenz berücksichtigen kann (vgl. [Ashish Vaswani \(2017\)](#)).

Ein klassischer Transformer besteht aus einem Encoder und einem Decoder. Während der Encoder den Eingabetext in eine semantisch aussagekräftige Repräsentation überführt, dient der Decoder der Textgenerierung. In vielen modernen Anwendungen – darunter auch die in dieser Arbeit betrachteten Such- und Embedding-Modelle – wird ausschließlich der Encoder verwendet (vgl. [Ashish Vaswani \(2017\)](#)).

Transformer bilden die Grundlage der in diesem Projekt eingesetzten Sprach- und Embedding-Modelle und sind daher zentral für das Verständnis der folgenden Kapitel (vgl. [Ashish Vaswani \(2017\)](#)).

1.5 Bedeutung von Transformern für LLMs und Embeddings

Transformer-Modelle sind heute die Basis nahezu aller modernen Anwendungen der natürlichen Sprachverarbeitung. Sie bilden das Fundament großer Sprachmodelle (*Large Language Models, LLMs*) sowie leistungsfähiger Embedding-Modelle (vgl. [Ashish Vaswani \(2017\)](#), [Brown et al. \(2020\)](#), [Touvron et al. \(2023\)](#)).

Durch den Einsatz von Self-Attention können Transformer auch weit entfernte Abhängigkeiten innerhalb eines Textes erfassen. Dies ist insbesondere bei langen oder komplexen Dokumenten von entscheidender Bedeutung, wie sie etwa in wissenschaftlichen Publikationen vorkommen. Frühere Architekturen wie RNNs oder LSTMs konnten solche Zusammenhänge nur eingeschränkt abbilden (vgl. [Ashish Vaswani \(2017\)](#)).

Für Embedding-Modelle bieten Transformer den Vorteil, dass sie nicht nur einzelne Wörter, sondern die semantische Bedeutung ganzer Sätze oder Dokumente in dichten Vektorrepräsentationen erfassen. Transformer-Encoder eignen sich daher besonders für Aufgaben wie semantische Suche, Textklassifikation oder Empfehlungssysteme (vgl. [Nandakumar et al. \(2023\)](#)).

Moderne Embedding-Modelle wie *SPECTER2*, *Sentence-BERT* oder *E5* basieren auf Transformer-Encodern. Sie ermöglichen es, inhaltlich ähnliche Texte

im Vektorraum nahe beieinander abzulegen, während thematisch unterschiedliche Dokumente klar getrennt sind. Diese Eigenschaft ist essenziell für die semantische Suche, wie sie im Rahmen dieses Projekts zur wissenschaftlichen Literatursuche eingesetzt wird (vgl. [Nandakumar et al. \(2023\)](#)).

Zusammenfassend lässt sich festhalten, dass Transformer die technologische Grundlage moderner Sprachverarbeitung darstellen. Ohne diese Architektur wären leistungsfähige LLMs sowie präzise Embedding-Modelle nicht realisierbar, wodurch auch das in dieser Arbeit entwickelte Suchsystem in dieser Form nicht möglich wäre (vgl. [Ashish Vaswani \(2017\)](#), [Brown et al. \(2020\)](#), [Touvron et al. \(2023\)](#), [Nandakumar et al. \(2023\)](#)).

2 Einführung in Natural Language Processing (NLP)

Natural Language Processing (NLP) ist ein zentraler Teilbereich der Künstlichen Intelligenz, der sich mit der automatischen Verarbeitung und Analyse menschlicher Sprache beschäftigt. Ziel ist es, Texte so zu modellieren, dass Computersysteme sprachbasierte Aufgaben ausführen können, etwa das Beantworten von Suchanfragen, das Zusammenfassen von Texten oder die Klassifikation von Dokumenten. Moderne Anwendungen wie Suchmaschinen, Chatbots oder Sprachassistenten basieren maßgeblich auf Methoden des NLP (vgl. [Jurafsky & Martin \(2023\)](#)).

Während frühe NLP-Ansätze überwiegend auf regelbasierten oder statistischen Verfahren beruhten, wird heutiges NLP fast ausschließlich durch tiefe neuronale Netze geprägt. Eine zentrale Fragestellung dabei ist, wie sprachliche Bedeutung in numerischer Form repräsentiert werden kann. Diese numerischen Repräsentationen werden als Embeddings bezeichnet und bilden die Grundlage moderner NLP-Systeme.

2.1 Klassische NLP-Ansätze

Vor dem Aufkommen neuronaler Sprachmodelle wurden Texte hauptsächlich mithilfe statistischer Verfahren repräsentiert. Zu den bekanntesten Ansätzen zählen Bag-of-Words, TF-IDF sowie N-Gramm-Modelle. Diese Methoden basieren auf der Häufigkeit von Wörtern oder Wortfolgen, berücksichtigen jedoch weder semantische Beziehungen noch den sprachlichen Kontext.

So wird beispielsweise nicht erkannt, dass Begriffe wie „Auto“ und „Fahrzeug“ eine ähnliche Bedeutung haben oder dass ein Wort wie „Bank“ je nach Kontext unterschiedliche Bedeutungen annehmen kann (vgl. [Jurafsky & Martin \(2023\)](#)). Für einfache Klassifikations- oder Zählaufgaben können diese Modelle ausreichend sein, bei komplexeren Anwendungen wie semantischer Suche oder maschineller Übersetzung stoßen sie jedoch schnell an ihre Grenzen.

Diese Einschränkungen waren ein wesentlicher Motivationsfaktor für die Entwicklung semantischer Repräsentationen in Form von Embeddings.

2.2 Einführung in Embeddings

Da Computer ausschließlich mit numerischen Daten arbeiten, müssen sprachliche Informationen in eine geeignete mathematische Form überführt werden. Embeddings lösen dieses Problem, indem sie Wörter, Sätze oder ganze Dokumente als Vektoren in einem kontinuierlichen Vektorraum darstellen. Ziel dieser Repräsentation ist es, semantische Beziehungen geometrisch abzubilden.

Dabei gelten folgende grundlegende Prinzipien:

- Ähnliche Bedeutungen sollen durch ähnliche Vektoren repräsentiert werden.
- Unterschiedliche Bedeutungen sollen im Vektorraum weit voneinander entfernt liegen.
- Kontextinformationen sollen – sofern möglich – in die Repräsentation einfließen.

Embeddings bilden die Grundlage vieler moderner NLP-Anwendungen, insbesondere für Aufgaben wie Ähnlichkeitsberechnungen, Clustering oder semantische Suche. Auch in dieser Arbeit spielen Embeddings eine zentrale Rolle, da sie die Basis für den Vergleich wissenschaftlicher Texte darstellen.

2.3 Word Embeddings: Word2Vec und GloVe

Einen wesentlichen Fortschritt stellten Word-Embedding-Modelle wie Word2Vec dar. Diese Modelle ordnen jedem Wort einen festen Vektor zu und beruhen auf der Annahme, dass Wörter, die in ähnlichen Kontexten auftreten, auch ähnliche Bedeutungen haben. Dadurch entstehen semantische Strukturen im Vektorraum, die sich beispielsweise durch einfache Vektorrechnungen ausdrücken lassen:

$$\text{Koenig} - \text{Mann} + \text{Frau} \approx \text{Koenigin}$$

Modelle wie Word2Vec (vgl. [Mikolov et al. \(2013\)](#)) erfassen grundlegende semantische Beziehungen zwischen Wörtern sehr effektiv. Ein zentrales Problem dieser Ansätze besteht jedoch darin, dass jedes Wort unabhängig vom Kontext immer durch denselben Vektor repräsentiert wird. Mehrdeutige Wörter wie „Bank“ können somit nicht kontextabhängig interpretiert werden.

Diese Einschränkung führte zur Entwicklung kontextualisierter Embedding-Modelle.

2.4 Kontextualisierte Embeddings

Mit dem Einsatz tiefer neuronaler Netze entstanden Modelle, die Wortbedeutungen abhängig vom jeweiligen Kontext darstellen können. Ein bekanntes Beispiel ist ELMo, das für ein Wort unterschiedliche Vektoren erzeugt, je nachdem, in welchem Satz es vorkommt. Dadurch kann zwischen verschiedenen Bedeutungen desselben Wortes unterschieden werden.

Diese kontextualisierten Embeddings stellten einen wichtigen Zwischen- schritt dar und ebneten den Weg für leistungsfähigere Modelle auf Basis der Transformer-Architektur. Sie zeigten erstmals, dass eine dynamische, kontextabhängige Repräsentation von Sprache deutlich bessere Ergebnisse liefert als statische Wortvektoren.

2.5 Embeddings mit der Transformer-Architektur

Mit der Einführung der Transformer-Architektur wurden neue Maßstäbe in der Sprachverarbeitung gesetzt. Transformer-Encoder wie BERT (vgl. Devlin et al. (2018)) oder wissenschaftsspezifische Modelle wie SPECTER (vgl. Cohan et al. (2020)) erzeugen hochqualitative, kontextualisierte Embeddings, indem sie den gesamten Satz oder sogar vollständige Dokumente berücksichtigen.

Dies führt unter anderem zu folgenden Vorteilen:

- kontextabhängige Wortrepräsentationen,
- Satz- und Dokumentembeddings als einzelne Vektoren,
- präzisere semantische Modellierung,
- robuste Verarbeitung langer und komplexer Textstrukturen.

Insbesondere für wissenschaftliche Texte mit komplexer Terminologie und langen Abhängigkeiten sind Transformer-basierte Embeddings besonders geeignet. Aus diesem Grund kommen sie auch in dieser Arbeit zum Einsatz.

2.6 Relevanz für Tapyre Paper Search

Im Projekt *Tapyre Paper Search* werden Embeddings verwendet, um wissenschaftliche Publikationen in einem hochdimensionalen Vektorraum abzubilden. Dokumente mit ähnlichem inhaltlichem Schwerpunkt liegen dabei räumlich nahe beieinander, wodurch eine präzise semantische Suche ermöglicht wird.

Der Einsatz spezialisierter Modelle wie SPECTER₂, die gezielt auf wissenschaftlichen Texten trainiert wurden, verbessert die Erkennung fachlicher Zusammenhänge zusätzlich. Die in diesem Kapitel beschriebenen NLP- und Embedding-Konzepte bilden somit die theoretische Grundlage für die im weiteren Verlauf der Arbeit vorgestellten Such- und Vergleichsmechanismen.

3 Einführung in Agentic AI

Agentic AI beschreibt ein modernes Paradigma der künstlichen Intelligenz, bei dem Modelle nicht nur reaktiv auf Eingaben antworten, sondern eigenständig Ziele verfolgen, Handlungen planen, Werkzeuge einsetzen und komplexe Aufgaben in mehreren Schritten ausführen. Ein *Agent* ist dabei ein KI-System, das Informationen beschafft, Entscheidungen trifft, Aktionen ausführt und auf Basis neuer Beobachtungen sein weiteres Vorgehen anpasst (vgl. [Yao et al. \(2022\)](#)).

Im Gegensatz zu klassischen Sprachmodellen, die primär als textbasierte Antwortgeneratoren fungieren, agieren agentische Systeme proaktiv und interaktiv. Sie kombinieren Sprachverarbeitung mit Planungs-, Entscheidungs- und Ausführungsmechanismen und können dadurch komplexe Workflows selbstständig bearbeiten.

3.1 ReAct: Reasoning + Acting

Ein grundlegender Ansatz innerhalb von Agentic AI ist das ReAct-Framework (Reasoning + Acting). Dieser Ansatz kombiniert explizite Gedankenschritte (*Reasoning*) mit konkreten Aktionen (*Acting*). ReAct wurde von Yao et al. vorgestellt und zielt darauf ab, Entscheidungsprozesse transparent mit tatsächlichen Handlungen zu verbinden (vgl. [Yao et al. \(2022\)](#)).

Ein typischer ReAct-Agent arbeitet in einer zyklischen Struktur:

1. Der Agent formuliert einen Gedankenschritt (*Thought*), um das weitere Vorgehen zu planen.
2. Er führt eine Aktion aus, beispielsweise eine Suchanfrage oder API-Abfrage.

3. Er erhält eine Beobachtung (*Observation*) als Ergebnis der Aktion.
4. Auf Basis dieser Information plant er den nächsten Schritt.

Diese Schleife ermöglicht es Agenten, komplexe Aufgaben in überschaubare Teilschritte zu zerlegen und flexibel auf neue Informationen zu reagieren.

3.2 Tool Usage

Ein zentrales Merkmal agentischer Systeme ist die Fähigkeit, externe Werkzeuge (*Tools*) zu verwenden. Diese Tools erweitern die Fähigkeiten eines Sprachmodells erheblich, da sie den Zugriff auf externe Datenquellen und Rechenressourcen ermöglichen. Typische Beispiele sind:

- Datenbanken (z. B. Qdrant, MySQL),
- Web-APIs (z. B. arXiv oder Semantic Scholar),
- Dateisysteme,
- Suchfunktionen,
- Ausführung von Python-Skripten.

Der Agent wählt ein geeignetes Tool aus, übergibt die notwendigen Parameter, interpretiert die Ergebnisse und nutzt diese als Grundlage für weitere Entscheidungen. Dadurch fungiert das Sprachmodell als eine Art Steuerinstanz, die verschiedene Systeme orchestriert.

Im Kontext dieser Arbeit ermöglicht Tool Usage beispielsweise den Zugriff auf Desktop-Apps.

3.3 Model Context Protocol (MCP)

Das Model Context Protocol (MCP) ist ein offener Standard, der definiert, wie KI-Modelle sicher, strukturiert und nachvollziehbar mit Tools und externen Datenquellen interagieren können. MCP legt unter anderem fest:

- wie Tools formal beschrieben werden,
- wie Kontextinformationen an Modelle übergeben werden,

- wie Modelle Aktionen anfordern,
- und wie Ergebnisse standardisiert zurückgegeben werden.

Der von OpenAI vorgestellte Standard erleichtert die Integration agentischer Systeme in bestehende Software-Architekturen und reduziert den Implementierungsaufwand komplexer Pipelines (vgl. [OpenAI \(2024\)](#)). Für agentische Systeme wie Tapyre bietet MCP eine strukturierte Grundlage zur Anbindung externer Dienste.

3.4 Agent-to-Agent Kommunikation

In umfangreicheren agentischen Systemen arbeiten häufig mehrere Agenten gemeinsam an einer Aufgabe. Diese sogenannte Agent-to-Agent-Kommunikation ermöglicht eine Aufteilung komplexer Problemstellungen auf spezialisierte Rollen. Beispiele hierfür sind:

- ein Analyse-Agent zur Extraktion relevanter Informationen,
- ein Recherche-Agent zur Suche nach geeigneten Quellen,
- ein Planungs-Agent zur Koordination des Workflows,
- ein Evaluations-Agent zur Überprüfung der Ergebnisse.

Durch diese Spezialisierung steigt sowohl die Robustheit als auch die Skalierbarkeit des Gesamtsystems. Fehler oder Unsicherheiten einzelner Agenten können durch andere Agenten ausgeglichen werden.

3.5 RAG: Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) verbindet Sprachmodelle mit externem, strukturiertem Wissen. Anstatt Antworten ausschließlich aus dem internen Modellwissen zu generieren, ruft ein RAG-System zunächst relevante Dokumente ab und erzeugt darauf aufbauend eine fundierte Antwort. Der Ansatz wurde von Lewis et al. eingeführt (vgl. [Lewis et al. \(2020\)](#)).

Ein typischer RAG-Agent arbeitet in zwei Phasen:

Christian Vorhofer
Raphael Ladinig

1. **Retrieval:** Suche nach relevanten Dokumenten, beispielsweise über eine Vektorsuche in einer Datenbank wie Qdrant.
2. **Generation:** Erzeugung einer Antwort unter Einbeziehung der abgerufenen Inhalte.

Für Tapyre Paper Search ist dieser Ansatz essenziell, da wissenschaftliche Dokumente zunächst semantisch identifiziert und anschließend analysiert, zusammengefasst oder weiterverarbeitet werden.

3.6 Multi-Agent Systems

Multi-Agent Systems (MAS) bestehen aus mehreren autonomen, oft spezialisierten Agenten, die parallel oder kooperativ zusammenarbeiten. Solche Systeme bieten mehrere Vorteile:

- höhere Robustheit durch Aufgabenverteilung,
- bessere Skalierbarkeit,
- parallele Bearbeitung komplexer Probleme,
- gezielte Spezialisierung einzelner Agenten.

Multi-Agent-Systeme werden zunehmend in Forschung, Retrieval-Systemen und komplexen KI-Anwendungen eingesetzt. Auch für zukünftige Erweiterungen von Tapyre Paper Search bieten sie eine flexible und leistungsfähige Architekturgrundlage.

4 Grundkonzepte der verwendeten Technologien

Für die Umsetzung von Tapyre Paper Search werden mehrere moderne Software- und Infrastrukturtechnologien eingesetzt, die zusammen eine performante und erweiterbare Architektur ermöglichen. Dieses Kapitel erläutert die technischen Grundlagen und zeigt jeweils den praktischen Bezug zur Umsetzung des Systems (z. B. Deployment, Datenhaltung, semantische Suche, API-Schicht und GPU-beschleunigte Modellinferenz).

4.1 Docker und Containerisierung

Docker ist eine Plattform zur Containerisierung von Anwendungen (vgl. [Docker Inc. \(2025\)](#)). Im Gegensatz zu klassischen virtuellen Maschinen teilen sich Container den Kernel des Host-Betriebssystems, sind aber logisch voneinander isoliert. Diese Isolation wird durch mehrere Linux-Technologien erreicht, insbesondere Namespaces, Control Groups (cgroups) und Union-Filesystems (vgl. [Docker Inc. \(2025\)](#), [Quirós \(2024\)](#)):

- **Namespaces:** isolieren Prozesse, Netzwerk, Benutzer und Dateisysteme.
- **Control Groups (cgroups):** begrenzen CPU-, RAM- und I/O-Ressourcen.
- **Union-Filesystems** (z. B. OverlayFS): ermöglichen effiziente, layer-basierte Images.

Docker ermöglicht reproduzierbare Umgebungen, schnelle Deployments und konsistente Konfigurationen (vgl. [Docker Inc. \(2025\)](#)). Für Tapyre Paper Search ist dies insbesondere relevant, da Komponenten wie Qdrant, MySQL

und die Python-Dienste unabhängig voneinander, aber dennoch in einer einheitlichen Laufzeitumgebung betrieben werden können. Dadurch lassen sich Entwicklungs-, Test- und Produktionsumgebungen konsistent abbilden und Abhängigkeiten (z. B. Datenbankversionen) kontrollierbar halten.

4.2 MySQL als relationale Datenbank

MySQL ist ein relationales Datenbanksystem, das Daten strukturiert in Tabellen speichert. Das Datenmodell folgt einem relationalen Schema, bei dem Entitäten über Primär- und Fremdschlüssel miteinander verbunden sind. Für die interne Datenorganisation nutzt MySQL (InnoDB) insbesondere B⁺-Bäume, die als Clustered und Secondary Indexes realisiert sind (vgl. [Oracle Corporation \(2024\)](#), [Percona \(2024\)](#)):

- **Clustered Index:** InnoDB speichert Daten entlang des Primärschlüssels; die Tabelle ist selbst als B⁺-Baum organisiert.
- **Secondary Indexes:** weitere B⁺-Bäume, die Verweise auf die Primärschlüsselzeilen enthalten.
- **Effiziente Suche:** durch die logarithmische Höhe der B⁺-Bäume können Suchanfragen performant ausgeführt werden.

MySQL bietet außerdem ACID-Transaktionen, die Datenkonsistenz und Zuverlässigkeit sicherstellen (vgl. [Gray & Reuter \(1992\)](#)):

- **Atomicity:** Eine Transaktion wird entweder vollständig ausgeführt oder verworfen.
- **Consistency:** Integritätsregeln bleiben erhalten.
- **Isolation:** Parallel ausgeführte Transaktionen beeinflussen sich kontrolliert bzw. entsprechend des Isolation-Levels nicht unzulässig.
- **Durability:** bestätigte Änderungen bleiben dauerhaft gespeichert.

In Tapyre Paper Search wird MySQL zur Speicherung strukturierter Metadaten eingesetzt (z. B. Titel, Autoren, Kategorien, Importstatus). Das ermöglicht klassische Filter- und Verwaltungsabfragen (z. B. nach Datum, Kategorie oder Verarbeitungsstatus), während die semantische Suche separat über die Vektordatenbank erfolgt.

4.3 Qdrant und Approximate Nearest Neighbor Search

Qdrant ist eine spezialisierte Datenbank zur Speicherung und Suche von Vektorrepräsentationen (Embeddings) und ist für semantische Suchanwendungen optimiert (vgl. [Qdrant Technologies \(2024d\)](#)). Im Gegensatz zu relationalen Datenbanken steht hier nicht die textbasierte Volltextsuche im Vordergrund, sondern die Ähnlichkeitssuche in einem Vektorraum, in dem Dokumente durch numerische Repräsentationen beschrieben werden.

4.3.1 Speicherstruktur

Eine Qdrant-Collection besteht typischerweise aus Vektoren, Payload-Daten (Metadaten) sowie internen Segmenten zur Datenorganisation (vgl. [Qdrant Technologies \(2024b\)](#)):

- **Vektoren** (häufig 768 oder 1024 Dimensionen, abhängig vom Embedding-Modell),
- **Payload** wie Titel, DOI, Autoren oder Jahr,
- **Segmente** zur internen Aufteilung der Daten.

Diese Struktur ist auf schnelle Ähnlichkeitsabfragen und effiziente Datenzugriffe ausgelegt (vgl. [Qdrant Technologies \(2024e\)](#)). In Tapyre Paper Search erlaubt die Payload zudem das nachträgliche Filtern semantischer Treffer (z. B. nach Jahr oder Kategorie), während die eigentliche Ranking-Logik über Vektorsimilarität erfolgt.

4.3.2 Approximate Nearest Neighbor (ANN)

Da ein exakter Vergleich aller Vektoren bei großen Datenmengen ineffizient ist, verwenden Vektordatenbanken ANN-Algorithmen. Qdrant nutzt hierfür u. a. den HNSW-Index (Hierarchical Navigable Small World), einen graphbasierten ANN-Algorithmus (vgl. [Malkov & Yashunin \(2020\)](#), [Qdrant Technologies \(2024c,a\)](#)):

- mehrschichtige Graphstruktur,
- wenige Knoten auf höheren Ebenen (grobe Orientierung),
- viele Knoten auf unteren Ebenen (feine Suche),
- Navigation vom groben in den feinen Suchraum,
- schnelle Annäherung an die nächsten Nachbarn.

HNSW kombiniert hohe Geschwindigkeit mit hoher Treffergenauigkeit und eignet sich besonders für semantische Suchsysteme (vgl. [Malkov & Yashunin \(2020\)](#), [Qdrant Technologies \(2024e\)](#)). In Tapyre Paper Search ist dies zentral, da auch größere Paper-Sammlungen interaktiv durchsuchbar bleiben sollen.

4.3.3 Ähnlichkeitsmaße

Qdrant unterstützt verschiedene Distanz- bzw. Ähnlichkeitsmaße, darunter Kosinusähnlichkeit, euklidische Distanz und skalares Produkt (vgl. [Qdrant Technologies \(2024e\)](#)):

- **Kosinusähnlichkeit** (häufiger Standard für NLP-Embeddings),
- **Euklidische Distanz**,
- **Skalares Produkt** (Dot Product).

In Tapyre wird hauptsächlich die Kosinusähnlichkeit eingesetzt, da sie bei normalisierten Text-Embeddings eine robuste Näherung semantischer Ähnlichkeit bietet (vgl. [Qdrant Technologies \(2024e\)](#)).

4.4 Flask und REST-APIs

Flask ist ein leichtgewichtiges Webframework für Python und dient in Tapyre zur Bereitstellung von REST-APIs (vgl. [Ronacher & Contributors \(2024\)](#)). REST (*Representational State Transfer*) ist ein Architekturstil für verteilte Systeme, der u. a. von Fielding beschrieben wurde (vgl. [Fielding \(2000\)](#)). REST-basierte Schnittstellen verwenden standardisierte HTTP-Methoden:

Christian Vorhofer
Raphael Ladinig

- **GET:** Abfrage von Daten,
- **POST:** Erstellen neuer Ressourcen bzw. Auslösen von Verarbeitung,
- **PUT/PATCH:** Aktualisieren von Ressourcen,
- **DELETE:** Löschen von Ressourcen.

REST-APIs verwenden häufig JSON als Datenaustauschformat und sind zustandslos, d. h. jeder Request enthält alle für die Verarbeitung notwendigen Informationen (vgl. [Fielding \(2000\)](#)).

Ein vereinfachtes Beispiel für eine Anfrage an einen Embedding-Endpunkt:

```
POST /embed
{
    "text": "Deep learning improves scientific search."
}
```

In Tapyre Paper Search dient die API-Schicht als klar definierte Kommunikationsgrenze zwischen Komponenten (z. B. Frontend, Orchestrierung und Embedding-Service). Das erleichtert Tests, ermöglicht unabhängige Skalierung einzelner Dienste und reduziert Kopplung zwischen Systemteilen.

4.5 PyTorch und GPU-Beschleunigung

PyTorch ist ein Framework für Deep Learning und wird in Tapyre zur Berechnung von Embeddings eingesetzt (vgl. [Paszke et al. \(2019\)](#)). Da Transformer-Modelle wie SPECTER2 sehr rechenintensiv sind, wird die Ausführung auf GPUs genutzt, um Matrixoperationen massiv zu beschleunigen. CUDA stellt hierfür eine Plattform bereit, um entsprechende Operationen auf der Grafikkarte auszuführen (vgl. [NVIDIA Corporation \(2025\)](#)).

Relevante Vorteile von PyTorch sind unter anderem (vgl. [Paszke et al. \(2019\)](#)):

- dynamische Rechengraphen,
- breite Modell- und Tool-Unterstützung,
- nahtlose GPU-Nutzung,

- gute Integrierbarkeit in moderne NLP-Pipelines.

Für Tapyre Paper Search ist dies praktisch relevant, da die Embedding-Erzeugung typischerweise den größten Rechenanteil der Pipeline darstellt. GPU-Beschleunigung reduziert die Laufzeit der Indexierung und ermöglicht schnellere Aktualisierungen des Paper-Bestands.

4.6 Zusammenfassung

Docker stellt reproduzierbare Umgebungen für die komponentenbasierte Ausführung bereit (vgl. [Docker Inc. \(2025\)](#)), MySQL speichert strukturierte Metadaten effizient über Indizes und Transaktionen (vgl. [Oracle Corporation \(2024\)](#), [Gray & Reuter \(1992\)](#)), Qdrant ermöglicht performante semantische Vektorsuche mittels ANN/HNSW (vgl. [Qdrant Technologies \(2024d\)](#), [Malkov & Yashunin \(2020\)](#)), Flask dient als Kommunikationsschicht über REST (vgl. [Ronacher & Contributors \(2024\)](#), [Fielding \(2000\)](#)), und PyTorch führt rechenintensive Embedding-Modelle GPU-beschleunigt aus (vgl. [Paszke et al. \(2019\)](#), [NVIDIA Corporation \(2025\)](#)). Zusammen bilden diese Technologien die technische Grundlage für ein flexibles und leistungsfähiges Informationssystem wie Tapyre Paper Search.

5 Entwicklung von Tapyre als Agentic-AI-System

In diesem Kapitel wird die Entwicklung von Tapyre als *Agentic AI-System* beschrieben. Im Gegensatz zu klassischen, rein reaktiven Sprachmodellen, die ausschließlich nach dem Prinzip *Input → Output* arbeiten, nutzt Tapyre einen Agenten, der eigenständig Entscheidungen trifft, Tools aufruft und Aufgaben iterativ in einem ReAct-Loop (Reasoning + Acting) ausführt (vgl. [Yao et al. \(2022\)](#), [Schick et al. \(2023\)](#), [Wang et al. \(2024\)](#)).

Der Agent fungiert dabei als steuernde Instanz zwischen dem Sprachmodell, externen Werkzeugen und der Systemumgebung. Die Kopplung zwischen Kernsystem, LLM und Plugins ist bewusst lose gehalten, um eine hohe Erweiterbarkeit, Wartbarkeit und Austauschbarkeit einzelner Komponenten zu gewährleisten. Dieses Architekturprinzip orientiert sich an etablierten Entwurfsmustern modularer Softwaresysteme (vgl. [Gamma et al. \(1994\)](#)).

Im Folgenden werden die wichtigsten Bausteine dieser Architektur erläutert:

- abstrakte LLM-Schnittstelle und konkrete Implementierung für Ollama,
- der Agent auf Basis von LangChain und dem ReAct-Paradigma,
- das Plugin-Konzept und dessen Abbildung auf LangChain-Tools,
- dynamisches Laden der Plugins zur Laufzeit,
- ein konkretes Beispiel-Plugin (AppPlugin),
- lose Kopplung und systemweite Erweiterbarkeit.

5.1 Abstraktion der LLM-Schnittstelle

Um Tapyre unabhängig von einem konkreten Sprachmodell oder Anbieter zu halten, wird eine abstrakte LLM-Schnittstelle definiert. Jeder unterstützte LLM-Typ (z. B. Ollama oder cloudbasierte APIs) muss lediglich diese Schnittstelle implementieren. Dadurch kann das zugrunde liegende Sprachmodell ausgetauscht werden, ohne dass der Agent oder bestehende Plugins angepasst werden müssen.

Dieses Vorgehen folgt etablierten Architekturprinzipien wie Interface- und Factory-Abstraktionen, die eine klare Trennung von Schnittstelle und Implementierung vorsehen (vgl. [Gamma et al. \(1994\)](#)).

```

1  from abc import ABC, abstractmethod
2  from langchain_core.language_models.chat_models import BaseChatModel
3  class LLM(ABC):
4      @abstractmethod
5      def getLLM(self) -> BaseChatModel:
6          pass

```

Listing 5.1: Abstrakte LLM-Schnittstelle

Die Schnittstelle kapselt die Kommunikation mit dem Sprachmodell vollständig und stellt dem restlichen System eine einheitliche Interaktionsmöglichkeit zur Verfügung.

Eine konkrete Implementierung für das lokale LLM-Framework sieht wie folgt aus:

```

1  from abstractions.llm import LLM
2  from langchain_core.language_models.chat_models import BaseChatModel
3  from langchain_community.chat_models import ChatOllama
4
5  class OllamaLLM(LLM):
6      def __init__(self, model: str, host: str, temperature: float, max_tokens: int):
7          self.llm = ChatOllama(
8              model=model,
9              base_url=host,
10             temperature=temperature,
11             max_tokens=max_tokens
12         )
13     def getLLM(self) -> BaseChatModel:
14         return self.llm

```

Listing 5.2: OllamaLLM als konkrete Implementierung

Der restliche Systemkern arbeitet ausschließlich mit dem Interface LLM und erhält Instanzen von BaseChatModel. Welches konkrete Sprachmodell im Hintergrund verwendet wird, ist für den Agenten und die Plugins vollständig transparent.

5.2 Der Agent und der ReAct-Loop

Der Agent selbst ist ebenfalls als Abstraktion definiert und stellt lediglich eine zentrale Methode `ask` bereit, die eine Anfrage entgegennimmt und eine Antwort erzeugt:

```

1  from abc import ABC, abstractmethod
2  class Agent(ABC):
3      @abstractmethod
4      def ask(self, prompt: str) -> str:
5          pass

```

Listing 5.3: Abstrakte Agent-Schnittstelle

Diese minimale Schnittstelle verdeutlicht die konzeptionelle Rolle des Agenten: Er fungiert als vermittelnde Instanz zwischen Benutzeranfrage, Sprachmodell und verfügbaren Tools.

Die konkrete Implementierung `PluginAgent` basiert auf dem Framework LangChain und verwendet den Agententyp `ZERO_SHOT.REACT_DESCRIPTION`. Dieser Agententyp setzt explizit das ReAct-Paradigma um, bei dem das Modell interne Gedankenschritte (*Reasoning*) mit konkreten Aktionen (*Acting*) kombiniert (vgl. [Yao et al. \(2022\)](#), [LangChain \(2023\)](#)).

```

1  # simple_agent.py
2  from langchain.agents import initialize_agent, AgentType
3  from langchain.prompts import ChatPromptTemplate, SystemMessagePromptTemplate, HumanMessagePromptTemplate
4  from abstractions.agent import Agent
5  from abstractions.llm import LLM
6
7  class PluginAgent(Agent):
8      def __init__(self, tools: list, llm: LLM, system_prompt: str, verbose: bool):
9
10         self.llm = llm.getLLM()
11
12         self.prompt = ChatPromptTemplate.from_messages([
13             SystemMessagePromptTemplate.from_template(system_prompt),
14             HumanMessagePromptTemplate.from_template("{input}")
15         ])
16
17         self.agent = initialize_agent(
18             tools=tools,
19             llm=self.llm,
20             agent=AgentType.ZERO_SHOT.REACT_DESCRIPTION,
21             verbose=verbose,
22             handle_parsing_errors=True,
23             max_iterations=2,
24             early_stopping_method="generate"
25         )
26
27     def ask(self, prompt: str) -> str:
28         return self.agent.run(prompt)

```

Listing 5.4: PluginAgent mit ReAct-Agententyp

Zentrale Aspekte dieser Implementierung sind:

Christian Vorhofer
 Raphael Ladinig

- **ReAct-Loop:** Der Agent erzeugt intern eine Sequenz aus »Thought«, »Action« und »Observation«, wodurch Planung und Ausführung explizit miteinander verknüpft werden.
- **Tool-Auswahl:** Die verfügbare Tool-Liste wird dynamisch aus den geladenen Plugins generiert. Das LLM erhält ausschließlich deren Beschreibungen und entscheidet selbstständig, welches Tool geeignet ist.
- **Iterationsbegrenzung:** Die maximale Anzahl von Tool-Aufrufen wird begrenzt, um Endlosschleifen zu vermeiden und die Kontrolle über den Agentenlauf zu behalten (vgl. [Schick et al. \(2023\)](#)).

5.3 Plugins als Tools: Lose Kopplung durch Interfaces

Plugins stellen die eigentliche funktionale Erweiterbarkeit des Systems dar, etwa zum Starten von Anwendungen, zur Abfrage externer Datenquellen oder zur Interaktion mit dem Dateisystem. Sie sind über eine abstrakte Basisklasse definiert und können unabhängig vom Kernsystem implementiert werden.

```

1  from abc import ABC, abstractmethod
2  from langchain_core.tools import Tool as LCTool
3
4  class Plugin(ABC):
5      prefix: str
6      name: str
7      prompt: str
8
9      @property
10     def Promt(self) -> str: # noqa: N802
11         return self.prompt
12
13     @abstractmethod
14     def run(self, text: str) -> str:
15         pass
16
17     def full_name(self) -> str:
18         return self.name
19
20     def to_langchain(self, *, return_direct: bool = False):
21         tool = LCTool.from_function(
22             name=self.full_name(),
23             func=self.run,
24             description=self.prompt,
25         )
26
27         tool.return_direct = return_direct
28
29     def __call__(self, text: str) -> str:
30

```

31 `return self.run(text)`

Listing 5.5: Abstrakte Plugin-Basisklasse

Wesentliche Eigenschaften dieses Ansatzes sind:

- **Interface-basiertes Design:** Jedes Plugin implementiert lediglich die Methode `run()`.
- **Lose Kopplung:** Der Agent kennt ausschließlich die daraus erzeugten LangChain-Tools, nicht jedoch die konkrete Plugin-Implementierung.
- **Tool-Integration:** Mithilfe von `LCTool.from_function` wird die `run()`-Methode automatisch als Tool für den ReAct-Loop verfügbar gemacht (vgl. [LangChain \(2023\)](#)).

5.4 Dynamisches Laden der Plugins

Um neue Funktionalitäten ohne Änderungen am Hauptprogramm integrieren zu können, werden Plugins dynamisch zur Laufzeit geladen. Dieses Vorgehen entspricht gängigen Entwurfsmustern für modulare und erweiterbare Softwaresysteme (vgl. [Gamma et al. \(1994\)](#)).

```

1  from __future__ import annotations
2
3  import inspect
4  from importlib.util import module_from_spec, spec_from_file_location
5  from pathlib import Path
6  from typing import List
7  from abstractions.plugin import Plugin
8
9
10 class PluginLoader:
11     def __init__(self, plugins_dir: str | Path | None = None) -> None:
12         if plugins_dir is None:
13             plugins_dir = Path(__file__).parent / ".." / "plugins"
14         self.plugins_dir = Path(plugins_dir).resolve()
15
16     def load(self) -> List[Plugin]:
17         if not self.plugins_dir.exists():
18             print(f"[PluginLoader] Folder not Found: {self.plugins_dir}")
19             return []
20
21         plugins: List[Plugin] = []
22         for file in sorted(self.plugins_dir.glob("*.py")):
23             if file.name.startswith("_"):
24                 continue
25
26             mod = self._import_module(file)
27             if not mod:
28                 continue
29
30             for _, cls in inspect.getmembers(mod, inspect.isclass):
31                 if cls.__module__ != mod.__name__:
32                     continue
33                 if not issubclass(cls, Plugin) or cls is Plugin:

```

```

34         continue
35     if inspect.isabstract(cls):
36         continue
37     try:
38         instance = cls()
39         plugins.append(instance)
40     except TypeError as e:
41         print(f"[PluginLoader] Couldn't load a instance of {cls.__name__}: {e}")
42
43     return plugins
44
45 def _import_module(self, file: Path):
46     module_name = f"plugins_{file.stem}_abs(hash(str(file)))"
47     spec = spec_from_file_location(module_name, file)
48     if not spec or not spec.loader:
49         print(f"[PluginLoader] Spec failed for: {file}")
50     return None
51     mod = module_from_spec(spec)
52     try:
53         spec.loader.exec_module(mod) # type: ignore[attr-defined]
54     return mod
55     except Exception as e:
56         print(f"[PluginLoader] Error while importing {file}: {e}")
57     return None

```

Listing 5.6: Dynamischer PluginLoader

Die Vorteile dieses Ansatzes sind:

- pluginbasierte Erweiterbarkeit ohne Neukompilierung,
- keine Änderungen am Kernsystem erforderlich,
- automatische Erkennung neuer Plugins mittels Reflection und Introspection.

5.5 Beispiel: AppPlugin zur Steuerung lokaler Anwendungen

Ein konkretes Beispiel für ein Plugin ist das AppPlugin. Dieses liest installierte Desktop-Anwendungen aus .desktop-Dateien aus und ermöglicht es dem Agenten, Anwendungen auf Benutzeranfrage zu starten.

```

1 from abstractions.plugin import Plugin
2 import subprocess
3 import os
4 import configparser
5 import shlex
6
7
8 class AppPlugin(Plugin):
9     def __init__(self):
10         self.prefix = "launch"
11         self.name = "AppLauncher"
12         self.prompt = (
13             "Opens a specified application, you got the following apps to choose"
14             + self.get_all_apps()
15             + "."

```

Christian Vorhofer
Raphael Ladinig

```

16     )
17
18     self.app_map = self._build_app_map()
19
20     def _build_app_map(self):
21         desktop_dirs = set()
22
23         user_dir = os.path.expanduser("~/local/share/applications")
24         desktop_dirs.add(user_dir)
25
26         xdg_data_dirs_env = os.environ.get("XDG_DATA_DIRS")
27
28         if xdg_data_dirs_env:
29             for data_dir in xdg_data_dirs_env.split(":"):
30                 if data_dir:
31                     app_dir = os.path.join(data_dir, "applications")
32                     desktop_dirs.add(app_dir)
33
34         app_map = {}
35         for directory in desktop_dirs:
36             if os.path.exists(directory):
37                 for file in os.listdir(directory):
38                     if file.endswith(".desktop"):
39                         file_path = os.path.join(directory, file)
40                         try:
41                             config = configparser.ConfigParser(interpolation=None)
42                             config.read(file_path, encoding="utf-8")
43
44                             if "Desktop Entry" in config:
45                                 de = config["Desktop Entry"]
46
47                                 if de.get("NoDisplay", "false").lower() == "true":
48                                     continue
49                                 if de.get("Hidden", "false").lower() == "true":
50                                     continue
51
52                                 name = de.get("Name")
53                                 exec_cmd = de.get("Exec")
54
55                                 if name and exec_cmd:
56                                     exec_cmd = self._cleanup_exec(exec_cmd)
57                                     app_map[name] = exec_cmd
58
59             except Exception:
60                 pass
61
62     return app_map
63
64     def _cleanup_exec(self, exec_cmd: str) -> str:
65         exec_cmd = exec_cmd.replace("%%", "%")
66         for code in ("%f", "%F", "%u", "%U", "%i", "%c", "%k"):
67             exec_cmd = exec_cmd.replace(code, "")
68         return exec_cmd.strip()
69
70     def get_all_apps(self) -> str:
71         return ";" .join(sorted(self._build_app_map().keys()))
72
73     def run(self, text: str):
74         text = text.strip()
75         try:
76             exec_cmd = self._find_exec_cmd(text)
77             if not exec_cmd:
78                 return f"App '{text}' wurde nicht gefunden."
79
80             parts = shlex.split(exec_cmd)
81             subprocess.Popen(
82                 parts, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
83             )
84             return f"Successfully started '{text}'"
85         except Exception as e:
86             return f"Error Starting '{text}': {e}"
87
88     def _find_exec_cmd(self, text: str) -> str:
89         if text in self.app_map:
90             return self.app_map[text]
91
92         for name, cmd in self.app_map.items():

```

```
91         if name.lower() == text.lower():
92             return cmd
93
94     for name, cmd in self.app_map.items():
95         if text.lower() in name.lower():
96             return cmd
97
98     return ""
```

Listing 5.7: AppPlugin als konkretes Plugin

Das Plugin implementiert die abstrakte Plugin-Schnittstelle und wird dem Agenten als Tool zur Verfügung gestellt. Im ReAct-Loop kann das Sprachmodell selbstständig entscheiden, wann der Aufruf dieses Tools sinnvoll ist (vgl. [Yao et al. \(2022\)](#), [LangChain \(2023\)](#)).

5.6 Zusammenspiel von Agent, Plugins und ReAct-Loop

Der typische Ablauf einer Anfrage kombiniert mehrere etablierte Konzepte der Agentic AI:

1. ReAct-basiertes Reasoning (vgl. [Yao et al. \(2022\)](#)),
2. LLM-gestützte Tool-Nutzung (vgl. [Schick et al. \(2023\)](#)),
3. modulare Softwarearchitekturen (vgl. [Gamma et al. \(1994\)](#)),
4. agentische Selbstorganisation (vgl. [Wang et al. \(2024\)](#)),
5. optional: multi-agentische Koordination (vgl. [Du et al. \(2023\)](#), [Hong et al. \(2023\)](#)).

Durch diese Architektur wird Tapyre zu einem vollwertigen *Agentic AI*-System. Das Sprachmodell übernimmt die Rolle einer intelligenten Steuerungsinstanz, die eigenständig plant, Tools auswählt und Entscheidungen iterativ weiterentwickelt, während die Plugin-Struktur eine maximale Erweiterbarkeit des Gesamtsystems sicherstellt.

6 Architektur und Implementierung von Tapyre Paper Search

In diesem Kapitel wird die Architektur und Implementierung von Tapyre Paper Search beschrieben. Das System dient der automatisierten Verarbeitung, Indexierung und semantischen Durchsuchung wissenschaftlicher Publikationen. Ziel ist es, große Mengen an Forschungsarbeiten aus unterschiedlichen Quellen strukturiert aufzubereiten und sowohl klassisch über Metadaten als auch semantisch über Vektorrepräsentationen durchsuchbar zu machen.

Die Architektur folgt einem modularen Ansatz mit klar getrennten Verantwortlichkeiten für Datenbeschaffung, Textverarbeitung, Embedding-Erzeugung, Speicherung und Orchestrierung. Dieses Design orientiert sich an etablierten Softwarearchitekturprinzipien und ermöglicht eine hohe Wartbarkeit, Erweiterbarkeit sowie den einfachen Austausch einzelner Komponenten.

Abgrenzung zu klassischen Big-Data-Architekturen Im Rahmen dieser Diplomarbeit wurde bewusst auf den Einsatz klassischer Big-Data-Technologien wie Hadoop, Spark oder verteilte Data-Lake-Architekturen verzichtet. Der Grund dafür liegt in den begrenzten zeitlichen, personellen und infrastrukturellen Ressourcen, die im Rahmen einer Diplomarbeit realistisch zur Verfügung stehen.

Stattdessen wurde ein ressourceneffizienter Ansatz gewählt, der sich auf die für die Umsetzung des Projekts tatsächlich notwendigen Daten und Verarbeitungsschritte konzentriert. Es werden ausschließlich jene Informationen

gespeichert, die für die semantische Suche und Verwaltung wissenschaftlicher Publikationen erforderlich sind. Dies umfasst insbesondere strukturierte Metadaten sowie kompakte Vektorrepräsentationen der Inhalte.

Dieser bewusste Verzicht auf umfangreiche Big-Data-Frameworks reduziert die Komplexität des Systems erheblich, erleichtert Deployment und Wartung und ermöglicht eine vollständige Umsetzung auf begrenzter Hardware. Gleichzeitig bleibt die Architektur modular und erweiterbar, sodass bei steigenden Datenmengen oder erweiterten Ressourcen eine spätere Skalierung und Integration zusätzlicher Technologien möglich wäre.

Im Folgenden werden die zentralen Bausteine der Architektur erläutert:

- abstrakte Kernschnittstellen für Datenquellen, Embedding-Modelle und Datenbanken,
- konkrete Implementierungen für arXiv, Specter2, Qdrant und MySQL,
- die PDF-Verarbeitung als zentrale technische Herausforderung,
- die Pipeline zur Orchestrierung des Gesamtprozesses,
- das Zusammenspiel von strukturierter und semantischer Suche.

6.1 Abstraktion der Datenquellen

Um unterschiedliche wissenschaftliche Datenquellen flexibel integrieren zu können, wird eine abstrakte Schnittstelle für Datenprovider definiert. Diese legt fest, wie neue Dokumente geladen und bereitgestellt werden, ohne dass der restliche Verarbeitungsprozess von der konkreten Quelle abhängig ist.

```

1  from abc import ABC, abstractmethod
2
3  class DataProvider(ABC):
4      @abstractmethod
5      def next(self):
6          pass
7
8      @abstractmethod
9      def hasNext(self) -> bool:
10         pass

```

Listing 6.1: Abstrakte Schnittstelle für Datenquellen

Die Schnittstelle definiert ein einheitliches Vertragsmodell für den Zugriff auf Papers und deren Metadaten. Dadurch können neue Quellen, etwa

PubMed oder lokale Archive, ergänzt werden, ohne bestehende Komponenten anpassen zu müssen. Dieses Vorgehen entspricht dem Open-Closed-Prinzip, bei dem Software für Erweiterungen offen, für Änderungen jedoch geschlossen bleibt.

6.2 ArXiv als konkrete Datenquelle

Eine konkrete Implementierung dieser Schnittstelle stellt der Datenprovider für arXiv dar. Dieser ist für das Abrufen von Metadaten sowie das Herunterladen der zugehörigen PDF-Dokumente verantwortlich.

```

1 import os
2 import re
3 import time
4 import json
5 import requests
6 from pathlib import Path
7
8 from src.core.data_provider import DataProvider
9 from src.impl.logger import get_logger
10
11
12 class ArxivDataProvider(DataProvider):
13     def __init__(self,
14                  first_id: str = "",
15                  last_id: str = "",
16                  rate_limit_seconds: float = 3.0,
17                  max_retries: int = 3,
18                  ):
19         # Logger for traceable runtime diagnostics
20         self.logger = get_logger(__name__)
21         self.logger.info("Initializing ArxivDataProvider")
22
23         # Configuration of the arXiv ID range
24         self.first_id = first_id
25         self.last_id = last_id
26
27         # Current processing state (persisted to allow resume after restart)
28         self.current_id = first_id
29         self.last_pull = 0.0           # Timestamp of last HTTP request (rate limiting)
30         self.finished = False        # Indicates whether the ID range is fully processed
31
32         # Runtime parameters for stable long-running execution
33         self.rate_limit_seconds = rate_limit_seconds
34         self.max_retries = max_retries
35
36         # Path to persistent state file (can be overridden via environment variable)
37         state_path = os.getenv("ARXIV_STATE_FILE", "/app/state/arxiv_state.json")
38         self.state_file = Path(state_path)
39
40         # Ensure that the state directory exists
41         self.state_file.parent.mkdir(parents=True, exist_ok=True)
42
43         # Restore previous processing state if available
44         self._load_state()
45
46     def _load_state(self) -> None:
47         """Load the last persisted processing state to resume after a restart."""
48         if not self.state_file.exists():
49             self.logger.info("[state] No state file found, starting from first_id.")
50
51         return

```

```

52
53     try:
54         with self.state_file.open("r", encoding="utf-8") as f:
55             state = json.load(f)
56
57             # Restore only the required fields
58             saved_current = state.get("current_id")
59             saved_finished = state.get("finished", False)
60
61             if saved_current:
62                 self.logger.info(
63                     f"[state] Restoring state from {self.state_file}: "
64                     f"current_id={saved_current}, finished={saved_finished}"
65                 )
66                 self.current_id = saved_current
67                 self.finished = saved_finished
68     except Exception as e:
69         # State loading errors must not crash the pipeline
70         self.logger.error(f"[state] Failed to load state file: {e}", exc_info=True)
71
72     def _save_state(self) -> None:
73         """Persist the current processing state atomically."""
74         tmp_file = self.state_file.with_suffix(".tmp")
75
76         # Minimal state required for an exact resume
77         data = {
78             "current_id": self.current_id,
79             "first_id": self.first_id,
80             "last_id": self.last_id,
81             "finished": self.finished,
82             "last_pull": self.last_pull,
83         }
84
85         try:
86             # Write to a temporary file first
87             with tmp_file.open("w", encoding="utf-8") as f:
88                 json.dump(data, f)
89
90             # Atomically replace the previous state file
91             os.replace(tmp_file, self.state_file)
92
93             self.logger.debug(
94                 f"[state] Saved state: current_id={self.current_id}, finished={self.finished}"
95             )
96     except Exception as e:
97         self.logger.error(f"[state] Failed to save state: {e}", exc_info=True)
98
99         # Cleanup temporary file if necessary
100        if tmp_file.exists():
101            try:
102                tmp_file.unlink()
103            except OSError:
104                pass
105
106    def hasNext(self) -> bool:
107        """Return True if further documents are available in the configured ID range."""
108        self.logger.debug(f"[hasNext] Current ID: {self.current_id}, finished={self.finished}")
109        return not self.finished
110
111    def next(self):
112        """
113            Fetch the next available PDF from arXiv.
114
115            Workflow:
116            1) Compute the next arXiv ID
117            2) Download the corresponding PDF
118            3) Persist progress
119            4) Skip IDs without an available document
120        """
121        if self.finished:
122            self.logger.info("[next] No more IDs to process      finished.")
123            return None
124
125        while True:
126            # Compute the next candidate ID

```

```

127     next_id = self._get_next_id(self.current_id)
128     self.logger.debug(f"[next] Next ID candidate: {next_id}")
129
130     # Termination conditions
131     if next_id == self.last_id or next_id is None:
132         self.logger.info("[next] Reached last ID or invalid next ID      marking finished.")
133         self.finished = True
134         self._save_state()
135         return None
136
137     self.current_id = next_id
138
139     # Download the PDF with rate limiting and retry logic
140     pdf_data = self._fetch_pdf(next_id)
141
142     if pdf_data and len(pdf_data) > 0:
143         self.logger.info(
144             f"[next] Successfully fetched PDF for {next_id} ({len(pdf_data)} bytes)"
145         )
146         self._save_state()
147         break
148
149     # If no valid document was found, continue with the next ID
150     self.logger.warning(f"[next] No valid PDF found for {next_id}, continuing...")
151
152     return next_id, pdf_data
153
154 def _get_next_id(self, current_id: str) -> str | None:
155     """
156     Compute the next arXiv identifier.
157
158     Format: arXiv:YYMM.NNNNN (optional version suffix vX).
159     Handles numeric roll-over and basic validity checks.
160     """
161     match = re.match(r'arXiv:(\d{2})(\d{2})\.(\d{4,5})(?:v\d+)?', current_id)
162     if not match:
163         self.logger.warning(f"[get_next_id] Invalid current ID format: {current_id}")
164         return None
165
166     yy, mm, num = map(int, match.groups())
167
168     ROLLOVER_LIMIT = 10000
169     num += 1
170
171     if num >= ROLLOVER_LIMIT:
172         num = 0
173         mm += 1
174         if mm > 12:
175             mm = 1
176             yy += 1
177
178         # Safety check to prevent invalid ranges or infinite loops
179         if yy > 99 or (yy == 7 and mm < 4):
180             return None
181
182     next_id = f'arXiv:{yy:02d}{mm:02d}.{num:05d}'
183     return next_id
184
185 def _fetch_pdf(self, paper_id: str) -> bytes:
186     """
187     Download a PDF from arXiv.
188
189     Features:
190     - Rate limiting between requests
191     - Retry strategy for transient failures
192     - Immediate abort on 404 (document does not exist)
193     """
194     attempts = 0
195     while attempts < self.max_retries:
196         # Enforce rate limiting
197         now = time.time()
198         wait_time = self.last_pull + self.rate_limit_seconds - now
199         if wait_time > 0:
200             self.logger.debug(f"[fetch_pdf] Rate limiting: sleeping {wait_time:.2f}s")
201             time.sleep(wait_time)

```

```

202         self.last_pull = time.time()
203
204     # Extract numeric arXiv ID for URL construction
205     match = re.match(r'arXiv:(\d{4,6}.\d{4,5})', paper_id)
206     if not match:
207         self.logger.error(f"[fetch_pdf] Invalid arXiv ID format: {paper_id}")
208         return b""
209
210     arxiv_id = match.group(1)
211     url = f"https://arxiv.org/pdf/{arxiv_id}.pdf"
212     self.logger.info(
213         f"[fetch_pdf] Fetching PDF from {url} (attempt {attempts+1}/{self.max_retries})"
214     )
215
216
217     try:
218         response = requests.get(url, timeout=10)
219         status = response.status_code
220
221         if status == 200:
222             self.logger.debug(
223                 f"[fetch_pdf] 200 OK for {arxiv_id} ({len(response.content)} bytes)"
224             )
225             return response.content
226
227         elif status == 404:
228             # Document does not exist    no retries required
229             self.logger.warning(f"[fetch_pdf] 404 Not Found for {arxiv_id}")
230             return b""
231
232         else:
233             # Unexpected HTTP status    retry with large backoff
234             attempts += 1
235             backoff = 5 * 60 * 60
236             self.logger.critical(
237                 f"[fetch_pdf] Unexpected HTTP {status} for {arxiv_id}, "
238                 f"retrying in {backoff}s (attempt {attempts}/{self.max_retries})"
239             )
240             time.sleep(backoff)
241
242     except requests.exceptions.RequestException as e:
243         # Transient network error    retry with increasing backoff
244         attempts += 1
245         backoff = min(600, 100 * attempts)
246         self.logger.error(
247             f"[fetch_pdf] Request exception for {arxiv_id}: {e}, "
248             f"retrying in {backoff}s (attempt {attempts}/{self.max_retries})",
249             exc_info=True
250         )
251         time.sleep(backoff)
252
253     self.logger.critical(f"[fetch_pdf] Max retries reached for {paper_id}, giving up.")
254     return b""

```

Listing 6.2: arXivDataProvider zur Anbindung der arXiv-API

Der Provider verarbeitet unter anderem Titel, Autoren, Abstracts, Kategorien sowie die URL des PDF-Dokuments. Durch die klare Trennung zwischen Datenbeschaffung und nachgelagerter Verarbeitung bleibt das System robust gegenüber API-Änderungen und leicht auf weitere Quellen übertragbar.

Christian Vorhofer
 Raphael Ladinig

6.3 PDF-Verarbeitung und Textextraktion

Die Umwandlung wissenschaftlicher PDF-Dokumente in maschinenlesbaren Text stellt eine der größten technischen Herausforderungen dar. PDFs enthalten häufig mehrspaltige Layouts, mathematische Formeln, Fußnoten sowie Seitenköpfe, die eine saubere Textextraktion erschweren.

```

1  from src.core.pdf_converter import PdfConverter
2  from src.impl.logger import get_logger
3
4  import fitz
5  from datetime import datetime, timedelta, timezone
6  import re
7  from typing import Optional
8  import unicodedata
9
10
11 class FitzPdfConverter(PdfConverter):
12     def __init__(self):
13         # Logger for traceable PDF processing and error diagnostics
14         self.logger = get_logger(__name__)
15         self.logger.debug("FitzPdfConverter initialized.")
16
17     def pdf_to_string(self, pdf) -> str:
18         """
19             Extract plain text from a PDF.
20
21             The input can either be a file path or raw PDF bytes.
22             Text is extracted page by page to allow partial recovery
23             in case of corrupted or problematic pages.
24         """
25
26         # Validate input type
27         if isinstance(pdf, str):
28             self.logger.info("Opening PDF from file path.")
29         elif isinstance(pdf, bytes):
30             self.logger.info("Opening PDF from bytes stream.")
31             self.logger.debug("PDF bytes length: %s", len(pdf))
32         else:
33             self.logger.error("Invalid input type for pdf_to_string: %s", type(pdf))
34             raise ValueError("Input must be a file path or PDF binary content")
35
36         doc = None
37         full_text = ""
38         error_pages = 0
39
40         try:
41             # Open PDF either from file system or in-memory stream
42             if isinstance(pdf, str):
43                 doc = fitz.open(pdf)
44             else:
45                 doc = fitz.open(stream=pdf, filetype="pdf")
46
47             page_count = len(doc)
48             self.logger.debug("PDF opened successfully. Page count: %d", page_count)
49
50             # Extract text page by page to isolate failures
51             for i, page in enumerate(doc, start=1):
52                 try:
53                     text = page.get_text()
54                     full_text += text
55
56                     # Periodic progress logging for large documents
57                     if i % 10 == 0 or i == page_count:
58                         self.logger.debug(
59                             "Extracted text up to page %d/%d (current page length=%d).",
60                             i, page_count, len(text)
61                         )
62                 except Exception as e:
63                     error_pages += 1

```

```

63             self.logger.error(
64                 "Failed to extract text from page %d/%d: %s      skipping this page.",
65                 i, page_count, e
66             )
67             continue
68
69         if error_pages > 0:
70             self.logger.warning(
71                 "Text extraction finished with %d page errors (total pages=%d).",
72                 error_pages, page_count
73             )
74
75         self.logger.info(
76             "Finished extracting text from PDF. Total length: %d characters.",
77             len(full_text)
78         )
79         return full_text
80
81     finally:
82         # Ensure the document is always closed
83         if doc is not None:
84             doc.close()
85         self.logger.debug("PDF document closed after text extraction.")
86
87     def pdf_metadata(self, pdf) -> dict:
88         """
89             Extract selected metadata fields from a PDF.
90
91             Metadata is normalized and date fields are parsed
92             into timezone-aware datetime objects.
93         """
94
95         # Validate input type
96         if isinstance(pdf, str):
97             self.logger.info("Opening PDF from file path for metadata.")
98         elif isinstance(pdf, bytes):
99             self.logger.info("Opening PDF from bytes stream for metadata.")
100            self.logger.debug("PDF bytes length: %s", len(pdf))
101        else:
102            self.logger.error("Invalid input type for pdf_metadata: %s", type(pdf))
103            raise ValueError("Input must be a file path or PDF binary content")
104
105        doc = None
106        try:
107            # Open PDF for metadata access
108            try:
109                if isinstance(pdf, str):
110                    doc = fitz.open(pdf)
111                else:
112                    doc = fitz.open(stream=pdf, filetype="pdf")
113            except Exception as e:
114                self.logger.error(
115                    "Failed to open PDF for metadata extraction: %s", e, exc_info=True
116                )
117            return {}
118
119        metadata = doc.metadata or {}
120        self.logger.debug("Raw metadata keys: %s", list(metadata.keys()))
121
122        # Whitelist of metadata fields relevant for downstream processing
123        expected_keys = [
124            'title', 'author', 'subject', 'keywords',
125            'creator', 'producer', 'creationDate',
126            'modDate', 'trapped'
127        ]
128
129        complete_metadata = {key: metadata.get(key) for key in expected_keys}
130
131        # Parse PDF date fields into datetime objects
132        if complete_metadata.get('creationDate'):
133            original = complete_metadata['creationDate']
134            parsed = self.parse_pdf_date(original)
135            complete_metadata['creationDate'] = parsed
136            self.logger.debug("Parsed creationDate: %s -> %s", original, parsed)
137
138        if complete_metadata.get('modDate'):
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589

```

```

138         original = complete_metadata['modDate']
139         parsed = self.parse_pdf_date(original)
140         complete_metadata['modDate'] = parsed
141         self.logger.debug("Parsed modDate: %s -> %s", original, parsed)
142
143         self.logger.info("Metadata extracted successfully.")
144         return complete_metadata
145
146     finally:
147         # Always close the document
148         if doc is not None:
149             doc.close()
150             self.logger.debug("PDF document closed after metadata extraction.")
151
152     def clean_string(self, text: str) -> str:
153         """
154             Normalize and clean extracted text.
155
156             This includes Unicode normalization, whitespace cleanup,
157             dash normalization and removal of invisible characters.
158         """
159         before_len = len(text)
160
161         # Normalize Unicode characters to a canonical form
162         text = unicodedata.normalize("NFKC", text)
163
164         # Normalize line endings
165         text = text.replace('\r\n', '\n').replace('\r', '\n')
166
167         # Replace uncommon Unicode whitespace characters
168         WHITESPACE_CHARS = [
169             "\u00AO",
170             "\u2007",
171             "\u202F",
172             "\u2009",
173             "\u2002", "\u2003", "\u2004", "\u2005", "\u2006",
174             "\u2008", "\u200A",
175             "\u3000",
176             "\u180E",
177             "\u200B", "\u200C", "\u200D", "\u2060",
178         ]
179         for ch in WHITESPACE_CHARS:
180             text = text.replace(ch, ' ')
181
182         # Normalize dash variants
183         text = text.replace('‐', '‐')
184         text = text.replace('‐', '‐')
185         text = text.replace('‐', '‐')
186
187         # Remove soft hyphen
188         text = text.replace('‐', '')
189
190         # Flatten text into a single line
191         text = text.replace('\n', ' ')
192
193         # Collapse repeated whitespace
194         text = ' '.join(text.split())
195
196         after_len = len(text)
197         self.logger.debug("Cleaned string: length %d -> %d.", before_len, after_len)
198         return text
199
200     def chunk_string(self, text: str, chunk_size=500, overlap=50) -> list[str]:
201         """
202             Split text into overlapping chunks.
203
204             This is primarily used to prepare text for downstream
205             NLP tasks such as embeddings or retrieval.
206         """
207         self.logger.info(
208             "Chunking text with chunk_size=%d and overlap=%d.", chunk_size, overlap
209         )
210
211         chunks = []
212         start = 0

```

```

213         text_len = len(text)
214
215         while start < text_len:
216             end = min(start + chunk_size, text_len)
217             chunks.append(text[start:end])
218             start += chunk_size - overlap
219
220             self.logger.info(
221                 "Produced %d chunks from text of length %d.", len(chunks), text_len
222             )
223
224     return chunks
225
226     def parse_pdf_date(self, pdf_date_str) -> Optional[datetime]:
227         """
228             Parse PDF date strings of the form:
229             D:YYYYMMDDHHmmSS+HH'mm'
230
231             Returns a timezone-aware datetime or None if parsing fails.
232         """
233         if not pdf_date_str or not str(pdf_date_str).startswith('D:'):
234             self.logger.warning(
235                 "PDF date string missing or not starting with 'D:: %s", pdf_date_str
236             )
237             return None
238
239         original = pdf_date_str
240         pdf_date_str = pdf_date_str[2:]
241
242         match = re.match(
243             r"(\d{4})(\d{2})(\d{2})(\d{2})(\d{2})(\d{2})([+-])(\d{2})'?(\\d{2})?'?", pdf_date_str
244         )
245
246         if not match:
247             self.logger.warning("Failed to parse PDF date string: %s", original)
248             return None
249
250         year, month, day, hour, minute, second, tz_sign, tz_hour, tz_minute = match.groups()
251         dt = datetime(int(year), int(month), int(day), int(hour), int(minute), int(second))
252
253         offset = timedelta(hours=int(tz_hour or 0), minutes=int(tz_minute or 0))
254         if tz_sign == '-':
255             offset = -offset
256
257         parsed = dt.replace(tzinfo=timezone(offset))
258         self.logger.debug("Parsed PDF date '%s' into datetime '%s'.", original, parsed)
259
260     return parsed

```

Listing 6.3: PDF-zu-Text-Konvertierung mit PyMuPDF

Die Implementierung basiert auf der Bibliothek PyMuPDF (fitz) und extrahiert den Text seitenweise. Dabei wird bewusst auf komplexe Layout-Rekonstruktionen verzichtet, um eine hohe Verarbeitungsgeschwindigkeit zu gewährleisten. Diese Lösung stellt einen praxisnahen Kompromiss zwischen Textqualität und Performance dar und eignet sich besonders für die Verarbeitung großer Paper-Sammlungen, wie sie in diesem Projekt anfallen.

Christian Vorhofer
 Raphael Ladinig

6.4 Abstraktion der Embedding-Erzeugung

Für die semantische Suche müssen Texte in hochdimensionale Vektoren überführt werden. Um unterschiedliche Embedding-Modelle flexibel einsetzen zu können, wird eine abstrakte Embedder-Schnittstelle definiert.

```

1  from abc import ABC, abstractmethod
2
3  class Embedder(ABC):
4      @abstractmethod
5      def embed(self, text: str):
6          pass

```

Listing 6.4: Abstrakte Embedder-Schnittstelle

Diese Schnittstelle kapselt die konkrete Modellimplementierung vollständig. Dadurch können verschiedene Modelle getestet, ausgetauscht oder parallel verwendet werden, ohne dass Änderungen an der Pipeline oder den Datenbankschichten notwendig sind. Diese Abstraktion ist insbesondere für experimentelle Evaluierungen von Vorteil.

6.5 Specter2 als semantisches Embedding-Modell

Als konkrete Implementierung kommt das Embedding-Modell Specter2 zum Einsatz. Dieses Transformer-basierte Modell wurde speziell für wissenschaftliche Texte trainiert und erzeugt Vektorrepräsentationen, die den inhaltlichen Kern eines Papers erfassen.

```

1  from src.core.embedder import Embedder
2  from src.impl.logger import get_logger
3
4  from typing import List, Union
5  from transformers import AutoTokenizer, AutoModel
6  import torch
7
8
9  class Specter2Embedder(Embedder):
10     def __init__(self, model_name: str = "allenai/specter2_base"):
11         # Logger for model loading, device selection and runtime diagnostics
12         self.logger = get_logger(__name__)
13         self.logger.info("[Specter2Embedder] Initializing model: %s", model_name)
14
15         # Select the most suitable computation device (CUDA > MPS > CPU)
16         if torch.cuda.is_available():
17             self.device = torch.device("cuda")
18             device_name = torch.cuda.get_device_name(0)
19             cap = torch.cuda.get_device_capability(0)
20             self.logger.info(
21                 "[CUDA] device=%s cap=%d.%d torch_cuda=%s",
22                 device_name, cap[0], cap[1], torch.version.cuda
23             )
24

```

```

25         # Enable cuDNN autotuning for improved performance on fixed input sizes
26         torch.backends.cudnn.benchmark = True
27
28     elif torch.backends.mps.is_available():
29         # Apple Metal backend (e.g. macOS with Apple Silicon)
30         self.device = torch.device("mps")
31         device_name = "Apple MPS (Metal)"
32
33     else:
34         # Fallback for systems without hardware acceleration
35         self.device = torch.device("cpu")
36         device_name = "CPU"
37
38     self.logger.info("[Specter2Embedder] Using device: %s", device_name)
39
40     try:
41         # Load tokenizer and model weights from Hugging Face
42         self.tokenizer = AutoTokenizer.from_pretrained(model_name)
43         self.model = AutoModel.from_pretrained(
44             model_name,
45             use_safetensors=True,
46             trust_remote_code=False,
47         )
48
49         # Move model to selected device and switch to inference mode
50         self.model.to(self.device)
51         self.model.eval()
52
53         self.logger.info("[Specter2Embedder] Model and tokenizer loaded successfully.")
54
55     except Exception as e:
56         # Model loading errors are considered fatal
57         self.logger.exception(
58             "[Specter2Embedder] Failed to load model '%s': %s", model_name, e
59         )
60         raise
61
62     @torch.inference_mode()
63     def embed(self, text: Union[str, List[str]]) -> torch.Tensor:
64         """
65             Generate vector embeddings for one or multiple input texts.
66
67             The method supports both single strings and batches and returns
68             a tensor of embeddings suitable for downstream retrieval or indexing.
69             """
70
71         # Normalize input to batch format
72         if isinstance(text, str):
73             self.logger.debug("[Specter2Embedder] Received single string for embedding.")
74             text = [text]
75         else:
76             self.logger.debug(
77                 "[Specter2Embedder] Received batch of %d texts for embedding.", len(text)
78             )
79
80         try:
81             # Tokenize input text without truncation (document-level embeddings)
82             encoded_input = self.tokenizer(
83                 text,
84                 padding=True,
85                 truncation=False,
86                 return_tensors="pt",
87             )
88
89             # Move tokenized input to the selected device
90             encoded_input = {
91                 k: v.to(self.device, non_blocking=True)
92                 for k, v in encoded_input.items()
93             }
94
95             self.logger.debug(
96                 "[Specter2Embedder] Tokenized input successfully (seq_len=%d).",
97                 int(encoded_input["input_ids"].shape[1])
98             )
99
# Use mixed precision on CUDA for faster inference

```

Christian Vorhofer
Raphael Ladinig

```

100     if self.device.type == "cuda":
101         with torch.amp.autocast(device_type="cuda", dtype=torch.float16):
102             model_output = self.model(**encoded_input)
103         else:
104             model_output = self.model(**encoded_input)
105
106         # Use the CLS token representation as document embedding
107         embeddings = model_output.last_hidden_state[:, 0, :]
108         self.logger.debug(
109             "[Specter2Embedder] Generated embeddings shape: %s",
110             tuple(embeddings.shape)
111         )
112
113         # Detach from computation graph and move to CPU for safe downstream usage
114         return embeddings.detach().to("cpu")
115
116     except Exception as e:
117         # Embedding failures are logged and propagated to the caller
118         self.logger.exception("[Specter2Embedder] Error during embedding: %s", e)
119         raise

```

Listing 6.5: Specter2Embedder zur Erzeugung semantischer Vektoren

Durch den Einsatz eines domänenspezifischen Modells wird eine deutlich bessere semantische Abbildung wissenschaftlicher Inhalte erreicht als mit generischen Sprachmodellen. Dies ist insbesondere für die thematische Suche und Ähnlichkeitsbewertung von Publikationen entscheidend.

6.6 Abstraktion der Datenhaltung

Das System unterscheidet bewusst zwischen strukturierter Datenhaltung für Metadaten und vektorbasierten Repräsentationen für semantische Suche. Eine abstrakte Datenbankschnittstelle definiert die hierfür notwendigen Operationen.

```

1  from abc import ABC, abstractmethod
2
3  class Database(ABC):
4      def __init__(self):
5          self.connect()
6
7      @abstractmethod
8      def connect(self):
9          pass
10
11     def __del__(self):
12         self.disconnect()
13
14     @abstractmethod
15     def disconnect(self):
16         pass

```

Listing 6.6: Abstrakte Datenbankschnittstelle

Diese Trennung erlaubt es, unterschiedliche Datenbanktypen gezielt für ihre jeweiligen Stärken einzusetzen und bei Bedarf auszutauschen.

Christian Vorhofer
 Raphael Ladinig

6.7 MySQL für strukturierte Metadaten

Metadaten wie Titel, Autoren, Kategorien oder Statusinformationen werden in einer relationalen Datenbank gespeichert.

```

1 import os
2 from datetime import datetime
3
4 from sqlalchemy import create_engine, text
5 from sqlalchemy.orm import sessionmaker, scoped_session
6 from sqlalchemy.exc import SQLAlchemyError, IntegrityError, DataError
7
8 from src.models.paper import Paper
9 from src.models.base import Base
10 from src.core.database import Database
11 from src.impl.logger import get_logger
12
13
14 class MySQLDatabase(Database):
15     def __init__(self):
16         # Logger for connection lifecycle and database operations
17         self.logger = get_logger(__name__)
18         self.logger.info("[MySQLDatabase] Initialization started")
19
20         # Read database configuration from environment variables
21         user = os.getenv("MYSQL_USER")
22         password = os.getenv("MYSQL_PASSWORD")
23         host = os.getenv("MYSQL_HOST", "mysql_db")
24         database = os.getenv("MYSQL_DATABASE")
25
26         self.logger.debug(
27             "[MySQLDatabase] Env vars - USER: %s, HOST: %s, DB: %s",
28             user, host, database
29         )
30
31         # Ensure mandatory configuration values are present
32         if not all([user, password, database]):
33             raise ValueError(
34                 "[MySQLDatabase] Missing env vars: MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE"
35             )
36
37         # Construct SQLAlchemy connection URL
38         self.db_url = f"mysql+pymysql://user:{password}@{host}/{database}"
39         self.logger.info("[MySQLDatabase] Connection URL: %s", self.db_url)
40
41         self.engine = None
42         self.Session = None
43
44     try:
45         # Initialize base Database class
46         super().__init__()
47         self.logger.info("[MySQLDatabase] super().__init__ successful")
48     except Exception as e:
49         self.logger.exception("[MySQLDatabase] Error calling super(): %s", e)
50         raise
51
52     try:
53         # Actual connection setup is deferred to connect()
54         self.logger.info("[MySQLDatabase] Connection established successfully")
55     except Exception as e:
56         self.logger.exception("[MySQLDatabase] Error while establishing connection: %s", e)
57         raise
58
59     def connect(self):
60         """
61             Create SQLAlchemy engine and scoped session.
62             Also ensures that all database tables are created.
63             """
64             self.logger.info("[MySQLDatabase] Creating engine and session...")
65
66         # pool_pre_ping avoids stale connections in long-running applications

```

Christian Vorhofer
 Raphael Ladinig

```

67         self.engine = create_engine(
68             self.db_url,
69             echo=False,
70             pool_pre_ping=True
71         )
72         self.Session = scoped_session(sessionmaker(bind=self.engine))
73
74         self.logger.info("[MySQLDatabase] Engine and Session created. Creating tables...")
75         Base.metadata.create_all(self.engine)
76         self.logger.info("[MySQLDatabase] Tables created")
77
78     def disconnect(self):
79         """Cleanly dispose database resources."""
80         self.logger.info("[MySQLDatabase] Disconnecting from database...")
81
82         if self.Session:
83             self.Session.remove()
84         if self.engine:
85             self.engine.dispose()
86
87         self.logger.info("[MySQLDatabase] Disconnected")
88
89     def get_session(self):
90         """Return a new scoped SQLAlchemy session."""
91         return self.Session()
92
93     def add_paper(
94         self,
95         arxiv_id: str,
96         text: str,
97         title: str = None,
98         author: str = None,
99         subject: str = None,
100        keywords: str = None,
101        creator: str = None,
102        producer: str = None,
103        creation_date: datetime = None,
104        modification_date: datetime = None,
105        trapped: str = None,
106        skip_if_exists: bool = True,
107    ):
108        """
109        Insert a paper into the database.
110
111        If skip_if_exists is enabled, the method checks for an existing
112        entry and avoids duplicate inserts.
113        """
114        session = self.get_session()
115        try:
116            # Optional existence check to avoid duplicate inserts
117            if skip_if_exists and self.paper_exists(arxiv_id):
118                self.logger.info(
119                    "[MySQLDatabase] Paper %s already exists, skipping insert.", arxiv_id
120                )
121                return arxiv_id
122
123            self.logger.info("[MySQLDatabase] Inserting paper: %s", arxiv_id)
124
125            paper = Paper(
126                arxiv_id=arxiv_id,
127                text=text,
128                title=title,
129                author=author,
130                subject=subject,
131                keywords=keywords,
132                creator=creator,
133                producer=producer,
134                creation_date=creation_date,
135                modification_date=modification_date,
136                trapped=trapped,
137            )
138
139            session.add(paper)
140            session.commit()
141

```

```

142         self.logger.info("[MySQLDatabase] Paper saved: %s", arxiv_id)
143         return arxiv_id
144
145     except IntegrityError as e:
146         # Constraint violation (e.g. duplicate primary key)
147         session.rollback()
148         self.logger.warning(
149             "[MySQLDatabase] IntegrityError on %s, skipping. Detail: %s",
150             arxiv_id, e
151         )
152         return None
153
154     except DataError as e:
155         # Typically triggered by invalid or oversized field values
156         session.rollback()
157         author_len = len(author) if author is not None else 0
158         self.logger.warning(
159             "[MySQLDatabase] DataError on %s (author_len=%d), skipping. Detail: %s",
160             arxiv_id, author_len, e
161         )
162         return None
163
164     except SQLAlchemyError as e:
165         # Catch-all for unexpected database errors
166         session.rollback()
167         self.logger.exception(
168             "[MySQLDatabase] Unexpected SQLAlchemyError on %s: %s",
169             arxiv_id, e
170         )
171         return None
172
173     finally:
174         session.close()
175
176 def get_paper_by_arxiv_id(self, arxiv_id: str):
177     """Fetch a single paper by its arXiv identifier."""
178     session = self.get_session()
179     try:
180         self.logger.info("[MySQLDatabase] Fetching paper by ID: %s", arxiv_id)
181         return session.query(Paper).filter_by(arxiv_id=arxiv_id).first()
182     finally:
183         session.close()
184
185 def get_all_papers(self):
186     """Return all stored papers."""
187     session = self.get_session()
188     try:
189         self.logger.info("[MySQLDatabase] Fetching all papers")
190         return session.query(Paper).all()
191     finally:
192         session.close()
193
194 def delete_paper(self, arxiv_id: str):
195     """Delete a paper by its arXiv ID."""
196     session = self.get_session()
197     try:
198         self.logger.info("[MySQLDatabase] Deleting paper with ID: %s", arxiv_id)
199
200         paper = session.query(Paper).filter_by(arxiv_id=arxiv_id).first()
201         if paper:
202             session.delete(paper)
203             session.commit()
204             self.logger.info("[MySQLDatabase] Paper deleted: %s", arxiv_id)
205             return True
206
207         self.logger.info("[MySQLDatabase] Paper not found: %s", arxiv_id)
208         return False
209
210     except SQLAlchemyError as e:
211         session.rollback()
212         self.logger.exception(
213             "[MySQLDatabase] Error deleting paper %s: %s",
214             arxiv_id, e
215         )
216         return False

```

```

217     finally:
218         session.close()
219
220     def get_statistics(self):
221         """
222             Gather basic database statistics such as number of papers,
223             date range and approximate database size.
224         """
225         session = self.get_session()
226         try:
227             self.logger.info("[MySQLDatabase] Gathering database statistics")
228
229             total_papers = session.query(Paper).count()
230             latest_paper = session.query(Paper).order_by(Paper.creation_date.desc()).first()
231             earliest_paper = session.query(Paper).order_by(Paper.creation_date.asc()).first()
232
233             # Resolve database name for size query
234             db_name = (
235                 self.engine.url.database if self.engine else None
236                 ) or os.getenv("MYSQL_DATABASE")
237
238             db_size = None
239             if self.engine and db_name:
240                 with self.engine.connect() as conn:
241                     result = conn.execute(
242                         text("""
243                             SELECT ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
244                             FROM information_schema.tables
245                             WHERE table_schema = :db
246                         """),
247                         {"db": db_name},
248                     )
249                     db_size = result.scalar_one_or_none()
250
251             stats = {
252                 "total_papers": total_papers,
253                 "latest_paper_date": latest_paper.creation_date if latest_paper else None,
254                 "earliest_paper_date": earliest_paper.creation_date if earliest_paper else None,
255                 "database_size_mb": db_size,
256             }
257
258             self.logger.info("[MySQLDatabase] Statistics: %s", stats)
259             return stats
260
261         finally:
262             session.close()
263
264     def paper_exists(self, arxiv_id: str) -> bool:
265         """
266             Check whether a paper with the given arXiv ID exists.
267         """
268         session = self.get_session()
269         try:
270             return session.get(Paper, arxiv_id) is not None
271         finally:
272             session.close()

```

Listing 6.7: MySQL-Datenbankanbindung für Paper-Metadaten

Relationale Datenbanken eignen sich besonders für konsistente Speicherung, relationale Abfragen und Filteroperationen, wie sie bei Metadaten häufig erforderlich sind.

Christian Vorhofer
 Raphael Ladinig

6.8 Qdrant als Vektordatenbank

Für die semantische Suche werden die erzeugten Embeddings in einer spezialisierten Vektordatenbank gespeichert. Hierfür kommt Qdrant zum Einsatz, das effiziente Approximate-Nearest-Neighbor-Suchen ermöglicht.

```

1 import os
2 import uuid
3 import time
4
5 from qdrant_client import QdrantClient
6 from qdrant_client.http.models import PointStruct
7 from qdrant_client.models import Distance, VectorParams
8
9 from src.core.database import Database
10 from src.impl.logger import get_logger
11
12
13 class QdrantDatabase(Database):
14     def __init__(self):
15         # Logger for connection lifecycle and vector operations
16         self.logger = get_logger(__name__)
17
18         # Configuration via environment variables
19         self.collection_name = os.getenv("QDRANT_COLLECTION", "chunks")
20         self.host = os.getenv("QDRANT_HOST", "qdrant_db")
21         self.port = int(os.getenv("QDRANT_PORT", 6333))
22         self.vector_size = int(os.getenv("VECTOR_SIZE", 768))
23
24         self.client = None
25
26         self.logger.info(
27             "[QdrantDatabase] Initialized with host=%s, port=%d, collection=%s, vector_size=%d",
28             self.host,
29             self.port,
30             self.collection_name,
31             self.vector_size,
32         )
33
34     try:
35         # Initialize base Database abstraction
36         super().__init__()
37         self.logger.debug("[QdrantDatabase] Super init successful")
38     except Exception as e:
39         self.logger.exception("[QdrantDatabase] Error during super init: %s", e)
40         raise
41
42     def connect(self):
43         """
44             Establish a connection to Qdrant and ensure that the target collection exists.
45             If the collection does not exist, it is created with the configured vector size
46             and cosine similarity as distance metric.
47         """
48         self.logger.info(
49             "[QdrantDatabase] Connecting to Qdrant at %s:%d...", self.host, self.port
50         )
51         self.client = QdrantClient(host=self.host, port=self.port)
52
53     try:
54         # Check whether the collection already exists
55         self.client.get_collection(self.collection_name)
56         self.logger.info(
57             "[QdrantDatabase] Collection '%s' already exists.", self.collection_name
58         )
59     except Exception:
60         # Collection does not exist      create a new one
61         self.logger.warning(
62             "[QdrantDatabase] Collection '%s' not found. Creating a new one...", self.collection_name,
63         )

```



```

140         payload={
141             "arxiv_id": arxiv_id,
142             "text": text,
143         },
144     )
145
146     attempt = 0
147     while attempt < max_retries:
148         attempt += 1
149         try:
150             # Upsert the point into the collection (synchronous write)
151             self.client.upsert(
152                 collection_name=self.collection_name,
153                 points=[point],
154                 wait=True,
155             )
156             self.logger.info(
157                 "[QdrantDatabase] Added chunk %s for paper %s (attempt %d/%d)",
158                 pk,
159                 arxiv_id,
160                 attempt,
161                 max_retries,
162             )
163         return pk
164
165     except (
166         ResponseHandlingException,
167         httpx.RemoteProtocolError,
168         httpcore.RemoteProtocolError,
169     ) as e:
170         # Transient transport-level errors      retry with exponential backoff
171         if attempt >= max_retries:
172             self.logger.exception(
173                 "[QdrantDatabase] Giving up on chunk %s for paper %s after %d attempts: %s",
174                 pk,
175                 arxiv_id,
176                 attempt,
177                 e,
178             )
179         raise
180
181     delay = min(base_delay * (2 ** (attempt - 1)), 30.0)
182     self.logger.warning(
183         "[QdrantDatabase] Transient error when upserting chunk %s "
184         "for paper %s (attempt %d/%d): %s      retrying in %.1fs",
185         pk,
186         arxiv_id,
187         attempt,
188         max_retries,
189         e,
190         delay,
191     )
192     time.sleep(delay)
193
194     except Exception as e:
195         # Non-recoverable error      do not retry
196         self.logger.exception(
197             "[QdrantDatabase] Failed to upsert chunk %s for paper %s (no retry): %s",
198             pk,
199             arxiv_id,
200             e,
201         )
202     raise
203
204 def get_similar(self, embedding: list[float], top_k: int = 5):
205     """
206         Perform a similarity search for the given embedding and return the
207         top_k most similar chunks.
208     """
209     self.logger.debug(
210         "[QdrantDatabase] Searching for top %d similar embeddings.", top_k
211     )
212
213     # Normalize embedding format
214     if hasattr(embedding, "tolist"):

```

```

215         embedding = embedding.tolist()
216
217     if isinstance(embedding, list) and isinstance(embedding[0], list):
218         embedding = embedding[0]
219
220     try:
221         search_result = self.client.search(
222             collection_name=self.collection_name,
223             query_vector=embedding,
224             limit=top_k,
225         )
226         self.logger.info(
227             "[QdrantDatabase] Found %d similar vectors.", len(search_result)
228         )
229     return search_result
230 except Exception as e:
231     self.logger.exception(
232         "[QdrantDatabase] Error during similarity search: %s", e
233     )
234     raise
235
236 def get_statistics(self):
237     """
238     Retrieve basic statistics for the configured Qdrant collection.
239     """
240     self.logger.info(
241         "[QdrantDatabase] Fetching collection statistics for '%s'",
242         self.collection_name,
243     )
244     try:
245         stats = self.client.get_collection(self.collection_name)
246         data = {
247             "points_count": stats.points_count,
248             "segments_count": getattr(stats, "segments_count", None),
249         }
250         self.logger.info("[QdrantDatabase] Statistics: %s", data)
251     return data
252 except Exception as e:
253     self.logger.exception(
254         "[QdrantDatabase] Failed to fetch collection stats: %s", e
255     )
256     raise

```

Listing 6.8: Qdrant-Datenbank für semantische Suche

Durch den Einsatz HNSW-basierter Indexstrukturen können auch sehr große Paper-Sammlungen performant durchsucht werden, was eine zentrale Voraussetzung für semantische Suchanwendungen darstellt.

6.9 Pipeline zur Orchestrierung des Gesamtprozesses

Die Pipeline bildet das zentrale Orchestrierungselement des Systems. Sie verbindet Datenquelle, PDF-Verarbeitung, Embedding-Erzeugung und Speicherung zu einem konsistenten und reproduzierbaren Ablauf.

```

1 from tqdm.auto import tqdm
2 import os

```

Christian Vorhofer
 Raphael Ladinig

```

3   from src.impl.simple_data_provider import SimpleDataProvider
4   from src.impl.fitz_pdf_converter import FitzPdfConverter
5   from src.impl.specter_2_embedder import Specter2Embedder
6   from src.impl.mysql_database import MySQLDatabase
7   from src.impl.qdrant_database import QdrantDatabase
8   from src.impl.logger import get_logger
9
10
11
12  class Pipeline:
13      def __init__(self, relational_db, vector_db, data_provider, pdf_converter, embedder):
14          """
15              Central processing pipeline.
16
17              Orchestrates the full workflow:
18                  PDF conversion      text chunking
19                  embedding generation    persistence in relational and vector databases.
20          """
21          self.mysql_db = relational_db
22          self.qdrant_db = vector_db
23          self.data_provider = data_provider
24          self.pdf_converter = pdf_converter
25          self.embedder = embedder
26
27          # Logger for high-level pipeline execution and progress reporting
28          self.logger = get_logger(__name__)
29          self.logger.debug(
30              "Pipeline initialized with %s, %s, %s, %s, %s",
31              type(relational_db).__name__,
32              type(vector_db).__name__,
33              type(data_provider).__name__,
34              type(pdf_converter).__name__,
35              type(embedder).__name__,
36          )
37
38          # Allows disabling progress bars in non-interactive environments
39          self._tqdm_disable = False
40
41          # Custom progress bar formats for papers and chunks
42          self._paper_bar_format = (
43              "{l_bar}{bar} {n_fmt} papers [{elapsed}<{remaining}, {rate_fmt}]"
44          )
45          self._chunk_bar_format = (
46              " {l_bar}{bar} {n_fmt}/{total_fmt} chunks "
47              "[{elapsed}<{remaining}, {rate_fmt}{postfix}]"
48          )
49
50      def process(self):
51          """
52              Execute the full processing pipeline.
53
54              Iterates over all available papers, extracts and cleans text,
55              generates embeddings for text chunks and stores the results
56              in both relational and vector databases.
57          """
58          self.logger.info("Pipeline processing started.")
59
60          # Outer progress bar: paper-level processing
61          with tqdm(
62              desc="Papers",
63              unit="paper",
64              dynamic_ncols=True,
65              mininterval=0.5,
66              bar_format=self._paper_bar_format,
67              disable=self._tqdm_disable,
68          ) as paper_bar:
69
70              while self.data_provider.hasNext():
71                  # Fetch next paper (arXiv ID + raw PDF bytes)
72                  result = self.data_provider.next()
73                  if result is None:
74                      self.logger.warning(
75                          "Data provider returned None      no more papers or an error occurred."
76                      )
77                      break

```

Christian Vorhofer
Raphael Ladinig

```

78
79     arxiv_id, pdf_data = result
80     if arxiv_id is None or pdf_data is None:
81         self.logger.warning(
82             "Received invalid paper result (arxiv_id=%s, pdf_data=%s). Skipping...",
83             arxiv_id,
84             "None" if pdf_data is None else "bytes",
85         )
86         continue
87
88     # Avoid duplicate processing
89     if self.mysql_db.paper_exists(arxiv_id):
90         self.logger.info("Paper %s already exists      skipping.", arxiv_id)
91         continue
92
93     self.logger.info("Processing paper: %s", arxiv_id)
94
95     # Extract plain text from the PDF
96     try:
97         text = self.pdf_converter.pdf_to_string(pdf_data)
98     except Exception as e:
99         self.logger.error(
100             "Failed to extract text for %s: %s",
101             arxiv_id,
102             e,
103             exc_info=True,
104         )
105         continue
106
107     if not text or not text.strip():
108         self.logger.warning(
109             "No text extracted for %s      skipping.", arxiv_id
110         )
111         continue
112
113     # Extract PDF metadata (best-effort)
114     try:
115         metadata = self.pdf_converter.pdf_metadata(pdf_data)
116     except Exception as e:
117         self.logger.error(
118             "Failed to extract metadata for %s: %s",
119             arxiv_id,
120             e,
121             exc_info=True,
122         )
123         metadata = {}
124
125     # Clean and chunk the extracted text
126     cleaned_text = self.pdf_converter.clean_string(text)
127     chunks = self.pdf_converter.chunk_string(cleaned_text)
128
129     if not chunks:
130         self.logger.warning(
131             "No chunks generated for %s      skipping embedding.", arxiv_id
132         )
133         paper_bar.update(1)
134         continue
135
136     # Persist structured paper data in the relational database
137     self.mysql_db.add_paper(
138         arxiv_id=arxiv_id,
139         text=text,
140         title=metadata.get("title"),
141         author=metadata.get("author"),
142         subject=metadata.get("subject"),
143         keywords=metadata.get("keywords"),
144         creator=metadata.get("creator"),
145         producer=metadata.get("producer"),
146         creation_date=metadata.get("creationDate"),
147         modification_date=metadata.get("modDate"),
148         trapped=metadata.get("trapped"),
149     )
150     self.logger.info(
151         "Created DB record for paper %s (%d chunks)",
152         arxiv_id,

```

```

153             len(chunks),
154         )
155
156     # Generate embeddings for all text chunks
157     try:
158         embeddings = self.embedder.embed(chunks)
159     except Exception as e:
160         self.logger.error(
161             "Failed to embed chunks for %s: %s",
162             arxiv_id,
163             e,
164             exc_info=True,
165         )
166         paper_bar.update(1)
167         continue
168
169     # Inner progress bar: chunk-level embedding & storage
170     with tqdm(
171         total=len(chunks),
172         desc=f"Embedding {arxiv_id}",
173         unit="chunk",
174         dynamic_ncols=True,
175         mininterval=0.1,
176         leave=False,
177         bar_format=self._chunk_bar_format,
178         disable=self._tqdm_disable,
179     ) as chunk_bar:
180
181         for idx, (chunk, embedding) in enumerate(
182             zip(chunks, embeddings), start=1
183         ):
184             # Store each chunk embedding in the vector database
185             self.qdrant_db.add_chunk(
186                 arxiv_id,
187                 embedding.tolist(),
188                 chunk,
189             )
190             chunk_bar.update(1)
191
192             # Periodic logging for large documents
193             if idx % 25 == 0 or idx == len(chunks):
194                 chunk_bar.set_postfix_str(f"last={idx}")
195                 self.logger.debug(
196                     "Embedded %d/%d chunks for %s.",
197                     idx,
198                     len(chunks),
199                     arxiv_id,
200                 )
201
202             paper_bar.set_postfix_str(
203                 f"last={arxiv_id} chunks={len(chunks)}"
204             )
205             paper_bar.update(1)
206
207         self.logger.info("Pipeline processing finished.")

```

Listing 6.9: Pipeline zur Verarbeitung und Indexierung von Papers

Der Gesamtprozess gliedert sich in folgende Schritte:

1. Abruf neuer Papers aus der Datenquelle,
2. Download und Textextraktion aus PDF-Dokumenten,
3. Aufteilung des Textes in Chunks und Erzeugung von Embeddings,
4. Speicherung von Metadaten und Vektorrepräsentationen.

Christian Vorhofer
 Raphael Ladinig

Durch die klare Trennung dieser Schritte bleibt die Pipeline leicht erweiterbar, testbar und für agentische Erweiterungen geeignet.

6.10 Zusammenspiel der Komponenten

Das Zusammenspiel der beschriebenen Komponenten ermöglicht eine skalierbare und flexible Paper-Search-Plattform. Während relationale Datenbanken effiziente Metadatenabfragen erlauben, stellt die Vektordatenbank eine leistungsfähige semantische Suche bereit. Die modulare Architektur erlaubt es, neue Datenquellen, Embedding-Modelle oder Datenbanken mit minimalem Implementierungsaufwand zu integrieren.

Damit bildet Tapyre Paper Search die technische Grundlage für eine moderne, agentenfähige Forschungsplattform, die klassische Informationsretrieval-Ansätze mit aktuellen Methoden der semantischen Suche kombiniert.

6.11 Analyse der Performance-Charakteristika

Diese Section analysiert beobachtete Performance-Effekte während des praktischen Betriebs von Tapyre Paper Search. Neben ressourcenbedingten Einschränkungen werden insbesondere Limitierungen bei der semantischen Suche, anfängliche Implementierungsineffizienzen sowie periodische Unterbrechungen des Systems betrachtet. Ziel ist es, die gemessenen Verläufe einzuordnen und deren Ursachen transparent darzustellen.

6.11.1 Limitierungen bei der semantischen Suche

Im praktischen Betrieb zeigte sich, dass die Indexierung und Speicherung der Embeddings auch bei großen Datenmengen stabil funktionierte. Ab einer Größenordnung von etwa 500 000 Publikationen traten die wesentlichen Einschränkungen jedoch bei der semantischen Abfrage (Querying) auf.

Christian Vorhofer
Raphael Ladinig

Die Antwortzeiten stiegen deutlich an und in einzelnen Fällen kam es zu Instabilitäten bis hin zu Abstürzen der Vektordatenbank Qdrant.

Die Ursache lag primär in den begrenzten verfügbaren Arbeitsspeicherressourcen der Serverumgebung. Da der Server parallel von mehreren Personen genutzt wurde, musste der verfügbare RAM für Qdrant stark limitiert werden. Unter diesen Bedingungen kann die effiziente Ausführung von ANN-Abfragen – insbesondere HNSW-basierter Suchverfahren – beeinträchtigt werden, da Indexstrukturen und Caches nicht vollständig im Speicher gehalten werden können. Dies führt zu vermehrten I/O-Zugriffen sowie deutlichen Latenzspitzen, wodurch sich die Query-Zeiten im Extremfall stark verlängern.

6.11.2 Anfängliche Implementierungsineffizienzen und Optimierungen

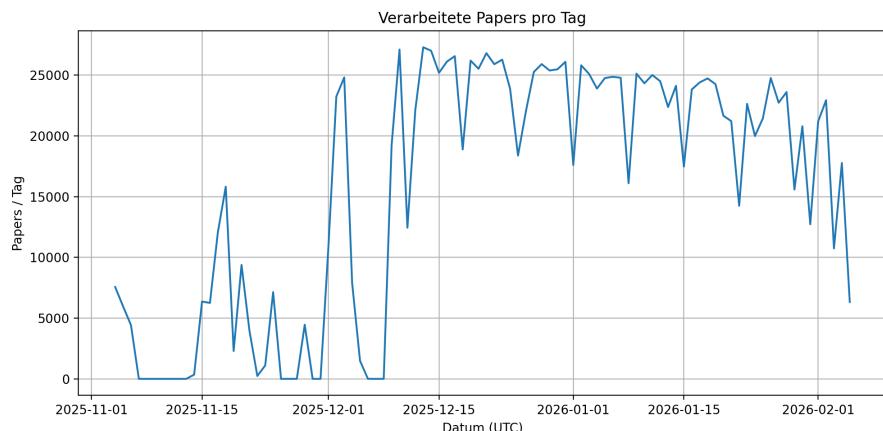


Abbildung 6.1: Anzahl der pro Tag verarbeiteten wissenschaftlichen Publikationen im Testbetrieb

Abbildung 6.1 zeigt die Anzahl der täglich verarbeiteten wissenschaftlichen Publikationen während des Testbetriebs. Zu Beginn des Beobachtungszeitraums ist eine vergleichsweise geringe Verarbeitungsrate erkennbar. Diese anfängliche Phase ist primär auf noch ineffiziente Implementierungen innerhalb der Pipeline zurückzuführen.

Insbesondere betraf dies nicht optimal gewählte Batch-Größen, redundante Verarbeitungsschritte sowie fehlende Parallelisierung einzelner Komponenten. Im Verlauf der Entwicklung wurden diese Schwachstellen schrittweise identifiziert und durch gezieltes Refactoring behoben werden. Die anschließend steigende Tagesleistung verdeutlicht, dass Performanceverbesserungen vor allem durch Implementierungsoptimierungen erreicht wurden und die zugrunde liegende Architektur grundsätzlich skalierbar ausgelegt ist.

6.11.3 Periodische Verarbeitungseinbrüche durch geplante Unterbrechungen

In Abbildung 6.1 sind zudem wiederkehrende Einbrüche in der täglichen Verarbeitungsrate erkennbar. Diese treten in regelmäßigen Abständen auf und sind insbesondere an Donnerstagen zu beobachten. Die Ursache hierfür liegt nicht in technischen Instabilitäten oder Fehlern des Systems, sondern in geplanten Unterbrechungen des Pipeline-Betriebs.

Aufgrund der gemeinsamen Nutzung der Serverinfrastruktur musste die Pipeline an Donnerstagen für mehrere Stunden deaktiviert werden, um Rechenressourcen für andere Nutzer freizugeben. Während dieser Zeit fand keine Indexierung oder Embedding-Erzeugung statt, was sich direkt in der reduzierten Tagesverarbeitung widerspiegelt. Diese Einbrüche sind daher als organisatorische Randbedingung des Entwicklungsumfelds zu interpretieren.

6.11.4 Kumulative Verarbeitung und Gesamtstabilität des Systems

Abbildung 6.2 zeigt die kumulative Anzahl der verarbeiteten Publikationen über die Zeit. Der nahezu monotone und annähernd lineare Anstieg bestätigt, dass der Import- und Indexierungsprozess zuverlässig und kontinuierlich durchgeführt werden konnte. Es traten keine systematischen Abbrüche oder Datenverluste auf.

Christian Vorhofer
Raphael Ladinig

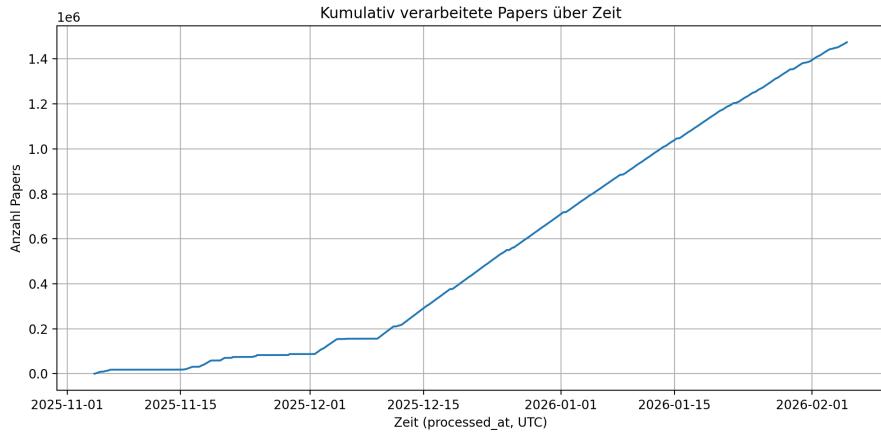


Abbildung 6.2: Kumulative Anzahl der verarbeiteten wissenschaftlichen Publikationen

Gleichzeitig verdeutlicht die Darstellung, dass die zunehmende Datensumme keinen negativen Einfluss auf die Stabilität der Pipeline hatte. Die später beobachteten Performance-Probleme betrafen primär die semantische Abfrageleistung der Vektordatenbank unter stark begrenzten Arbeitsspeicherressourcen, nicht jedoch die Speicherung oder Indexierung selbst.

6.11.5 Zusammenfassende Einordnung

Zusammenfassend lassen sich die beobachteten Performance-Charakteristika auf drei Hauptfaktoren zurückführen: anfängliche Implementierungs ineffizienzen, bewusst limitierte Hardware-Ressourcen sowie geplante Betriebsunterbrechungen. Keiner dieser Faktoren stellt eine grundsätzliche Einschränkung der Architektur von Tapyre Paper Search dar.

Vielmehr zeigen die Ergebnisse, dass das System auch unter realistischen und eingeschränkten Rahmenbedingungen stabil betrieben werden konnte. Die Analyse liefert wertvolle Erkenntnisse für zukünftige Optimierungen und zeigt auf, welche Maßnahmen bei erweitertem Ressourcenrahmen oder in einer dedizierten Betriebsumgebung sinnvoll wären.

Appendix

Tabellenverzeichnis

Abbildungsverzeichnis

| | | |
|-----|--|----|
| 6.1 | Anzahl der pro Tag verarbeiteten wissenschaftlichen Publikationen im Testbetrieb | 51 |
| 6.2 | Kumulative Anzahl der verarbeiteten wissenschaftlichen Publikationen | 52 |

Listings

| | | |
|------|--|----|
| 5.1 | Abstrakte LLM-Schnittstelle | 22 |
| 5.2 | OllamaLLM als konkrete Implementierung | 22 |
| 5.3 | Abstrakte Agent-Schnittstelle | 23 |
| 5.4 | PluginAgent mit ReAct-Agententyp | 23 |
| 5.5 | Abstrakte Plugin-Basisklasse | 24 |
| 5.6 | Dynamischer PluginLoader | 25 |
| 5.7 | AppPlugin als konkretes Plugin | 26 |
| 6.1 | Abstrakte Schnittstelle für Datenquellen | 30 |
| 6.2 | Initialisierung des ArxivDataProvider und Laden des Zustands | 31 |
| 6.3 | Persistieren des aktuellen Zustands (atomarer Write) | 32 |
| 6.4 | Iteration über arXiv-IDs und Abruf des nächsten verfügbaren PDFs | 34 |
| 6.5 | Berechnung der nächsten arXiv-ID inklusive Roll-Over | 35 |
| 6.6 | Download des PDFs mit Rate-Limiting und Retry-Mechanismus | 35 |
| 6.7 | PDF-zu-Text-Konvertierung mit PyMuPDF | 36 |
| 6.8 | Abstrakte Embedder-Schnittstelle | 39 |
| 6.9 | Specter2Embedder zur Erzeugung semantischer Vektoren | 39 |
| 6.10 | Abstrakte Datenbankschnittstelle | 41 |
| 6.11 | MySQL-Datenbankanbindung für Paper-Metadaten | 41 |
| 6.12 | Qdrant-Datenbank für semantische Suche | 44 |
| 6.13 | Pipeline zur Verarbeitung und Indexierung von Papers | 47 |

Literaturverzeichnis

Ashish Vaswani, Noam Shazeer, N. P. J. U. L. J. A. N. G. L. K. I. P. (2017), 'Arxiv', <https://arxiv.org/abs/1706.03762>. Zugriff 2025.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M. et al. (2020), Language models are few-shot learners, in 'Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)'.

Chen, W. (2023), 'Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks', *arXiv preprint arXiv:2305.10200* .

Cohan, A., Feldman, S., Beltagy, I., Downey, D. & Weld, D. (2020), 'Specter: Document-level representation learning for scientific papers', *arXiv preprint arXiv:2004.07180* .

URL: <https://arxiv.org/abs/2004.07180>

Couper, M. P. (2001), 'Web Survey Research: Challenges and Opportunities', Proceedings of the Annual Meeting of the American Statistical Association.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Diekmann, A. (1999), *Empirische Sozialforschung. Grundlagen, Methoden, Anwendungen*, fifth edn, Rowohlt Enzyklopädie, Reinbeck bei Hamburg.

Dillman, D. A., Tortora, R. & Bowker, D. (1998), Principles for Constructing Web Surveys, Technical report, SESRC.

Docker Inc. (2025), 'Docker engine security', <https://docs.docker.com/engine/security/>. Zugriff am: 05.12.2025.

- Du, N., Liu, H., Li, Y. et al. (2023), 'Improving factuality and reasoning in language models through multiagent debate', *arXiv preprint arXiv:2305.14325*.
- Fielding, R. T. (2000), Architectural Styles and the Design of Network-based Software Architectures, PhD thesis, University of California, Irvine.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
- Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press. <http://www.deeplearningbook.org>.
- Gray, J. & Reuter, A. (1992), *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann.
- Hochreiter, S. & Schmidhuber, J. (1997), long short-term memory. <https://www.bioinf.jku.at/publications/older/2604.pdf>.
- Hong, B., Tang, Y., Li, H. et al. (2023), 'Metagpt: Meta programming for multi-agent collaborative framework', *arXiv preprint arXiv:2308.00352*.
- Jurafsky, D. & Martin, J. H. (2023), *Speech and Language Processing*. Online version.
URL: <https://web.stanford.edu/jurafsky/slp3/>
- LangChain (2023), 'Langchain documentation', <https://python.langchain.com/>. Accessed 2024.
- Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* 86(11), 2278–2324.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N. et al. (2020), 'Retrieval-augmented generation for knowledge-intensive nlp tasks', *arXiv preprint arXiv:2005.11401*.
URL: <https://arxiv.org/abs/2005.11401>
- Loc (2004), 'Corporate value statement'. Einstiegsseite zum Unternehmensleitbild.
URL: <http://www.lockheedmartin.com/wms/findPage.do>

Malkov, Y. A. & Yashunin, D. A. (2020), 'Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs', *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42(4), 824–836.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781*.
URL: <https://arxiv.org/abs/1301.3781>

Nachname, V. (2024), 'Titel der Webseite', <https://www.beispielseite.de>. Zugriff am 15. November 2024.

Nandakumar, K., Cohan, A., Feldman, S., Downey, D. & Beltagy, I. (2023), Specter2: Building better document-level representations, in 'Findings of the Association for Computational Linguistics (ACL)'.

NVIDIA Corporation (2025), *CUDA C++ Programming Guide*. Zugriff am: 05.12.2025.

OpenAI (2024), 'Model context protocol', <https://github.com/modelcontextprotocol>. Accessed 2024.

Oracle Corporation (2024), *MySQL 8.4 Reference Manual: InnoDB Index Types*. Zugriff am: 05.12.2025.
URL: <https://dev.mysql.com/doc/refman/8.4/en/innodb-index-types.html>

Paszke, A., Gross, S., Massa, F. et al. (2019), Pytorch: An imperative style, high-performance deep learning library, in 'Advances in Neural Information Processing Systems 32 (NeurIPS 2019)'.

Percona (2024), 'Understanding mysql indexes: Types, benefits, and best practices'. Zugriff am: 05.12.2025.
URL: <https://www.percona.com/blog/understanding-mysql-indexes-types-best-practices/>

Qdrant Technologies (2024a), 'Hnsw indexing fundamentals', <https://qdrant.tech/course/essentials/day-2/what-is-hnsw/>. Zugriff am: 05.12.2025.

Qdrant Technologies (2024b), 'Qdrant concepts: Collections', <https://qdrant.tech/documentation/concepts/collections/>. Zugriff am: 05.12.2025.

Qdrant Technologies (2024c), 'Qdrant concepts: Indexing', <https://qdrant.tech/documentation/concepts/indexing/>. Zugriff am: 05.12.2025.

Qdrant Technologies (2024d), 'Qdrant documentation', <https://qdrant.tech/documentation/>. Zugriff am: 05.12.2025.

Qdrant Technologies (2024e), 'Similarity search in qdrant', <https://qdrant.tech/documentation/concepts/search/>. Zugriff am: 05.12.2025.

Quirós, G. (2024), 'How containers work: Layers, overlayfs, namespaces & cgroups'. Zugriff am: 05.12.2025.

URL: <https://k8studio.io/tutorials/container-architecture-namespaces-cgroups-overlayfs/>

Reips, U.-D. (2002), 'Standards for Internet-Based Experimenting', *Experimental Psychology* 1(4), 243–256.

Ronacher, A. & Contributors, F. (2024), 'Flask documentation', <https://flask.palletsprojects.com/>. Zugriff am: 05.12.2025.

Schick, T., Dwivedi-Yu, J., Vu, T. et al. (2023), 'Toolformer: Language models can teach themselves to use tools', *arXiv preprint arXiv:2302.04761*.

URL: <https://arxiv.org/abs/2302.04761>

Titel der Website (n.d.), Website. (Abgerufen am: 2017-07-11).

URL: <http://www.musterseite.de>

Touvron, H., Lavril, T., Izacard, G., Martinet, X. et al. (2023), 'Llama: Open and efficient foundation language models', *arXiv preprint arXiv:2302.13971*.

Wang, Z., Zhu, C., Lin, Y. & Zhou, J. (2024), 'A survey on agentic large language models', *arXiv preprint arXiv:2401.05561*.

Yao, S., Deng, J. Z., Zhou, J., Wang, A., Lo, Y. & Goodman, N. (2022), 'React: Synergizing reasoning and acting in language models', *arXiv preprint arXiv:2210.03629*.

URL: <https://arxiv.org/abs/2210.03629>