

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 Convolutional Neural Networks (CNN)	3
1.4 Rekurrente Neuronale Netze (RNN, LSTM)	4
1.5 Die Transformer-Architektur	4
1.6 Bedeutung von Transformern für LLMs und Embeddings	6
2 Einführung in Natural Language Processing (NLP)	9
2.1 Klassische NLP-Ansätze	9
2.2 Einführung in Embeddings	10
2.3 Word Embeddings: Word2Vec und GloVe	10
2.4 Kontextualisierte Embeddings	11
2.5 Embeddings mit der Transformer-Architektur	11
2.6 Relevanz für Tapyre Paper Search	11
3 Einführung in Agentic AI	13
3.1 ReAct: Reasoning + Acting	13
3.2 Tool Usage	14
3.3 Model Context Protocol (MCP)	14
3.4 Agent-to-Agent Kommunikation	15
3.5 RAG: Retrieval-Augmented Generation	15
3.6 Multi-Agent Systems	16
4 Grundkonzepte der verwendeten Technologien	17
4.1 Docker und Containerisierung	17
4.2 MySQL als relationale Datenbank	18

4.3	Qdrant und Approximate Nearest Neighbor Search	18
4.4	Flask und REST-APIs	20
4.5	PyTorch und GPU-Beschleunigung	21
4.6	Zusammenfassung	21
5	Entwicklung von Tapyre als Agentic-AI-System	23
5.1	Abstraktion der LLM-Schnittstelle	23
5.2	Der Agent und der ReAct-Loop	24
5.3	Plugins als Tools: Loose Coupling durch Interfaces	26
5.4	Dynamisches Laden der Plugins	27
5.5	Beispiel: AppPlugin zur Steuerung lokaler Anwendungen . .	28
5.6	Zusammenspiel: Agent, Plugins und ReAct-Loop	30
6	Architektur und Implementierung von Tapyre Paper Search	31
6.1	Abstraktion der Datenquellen	32
6.2	arXiv als konkrete Datenquelle	32
6.3	PDF-Verarbeitung und Textextraktion	35
6.4	Abstraktion der Embedding-Erzeugung	38
6.5	Specter2 als semantisches Embedding-Modell	39
6.6	Abstraktion der Datenhaltung	40
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
Literaturverzeichnis		59

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“ des neuronalen Netzes ist in den Gewichten und Bias-Werten gespeichert, die während des Trainingsprozesses mithilfe von Optimierungsverfahren wie dem Gradientenabstieg angepasst werden.

Die Fähigkeit eines einzelnen Neurons, eine lineare Entscheidungsgrenze zu modellieren, wurde bereits früh durch das Perzeptron-Modell demonstriert. Erst durch die Kombination vieler Neuronen in tieferen Netzen wurde es möglich, hochkomplexe Muster wie Bildmerkmale oder sprachliche Zusammenhänge effizient zu verarbeiten. Damit stellen künstliche Neuronen 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.

[Goodfellow et al. \(2016\)](#)

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 bereits die wichtigste Eigenschaft: Informationen fließen nur in eine Richtung, nämlich vom Eingang (*Input Layer*) über eine oder mehrere verdeckte Schichten (*Hidden Layers*) zum Ausgang (*Output Layer*). Es gibt keine Rückkopplungen oder Schleifen.

Ein FNN besteht aus vielen künstlichen Neuronen, die miteinander verbunden sind. Jedes Neuron berechnet aus seinen Eingaben eine gewichtete Summe und wendet anschließend eine Aktivierungsfunktion wie ReLU, Sigmoid oder Tanh an. Dadurch kann das Netzwerk auch komplexe, nicht-lineare Zusammenhänge erkennen.

Das Netzwerk „lernt“, indem es seine Gewichte anpasst. Dies geschieht über ein Verfahren namens *Backpropagation*. Dabei wird gemessen, wie weit die Vorhersage des Netzes vom tatsächlichen Ergebnis entfernt ist. Dieser Fehler wird dann genutzt, um die Gewichte so zu verändern, dass das Modell in zukünftigen Durchläufen bessere Ergebnisse liefert.

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 lassen sich als Weiterentwicklungen dieses grundlegenden Prinzips verstehen.

[Goodfellow et al. \(2016\)](#)

1.3 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) sind eine spezielle Art von neuronalen Netzen, die besonders gut für die Verarbeitung von Bildern geeignet sind. Im Gegensatz zu einfachen Feed-Forward-Netzen berücksichtigen CNNs die räumliche Struktur von Daten. Das bedeutet, dass sie Muster wie Kanten, Formen oder Texturen erkennen können – unabhängig davon, wo sie im Bild auftreten.

Der wichtigste Baustein eines CNN ist die *Convolutional Layer*. In dieser Schicht wandern kleine Filter (auch *Kerne* oder *Kernels* genannt) über das Bild und berechnen lokale Merkmale. Ein einzelner Filter kann zum Beispiel lernen, horizontale Kanten zu erkennen, während ein anderer runde Formen erkennt. Mehrere solcher Filter erzeugen sogenannte Feature Maps, die unterschiedliche Aspekte des Bildes hervorheben.

Zusätzlich zu den Faltungsschichten verwenden CNNs oft *Pooling-Schichten*. Diese verkleinern die Bilddarstellung, indem sie z. B. aus einem 2×2 -Bereich nur den größten Wert übernehmen (*Max-Pooling*). Dadurch wird das Modell robuster gegenüber kleinen Verschiebungen im Bild und reduziert gleichzeitig die Anzahl der Parameter.

Am Ende eines CNNs befinden sich meist ein oder mehrere vollständig verbundene Schichten (*Fully Connected Layers*), die auf Basis der erkannten Merkmale eine Entscheidung treffen, zum Beispiel welche Klasse ein Bild hat.

CNNs haben die Bildverarbeitung revolutioniert und sind nach wie vor ein zentraler Bestandteil moderner Computer-Vision-Systeme. Für Textverarbeitung werden sie allerdings seltener eingesetzt, da Sprache eher eine Sequenz als eine zweidimensionale Struktur ist.

[Lecun et al. \(1998\)](#)

1.4 Rekurrente Neuronale Netze (RNN, LSTM)

Rekurrente Neuronale Netze (RNNs) wurden entwickelt, um Daten zu verarbeiten, die aus Sequenzen bestehen – zum Beispiel Texte, Sprache, Musik oder Zeitreihen. Im Gegensatz zu Feed-Forward- oder Convolutional-Netzen besitzen RNNs eine Rückkopplung: Ein Teil der Ausgabe eines Zeitschrittes wird als Eingabe in den nächsten Schritt zurückgeführt. Dadurch können RNNs Informationen aus früheren Zeitpunkten speichern und haben eine Art „Gedächtnis“.

Ein einfaches RNN verarbeitet zu jedem Zeitpunkt einen Eingangswert und kombiniert diesen mit dem vorherigen Zustand des Netzwerks. Das Verfahren funktioniert gut bei kurzen Sequenzen, hat jedoch Schwierigkeiten bei langen Abhängigkeiten. Das liegt am sogenannten *Vanishing Gradient Problem*, bei dem wichtige Informationen beim Training schnell verloren gehen.

Um diese Schwächen auszugleichen, wurden LSTMs (Long Short-Term Memory) entwickelt. LSTMs besitzen spezielle Schaltelemente, sogenannte *Gates*. Diese Gates entscheiden, welche Informationen gespeichert, weitergegeben oder gelöscht werden. Dadurch können LSTMs deutlich länger relevante Zusammenhänge behalten und sind stabiler beim Training als einfache RNNs.

LSTMs wurden viele Jahre erfolgreich im Bereich der Sprachverarbeitung eingesetzt, zum Beispiel für maschinelle Übersetzung oder Textklassifikation. Heute spielen sie jedoch eine deutlich kleinere Rolle, da Transformer-Modelle effizienter trainierbar sind und besser mit langen Texten umgehen können.

Hochreiter & Schmidhuber (1997)

1.5 Die Transformer-Architektur

Transformer-Modelle wurden im Jahr 2017 mit dem Paper *Attention Is All You Need* eingeführt und haben die Sprachverarbeitung grundlegend

Christian Vorhofer
Raphael Ladinig

verändert. Im Gegensatz zu RNNs und LSTMs verarbeiten Transformer die Eingabe nicht schrittweise, sondern betrachten alle Wörter eines Satzes gleichzeitig. Dadurch können sie wesentlich besser mit langen Texten umgehen und lassen sich effizient auf modernen GPUs parallelisieren.

Das zentrale Konzept eines Transformers ist die sogenannte Self-Attention. Diese Technik ermöglicht es dem Modell, zu bestimmen, welche Wörter in einem Satz für die Bedeutung eines anderen Wortes wichtig sind. Ein Beispiel: Im Satz „Der Hund jagt die Katze, weil sie schnell ist“ muss das Modell erkennen, dass sich „sie“ auf „die Katze“ bezieht. Self-Attention macht genau das möglich, indem jedes Wort auf alle anderen Wörter „aufmerksam“ werden kann.

Ein weiterer wichtiger Bestandteil ist die Multi-Head Attention. Hierbei nutzt das Modell mehrere Attention-Mechanismen gleichzeitig, die jeweils unterschiedliche Arten von Beziehungen lernen können – etwa grammatische Strukturen, thematische Zusammenhänge oder Referenzen im Text. Diese Informationen werden anschließend kombiniert, um ein besonders aussagekräftiges Gesamtverständnis zu erzeugen.

Da Transformer keine natürliche Reihenfolge wie RNNs haben, benötigen sie Positional Encodings, die beschreiben, an welcher Stelle ein Wort im Satz steht. Diese Positionsinformationen werden zu den Eingabedaten addiert, sodass das Modell die Struktur des Satzes versteht.

Ein klassischer Transformer besteht aus zwei Teilen: einem Encoder, der den Text verarbeitet und in eine nützliche Repräsentation (Embedding) umwandelt, und einem Decoder, der beispielsweise Text generieren kann. Bei modernen Sprachmodellen wie GPT wird meist nur der Decoder verwendet, während für Suchsysteme wie Tapyre-Paper-Search ausschließlich der Encoder relevant ist.

Transformer haben sich aufgrund ihrer Genauigkeit, Skalierbarkeit und Effizienz als Standard für alle modernen NLP-Systeme durchgesetzt. Sie bilden die Grundlage großer Sprachmodelle (LLMs) und leistungsstarker Embedding-Modelle, wie sie auch in diesem Projekt verwendet werden.

[Ashish Vaswani \(2017\)](#)

Christian Vorhofer
Raphael Ladinig

1.6 Bedeutung von Transformern für LLMs und Embeddings

Transformer-Modelle spielen heute eine zentrale Rolle in fast allen Bereichen der Sprachverarbeitung. Sie bilden die Grundlage großer Sprachmodelle (*Large Language Models, LLMs*) wie GPT, LLaMA oder PaLM und sind außerdem der Standard für die Erzeugung hochwertiger Text-Embeddings. Der Grund dafür liegt in den besonderen Eigenschaften der Transformer-Architektur.

Durch den Einsatz von Self-Attention können Transformer Zusammenhänge zwischen weit entfernten Wörtern erkennen, was besonders wichtig für längere Texte, komplexe Satzstrukturen oder wissenschaftliche Dokumente ist. Während frühere Modelle wie RNNs oder LSTMs oft Schwierigkeiten hatten, Informationen über viele Wörter hinweg zu behalten, können Transformer den gesamten Kontext gleichzeitig berücksichtigen. Dies führt zu deutlich besseren Ergebnissen bei allen Aufgaben, die ein tiefes Textverständnis erfordern.

Für Embedding-Modelle – also Modelle, die Texte in numerische Vektoren umwandeln – bieten Transformer einen weiteren entscheidenden Vorteil: Sie erzeugen Repräsentationen, die nicht nur die Bedeutung einzelner Wörter, sondern die gesamte semantische Struktur eines Satzes oder Dokuments erfassen. Deshalb eignen sich Transformer-Encoder besonders gut für Suchsysteme, Klassifikationsaufgaben oder Recommendation-Systeme.

Moderne Embedding-Modelle wie *SPECTER2*, *Sentence-BERT* oder *E5* basieren alle auf Transformer-Encodern. Sie ermöglichen es, dass ähnliche Texte in einem Vektorraum nahe beieinander liegen, während unterschiedliche Inhalte klar voneinander getrennt sind. Diese Eigenschaft ist essenziell für semantische Suche, wie sie auch in diesem Projekt eingesetzt wird.

Zusammenfassend lässt sich sagen, dass Transformer die Grundlage moderner Sprachverarbeitung bilden. Ohne Transformer wären sowohl leistungsfähige LLMs als auch präzise Embedding-Modelle nicht möglich – und Systeme wie Tapyre Paper Search könnten in dieser Form nicht existieren.

Ashish Vaswani (2017) Brown et al. (2020) Touvron et al. (2023) Nandakumar et al. (2023)

Christian Vorhofer
Raphael Ladinig

2 Einführung in Natural Language Processing (NLP)

Natural Language Processing (NLP) ist ein zentraler Bereich der Künstlichen Intelligenz, der sich mit der automatischen Verarbeitung menschlicher Sprache beschäftigt. Ziel ist es, Texte so zu analysieren und zu interpretieren, dass Computer sprachbasierte Aufgaben ausführen können – etwa Suchanfragen beantworten, Texte zusammenfassen oder Dokumente klassifizieren. Moderne Systeme wie Suchmaschinen, Chatbots oder Sprachassistenten bauen maßgeblich auf Methoden des NLP auf [Jurafsky & Martin \(2023\)](#).

Während frühe Ansätze vor allem statistische Modelle nutzten, basiert das heutige NLP überwiegend auf tiefen neuronalen Netzen. Eine entscheidende Rolle spielt dabei die Frage, wie Bedeutungen mathematisch repräsentiert werden können. Diese Repräsentationen werden als Embeddings bezeichnet.

2.1 Klassische NLP-Ansätze

Vor dem Aufkommen neuronaler Modelle wurden Texte meist mithilfe statistischer Verfahren dargestellt. Typische Beispiele sind Bag-of-Words, TF-IDF und N-Gramme. Diese Methoden berücksichtigen jedoch weder semantische Beziehungen noch Kontextinformationen. So wird nicht erkannt, dass „Auto“ und „Fahrzeug“ ähnliche Bedeutungen haben oder dass „Bank“ sowohl ein Sitzmöbel als auch ein Finanzinstitut bezeichnen kann [Jurafsky & Martin \(2023\)](#).

Für einfache Klassifikationsaufgaben sind solche Modelle oft ausreichend, stoßen jedoch bei komplexeren Anwendungen – etwa semantischer Suche oder Übersetzung – schnell an ihre Grenzen.

2.2 Einführung in Embeddings

Da Computer ausschließlich mit numerischen Daten arbeiten, müssen Texte in Zahlen überführt werden. Embeddings lösen dieses Problem, indem sie Wörter, Sätze oder ganze Dokumente als Vektoren in einem kontinuierlichen Raum darstellen. Dabei gilt:

- Ähnliche Bedeutungen sollen ähnliche Vektoren besitzen.
- Unterschiedliche Bedeutungen sollen weit voneinander entfernt liegen.
- Kontextinformationen sollen möglichst berücksichtigt werden.

Embeddings bilden die Grundlage vieler moderner NLP-Systeme und ermöglichen semantische Ähnlichkeitsanalysen sowie inhaltliche Textvergleiche.

2.3 Word Embeddings: Word2Vec und GloVe

Einen bedeutenden Fortschritt stellten Word Embeddings wie Word2Vec dar. Diese Modelle ordnen jedem Wort einen festen Vektor zu und basieren auf der Idee, dass Wörter, die in ähnlichen Kontexten auftreten, ähnliche Repräsentationen erhalten. Dadurch entstehen semantische Strukturen wie:

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

Word2Vec [Mikolov et al. \(2013\)](#) und ähnliche Ansätze erfassen grundlegende semantische Beziehungen, ignorieren jedoch die Mehrdeutigkeit von Wörtern: Das Wort „Bank“ hat stets denselben Vektor, unabhängig vom Kontext.

2.4 Kontextualisierte Embeddings

Mit tiefen neuronalen Netzen entstanden Modelle, die Wortbedeutungen kontextabhängig repräsentieren. Ein Beispiel dafür ist ELMo, das für jedes Wort unterschiedliche Vektoren erzeugt – abhängig vom Satz, in dem es vorkommt. Diese Ansätze bilden eine Übergangsphase zwischen klassischen Embeddings und modernen Transformer-Modellen.

2.5 Embeddings mit der Transformer-Architektur

Mit der Einführung der Transformer-Architektur wurden neue Maßstäbe gesetzt. Transformer-Encoder wie BERT [Devlin et al. \(2018\)](#) oder wissenschaftsspezifische Modelle wie SPECTER [Cohan et al. \(2020\)](#) erzeugen hochqualitative, kontextualisierte Embeddings, indem sie den gesamten Satz oder sogar das gesamte Dokument berücksichtigen.

Dies führt zu:

- kontextabhängigen Wortvektoren,
- Satz- und Dokumentrepräsentationen als einzelne Vektoren,
- präziser semantischer Modellierung,
- robuster Erfassung langer und komplexer Zusammenhänge.

Transformer-Embeddings sind daher besonders gut geeignet, um wissenschaftliche Texte mit ihren komplexen Begrifflichkeiten und Strukturmerkmalen zu verarbeiten.

2.6 Relevanz für Tapyre Paper Search

Im Projekt *Tapyre Paper Search* dienen Embeddings dazu, wissenschaftliche Artikel in einem Vektorraum abzubilden. Dokumente mit thematischen Ähnlichkeiten liegen darin räumlich nahe beieinander, was eine präzise semantische Suche ermöglicht. Spezialisierte Modelle wie SPECTER2, die auf

wissenschaftlichen Publikationen trainiert wurden, verbessern die Erkennung fachlicher Zusammenhänge nochmals deutlich.

3 Einführung in Agentic AI

Agentic AI beschreibt ein neues Paradigma der künstlichen Intelligenz, bei dem Modelle nicht nur Antworten generieren, sondern eigenständig Handlungen planen, Tools verwenden, Entscheidungen treffen und komplexe Aufgaben in mehreren Schritten ausführen. Ein *Agent* ist dabei ein KI-System, das aktiv Ziele verfolgt, Informationen beschafft, Aktionen ausführt und basierend auf den Ergebnissen weitere Schritte plant. Während klassische Sprachmodelle rein reaktiv arbeiten, agiert Agentic AI proaktiv und interaktiv.

3.1 ReAct: Reasoning + Acting

Ein grundlegender Ansatz innerhalb von Agentic AI ist das ReAct-Framework (Reasoning + Acting). Dabei führt ein Agent nicht nur interne Überlegungen (*Reasoning*) aus, sondern trifft auch explizite Entscheidungen und führt konkrete Aktionen (*Acting*) aus. ReAct wurde von Yao et al. vorgestellt [Yao et al. \(2022\)](#).

Ein typischer ReAct-Agent arbeitet in zyklischer Struktur:

1. Der Agent überlegt (*Thought*), wie er vorgehen soll.
2. Er führt eine Aktion aus, z. B. eine API-Anfrage.
3. Er erhält eine Beobachtung (*Observation*).
4. Basierend darauf plant er den nächsten Schritt.

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

3.2 Tool Usage

Ein wesentliches Merkmal agentischer Systeme ist die Fähigkeit, externe Werkzeuge (*Tools*) einzusetzen. Beispiele hierfür sind:

- Datenbanken (z. B. Qdrant, MySQL),
- Web-APIs (arXiv, Semantic Scholar),
- Dateisysteme,
- Suchfunktionen,
- Python-Skripte.

Der Agent wählt das passende Tool aus, übergibt Parameter, interpretiert die Ergebnisse und nutzt diese, um weitere Entscheidungen zu treffen. Dadurch wird das Sprachmodell zu einer Art Steuerzentrale, die verschiedene Systeme koordinieren kann.

3.3 Model Context Protocol (MCP)

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

- wie Tools strukturiert beschrieben werden,
- wie Kontext an Modelle übergeben wird,
- wie Modelle Aktionen anfordern,
- und wie Ergebnisse standardisiert zurückgegeben werden.

Der offene Standard von OpenAI [OpenAI \(2024\)](#) ermöglicht es, Agenten flexibel in Software-Systeme einzubetten und komplexe Pipelines ohne individuelle Integrationslogik anzubinden.

3.4 Agent-to-Agent Kommunikation

In größeren Systemen arbeiten oft mehrere Agenten gemeinsam an einer Aufgabe. Diese Agent-to-Agent-Kommunikation kann genutzt werden, um Aufgaben zu verteilen, Wissen auszutauschen oder verschiedene Strategien zu evaluieren. Beispiele hierfür sind:

- ein Analyse-Agent extrahiert Daten,
- ein Recherche-Agent sucht passende Quellen,
- ein Planungs-Agent entscheidet über das weitere Vorgehen,
- ein Evaluations-Agent überprüft Ergebnisse.

Durch die Spezialisierung der Rollen entsteht eine höhere Robustheit und Skalierbarkeit.

3.5 RAG: Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) verbindet Sprachmodelle mit externem Wissen. Statt Antworten frei zu generieren, sucht ein RAG-System zuerst nach relevanten Dokumenten und erzeugt anschließend eine fundierte Antwort auf Basis dieser Inhalte. Der Ansatz wurde von Lewis et al. eingeführt [Lewis et al. \(2020\)](#).

Ein RAG-Agent arbeitet typischerweise wie folgt:

1. Retrieval: Suche nach relevanten Dokumenten, z. B. über Vektorschre in Qdrant.
2. Generation: Erzeugung einer Antwort mithilfe des Sprachmodells, unter Nutzung der gefundenen Informationen.

Für Tapyre Paper Search ist dieser Ansatz essenziell, da wissenschaftliche Dokumente zuerst semantisch abgerufen und dann weiter analysiert oder zusammengefasst werden.

3.6 Multi-Agent Systems

Multi-Agent-Systems (MAS) bestehen aus mehreren spezialisierten Agenten, die parallel oder kooperativ arbeiten. Vorteile solcher Systeme umfassen:

- höhere Robustheit durch Rollenverteilung,
- bessere Skalierbarkeit,
- parallele Problemlösung,
- Spezialisierung auf Teilprobleme.

MAS werden zunehmend in Forschung, Retrieval-Systemen und komplexen KI-Anwendungen eingesetzt.

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 bilden. Dieses Kapitel erläutert die wichtigsten technischen Grundlagen und erklärt insbesondere, wie Daten gespeichert, verarbeitet und über REST-Schnittstellen ausgetauscht werden.

4.1 Docker und Containerisierung

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

- Namespaces: isolieren Prozesse, Netzwerke, 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. Für Projekte wie Tapyre bedeutet dies, dass Komponenten wie Qdrant, MySQL oder Python-Anwendungen unabhängig voneinander, aber dennoch einheitlich ausgeführt werden können.

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) hauptsächlich B+-Bäume, die als Clustered und Secondary Indexes realisiert sind [Oracle Corporation \(2024\)](#), [Percona \(2024\)](#):

- Clustered Index: InnoDB speichert Daten entlang des Primärschlüssels. Die Tabelle ist selbst ein B+-Baum.
- Secondary Indexes: weitere B+-Bäume, die Zeiger auf die Primärzeilen enthalten.
- Effiziente Suche: Durch die logarithmische Höhe der B+-Bäume können Suchanfragen schnell ausgeführt werden.

MySQL bietet außerdem ACID-Transaktionen, die Datenkonsistenz garantieren und in der klassischen Datenbankliteratur ausführlich beschrieben werden [Gray & Reuter \(1992\)](#):

- Atomicity: Eine Transaktion wird entweder vollständig ausgeführt oder verworfen.
- Consistency: Alle Daten erfüllen definierte Integritätsregeln.
- Isolation: Gleichzeitige Transaktionen beeinflussen sich nicht.
- Durability: Bestätigte Änderungen bleiben dauerhaft gespeichert.

In Tapyre wird MySQL zur Speicherung von Metadaten eingesetzt.

4.3 Qdrant und Approximate Nearest Neighbor Search

Qdrant ist eine spezialisierte Datenbank zur Speicherung und Suche von Vektorrepräsentationen (Embeddings) und wird explizit als Vektor-Datenbank für semantische Suche entwickelt [Qdrant Technologies \(2024d\)](#). Im Gegensatz zu relationalen Datenbanken speichert Qdrant keine Texte, sondern numerische Vektoren, die die Bedeutung eines Dokuments darstellen.

Christian Vorhofer
Raphael Ladinig

Speicherstruktur

Eine Qdrant-Collection besteht aus [Qdrant Technologies \(2024b\)](#):

- Vektoren (meist 768 oder 1024 Dimensionen),
- Payload-Daten wie Titel, DOI, Autoren oder Jahr,
- Segmenten zur internen Aufteilung der Daten.

Diese Struktur ist optimiert für sequentielles Lesen und schnelle Ähnlichkeitsabfragen [Qdrant Technologies \(2024e\)](#).

Approximate Nearest Neighbor (ANN)

Da ein exakter Vergleich aller Vektoren bei großen Datenmengen ineffizient wäre, verwendet Qdrant Approximate Nearest Neighbor (ANN)-Algorithmen. Der wichtigste davon ist der HNSW-Index (Hierarchical Navigable Small World), ein Graph-basierter ANN-Algorithmus [Malkov & Yashunin \(2020\)](#), [Qdrant Technologies \(2024c,a\)](#):

- eine mehrschichtige Graphstruktur,
- wenige Knoten auf höheren Ebenen (grobe Orientierung),
- viele Knoten auf unteren Ebenen (feine Suche),
- Navigation vom groben zum feinen Bereich,
- ermöglicht extrem schnelle Annäherung an das richtige Suchergebnis.

HNSW kombiniert hohe Geschwindigkeit mit hoher Genauigkeit und eignet sich besonders gut für semantische Suchsysteme [Malkov & Yashunin \(2020\)](#), [Qdrant Technologies \(2024e\)](#).

Ähnlichkeitsmaße

Qdrant unterstützt verschiedene Metriken [Qdrant Technologies \(2024e\)](#):

- Kosinusähnlichkeit (Standard für NLP-Modelle),
- Euklidische Distanz,
- Skalares Produkt (Dot Product).

In Tapyre kommt hauptsächlich die Kosinusähnlichkeit zum Einsatz, da sie die semantische Nähe zwischen Dokumenten besonders gut abbildet.

4.4 Flask und REST-APIs

Flask ist ein leichtgewichtiges Webframework für Python und dient in Tapyre zur Bereitstellung von REST-APIs [Ronacher & Contributors \(2024\)](#). Eine REST-API (*Representational State Transfer*) basiert auf einem Architekturstil für verteilte Hypermedia-Systeme, der von Fielding in seiner Dissertation beschrieben wurde [Fielding \(2000\)](#). Sie ermöglicht die Kommunikation zwischen Anwendungsteilen über standardisierte HTTP-Methoden:

- GET: Anfrage von Daten
- POST: Erstellen neuer Daten
- PUT/PATCH: Aktualisieren von Daten
- DELETE: Löschen von Daten

REST-APIs verwenden häufig JSON als Datenaustauschformat und sind zustandslos: Jeder Request enthält alle notwendigen Informationen, um verarbeitet zu werden [Fielding \(2000\)](#).

Ein typischer Beispiel-Request:

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

Die API ruft daraufhin den Embedding-Prozess auf, speichert das Ergebnis oder gibt es zurück. Dieser Ansatz ermöglicht modulare, wartbare und gut testbare Kommunikationsstrukturen.

Christian Vorhofer
Raphael Ladinig

4.5 PyTorch und GPU-Beschleunigung

PyTorch ist ein Framework für Deep Learning und wird verwendet, um Embeddings zu berechnen [Paszke et al. \(2019\)](#). Da Transformer-Modelle wie SPECTER2 hunderte Millionen Parameter besitzen, werden GPUs genutzt, um Berechnungen massiv zu beschleunigen. CUDA ermöglicht hierbei die Ausführung linearer Algebraoperationen direkt auf der Grafikkarte [NVIDIA Corporation \(2025\)](#).

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

- dynamische Rechengraphen,
- große Modellbibliothek,
- nahtlose GPU-Unterstützung,
- Integration in moderne NLP-Pipelines.

4.6 Zusammenfassung

Docker stellt reproduzierbare Umgebungen bereit [Docker Inc. \(2025\)](#), MySQL speichert strukturierte Daten effizient über B+-Bäume und Clustered Indexes [Oracle Corporation \(2024\)](#), Qdrant ermöglicht schnelle semantische Vektorsuche mithilfe von HNSW und ANN [Qdrant Technologies \(2024d\)](#), [Malkov & Yashunin \(2020\)](#), Flask dient als Kommunikationsschicht über REST-APIs [Ronacher & Contributors \(2024\)](#), [Fielding \(2000\)](#), und PyTorch führt rechenintensive Transformer-Modelle auf GPUs mithilfe von CUDA aus [Paszke et al. \(2019\)](#), [NVIDIA Corporation \(2025\)](#). Diese Technologien bilden zusammen die Basis für ein modernes, 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 (Input → Output) arbeitet Tapyre mit einem Agenten, der eigenständig Tools aufrufen kann, in einem ReAct-Loop (Reasoning + Acting) entscheidet und seine Funktionalität über Plugins dynamisch erweitert [Yao et al. \(2022\)](#), [Schick et al. \(2023\)](#), [Wang et al. \(2024\)](#). Die Kopplung zwischen Kernsystem, LLM und Plugins ist dabei bewusst lose gehalten, um das System leicht erweiterbar und wartbar zu machen [Gamma et al. \(1994\)](#).

Im Folgenden werden die wichtigsten Bausteine erläutert:

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

5.1 Abstraktion der LLM-Schnittstelle

Um das System unabhängig von einem konkreten Sprachmodell oder Anbieter zu halten, wird eine abstrakte LLM-Schnittstelle definiert. Jeder LLM-Typ (z. B. Ollama, OpenAI, etc.) muss nur diese Schnittstelle implementieren. Dadurch kann das Modell später einfach ausgetauscht werden, ohne dass

der Agent oder die Plugins angepasst werden müssen. Dieses Prinzip folgt etablierten Architekturmustern wie Interface- und Factory-Abstraktionen [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

Eine konkrete Implementierung für Ollama sieht dann 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 Rest des Systems kennt nur das Interface LLM und arbeitet mit BaseChatModel-Instanzen. Welches konkrete Modell dahinter steckt, ist für den Agenten und die Plugins transparent.

5.2 Der Agent und der ReAct-Loop

Der eigentliche Agent ist als Abstraktion definiert und besitzt lediglich eine Methode ask, die eine Anfrage entgegennimmt und eine Antwort zurückliefert:

```

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

Die konkrete Implementation `PluginAgent` verwendet `LangChain` und das Agentenmodell `ZERO_SHOT.REACT_DESCRIPTION`. Dieses Agentenmodell setzt das ReAct-Prinzip um: Das Modell plant in Gedanken (*Reasoning*), ruft bei Bedarf Tools auf (*Acting*) und verarbeitet die Ergebnisse iterativ weiter [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         self.llm = llm.getLLM()
10
11         self.prompt = ChatPromptTemplate.from_messages([
12             SystemMessagePromptTemplate.from_template(system_prompt),
13             HumanMessagePromptTemplate.from_template("{input}")
14         ])
15
16         self.agent = initialize_agent(
17             tools=tools,
18             llm=self.llm,
19             agent=AgentType.ZERO_SHOT.REACT_DESCRIPTION,
20             verbose=verbose,
21             handle_parsing_errors=True,
22             max_iterations=2,
23             early_stopping_method="generate"
24         )
25
26
27     def ask(self, prompt: str) -> str:
28         return self.agent.run(prompt)

```

Listing 5.4: `PluginAgent` mit ReAct-Agententyp

Wichtige Aspekte:

- ReAct-Loop: `AgentType.ZERO_SHOT.REACT_DESCRIPTION` sorgt dafür, dass das LLM selbst entscheidet, wann ein Tool benutzt werden soll. Es erzeugt intern eine Folge aus »Thought«, »Action« und »Observation«.
- Tool-Auswahl: Die Liste `tools` wird später aus den geladenen Plugins erzeugt. Das LLM sieht nur die Tool-Beschreibungen und entscheidet basierend darauf, welches Tool geeignet ist.
- `max_iterations`: Begrenzung des ReAct-Loops auf zwei Tool-Aufrufe, um Endlosschleifen zu vermeiden [Schick et al. \(2023\)](#).

5.3 Plugins als Tools: Loose Coupling durch Interfaces

Plugins stellen die eigentliche Funktionalität des Systems dar (z. B. das Starten von Programmen, Suchen in Datenbanken, etc.). Sie sind über eine abstrakte Basisklasse definiert und können damit beliebig erweitert werden, ohne dass der Kern des Systems angepasst werden muss. Die Architektur folgt klassischen Prinzipien der losen Kopplung [Gamma et al. \(1994\)](#).

```

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         return tool
29
30     def __call__(self, text: str) -> str:
31         return self.run(text)

```

Listing 5.5: Abstrakte Plugin-Basisklasse

Wesentliche Punkte:

- Interface-basiert: Plugins müssen nur `run()` implementieren.
- Loose Coupling: Der Agent kennt nur die LangChain-Tools, nicht die konkrete Plugin-Klasse.
- Tool-Integration: `LCTool.from_function` macht aus `run()` ein Tool, das im ReAct-Loop genutzt werden kann [LangChain \(2023\)](#).

5.4 Dynamisches Laden der Plugins

Damit neue Plugins hinzugefügt werden können, ohne das Hauptprogramm zu ändern, werden sie dynamisch zur Laufzeit geladen. Dieses Prinzip folgt gängigen Entwurfsmustern für modulare und erweiterbare Systeme [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

Vorteile dieses Ansatzes:

- Plugin-basiertes Design: Funktionen werden modular ergänzt.
- Keine Codeänderung im Kern: Neue Funktionalitäten sind sofort nutzbar.
- Reflection & Introspection: Automatische Erkennung von Plugin-Klassen.

5.5 Beispiel: AppPlugin zur Steuerung lokaler Anwendungen

Ein konkretes Plugin ist das AppPlugin, das installierte Desktop-Anwendungen aus .desktop-Dateien ausliest und auf Kommando starten kann. Es implementiert die Plugin-Schnittstelle und stellt damit ein Werkzeug bereit, das der Agent im ReAct-Loop selbstständig nutzen kann [Yao et al. \(2022\)](#), [LangChain \(2023\)](#).

```

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             + "."
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"):

```

Christian Vorhofer
 Raphael Ladinig

```

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():
93             if name.lower() == text.lower():
94                 return cmd
95
96         for name, cmd in self.app_map.items():
97             if text.lower() in name.lower():
98                 return cmd
99
100    return ""

```

Listing 5.7: AppPlugin als konkretes Plugin

5.6 Zusammenspiel: Agent, Plugins und ReAct-Loop

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

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

Diese Architektur macht Tapyre zu einem echten *Agentic AI*-System: Das LLM dient als Steuerungsinstanz, die eigenständig Tools auswählt, Aktionen plant und iterativ Entscheidungen trifft, während die Plugin-Struktur maximale Erweiterbarkeit 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 (Metadaten) als auch semantisch (Vektorrepräsentationen) durchsuchbar zu machen.

Die Architektur folgt einem modularen Ansatz mit klar getrennten Verantwortlichkeiten für Datenbeschaffung, Textverarbeitung, Embedding-Erzeugung, Speicherung und Orchestrierung. Dadurch ist das System leicht erweiterbar, wartbar und auf unterschiedliche Datenquellen sowie Embedding-Modelle anpassbar.

Im Folgenden werden die zentralen Bausteine erläutert:

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

6.1 Abstraktion der Datenquellen

Um unterschiedliche Paper-Quellen einbinden zu können, wird eine abstrakte Schnittstelle für Datenprovider definiert. Diese legt fest, wie neue Dokumente geladen und bereitgestellt werden, ohne den restlichen Verarbeitungsprozess zu beeinflussen.

```

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

Durch diese Abstraktion können neue Datenquellen (z.B. PubMed oder lokale Archive) ergänzt werden, ohne Änderungen an der Pipeline oder den Datenbankschichten vorzunehmen. Das System folgt damit dem Open-Closed-Prinzip etablierter Softwarearchitekturen.

6.2 arXiv als konkrete Datenquelle

Eine konkrete Implementierung dieser Schnittstelle stellt der arXiv-Datenprovider dar. Er übernimmt das Abrufen von Metadaten sowie das Herunterladen der zugehörigen PDF-Dokumente.

```

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         self.logger = get_logger(__name__)
20         self.logger.info("Initializing ArxivDataProvider")
21
22

```

Christian Vorhofer
 Raphael Ladinig

```

23         self.first_id = first_id
24         self.last_id = last_id
25         self.current_id = first_id
26         self.last_pull = 0.0
27         self.finished = False
28         self.rate_limit_seconds = rate_limit_seconds
29         self.max_retries = max_retries
30
31     state_path = os.getenv("ARXIV_STATE_FILE", "/app/state/arxiv_state.json")
32     self.state_file = Path(state_path)
33
34     self.state_file.parent.mkdir(parents=True, exist_ok=True)
35
36     self._load_state()
37
38     def _load_state(self) -> None:
39         if not self.state_file.exists():
40             self.logger.info("[state] No state file found, starting from first_id.")
41             return
42
43         try:
44             with self.state_file.open("r", encoding="utf-8") as f:
45                 state = json.load(f)
46
47             saved_current = state.get("current_id")
48             saved_finished = state.get("finished", False)
49
50             if saved_current:
51                 self.logger.info(
52                     f"[state] Restoring state from {self.state_file}: "
53                     f"current_id={saved_current}, finished={saved_finished}"
54                 )
55                 self.current_id = saved_current
56                 self.finished = saved_finished
57         except Exception as e:
58             self.logger.error(f"[state] Failed to load state file: {e}", exc_info=True)
59
60     def _save_state(self) -> None:
61         tmp_file = self.state_file.with_suffix(".tmp")
62         data = {
63             "current_id": self.current_id,
64             "first_id": self.first_id,
65             "last_id": self.last_id,
66             "finished": self.finished,
67             "last_pull": self.last_pull,
68         }
69
70         try:
71             with tmp_file.open("w", encoding="utf-8") as f:
72                 json.dump(data, f)
73
74             os.replace(tmp_file, self.state_file)
75             self.logger.debug(
76                 f"[state] Saved state: current_id={self.current_id}, finished={self.finished}"
77             )
78         except Exception as e:
79             self.logger.error(f"[state] Failed to save state: {e}", exc_info=True)
80             if tmp_file.exists():
81                 try:
82                     tmp_file.unlink()
83                 except OSError:
84                     pass
85
86     def hasNext(self) -> bool:
87         self.logger.debug(f"[hasNext] Current ID: {self.current_id}, finished={self.finished}")
88         return not self.finished
89
90     def next(self):
91         """Fetch the next PDF from arXiv sequentially."""
92         if self.finished:
93             self.logger.info("[next] No more IDs to process      finished.")
94             return None
95
96         while True:
97             next_id = self._get_next_id(self.current_id)

```

```

98     self.logger.debug(f"[next] Next ID candidate: {next_id}")
99
100    if next_id == self.last_id or next_id is None:
101        self.logger.info("[next] Reached last ID or invalid next ID      marking finished.")
102        self.finished = True
103        self._save_state()
104        return None
105
106    self.current_id = next_id
107    pdf_data = self._fetch_pdf(next_id)
108
109    if pdf_data and len(pdf_data) > 0:
110        self.logger.info(
111            f"[next] Successfully fetched PDF for {next_id} ({len(pdf_data)} bytes)"
112        )
113        self._save_state()
114        break
115
116    self.logger.warning(f"[next] No valid PDF found for {next_id}, continuing...")
117
118    return next_id, pdf_data
119
120    def _get_next_id(self, current_id: str) -> str | None:
121        """Increment arXiv ID and force rollover at 10000."""
122        match = re.match(r'arXiv:(\d{2})(\d{2})\.(\d{4,5})(?:v\d+)?', current_id)
123        if not match:
124            self.logger.warning(f"[get_next_id] Invalid current ID format: {current_id}")
125            return None
126
127        yy, mm, num = map(int, match.groups())
128
129        ROLLOVER_LIMIT = 10000
130        num += 1
131
132        if num >= ROLLOVER_LIMIT:
133            num = 0
134            mm += 1
135            if mm > 12:
136                mm = 1
137                yy += 1
138
139            if yy > 99 or (yy == 7 and mm < 4):
140                return None
141
142        next_id = f'arXiv:{yy:02d}{mm:02d}.{num:05d}'
143        return next_id
144
145    def _fetch_pdf(self, paper_id: str) -> bytes:
146        """Download PDF from arXiv with retries and rate limiting."""
147        attempts = 0
148        while attempts < self.max_retries:
149            now = time.time()
150            wait_time = self.last_pull + self.rate_limit_seconds - now
151            if wait_time > 0:
152                self.logger.debug(f"[fetch_pdf] Rate limiting: sleeping {wait_time:.2f}s")
153                time.sleep(wait_time)
154
155            self.last_pull = time.time()
156
157            match = re.match(r'arXiv:(\d{4,6})\.\d{4,5}', paper_id)
158            if not match:
159                self.logger.error(f"[fetch_pdf] Invalid arXiv ID format: {paper_id}")
160                return b""
161
162            arxiv_id = match.group(1)
163            url = f"https://arxiv.org/pdf/{arxiv_id}.pdf"
164            self.logger.info(
165                f"[fetch_pdf] Fetching PDF from {url} (attempt {attempts+1}/{self.max_retries})"
166            )
167
168            try:
169                response = requests.get(url, timeout=10)
170                status = response.status_code
171
172                if status == 200:

```

```

173         self.logger.debug(
174             f"[fetch_pdf] 200 OK for {arxiv_id} ({len(response.content)} bytes)"
175         )
176         return response.content
177
178     elif status == 404:
179         self.logger.warning(f"[fetch_pdf] 404 Not Found for {arxiv_id}")
180         return b""
181
182     else:
183         attempts += 1
184         backoff = 5 * 60 * 60
185         self.logger.critical(
186             f"[fetch_pdf] CRITICAL: Unexpected HTTP {status} for {arxiv_id}, "
187             f"retrying in {backoff}s (attempt {attempts}/{self.max_retries})"
188         )
189         time.sleep(backoff)
190
191     except requests.exceptions.RequestException as e:
192         attempts += 1
193         backoff = min(600, 100 * attempts)
194         self.logger.error(
195             f"[fetch_pdf] Request exception for {arxiv_id}: {e}, "
196             f"retrying in {backoff}s (attempt {attempts}/{self.max_retries})",
197             exc_info=True
198         )
199         time.sleep(backoff)
200
201     self.logger.critical(f"[fetch_pdf] Max retries reached for {paper_id}, giving up.")
202     return b""

```

Listing 6.2: arXivDataProvider zur Anbindung der arXiv-API

Der Provider verarbeitet unter anderem Titel, Autoren, Abstracts, Kategorien sowie die PDF-URL eines Papers. Durch die Trennung von Metadatenbeschaffung und nachgelagerter Textverarbeitung bleibt die Architektur flexibel gegenüber Änderungen der Datenquelle.

6.3 PDF-Verarbeitung und Textextraktion

Die Umwandlung von wissenschaftlichen PDFs in maschinenlesbaren Text stellt eine zentrale technische Herausforderung dar. Wissenschaftliche Dokumente enthalten häufig mehrspaltige Layouts, Formeln, Fußnoten und Seitenheader, die eine robuste Textextraktion erschweren.

```

1  from src.core.pdf_converter import PdfConverter
2  from src.impl.logger import get_logger
3  import fitz
4  from datetime import datetime, timedelta, timezone
5  import re
6  from typing import Optional
7  import unicodedata
8
9
10 class FitzPdfConverter(PdfConverter):
11     def __init__(self):
12         self.logger = get_logger(__name__)
13         self.logger.debug("FitzPdfConverter initialized.")
14

```

```

15     def pdf_to_string(self, pdf) -> str:
16         if isinstance(pdf, str):
17             self.logger.info("Opening PDF from file path.")
18         elif isinstance(pdf, bytes):
19             self.logger.info("Opening PDF from bytes stream.")
20             self.logger.debug("PDF bytes length: %s", len(pdf))
21         else:
22             self.logger.error("Invalid input type for pdf_to_string: %s", type(pdf))
23             raise ValueError("Input must be a file path or PDF binary content")
24
25         doc = None
26         full_text = ""
27         error_pages = 0
28
29         try:
30             if isinstance(pdf, str):
31                 doc = fitz.open(pdf)
32             else:
33                 doc = fitz.open(stream=pdf, filetype="pdf")
34
35             page_count = len(doc)
36             self.logger.debug("PDF opened successfully. Page count: %d", page_count)
37
38             for i, page in enumerate(doc, start=1):
39                 try:
40                     text = page.get_text()
41                     full_text += text
42                     if i % 10 == 0 or i == page_count:
43                         self.logger.debug(
44                             "Extracted text up to page %d/%d (current page length=%d).",
45                             i, page_count, len(text)
46                         )
47                 except Exception as e:
48                     error_pages += 1
49                     self.logger.error(
50                         "Failed to extract text from page %d/%d: %s      skipping this page.",
51                         i, page_count, e
52                     )
53                     continue
54
55             if error_pages > 0:
56                 self.logger.warning(
57                     "Text extraction finished with %d page errors (total pages=%d).",
58                     error_pages, page_count
59                 )
60
61             self.logger.info(
62                 "Finished extracting text from PDF. Total length: %d characters.",
63                 len(full_text)
64             )
65         return full_text
66
67     finally:
68         if doc is not None:
69             doc.close()
70             self.logger.debug("PDF document closed after text extraction.")
71
72     def pdf_metadata(self, pdf) -> dict:
73         if isinstance(pdf, str):
74             self.logger.info("Opening PDF from file path for metadata.")
75         elif isinstance(pdf, bytes):
76             self.logger.info("Opening PDF from bytes stream for metadata.")
77             self.logger.debug("PDF bytes length: %s", len(pdf))
78         else:
79             self.logger.error("Invalid input type for pdf_metadata: %s", type(pdf))
80             raise ValueError("Input must be a file path or PDF binary content")
81
82         doc = None
83         try:
84             if isinstance(pdf, str):
85                 doc = fitz.open(pdf)
86             else:
87                 doc = fitz.open(stream=pdf, filetype="pdf")
88         except Exception as e:

```

Christian Vorhofer
 Raphael Ladinig

```

90         self.logger.error(
91             "Failed to open PDF for metadata extraction: %s", e, exc_info=True
92         )
93     return {}
94
95     metadata = doc.metadata or {}
96     self.logger.debug("Raw metadata keys: %s", list(metadata.keys()))
97
98     expected_keys = [
99         'title', 'author', 'subject', 'keywords',
100        'creator', 'producer', 'creationDate',
101        'modDate', 'trapped'
102    ]
103
104    complete_metadata = {key: metadata.get(key) for key in expected_keys}
105
106    if complete_metadata.get('creationDate'):
107        original = complete_metadata['creationDate']
108        parsed = self.parse_pdf_date(original)
109        complete_metadata['creationDate'] = parsed
110        self.logger.debug("Parsed creationDate: %s -> %s", original, parsed)
111
112    if complete_metadata.get('modDate'):
113        original = complete_metadata['modDate']
114        parsed = self.parse_pdf_date(original)
115        complete_metadata['modDate'] = parsed
116        self.logger.debug("Parsed modDate: %s -> %s", original, parsed)
117
118    self.logger.info("Metadata extracted successfully.")
119    return complete_metadata
120
121 finally:
122     if doc is not None:
123         doc.close()
124     self.logger.debug("PDF document closed after metadata extraction.")
125
126 def clean_string(self, text: str) -> str:
127     before_len = len(text)
128     text = unicodedata.normalize("NFKC", text)
129
130     text = text.replace('\r\n', '\n').replace('\r', '\n')
131
132     WHITESPACE_CHARS = [
133         "\u00A0",
134         "\u2007",
135         "\u202F",
136         "\u2009",
137         "\u2002", "\u2003", "\u2004", "\u2005", "\u2006",
138         "\u2008", "\u200A",
139         "\u3000",
140         "\u180E",
141         "\u200B", "\u200C", "\u200D", "\u2060",
142     ]
143     for ch in WHITESPACE_CHARS:
144         text = text.replace(ch, ' ')
145
146     text = text.replace('  ', '-')
147     text = text.replace('  ', '--')
148     text = text.replace('  ', '---')
149     text = text.replace('  ', '')
150
151     text = text.replace('\n', ' ')
152
153     text = ' '.join(text.split())
154     after_len = len(text)
155     self.logger.debug("Cleaned string: length %d -> %d.", before_len, after_len)
156     return text
157
158 def chunk_string(self, text: str, chunk_size=500, overlap=50) -> list[str]:
159     self.logger.info("Chunking text with chunk_size=%d and overlap=%d.", chunk_size, overlap)
160     chunks = []
161     start = 0
162     text_len = len(text)
163     while start < text_len:
164         end = min(start + chunk_size, text_len)

```

```

165         chunks.append(text[start:end])
166         start += chunk_size - overlap
167
168     self.logger.info("Produced %d chunks from text of length %d.", len(chunks), text_len)
169     return chunks
170
171     def parse_pdf_date(self, pdf_date_str) -> Optional[datetime]:
172         if not pdf_date_str or not str(pdf_date_str).startswith('D:'):
173             self.logger.warning("PDF date string missing or not starting with 'D:' %s", pdf_date_str)
174             return None
175
176         original = pdf_date_str
177         pdf_date_str = pdf_date_str[2:]
178
179         match = re.match(
180             r"(\d{4})(\d{2})(\d{2})(\d{2})(\d{2})([+-])(\d{2})'?(\\d{2})?'?", pdf_date_str
181         )
182
183         if not match:
184             self.logger.warning("Failed to parse PDF date string: %s", original)
185             return None
186
187         year, month, day, hour, minute, second, tz_sign, tz_hour, tz_minute = match.groups()
188         dt = datetime(int(year), int(month), int(day), int(hour), int(minute), int(second))
189
190         offset = timedelta(hours=int(tz_hour or 0), minutes=int(tz_minute or 0))
191         if tz_sign == '-':
192             offset = -offset
193
194         parsed = dt.replace(tzinfo=timezone(offset))
195         self.logger.debug("Parsed PDF date '%s' into datetime '%s'.", original, parsed)
196
197     return parsed

```

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

Die Implementierung basiert auf PyMuPDF (fitz) und extrahiert den Text seitenweise. Diese Lösung stellt einen praktikablen Kompromiss zwischen Performance und Textqualität dar und eignet sich insbesondere für große Paper-Sammlungen.

6.4 Abstraktion der Embedding-Erzeugung

Die semantische Suche erfordert die Umwandlung von Text in hochdimensionale Vektoren. Um unterschiedliche Modelle 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

Christian Vorhofer
 Raphael Ladinig

Diese Abstraktion erlaubt es, verschiedene Embedding-Modelle auszutauschen oder parallel zu evaluieren, ohne Änderungen an der restlichen Systemarchitektur vorzunehmen.

6.5 Specter2 als semantisches Embedding-Modell

Als konkrete Implementierung kommt Specter2 zum Einsatz, ein speziell für wissenschaftliche Texte entwickeltes Transformer-Modell. Es erzeugt Vektoren, die den semantischen Inhalt eines Papers erfassen und sich besonders für wissenschaftliche Suchanwendungen eignen.

```

1  from src.core.embedder import Embedder
2  from src.impl.logger import get_logger
3  from typing import List, Union
4  from transformers import AutoTokenizer, AutoModel
5  import torch
6
7
8  class Specter2Embedder(Embedder):
9      def __init__(self, model_name: str = "allenai/specter2_base"):
10          self.logger = get_logger(__name__)
11          self.logger.info("[Specter2Embedder] Initializing model: %s", model_name)
12
13          if torch.cuda.is_available():
14              self.device = torch.device("cuda")
15              device_name = torch.cuda.get_device_name(0)
16              cap = torch.cuda.get_device_capability(0)
17              self.logger.info("[CUDA] device=%s cap=%d%d torch_cuda=%s",
18                               device_name, cap[0], cap[1], torch.version.cuda)
19              torch.backends.cudnn.benchmark = True
20
21          elif torch.backends.mps.is_available():
22              self.device = torch.device("mps")
23              device_name = "Apple MPS (Metal)"
24
25          else:
26              self.device = torch.device("cpu")
27              device_name = "CPU"
28
29          self.logger.info("[Specter2Embedder] Using device: %s", device_name)
30
31      try:
32          self.tokenizer = AutoTokenizer.from_pretrained(model_name)
33          self.model = AutoModel.from_pretrained(
34              model_name,
35              use_safetensors=True,
36              trust_remote_code=False,
37          )
38          self.model.to(self.device)
39          self.model.eval()
40          self.logger.info("[Specter2Embedder] Model and tokenizer loaded successfully.")
41      except Exception as e:
42          self.logger.exception("[Specter2Embedder] Failed to load model '%s': %s", model_name, e)
43          raise
44
45      @torch.inference_mode()
46      def embed(self, text: Union[str, List[str]]) -> torch.Tensor:
47          if isinstance(text, str):
48              self.logger.debug("[Specter2Embedder] Received single string for embedding.")
49              text = [text]
50
51      try:

```

```

52         encoded_input = self.tokenizer(
53             text,
54             padding=True,
55             truncation=False,
56             return_tensors="pt",
57         )
58         encoded_input = {k: v.to(self.device, non_blocking=True) for k, v in encoded_input.items()}
59
60         self.logger.debug(
61             "[Specter2Embedder] Tokenized input successfully (seq_len=%d).",
62             int(encoded_input["input_ids"].shape[1])
63         )
64
65         if self.device.type == "cuda":
66             with torch.amp.autocast(device_type="cuda", dtype=torch.float16):
67                 model_output = self.model(**encoded_input)
68             else:
69                 model_output = self.model(**encoded_input)
70
71         embeddings = model_output.last_hidden_state[:, 0, :] # CLS
72         self.logger.debug("[Specter2Embedder] Generated embeddings shape: %s", tuple(embeddings.shape))
73
74         return embeddings.detach().to("cpu")
75
76     except Exception as e:
77         self.logger.exception("[Specter2Embedder] Error during embedding: %s", e)
78         raise
79

```

Listing 6.5: Specter2Embedder zur Erzeugung semantischer Vektoren

Durch die Verwendung eines domänenspezifischen Modells wird eine deutlich bessere semantische Repräsentation erzielt als mit generischen Sprachmodellen.

6.6 Abstraktion der Datenhaltung

Das System unterscheidet bewusst zwischen strukturierter Datenhaltung (Metadaten) und semantischer Vektorspeicherung. Eine abstrakte Datenbankschnittstelle definiert dabei die 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

Christian Vorhofer
 Raphael Ladinig

Diese Trennung erlaubt es, relationale und vektorbasierte Datenbanken gezielt für ihre jeweiligen Stärken einzusetzen.

6.7 MySQL für strukturierte Metadaten

Metadaten wie Titel, Autoren, Kategorien und Statusinformationen werden in einer relationalen Datenbank gespeichert. Die konkrete Implementierung basiert auf MySQL.

```

1 import os
2 from datetime import datetime
3 from sqlalchemy import create_engine, text
4 from sqlalchemy.orm import sessionmaker, scoped_session
5 from sqlalchemy.exc import SQLAlchemyError, IntegrityError, DataError
6
7 from src.models.paper import Paper
8 from src.models.base import Base
9 from src.core.database import Database
10 from src.impl.logger import get_logger
11
12
13 class MySQLDatabase(Database):
14     def __init__(self):
15         self.logger = get_logger(__name__)
16         self.logger.info("[MySQLDatabase] Initialization started")
17
18         user = os.getenv("MYSQL_USER")
19         password = os.getenv("MYSQL_PASSWORD")
20         host = os.getenv("MYSQL_HOST", "mysql_db")
21         database = os.getenv("MYSQL_DATABASE")
22
23         self.logger.debug("[MySQLDatabase] Env vars - USER: %s, HOST: %s, DB: %s", user, host, database)
24
25         if not all([user, password, database]):
26             raise ValueError("[MySQLDatabase] Missing env vars: MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE")
27
28         self.db_url = f"mysql+pymysql://{user}:{password}@{host}/{database}"
29         self.logger.info("[MySQLDatabase] Connection URL: %s", self.db_url)
30
31         self.engine = None
32         self.Session = None
33
34     try:
35         super().__init__()
36         self.logger.info("[MySQLDatabase] super().__init__ successful")
37     except Exception as e:
38         self.logger.exception("[MySQLDatabase] Error calling super(): %s", e)
39         raise
40
41     try:
42         self.logger.info("[MySQLDatabase] Connection established successfully")
43     except Exception as e:
44         self.logger.exception("[MySQLDatabase] Error while establishing connection: %s", e)
45         raise
46
47     def connect(self):
48         self.logger.info("[MySQLDatabase] Creating engine and session...")
49         self.engine = create_engine(self.db_url, echo=False, pool_pre_ping=True)
50         self.Session = scoped_session(sessionmaker(bind=self.engine))
51         self.logger.info("[MySQLDatabase] Engine and Session created. Creating tables...")
52         Base.metadata.create_all(self.engine)
53         self.logger.info("[MySQLDatabase] Tables created")
54
55     def disconnect(self):

```

Christian Vorhofer
 Raphael Ladinig

```

56         self.logger.info("[MySQLDatabase] Disconnecting from database...")
57     if self.Session:
58         self.Session.remove()
59     if self.engine:
60         self.engine.dispose()
61     self.logger.info("[MySQLDatabase] Disconnected")
62
63     def get_session(self):
64         return self.Session()
65
66     def add_paper(
67         self,
68         arxiv_id: str,
69         text: str,
70         title: str = None,
71         author: str = None,
72         subject: str = None,
73         keywords: str = None,
74         creator: str = None,
75         producer: str = None,
76         creation_date: datetime = None,
77         modification_date: datetime = None,
78         trapped: str = None,
79         skip_if_exists: bool = True,
80     ):
81         session = self.get_session()
82         try:
83             if skip_if_exists and self.paper_exists(arxiv_id):
84                 self.logger.info("[MySQLDatabase] Paper %s already exists, skipping insert.", arxiv_id)
85                 return arxiv_id
86
87             self.logger.info("[MySQLDatabase] Inserting paper: %s", arxiv_id)
88             paper = Paper(
89                 arxiv_id=arxiv_id,
90                 text=text,
91                 title=title,
92                 author=author,
93                 subject=subject,
94                 keywords=keywords,
95                 creator=creator,
96                 producer=producer,
97                 creation_date=creation_date,
98                 modification_date=modification_date,
99                 trapped=trapped,
100            )
101            session.add(paper)
102            session.commit()
103            self.logger.info("[MySQLDatabase] Paper saved: %s", arxiv_id)
104            return arxiv_id
105
106        except IntegrityError as e:
107            session.rollback()
108            self.logger.warning("[MySQLDatabase] IntegrityError on %s, skipping. Detail: %s", arxiv_id, e)
109            return None
110
111        except DataError as e:
112            session.rollback()
113            author_len = len(author) if author is not None else 0
114            self.logger.warning("[MySQLDatabase] DataError on %s (author_len=%d), skipping. Detail: %s", arxiv_id, author_len)
115            return None
116
117        except SQLAlchemyError as e:
118            session.rollback()
119            self.logger.exception("[MySQLDatabase] Unexpected SQLAlchemyError on %s: %s", arxiv_id, e)
120            return None
121
122        finally:
123            session.close()
124
125    def get_paper_by_arxiv_id(self, arxiv_id: str):
126        session = self.get_session()
127        try:
128            self.logger.info("[MySQLDatabase] Fetching paper by ID: %s", arxiv_id)
129            return session.query(Paper).filter_by(arxiv_id=arxiv_id).first()
130        finally:

```

```

131         session.close()
132
133     def get_all_papers(self):
134         session = self.get_session()
135         try:
136             self.logger.info("[MySQLDatabase] Fetching all papers")
137             return session.query(Paper).all()
138         finally:
139             session.close()
140
141     def delete_paper(self, arxiv_id: str):
142         session = self.get_session()
143         try:
144             self.logger.info("[MySQLDatabase] Deleting paper with ID: %s", arxiv_id)
145             paper = session.query(Paper).filter_by(arxiv_id=arxiv_id).first()
146             if paper:
147                 session.delete(paper)
148                 session.commit()
149                 self.logger.info("[MySQLDatabase] Paper deleted: %s", arxiv_id)
150             return True
151         except SQLAlchemyError as e:
152             self.logger.exception("[MySQLDatabase] Error deleting paper %s: %s", arxiv_id, e)
153             return False
154         finally:
155             session.close()
156
157     def get_statistics(self):
158         session = self.get_session()
159         try:
160             self.logger.info("[MySQLDatabase] Gathering database statistics")
161             total_papers = session.query(Paper).count()
162             latest_paper = session.query(Paper).order_by(Paper.creation_date.desc()).first()
163             earliest_paper = session.query(Paper).order_by(Paper.creation_date.asc()).first()
164
165             db_name = (self.engine.url.database if self.engine else None) or os.getenv("MYSQL_DATABASE")
166
167             db_size = None
168             if self.engine and db_name:
169                 with self.engine.connect() as conn:
170                     result = conn.execute(
171                         text("""
172                             SELECT ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
173                             FROM information_schema.tables
174                             WHERE table_schema = :db
175                             """
176                         ),
177                         {"db": db_name},
178                     )
179                     db_size = result.scalar_one_or_none()
180
181             stats = {
182                 "total_papers": total_papers,
183                 "latest_paper_date": latest_paper.creation_date if latest_paper else None,
184                 "earliest_paper_date": earliest_paper.creation_date if earliest_paper else None,
185                 "database_size_mb": db_size,
186             }
187             self.logger.info("[MySQLDatabase] Statistics: %s", stats)
188             return stats
189         finally:
190             session.close()
191
192     def paper_exists(self, arxiv_id: str) -> bool:
193         session = self.get_session()
194         try:
195             return session.get(Paper, arxiv_id) is not None
196         finally:
197             session.close()

```

Listing 6.7: MySQL-Datenbankanbindung für Paper-Metadaten

Relationale Datenbanken eignen sich besonders für konsistente Speicherung,

Christian Vorhofer
 Raphael Ladinig

Filterung und relationale Abfragen auf Metadaten.

6.8 Qdrant als Vektordatenbank

Für die semantische Suche werden die Embeddings in einer spezialisierten Vektordatenbank gespeichert. Qdrant ermöglicht effiziente Approximate-Nearest-Neighbor-Suche auf hochdimensionalen Vektoren.

```

1  import os
2  import uuid
3  from qdrant_client import QdrantClient
4  from qdrant_client.http.models import PointStruct
5  from qdrant_client.models import Distance, VectorParams
6  from src.core.database import Database
7  from src.impl.logger import get_logger
8
9
10 class QdrantDatabase(Database):
11     def __init__(self):
12         self.logger = get_logger(__name__)
13         self.collection_name = os.getenv("QDRANT_COLLECTION", "chunks")
14         self.host = os.getenv("QDRANT_HOST", "qdrant_db")
15         self.port = int(os.getenv("QDRANT_PORT", 6333))
16         self.vector_size = int(os.getenv("VECTOR_SIZE", 768))
17         self.client = None
18
19         self.logger.info(
20             "[QdrantDatabase] Initialized with host=%s, port=%d, collection=%s, vector_size=%d",
21             self.host, self.port, self.collection_name, self.vector_size
22         )
23
24     try:
25         super().__init__()
26         self.logger.debug("[QdrantDatabase] Super init successful")
27     except Exception as e:
28         self.logger.exception("[QdrantDatabase] Error during super init: %s", e)
29         raise
30
31     def connect(self):
32         self.logger.info("[QdrantDatabase] Connecting to Qdrant at %s:%d...", self.host, self.port)
33         self.client = QdrantClient(host=self.host, port=self.port)
34
35     try:
36         self.client.get_collection(self.collection_name)
37         self.logger.info("[QdrantDatabase] Collection '%s' already exists.", self.collection_name)
38     except Exception:
39         self.logger.warning("[QdrantDatabase] Collection '%s' not found. Creating a new one...", self.collection_name)
40         try:
41             self.client.recreate_collection(
42                 collection_name=self.collection_name,
43                 vectors_config=VectorParams(
44                     size=self.vector_size,
45                     distance=Distance.COSINE
46                 )
47             )
48             self.logger.info("[QdrantDatabase] Collection '%s' created successfully.", self.collection_name)
49         except Exception as e:
50             self.logger.exception("[QdrantDatabase] Failed to create collection '%s': %s", self.collection_name, e)
51             raise
52
53     def disconnect(self):
54         if self.client:
55             self.client = None
56             self.logger.info("[QdrantDatabase] Disconnected from Qdrant.")
57         else:

```

Christian Vorhofer
 Raphael Ladinig

```

58         self.logger.warning("[QdrantDatabase] Disconnect called, but client was already None.")
59
60     def add_chunk(
61         self,
62         arxiv_id: str,
63         embedding,
64         text: str,
65         max_retries: int = 3,
66         base_delay: float = 1.0,
67     ):
68         if not arxiv_id or embedding is None or text is None:
69             self.logger.error(
70                 "[QdrantDatabase] Missing required fields (arxiv_id=%s, embedding=%s, text=%s).",
71                 arxiv_id,
72                 type(embedding),
73                 "present" if text else "None",
74             )
75             raise ValueError("Both arxiv_id, embedding, and text are required.")
76
77         if hasattr(embedding, "tolist"):
78             embedding_list = embedding.tolist()
79         else:
80             embedding_list = embedding
81
82         if isinstance(embedding_list[0], (list, tuple)):
83             embedding_list = embedding_list[0]
84
85         pk = str(uuid.uuid4())
86         self.logger.debug(
87             "[QdrantDatabase] Adding chunk with ID %s for paper %s", pk, arxiv_id
88         )
89
90         point = PointStruct(
91             id=pk,
92             vector=embedding_list,
93             payload={
94                 "arxiv_id": arxiv_id,
95                 "text": text,
96             },
97         )
98
99         attempt = 0
100        while attempt < max_retries:
101            attempt += 1
102            try:
103                self.client.upsert(
104                    collection_name=self.collection_name,
105                    points=[point],
106                    wait=True,
107                )
108                self.logger.info(
109                    "[QdrantDatabase] Added chunk %s for paper %s (attempt %d/%d)",
110                    pk,
111                    arxiv_id,
112                    attempt,
113                    max_retries,
114                )
115            return pk
116
117        except (
118            ResponseHandlingException,
119            httpx.RemoteProtocolError,
120            httpcore.RemoteProtocolError,
121        ) as e:
122            if attempt >= max_retries:
123                self.logger.exception(
124                    "[QdrantDatabase] Giving up on chunk %s for paper %s after %d attempts: %s",
125                    pk,
126                    arxiv_id,
127                    attempt,
128                    e,
129                )
130                raise
131
132        delay = min(base_delay * (2 ** (attempt - 1)), 30.0)

```

```

133         self.logger.warning(
134             "[QdrantDatabase] Transient error when upserting chunk %s "
135             "for paper %s (attempt %d/%d): %s      retrying in %.1fs",
136             pk,
137             arxiv_id,
138             attempt,
139             max_retries,
140             e,
141             delay,
142         )
143         time.sleep(delay)
144
145     except Exception as e:
146         self.logger.exception(
147             "[QdrantDatabase] Failed to upsert chunk %s for paper %s (no retry): %s",
148             pk,
149             arxiv_id,
150             e,
151         )
152         raise
153
154     def get_similar(self, embedding: list[float], top_k: int = 5):
155         self.logger.debug("[QdrantDatabase] Searching for top %d similar embeddings.", top_k)
156
157         if hasattr(embedding, "tolist"):
158             embedding = embedding.tolist()
159
160         if isinstance(embedding, list) and isinstance(embedding[0], list):
161             embedding = embedding[0]
162
163         try:
164             search_result = self.client.search(
165                 collection_name=self.collection_name,
166                 query_vector=embedding,
167                 limit=top_k
168             )
169             self.logger.info("[QdrantDatabase] Found %d similar vectors.", len(search_result))
170             return search_result
171         except Exception as e:
172             self.logger.exception("[QdrantDatabase] Error during similarity search: %s", e)
173             raise
174
175     def get_statistics(self):
176         self.logger.info("[QdrantDatabase] Fetching collection statistics for '%s'", self.collection_name)
177         try:
178             stats = self.client.get_collection(self.collection_name)
179             data = {
180                 "points_count": stats.points_count,
181                 "segments_count": getattr(stats, "segments_count", None),
182             }
183             self.logger.info("[QdrantDatabase] Statistics: %s", data)
184             return data
185         except Exception as e:
186             self.logger.exception("[QdrantDatabase] Failed to fetch collection stats: %s", e)
187             raise

```

Listing 6.8: Qdrant-Datenbank für semantische Suche

Durch den Einsatz von HNSW-basierten Indexstrukturen können auch sehr große Paper-Sammlungen performant durchsucht werden.

Christian Vorhofer
 Raphael Ladinig

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 Ablauf.

```

1  from tqdm.auto import tqdm
2  from src.impl.simple_data_provider import SimpleDataProvider
3  from src.impl.fitz.pdf_converter import FitzPdfConverter
4  from src.impl.specter_2_embedder import Specter2Embedder
5  from src.impl.mysql_database import MySQLDatabase
6  from src.impl.qdrant_database import QdrantDatabase
7  from src.impl.logger import get_logger
8  import os
9
10
11 class Pipeline:
12     def __init__(self, relational_db, vector_db, data_provider, pdf_converter, embedder):
13         self.mysql_db = relational_db
14         self.qdrant_db = vector_db
15         self.data_provider = data_provider
16         self.pdf_converter = pdf_converter
17         self.embedder = embedder
18         self.logger = get_logger(__name__)
19         self.logger.debug(
20             "Pipeline initialized with %s, %s, %s, %s, %s",
21             type(relational_db).__name__,
22             type(vector_db).__name__,
23             type(data_provider).__name__,
24             type(pdf_converter).__name__,
25             type(embedder).__name__
26         )
27
28         self._tqdm_disable = False
29
30         self._paper_bar_format = "{l_bar}{bar}! {n_fmt} papers [{elapsd}<{remaining}, {rate_fmt}]"
31         self._chunk_bar_format = (
32             "{l_bar}{bar}! {n_fmt}/{total_fmt} chunks "
33             "[{elapsd}<{remaining}, {rate_fmt}{postfix}]"
34         )
35
36     def process(self):
37         self.logger.info("Pipeline processing started.")
38
39         with tqdm(
40             desc="Papers",
41             unit="paper",
42             dynamic_ncols=True,
43             mininterval=0.5,
44             bar_format=self._paper_bar_format,
45             disable=self._tqdm_disable,
46         ) as paper_bar:
47             while self.data_provider.hasNext():
48                 result = self.data_provider.next()
49                 if result is None:
50                     self.logger.warning("Data provider returned None      no more papers or an error occurred.")
51                     break
52
53                 arxiv_id, pdf_data = result
54                 if arxiv_id is None or pdf_data is None:
55                     self.logger.warning(
56                         "Received invalid paper result (arxiv_id=%s, pdf_data=%s). Skipping...",
57                         arxiv_id,
58                         "None" if pdf_data is None else "bytes",
59                     )
60                     continue
61

```

```

62     if self.mysql_db.paper_exists(arxiv_id):
63         self.logger.info("Paper %s already exists      skipping.", arxiv_id)
64         continue
65
66     self.logger.info("Processing paper: %s", arxiv_id)
67
68     try:
69         text = self.pdf_converter.pdf_to_string(pdf_data)
70     except Exception as e:
71         self.logger.error("Failed to extract text for %s: %s", arxiv_id, e, exc_info=True)
72         continue
73
74     if not text or not text.strip():
75         self.logger.warning("No text extracted for %s      skipping.", arxiv_id)
76         continue
77
78     try:
79         metadata = self.pdf_converter.pdf_metadata(pdf_data)
80     except Exception as e:
81         self.logger.error("Failed to extract metadata for %s: %s", arxiv_id, e, exc_info=True)
82         metadata = {}
83
84     cleaned_text = self.pdf_converter.clean_string(text)
85     chunks = self.pdf_converter.chunk_string(cleaned_text)
86
87     if not chunks:
88         self.logger.warning("No chunks generated for %s      skipping embedding.", arxiv_id)
89         paper_bar.update(1)
90         continue
91
92     self.mysql_db.add_paper(
93         arxiv_id=arxiv_id,
94         text=text,
95         title=metadata.get("title"),
96         author=metadata.get("author"),
97         subject=metadata.get("subject"),
98         keywords=metadata.get("keywords"),
99         creator=metadata.get("creator"),
100        producer=metadata.get("producer"),
101        creation_date=metadata.get("creationDate"),
102        modification_date=metadata.get("modDate"),
103        trapped=metadata.get("trapped"),
104    )
105    self.logger.info("Created DB record for paper %s (%d chunks)", arxiv_id, len(chunks))
106
107    try:
108        embeddings = self.embedder.embed(chunks)
109    except Exception as e:
110        self.logger.error("Failed to embed chunks for %s: %s", arxiv_id, e, exc_info=True)
111        paper_bar.update(1)
112        continue
113
114    with tqdm(
115        total=len(chunks),
116        desc=f"Embedding {arxiv_id}",
117        unit="chunk",
118        dynamic_ncols=True,
119        mininterval=0.1,
120        leave=False,
121        bar_format=self._chunk_bar_format,
122        disable=self._tqdm_disable,
123    ) as chunk_bar:
124        for idx, (chunk, embedding) in enumerate(zip(chunks, embeddings), start=1):
125            self.qdrant_db.add_chunk(arxiv_id, embedding.tolist(), chunk)
126            chunk_bar.update(1)
127            if idx % 25 == 0 or idx == len(chunks):
128                chunk_bar.set_postfix_str(f" last={idx}")
129                self.logger.debug("Embedded %d/%d chunks for %s.", idx, len(chunks), arxiv_id)
130
131    paper_bar.set_postfix_str(f"last={arxiv_id} chunks={len(chunks)}")
132    paper_bar.update(1)
133

```

```
134     self.logger.info("Pipeline processing finished.")
```

Listing 6.9: Pipeline zur Verarbeitung und Indexierung von Papers

Der Ablauf gliedert sich dabei in folgende Schritte:

1. Abruf neuer Papers aus der Datenquelle,
2. Download und Textextraktion aus PDFs,
3. Chunking und Embedding der Texte,
4. Speicherung von Metadaten und Vektoren.

Durch die klare Trennung der einzelnen Schritte bleibt die Pipeline leicht erweiterbar und testbar.

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, 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.

Christian Vorhofer
Raphael Ladinig

Appendix

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

5.1	Abstrakte LLM-Schnittstelle	24
5.2	OllamaLLM als konkrete Implementierung	24
5.3	Abstrakte Agent-Schnittstelle	24
5.4	PluginAgent mit ReAct-Agententyp	25
5.5	Abstrakte Plugin-Basisklasse	26
5.6	Dynamischer PluginLoader	27
5.7	AppPlugin als konkretes Plugin	28
6.1	Abstrakte Schnittstelle für Datenquellen	32
6.2	arXivDataProvider zur Anbindung der arXiv-API	32
6.3	PDF-zu-Text-Konvertierung mit PyMuPDF	35
6.4	Abstrakte Embedder-Schnittstelle	38
6.5	Specter2Embedder zur Erzeugung semantischer Vektoren	39
6.6	Abstrakte Datenbankschnittstelle	40
6.7	MySQL-Datenbankanbindung für Paper-Metadaten	41
6.8	Qdrant-Datenbank für semantische Suche	44
6.9	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>