Hochschule für Technik, Wirtschaft und Kultur Leipzig

Fakultät Informatik, Mathematik und Naturwissenschaften

Bachelorstudiengang Informatik

Bachelorarbeit

zur Erlangung der akademischen Grades

Bachelor of Science (B.Sc.)

Vergleichsmethoden und Bewertung von Vektor-DBMS für KI

Eingereicht von: Lukas Westholt

[redacted]

Leipzig 22. Oktober 2024

Erstprüfer: Prof. Dr. Karsten Weicker, HTWK Leipzig

Zweitprüfer: [redacted] Exxeta AG

Abstract

KI-Suchsysteme oder -Chatbots sind gesellschaftliche Zukunftstechnologien und zunehmend auf die effiziente Verarbeitung unstrukturierter Daten in Form von Embeddings angewiesen. Sie haben unterschiedliche Anforderungen an Vektor-DBMS (VDBMS), deren Markt in den letzten fünf Jahren aufgrund steigender Auswahl unübersichtlich geworden ist.
Ziel dieser Arbeit ist die Erleichterung der mehrdimensionalen Wahl zwischen VDBMS mittels Daten der systematischen Evaluation von verhältnisskalierten Leistungsmetriken sowie nominalskalierten Merkmalen. Die Methodik umfasst ein eigens entwickeltes Benchmarking-System, welches Latenz, Durchsatz und Ressourcennutzung misst. Zugunsten der Reproduzierbarkeit des Systems wurde konsequent auf allgemein verfügbare Cloudumgebung per Infrastructure-as-Code und automatisierte Verfahren gesetzt. Zudem wird Vererbung und das Adapter-Design-Pattern genutzt, um die Erweiterbarkeit der Software sicherzustellen. Analysiert werden neun gängige VDBMS (darunter Qdrant, Redis und Chroma) für vier Anwendungsfälle in Industrie, Handel und Wissenschaft. Vergleichend werden publizierte Benchmarking-Systeme herangezogen. Um die Eignung von VDBMS individuell zu bewerten, wird ein algorithmusbasiertes Empfehlungssystem vorgestellt, bei dem neben den genannten Parametern fallspezifische Anforderungen wie Transaktionsunterstützung berücksichtigt werden.

Die Arbeit liefert Messdaten von zwei Testaufbauten und zeigt, dass Qdrant in den Kategorien Latenz, Abfragedurchsatz und Upload-Zeit eines Datensatzes die beste Leistung erzielt, auch bei hohen Präzisionsanforderungen. Redis überzeugt durch geringe Latenz, während Chroma Schwächen aufweist. Die Evaluation der nominalen Merkmale ergibt zwei Kategorien von VDBMS: 1. traditionelle DBMS mit Vektorerweiterung unterstützen beispielsweise Transaktion und 2. dezidierte VDBMS, welche nur abgeschwächte Konsistenz unterstützen. Das Empfehlungssystem wird für die Anwendungsfälle eingesetzt und ermittelt unterschiedliche Produktlösungen. Die Ergebnisse deduzieren eine Präzedenz für Qdrant, betonen präzise Selektion und unterstützen Entwickler:innen bei der Wahl des geeigneten VDBMS.

Abstract

AI search systems or chatbots are societal technologies of the future and are increasingly dependent on the efficient processing of unstructured data in the form of embeddings. They have different requirements for vector DBMSs (VDBMS), whose market has become congested in the last five years due to a broadening spectrum.
The aim of this work is to facilitate the multidimensional choice between VDBMSs using data from the systematic evaluation of ratio-scaled performance metrics and nominal-scaled features. The methods include a specifically developed benchmarking system that measures latency, throughput and resource utilisation. In favour of the reproducibility of the system, a generally available cloud environment via infrastructure-as-code and automated processes were consistently used. In addition, inheritance and the adapter design pattern are used to ensure the extensibility of the software. Nine common VDBMSs (including Qdrant, Redis and Chroma) are analysed for four use cases in industry, commerce and science. Published benchmarking systems are used for comparison. In order to evaluate the suitability of VDBMSs individually, an algorithm-based recommendation system is presented, in which case-specific requirements such as transaction support are taken into account in addition to the parameters mentioned.

We provide measurement data from two test setups and show that Qdrant achieves the best performance in the categories of latency, query throughput and upload time of a data set, especially with high precision requirements. Redis impresses with its low latency, while Chroma shows weaknesses. The evaluation of the nominal characteristics results in two categories of VDBMSs: 1. traditional DBMSs with vector extension support, for example, transaction and 2. dedicated VDBMSs, which only support weakened consistency. Our recommendation system is applied to the use cases and determines different product solutions. The results deduce a precedent for Qdrant, emphasise precise selection and support developers in choosing the appropriate VDBMS.

Inhaltsverzeichnis

1 Einleitung **1**

2 Embeddings und VDBMS **5**
2.1 Embeddings 5
2.2 Vektordatenbankmanagementsysteme 9
2.3 Verfahren zur Berechnung der Ähnlichkeit 12
2.4 Verfahren zur Berechnung des nächsten Nachbarn 14

3 Anforderungsanalyse **19**
3.1 Schlüsselszenarien zum Einsatz von VDBMS 19
3.2 Wahl und Eigenschaften der VDBMS 20
3.3 Vergleichsmetriken für Datenbanken 22
3.4 Weitere nominale Vergleichseigenschaften 24
3.5 Ergebnisse der Anforderungsanalyse 25

4 Architektur des Testsystems **27**
4.1 Management- und Benchmarking-System für VDBMS 27
4.2 Cloud-Infrastruktur 30
4.3 Rastersuche und Laufzeitanalyse 31

5 Evaluation **34**
5.1 Messdaten des ersten Aufbaus 34
5.2 Messdaten des zweiten Aufbaus 35
5.3 Leistungsvergleich der VDBMS 36
5.4 Empfehlungssystem 40
5.5 Auswertung der Schlüsselszenarien anhand des Empfehlungssystems 42

6 Zusammenfassung und Ausblick **44**

Glossar . . . . . **III**

Abkürzungsverzeichnis . . . . . **V**

Literatur . . . . . **VII**

Abbildungsverzeichnis . . . . . **XI**

Tabellenverzeichnis . . . . . **XII**

Anhang . . . . . **XIII**

A Resultate der Experimente: Datentabellen XIII

B Beispiel: VDBMS-Empfehlungssystem XVIII

C Arbeit mit Leipzig Corpora Collection XX

D Ausblick: GUI für Bereitstellung von VDBMS XXI

1 Einleitung

Mit dem Aufkommen der Popularität von *OpenAI GPT-4o* für *ChatGPT* und der damit verbundenen Möglichkeit, Fragen an hochgeladene Textdateien wie PDFs zu stellen, wächst auch bei anderen Firmen das Interesse an einer effizienten Verarbeitung und Verwaltung großer und unstrukturierter Datenmengen. Nutzer:innen kennen nun den Komfort und die Vorteile des Angebots von *OpenAI*, wollen aus Kosten- und Datenschutzgründen aber häufig ein eigenes dokumentenbasiertes Frage-Antwort-Modell entwickeln. Die *Exxeta AG* und andere Unternehmen, die eine KI-Strategie entwickeln, erweitern ihre Dienstleistungen durch den Einsatz von Vektordatenbanken (VDB). So steigt auch der Bedarf an VDB[1], da diese sich hier als geeignete Form der Datenverwaltung herausgestellt haben. Mit den verschiedenen Anforderungen und neuen Vektordatenbankmanagementsystemen (VDBMS) zeigt sich die Notwendigkeit eines Leistungsvergleichs der Produkte für die Entwicklung eines evidenzbasierten Entscheidungssystems zur Auswahl des geeigneten Produkts. In dieser Arbeit soll mithilfe von eigens entwickelten Methodiken eine systematische Untersuchung von Umfang und Performanz unter Nutzung und als Weiterentwicklung bestehender Ansätze vollzogen werden.

In der aktuellen Entwicklung von maschinellem Lernen gibt es einen Trend, der als „Einbettung von allem“ („Embed Everything“) bezeichnet wird (Di Sarah et al., 2021).[2] Diese Entwicklung ist sowohl für neue KI-Anwendungen als auch für Bestandsfälle in Industrie, Handel und Wissenschaft von Bedeutung. Daten werden in hochdimensionale Vektoren eingebettet, um sie in VDBMS zu verwalten. Um VDB in diesen Anwendungen einzusetzen, muss zunächst der Begriff der *Embeddings* und deren Rolle im maschinellen Lernen geklärt werden (Kapitel 2).
Embeddings sind Vektoren, die typischerweise aus 256 oder 512 Fließkommazahlen bestehen und eine kompakte Repräsentation semantischer Informationen von unstrukturierten Daten wie Text, Bild, Audio oder Video darstellen. Diese Informationen spiegeln sich in der Beziehung zwischen *Embeddings* wider, messbar als (euklidischer) Abstand. So liegen *Embeddings* eines ähnlichen Konzepts (z.B. zwei Bilder von Paradiesvögeln) trotz unterschiedlicher Darstellung (z.B. Pixel) im Raum nahe beieinander.

[1] Abkürzungen und Fachbegriffe werden im Abkürzungsverzeichnis oder Glossar erläutert.
[2] Begriff in Anlehnung an Boris Power, *Head of Applied Research* bei *OpenAI*. Link: https://x.com/BorisMPower/status/1604096153263296512 (Stand: 21. Oktober 2024).

Ein VDBMS ist darauf ausgelegt, diese *Embeddings* zu speichern, zu indexieren und eine effiziente Suche durchzuführen. Dies ist wichtig für KI-Anwendungen, beispielsweise bei der Rückwärtssuche, bei der ausgehend vom Eingabebild ähnliche Bilder basierend auf ihrem semantischen Inhalt gefunden werden sollen.

Das Abbilden unstrukturierter Daten in *Embeddings* wird hauptsächlich durch Modelle des maschinellen Lernens realisiert, ist aber nicht auf diese Technik beschränkt. Zur Abgrenzung zum KI-Kontext: Auch biometrische Messdaten (z.B. Fingerabdrücke) zur Identitätsverifikation können beispielsweise in VDBMS verwaltet werden.

Traditionelle Datenbanken, die für strukturierte Daten konzipiert sind, können dies nicht, da sie auf einem festen Schema basieren. Der Bedarf an Werkzeugen zur Verwaltung von *Embeddings* hat zur Entwicklung von VDBMS geführt, welche verteilt, cloud-native und horizontal skalierbar sein können. Johnson et al. (2017) betonen, dass Milliarden *Embeddings* verwaltet werden können, was eine neue Größenordnung darstellt. Diese Entwicklung zielt darauf ab, die Leistungsfähigkeit der Vektorisierung von Daten zu maximieren und das Konzept des „Einbettens der Welt“ weiter voranzutreiben.

Dass hierfür eine Notwendigkeit besteht, zeigt der Global DataSphere-Bericht von der International Data Corporation (IDC), welcher im Jahr 2022 (Rydning, zitiert nach Burgener und Rydning, 2022, S. 3 f.) prognostizierte, dass 2026 über 90 % der jährlich erzeugten Daten unstrukturiert sein werden (siehe Abbildung 1.1).

In der Anforderungsanalyse (Kapitel 3) werden zunächst vier vorstellbare Schlüsselszenarien zum Einsatz von VDBMS vorgestellt. Neun VDBMS sowie ihre Vergleichsmerkmale werden eingeführt, außerdem werden Hardwareressourcen benannt.

Für die in dieser Arbeit postulierte Leistungsmessung werden in Kapitel 4.1 und 4.2 Schwerpunkte in Robustheit (1), Reproduzierbarkeit (2) und Erweiterbarkeit (3) gesetzt. Für die Umsetzung dessen wird Automatisierung (1), Cloud- und Infrastructure-as-Code-Technologie (2) sowie Vererbung in objektorientierter Programmierung (3) genutzt. Um den Mehrwert der Ergebnisse dieser Methodik zu demonstrieren, führt Kapitel 5.4 zu einem Prototyp, dessen Kern ein algorithmusbasiertes Empfehlungssystem für VDBMS darstellt. Durch die Eingabe von szenariotypischen Parametern wird ein Score errechnet, welcher die Eignung der VDBMS für den spezifischen Zweck widerspiegelt. In Kapitel 4.3 wird die Ausführung des Benchmarks beschrieben sowie eine Laufzeitanalyse des Algorithmus ausgeführt.

Die Ergebnisse der Messung zweier Cloud-Aufbauten werden untersucht, verglichen und abgebildet (5.1, 5.2). Empfehlungen für die Schlüsselszenarien werden ausgesprochen. Auf Probleme bei der Ausführung und ihre Auswirkung auf die Testergebnisse wird eingegangen (5.3).

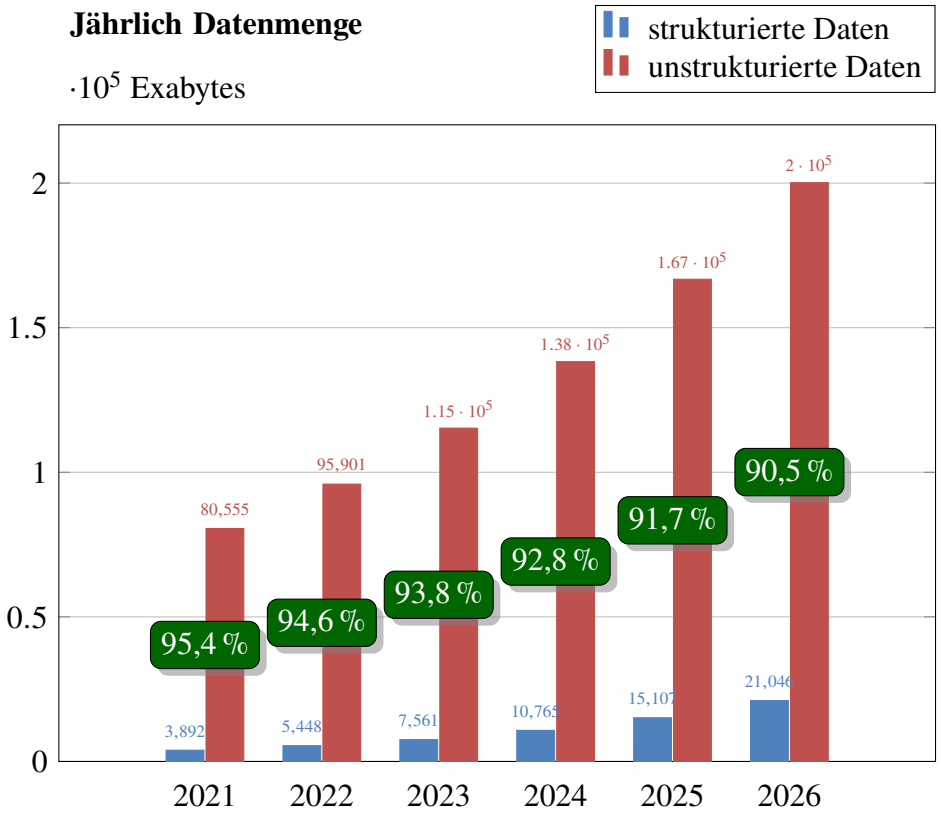


Abbildung 1.1: Prognose für jährliche Datenmenge aus dem Global DataSphere-Bericht von IDC von Rydning (2022, zitiert nach Burgener und Rydning, 2022, S. 3 f.). Ein Exabyte entspricht 1000 Petabyte. Der Wert im grünen Knoten steht für den Anteil der unstrukturierten Daten je Jahr.

Im letzten Kapitel folgt die Zusammenfassung der wichtigsten Ergebnisse und die Schlussfolgerung, dass das VDBMS *Qdrant* die höchste Leistung erbringt. Zudem erfolgt eine kritische Bewertung des Systems durch Verbesserungsvorschläge.

Die bisherige Forschung im Bereich von Vektordatenbanken und KI-Systemen zeigt auf, dass maschinelles Lernen und *Embeddings* entscheidend sind, um unstrukturierte Daten effizient zu verarbeiten. Es gibt bereits ein System, das sich mit dem Vergleich solcher Vektordatenbanksysteme beschäftigt. Es handelt sich um die Open-Source-Software von Qdrant Solutions GmbH (2022). Darüber hinaus untersucht der *ANN-Benchmark*, ebenfalls Open-Source, die darunterliegenden Algorithmen. Diese zwei Systeme werden in der Arbeit betrachtet und konkrete Ableitungen getroffen (siehe Kapitel 3).

Durch die reproduzierbaren Analysen der VDBMS und ihrer Leistungen hinsichtlich der aus der Literatur motivierten Metriken leistet diese Arbeit einen wissenschaftlichen Beitrag, da bisherige Untersuchungen zu den ähnlichen Datenbankmanagementsystemen (DBMS) teils widersprüchliche Ergebnisse lieferten (siehe Diskussion in Kapitel 4 über *MongoDB* und *Cassandra*).

In der Arbeit werden neue Metriken wie Latenz vorgestellt, die auf spezifische Anforderungen der Vektordatenbanken abzielen, welche in bestehenden Studien nicht hinreichend analysiert wurden. So werden bestehende Forschungslücken identifiziert, etwa im Hinblick auf die Nutzung von Hybridmodellen, der Kombination aus strukturierten und unstrukturierten Daten (Kapitel 2.2).

2 Embeddings und VDBMS

Wie in der Einleitung beschrieben, können KI-Modelle unstrukturierte Daten als Vektoren abbilden. Diese Vektoren spielen in der heutigen Datenverarbeitung als effiziente Repräsentation von Informationen eine immer größer werdende Rolle. Insbesondere in der Computerlinguistik (CL) sind *Embeddings* ein entscheidendes Werkzeug, um die semantische Struktur von Textdaten zu erfassen und zu analysieren.

In diesem Kapitel werden die grundlegenden Konzepte von *Embeddings* untersucht und ihre mathematische Grundlage und Anwendung erörtert, da ihre Speicherung in den VDBMS für den Leistungsvergleich dieser Arbeit notwendig ist. Außerdem werden VDBMS eingeführt und definiert, um sie im Folgenden vergleichen zu können: VDBMS ermöglichen die effiziente Speicherung und Abfrage hochdimensionaler numerischer Vektoren, insbesondere von *Embeddings* unstrukturierter Daten. Auf zwei Konfigurationsmöglichkeiten der VDBMS, Abstandsmetriken der Ähnlichkeitssuche und Indexierungsalgorithmen, wird genauer eingegangen. Diese sind maßgebend für die Eigenschaften der Datensätze (ersteres) und essenziell für die Leistung der Suchalgorithmen (zweiteres).

2.1 Embeddings

In der allgemeinen Topologie bezeichnet eine Einbettung eine injektive Abbildung, die die Struktur von X in einen anderen Raum Y überträgt (Kenyon-Dean et al., 2020, S. 1). Sie kann es ermöglichen, die Eigenschaften von X einfacher zu analysieren (Hocking & Young, 1988, S. 73).

Definition 1 Eine Funktion $f : X \to Y$ heißt **injektiv**, falls für alle $x, y \in X$ aus $f(x) = f(y)$ folgt $x = y$ (Berghammer, 2014, S. 115).

Bei der Worteinbettung handelt es sich um eine Technik aus dem Bereich der CL, bei der Textelemente (wie ein Wort, ein Satz oder ein Dokument, engl: *text chunks*) in einem Vektorraum abgebildet werden.
Im Grundmodell wird die Bedeutung eines Wortes anhand einer einfachen Funktion der

Häufigkeit seiner benachbarten Wörter bestimmt (Koexistenz). Diese Methode erzeugt lange Zählvektoren z, welche dünnbesetzt sind, also überwiegend aus Nullen bestehen, da viele Wörter, basierend auf einem Vokabular V, in den meisten Zusammenhängen nicht vorkommen, siehe Abbildung 2.1 (Jurafsky & Martin, 2009, S. 110). Die Dimensionalität des Zählvektors entspricht der Größe des Vokabulars $|z| = |V|$. Diese Größe variiert üblicherweise zwischen 10.000 und 50.000 Wörtern, basierend auf den häufigsten Begriffen im Trainingskorpus (Jurafsky & Martin, 2009, S. 113 f.). Koexistenz wird dort mit $\pm h$-Wörtern Versatz oder als in einem Dokument vorkommend definiert, wobei $h \in \mathbb{N}$ ist.
Weiterführend, aber hier nicht näher betrachtet, ist die *positive pointwise mutual information (PPMI)*-Gewichtung und das Paradox, dass Wörter, die häufig koexistieren (wie die Wörter Buch und Schule) aussagekräftiger sind als überrepräsentierte Wörter wie Artikel und Füllwörter.

	Jahr	Welt	Klassiker	...	Tatort
Buch	103	23	5	...	0
Schule	161	0	0	...	0
deutschen	0	0	4	...	0
Menschen	0	109	0	...	0
Glaube	0	0	0	...	0

Tabelle 2.1: Wort-Wort-Matrix mit Koexistenzen zwischen Wörtern des Wikipedia-Datensatzes der Leipzig Corpora Collection (Goldhahn et al., 2012). Koexistenzen eines Wortes sind diejenigen Wörter, die in definierter Nähe mit ihm vorkommen. In Anhang C wird die Erhebungsmethode näher erläutert. Tabelle nach Jurafsky und Martin, 2009, S. 113.

Da dünnbesetzte Vektoren nur auf Ko-Vorkommen basieren, versagen sie darin, die semantische Ähnlichkeit von Synonymen (wie glücklich und fröhlich) zu erkennen, wenn sie nicht explizit gemeinsam auftreten und damit distinkt und beziehungslos zueinander sind (Jurafsky & Martin, 2009, S. 121).

Dichte Vektoren sind in der Lage, semantische Ähnlichkeiten zwischen Wörtern besser zu erfassen. Im Gegensatz zu dünnbesetzten Vektoren sind dichte Vektoren (typischerweise $v \in \mathbb{R}^D$) niedrigdimensional (50 <= D <= 1000) (Jurafsky & Martin, 2009, S. 121 f.).

Diese dichten Vektoren sind ein Abbild der dünnbesetzten Zählvektoren. Die Abbildung wurde vor der Linguistic Society of America (LSA) ursprünglich *Embedding* genannt (Jurafsky & Martin, 2009, S. 135). Sie lautet $f\colon C \hookrightarrow \mathbb{R}^D$, wobei C die Menge aller Textelemente in Form von Zählvektoren ist ($z \in C \subset \mathbb{N}^{|V|}$).

$$f : \begin{bmatrix} 246 & 708 & 0 & 541 \cdots \end{bmatrix} \mapsto \begin{bmatrix} 0.8569711893469244 & 0.19933518586690668 & \cdots \end{bmatrix}$$

Diese Abbildung transformiert die Zählvektoren in eine Menge dichter Vektoren eines niedrigdimensionalen Vektorraums, der auch als latenter Raum (eng: latent space) bekannt ist, in dem semantische Beziehungen besser erfasst werden (Liu et al., 2019, S. 1) sowie kostengünstig verarbeitet werden können (Taipalus, 2024b, S. 1). Das Ziel dieser Methode ist es, die Semantik von Wörtern, also die linguistische Struktur, aus einem Referenztextkorpus (C) zu erhalten und dabei die ursprüngliche Dimension der Eingabedaten zu reduzieren. Das durch KI gelöste Optimierungsproblem besteht darin, die Abstands-Differenz

$$d(f(z_1), f(z_2)) - d(z_1, z_2) \tag{2.1}$$

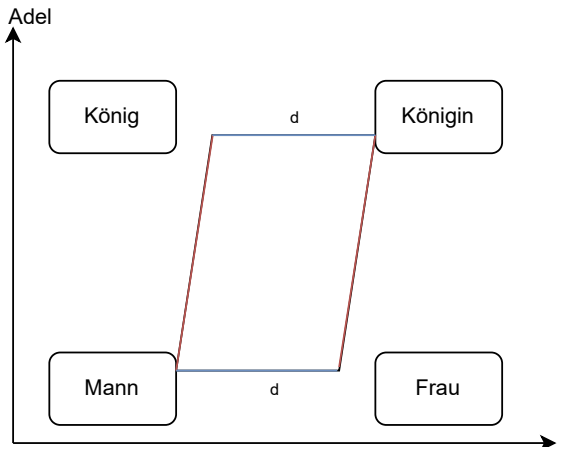
zwischen den Abständen innerhalb der dünnbesetzten Vektoren in den Eingabedaten und denen der dicht-besetzten Vektoren in den Ausgabedaten zu minimieren. Der Fehler dieser Komprimierung (loss function) ist äquivalent zur beschriebenen Abstandsdifferenz (Kenyon-Dean et al., 2020, S. 1 f.). So bleibt die semantische Ähnlichkeit als zentrale Eigenschaft der Textelemente in den dicht-besetzten Vektoren als Abstand bestehen.

Ein beispielhafter Lernalgorithmus zur Worteinbettung (f) ist Global Vectors for Word Representation (GloVe), welcher von der Stanford University als effiziente Methode zur Erzeugung von Wortvektoren entwickelt wurde und unüberwacht mit $n = 300$ und $|V| = 400.000$ lernt (Jeffrey Pennington et al., 2014, S. 1539). Ebenfalls erfolgreich ist *word2vec* (Jurafsky & Martin, 2009, S. 128).

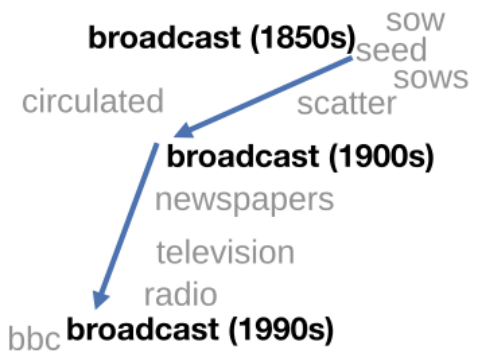
Mit einfachen algebraischen Operationen an den Wortvektoren wurde beispielsweise gezeigt, dass $\overrightarrow{\text{König}} - \overrightarrow{\text{Mann}} + \overrightarrow{\text{Frau}}$ einen Vektor ergibt, welcher der Vektorrepräsentation des Wortes Königin am nächsten kommt (Mikolov, Yih & Zweig, 2013, S. 1). Dieses Beispiel ist in der Abbildung 2.1a zu sehen. Die Nutzung der Analogie-Unterstrukturen des Vektorraums ist für Benchmark-Messungen interessant, um die Qualität des Algorithmus zu messen (Mikolov, Chen et al., 2013, S. 6). So konnte GloVe 75 % der 19544 Fragen wie „Athens is to Greece as Berlin is to ?“ des Benchmarks von Mikolov, Chen et al. (2013, S. 6) korrekt beantworten (Jeffrey Pennington et al., 2014, S. 1537).

Auch der *word2vec*-Algorithmus kann verwendet werden, um Wörter in Vektoren umgewandelt werden. Die Abbildung 2.1b zeigt eine Anwendung dessen in der CL. Durch die Visualisierung des Vektor-Zeit-Raumes ist ersichtlich, dass der Begriff *broadcast* in den 1850er Jahren „Samen ausstreuen“ bedeutete; mit dem Aufkommen von Fernsehen und Radio wandelte sich seine Bedeutung zu „Signalübertragung“.

Um den Begriff der Worteinbettung zu definieren, müssen die spezifischen Merkmale, durch die er gekennzeichnet ist, genauer betrachtet werden. Dazu wird zunächst eine mathematische



(a) Analogie der Vektorgleichung $\overrightarrow{\text{Königin}} - \overrightarrow{\text{König}} = \overrightarrow{\text{Frau}} - \overrightarrow{\text{Mann}} = d$. Abbildung nach Jurafsky und Martin, 2009, S. 129.



(b) Zweidimensionale t-SNE-Visualisierung des semantischen Wandels im Englischen unter Verwendung von *word2vec*-Vektoren. Quelle: Hamilton et al. (Übers. d. Verf., 2016, S. 2).

Abbildung 2.1: Linguistische Nutzung der Vektorräume

Abhandlung des historischen Worthintergrundes durchgeführt, um dann eine Überführung der Definition zum heutigen *Embedding* in der KI vorzunehmen.

Obwohl bei Worteinbettung das Wort *Einbettung* der Suffix ist, ist die Eigenschaft der Injektivität in den Algorithmen fragwürdig. Aus der Definition von der Injektivität (Definition 1) folgt, dass jeder dichte Vektor (Element der Zielmenge) höchstens einen dazugehörigen dünnen Vektor (Element der Definitionsmenge C) besitzen darf. Damit ein Algorithmus also eine Einbettung vornimmt, dürfen nie zwei oder mehr Textelemente auf denselben Vektor abgebildet werden.

Daher stellt sich die weiterführende Frage, ob die nicht-vollständige Funktion $f^{-1} : \mathbb{R}^D \to C$ für die Ausgaben der Einbettungsfunktion existiert. Die Einbettungsumkehr, das heißt die Rekonstruktion des vollständigen Textes aus dem Vektor, kann ähnlich zu Bildgenerierungsmodellen mehrschrittig erfolgreich per maschinellem Lernen erfolgen (Morris et al., 2023). Eine solche Vertiefung wird in dieser Arbeit nicht behandelt.

In diesem Kontext ist es wichtig, sich auf die Terminologie zu konzentrieren, die in der aktuellen Forschung verwendet wird. Während die LSA 1997 den Begriff *Embeddings* seiner mathematischen Bedeutung als Abbildung gerecht nutzt, beschreibt das Wort heute auch den resultierenden dichten Vektor (Jurafsky & Martin, 2009, S. 135).

Um Verwirrung zu vermeiden, wird in der vorliegenden Arbeit aufgrund der metonymischen Verschiebung ab sofort ausschließlich der Begriff *Embeddings* für dichte Vektoren verwendet. Trainierte Algorithmen, die zur Generierung von *Embeddings* eingesetzt werden, beispielsweise GloVe-Modelle, werden in der Literatur als *Embedding-Modelle* bezeichnet.

In der heutigen Nutzung werden *Embeddings* primär von spezialisierten neuronalen Netzwerken erzeugt, die auf CNN-Architektur oder Transformer-Architektur basieren, darunter große Sprachmodelle (Large Language Models, LLMs) wie GPT, BERT oder LLaMA. Diese Netzwerke erlauben die Transformation eines einzelnen Datentyps, beispielsweise Text, Bild oder Video, in Vektorrepräsentationen (W. Li et al., 2020, S. 1479 f.).

Ein fortgeschrittener Ansatz besteht darin, solche Vektoren in einem Multi-Modal-Raum zu vereinen. Dabei handelt es sich um einen gemeinsamen Darstellungsraum, in dem Daten unterschiedlicher Modalitäten in eine einheitliche Repräsentation überführt werden. Ähnliche Konzepte werden modalitätsunabhängig räumlich nah zueinander eingebettet, was die Interaktion zwischen Modalitäten, etwa durch Cross-Modal Retrieval, also die Suche in einer Modalität (z. B. Text) nach Inhalten in einer anderen Modalität (z. B. Audio), ermöglicht, siehe Abbildung 2.2.

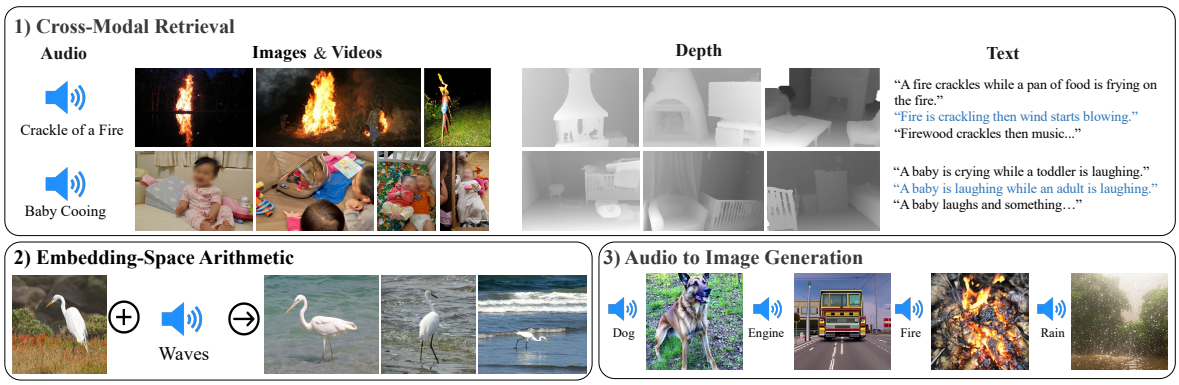


Abbildung 2.2: Der gemeinsame Einbettungsraum von IMAGEBIND ermöglicht neuartige multimodale Fähigkeiten: **1)** Cross-Modal Retrieval, das eine emergente Angleichung von Modalitäten wie Audio, Tiefe oder Text zeigt, die nicht gemeinsam beobachtet werden. **2)** Das Addieren von Einbettungen aus verschiedenen Modalitäten komponiert wie gewohnt deren Semantik. **3)** Audio-to-Image-Generierung mit einem vortrainierten DALLE-2 Decoder. Quelle: Girdhar et al. (Übers. d. Verf., 2023, S. 1).

2.2 Vektordatenbankmanagementsysteme

Im Gegensatz zu traditionellen relationalen Datenbanken, die hauptsächlich tabellarische Daten verwalten, werden in VDBMS numerische Vektoren, bestehend aus natürlichen, reellen oder komplexen Zahlen, verwaltet (Taipalus, 2024b, S. 1). So eignen sich VDBMS dazu, die *Embeddings* zu speichern. Datenbankmanagementsysteme für Vektordaten mit traditionellen Funktionen wie Abfrageoptimierung, Transaktionen, Skalierbarkeit, Fehlertoleranz, Datenschutz und Sicherheit sind besonders für E-Commerce- und Empfehlungssysteme gefragt (Pan et al., 2024, S. 1591). Um schnelle Abfragen zu ermöglichen, verwenden VDBMS

auch Indexierung. Beispielhafte Vektordaten für VDBMS sind neben den *Embeddings* für unstrukturierte Daten, wie Texte, Bilder, Videos und Produktdaten (z.B. für Empfehlungssysteme), auch Messdaten (z.B. Fingerabdrücke), Geodaten und Hashwerte außerhalb des KI-Kontexts. Bei Letzteren kann beispielsweise eine numerische Hashfunktion auf kleine Segmente einer ausführbaren Datei angewendet werden, um Malware zu identifizieren. Dabei wird nach Ähnlichkeiten zu bestehender Malware gesucht.

Da nach der Transformation in den Vektorraum durch das *Embedding-Modell* die Semantik der unstrukturierten Daten größtenteils erhalten bleibt, bilden Vektoren mit niedrigem Abstand zueinander ähnliche Daten ab. Diese Eigenschaft wird in der Ähnlichkeitssuche (engl: *similarity search*) der VDBMS genutzt: Zu einem Eingabevektor werden die k-nächsten Vektoren mit kleinstem Abstand gesucht, wobei k die gewählte Anzahl der als Antwort zurückgegebenen Vektoren ist. Diese Fähigkeit ist ein wichtiger Bestandteil von VDBMS. Sie wird z. B. bei Plagiatserkennung durch Wortähnlichkeit (Jurafsky & Martin, 2009, S. 121) oder im Kontext von Question-Answering bei Retrieval-Augmented Generation (RAG)-Systemen und bei Chatbots als Langzeitgedächtnis genutzt. In RAG-Systemen werden generative und retrival-basierte Ansätze kombiniert, um kontextbehaftete Antworten zu liefern. Zunächst wird das zu durchsuchende Kontextwissen als *Embeddings* in das VDBMS geladen. Wenn eine Anfrage Q eingeht, vektorisiert das System die Frage, um ihren semantischen Gehalt zu erfassen. Anschließend wird mit diesem *Embedding* eine Ähnlichkeitssuche im VDBMS initiiert, um k relevante Einträge abzurufen. Diese werden Q textuell angehängt und von dem LLM verarbeitet (Maryamah et al., 2024, S. 261). Langzeit-Gedächtnisse speichern historische Chats in VDBMS, damit das LLM personalisierte und kontextualisierte Antworten generieren kann.

VDBMS	**Datum**	**Lizenztyp**
Milvus	2019	Open Source (Apache 2.0)
Pinecone	2019	Proprietär
vearch	2019	Open Source (Apache 2.0)
Qdrant	2021	Open Source (Apache 2.0)
Google Vertex AI Matching Engine	2021	Proprietär
pgvector (PostgreSQL extension)	2021	Open Source (PostgreSQL License)
Zilliz Cloud (built on Milvus)	2021	Proprietär
Chroma	2022	Open Source (Apache 2.0)
Elasticsearch with kNN Plugin	2022	Open Source (Apache 2.0)
Redis Vector Similarity Search	2023	Open Source (MIT License)

Tabelle 2.2: Neue VDBMS(-Plugins) mit Erscheinungsdatum und Lizenztyp aus den letzten 5 Jahren. Die Daten werden anhand der frühesten GitHub-Veröffentlichung oder anderer Hinweise geschätzt. Die Daten geben das Jahr an, in dem die Vektorsuchfunktion erstmals im Produkt erschien (vgl. Pan et al., 2024, S. 1609).

Die Entwicklung von VDBMS hat sich in den letzten fünf Jahren stark beschleunigt. In diesem Zeitraum wurden mindestens 10 neue Systeme eingeführt, siehe Tabelle 2.2. Außerdem wurde *Milvus* (J. Wang et al., 2021) im Jahr 2022 komplett überarbeitet, siehe Guo et al. (2022, S. 1). Besonders ist, dass manche traditionelle Datenbanken ihren Funktionsumfang um Vektorsuche erweitern. Diese sind beispielsweise PostgreSQL, *Redis*, MongoDB, *OpenSearch* und *Elasticsearch*. Im Gegensatz dazu stehen Systeme mit Spezialisierung in Vektordaten (dedizierte VDBMS), was Vorteile in Datenverarbeitung und Indexierung ermöglicht. Im Kapitel 2 werden Beispiele zwecks Übersichtlichkeit auf die beiden VDBMS *Milvus* und *pgvector* begrenzt, um ein neueres und ein älteres, bzw. ein generisches und ein dezidiertes VDBMS abzudecken.

Um sie in weiteren Teilen der Arbeit referenzieren zu können, werden hier Merkmale von VDBMS besprochen. Eine für manche Szenarien wichtige Funktion, das Pre-Filtering, ist die Möglichkeit, vor der eigentlichen Suche Filter anzuwenden (Pan et al., 2024, S. 1605). Diese Methode erweist sich als deutlich effizienter als die Ergebnismenge nachträglich zu reduzieren, weil es die Datenmenge von Anfang an verringert und so Rechenressourcen, Zeit und Aufwand spart, was zu schnelleren und relevanteren Suchergebnissen führt. Allgemein wird dies als hybride Suche bezeichnet, bei der strukturierte und unstrukturierte Daten zusammen arbeiten. Ein Anwendungsbeispiel für hybride Suche sind E-Mails. In E-Mails gibt es im Header strukturierte Daten, welche relational gespeichert werden können und im Body unstrukturierte Daten, welche in *Embeddings* gespeichert werden können.

Single Instruction Multiple Data (SIMD)-Unterstützung, sowohl bei Prozessoren (beispielsweise AVX2) als auch bei VDBMS, spielt eine wesentliche Rolle, insbesondere im Hinblick auf hohe Leistung und Effizienz der Berechnung von Ähnlichkeiten. Veröffentlichte Berichte zu SIMD sind nur von *Qdrant*[3], *Milvus*[4] und *Weaviate*[5] verfügbar. Für die weiteren Untersuchungen in dieser Arbeit ist das Wissen über die SIMD-Unterstützung ohne Relevanz, da sich diese im Leistungsvergleich ohnehin auf die Messung auswirkt. Dadurch haben VDBMS mit SIMD-Unterstützung einen Vorteil und zudem einen niedrigeren Stromverbrauch durch höhere Effizienz. SIMD ermöglicht die parallele Verarbeitung von Daten sowie die Einsparung von arithmetischen Befehlen und verbessert dadurch die Leistung bei rechenintensiven Aufgaben um bis zu Faktor acht (Dilocker, 2023, 16:07), wie etwa bei der Berechnung von Abstandsmetriken in Vektorraum-Modellen, welche im folgenden Kapitel diskutiert wird. Ebenfalls große Leistungsverstärkung kann GPU-Support bringen. Johnson et al. (2017, S. 1)

[3]Link: https://github.com/qdrant/qdrant/blob/9c20e9e27960228019b4606f137ca82b42fc3e66/README.md#highlighted-features (Stand: 21. Oktober 2024).
[4]Link: https://github.com/milvus-io/web-content/blob/6dfdc91e1b244c6e3add04b12e06ba8008fccb4e/v2.4.x/site/en/reference/architecture/data_processing.md#index-building (Stand: 21. Oktober 2024).
[5]Link: https://github.com/weaviate/weaviate-io/blob/73861be9ea756c1df9fb94482f09d67795b73803/developers/weaviate/config-refs/distances.md#distance-implementations-and-optimizations (Stand: 21. Oktober 2024).

betonen, dass GPUs im Vergleich zu herkömmlichen CPUs für bestimmte Berechnungen eine 8,5-fache Beschleunigung ermöglichen.

2.3 Verfahren zur Berechnung der Ähnlichkeit

Die Anzahl der arithmetischen Operationen der Ähnlichkeitssuche ist durch die Ähnlichkeitsmaße geprägt. Sich damit zu beschäftigen, ermöglicht einen tieferen Einblick in die Funktionsweise von VDBMS. Für die Ähnlichkeitssuche gibt es verschiedene Rechnungen, die allesamt Abstandsmetriken im Vektorraum sind. Die VDBMS unterstützten jeweils eine Teilmenge dieser Abstandsmetriken. Die ausgewählte Metrik für den Abstand muss zum jeweiligen *Embedding-Modell* des LLM passen (Pan et al., 2024, S. 1596). Die drei typischen Maße sind der euklidische Abstand, das Skalarprodukt und der kosinusbasierte Abstand, beschrieben im Handbuch von *Milvus*.[6] Um den Aufwand der Umsetzung des Benchmarks abschätzen zu können, ist die Anzahl an arithmetischen Operationen relevant. Auch der verwendete Datensatz moduliert die Wahl des Ähnlichkeitsmaßes, z.B. bezogen auf die Normalisierung, wie am Ende des Kapitels beschrieben.

Euklidischer Abstand Seien $\mathbf{a} = \begin{bmatrix} a_0 & a_1 & \cdots & a_n \end{bmatrix}$ und $\mathbf{b} = \begin{bmatrix} b_0 & b_1 & \cdots & b_n \end{bmatrix}$ zwei Vektoren im n-dimensionalen euklidischen Raum, $s(a, b)$ die Ähnlichkeit und $d(a, b)$ der Abstand zwischen a und b, dann folgt die Rechnung des euklidischen Abstandes (2.2).

$$s_1(\mathbf{a}, \mathbf{b}) = d_1(\mathbf{a}, \mathbf{b}) = d_1(\mathbf{b}, \mathbf{a}) = \sqrt{\sum_{i=0}^{n-1} (b_i - a_i)^2} \tag{2.2}$$

Dabei gilt: Je kleiner s_1 ist, desto ähnlicher sind die beiden Vektoren.

Diese ist laut *Milvus* die am häufigsten verwendete Abstandsmetrik und sehr nützlich, wenn die Daten kontinuierlich sind.[6] *Milvus* berechnet den Wert ohne Anwendung der Quadratwurzel. Dies ist beim Vergleich von Entfernungen zweckmäßiger, da die letzte Quadratwurzel bei der Berechnung euklidischer Entfernungen obsolet ist, denn $d_1^2 > d_2^2 \iff d_1 > d_2$. Der Wert, der sich aus dieser Auslassung ergibt, ist das Quadrat des euklidischen Abstands (Spencer, 2014, S. 95).

Skalarprodukt Das Skalarprodukt (in der englischen Literatur auch *inner product* oder *dot product* genannt) ist ebenfalls ein Abstandmaß für zwei Vektoren und wie folgt definiert:

[6] Link: https://github.com/milvus-io/milvus-docs/blob/c918f06567b2b955360f08e273aeca52f9fb3bfb/site/en/reference/metric.md (Stand: 21. Oktober 2024).

$$s_2(\mathbf{a}, \mathbf{b}) = d_2(\mathbf{a}, \mathbf{b}) = \mathbf{a} \cdot \mathbf{b} = \sum_{i=0}^{n-1} a_i \cdot b_i \tag{2.3}$$

Dabei gilt: Je größer s_2 ist, desto ähnlicher sind die beiden Vektoren, sofern die Vektoren nicht null sind (Jurafsky & Martin, 2009, S. 114).

Insbesondere für den Vergleich nicht-normalisierter Daten oder bei hoher Relevanz von Längen- und Winkelähnlichkeit eignet sich das Skalarprodukt. Bei normalisierten Vektoren entspricht der Kosinus des Winkels (Kosinusähnlichkeit) zwischen zwei Vektoren ihrem Skalarprodukt (Singhal & Google, 2001, S. 36 f.). Daraus lässt sich schließen, dass bereits normalisierte Vektoren einfacher zu verarbeiten sind.

Kosinusähnlichkeit Die Kosinusähnlichkeit misst den Kosinus des Winkels zwischen zwei Vektoren.

$$s_3(\mathbf{a}, \mathbf{b}) = \cos(\theta) = \frac{s_2}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \frac{\mathbf{a} \cdot \mathbf{b}}{\sqrt{\mathbf{a}^2 \cdot \mathbf{b}^2}} = \frac{\sum_0^{n-1} (a_i \cdot b_i)}{\sqrt{\sum_0^{n-1} a_i^2} \cdot \sqrt{\sum_0^{n-1} b_i^2}} \tag{2.4}$$

$$d_3(\mathbf{a}, \mathbf{b}) = d_3(\mathbf{b}, \mathbf{a}) = 1 - s_3(\mathbf{a}, \mathbf{b}) \tag{2.5}$$

Die Kosinusähnlichkeit liegt immer im Intervall $[-1, 1]$. Zum Beispiel haben zwei genau gleichgerichtete Vektoren eine Kosinusähnlichkeit von 1, zwei orthogonale Vektoren eine Ähnlichkeit von 0 und zwei genau entgegengerichtete Vektoren eine Ähnlichkeit von -1.[6] Laut Jurafsky und Martin (2009, S. 114) ist dieses Ähnlichkeitsmaß das am häufigsten verwendete. So sind in der Literatur widersprüchliche Informationen zu finden.

Wenn die Elemente der Vektoren nicht negativ sind, dann gilt das eingeschränkte Interval $[0, 1]$. Hierbei gilt, je größer s_3 ist, desto ähnlicher sind die Vektoren.

Bei der Berechnung der Kosinusähnlichkeit ist das Skalarprodukt enthalten, sodass die Nutzung zwecks Effizienz nicht empfehlenswert ist. Bei nicht-normalisierten Vektoren oder unterschiedlichen Vektorlängen kann sich dennoch ein Anwendungsfall ergeben. Bei normalisierten Vektoren ist das Skalarprodukt oder der euklidische Abstand vorzuziehen. Letztendlich entscheidet jedoch das LLM in seiner Architektur über die richtige Wahl, so ist das zu nutzende Ähnlichkeitsmaß eine Eigenschaft der *Embeddings* bzw. des Datensatzes selbst.

Neben diesen drei Abstandsmetriken gibt es auch weitere, weniger übliche Abstandsmetriken wie den Hamming-Abstand oder die Manhattan-Distanz[7], welche hohe Performance, aber in den meisten Fällen schlechte Genauigkeit bieten.

[7] Link: https://github.com/spotify/annoy/blob/8a7e82cb537053926b0ac6ec132b9ccc875af40c/README.rst#summary-of-features (Stand: 21. Oktober 2024).

Für eine ausgewogene Darstellung von *Embeddings* wird hier kurz auf die Problematik der Modellabhängigkeit eingegangen:

Abhängigkeit zum Modell Abstandsmetriken setzen lediglich voraus, dass die Vektoren gleiche Dimensionen aufweisen (Sahoo et al., 2023, S. 3). Eine weitere wesentliche Einschränkung liegt darin, dass unterschiedliche neuronale Netze selten kompatible Merkmale erzeugen, selbst bei identischen Trainingsbedingungen. Mit neuen Trainingsdaten oder Modellarchitekturen wächst das Bestreben, das *Embedding-Modell* zu ersetzen, um die Leistung nachgelagerter Retrievalaufgaben zu verbessern (Y. Li et al., 2015, S. 2, 9). Wird ein neues Modell für die Berechnung der Anfrage genutzt, während alte Vektoren bestehen bleiben, kann dies zu falschen Ergebnissen führen. Um neue Modelle zu nutzen, müssen sämtliche *Embeddings* neu generiert werden, was als *Backfilling* bezeichnet wird (Shen et al., 2020, S. 2). Dieser rechenintensive Prozess erfolgt oft im Hintergrund: Während das alte Modell aktiv bleibt, werden neue Merkmale parallel berechnet (Jaeckle et al., 2023, S. 1 f.). Erst nach Abschluss des gesamten Aktualisierungsprozesses werden die neuen Merkmale eingesetzt. Dies stellt bei großen Systemen eine erhebliche Herausforderung dar und kann Monate dauern, was häufige Modellaktualisierungen erschwert.

2.4 Verfahren zur Berechnung des nächsten Nachbarn

Neben der Abstandsmetrik lassen viele VDBMS auch eine Auswahl des Vektorsuchverfahren zwecks Indexierung zu. In Datenbanken sind Indizes Datenstrukturen, die das Auffinden bestimmter Werte in einer Tabellenspalte beschleunigen. Um dies auf VDBMS übertragen, muss erst die per Index zu beschleunigende Aufgabe definiert werden:

Definition 2 Die Suche nach der Menge der Vektoren, bei der die Ähnlichkeit (siehe Abstandsmetrik) mit einem Abfragevektor am größten ist, wird **Suche nach dem nächsten Nachbarn** genannt (Jurafsky & Martin, 2009, S. 305).

In dieser Bachelorarbeit wird der Schwerpunkt auf die Untersuchung von In-Memory Indizes für Fließkommazahl-*Embeddings* gelegt. Die Arbeit konzentriert sich ausschließlich auf Algorithmen und Methoden, die für das Speichern und Verarbeiten von *Embeddings* direkt im Arbeitsspeicher optimiert sind. On-Disk-Algorithmen, wie sie etwa durch Systeme wie *Milvus* bereitgestellt werden, sind hingegen nicht Gegenstand der Betrachtung. Diese Entscheidung begründet sich in dem Bestreben, eine tiefgehende Analyse und Optimierung der In-Memory-Verfahren zu ermöglichen, die für Szenarien relevant sind, in denen niedrige Latenzzeiten und hohe Performance bei der Verarbeitung großer Datenmengen im Vordergrund stehen.

Algorithmen, die auf die Speicherung und das Abrufen von Daten auf Datenträgern ausgelegt sind, weichen in ihren Anforderungen und Implementierungen erheblich von den in dieser Arbeit betrachteten Ansätzen ab und werden daher bewusst ausgeklammert.

Grob lassen sich die Algorithmen in zwei Kategorien unterteilen:

Exakter k-Nächster Nachbar (kNN) Die Brute-Force-Methode besteht darin, jeden Anfragevektor mit allen Vektoren zu vergleichen, um das exakte Ergebnis zu erhalten. Dabei ist die zeitliche Komplexität $O(n \cdot m)$, wobei n die Dimension des Raums und m die Anzahl der Vektoren ist. Insbesondere bei Datensätzen mit Millionen oder Milliarden von Vektoren ist dieser Ansatz zu langsam.

Approximierter Nächster Nachbar (ANN) Um eine Beschleunigung zu erreichen, können ANN-Algorithmen eingesetzt werden. Im Folgenden werden zwei ANN-Algorithmen genauer betrachtet: der Inverted File (IVF)-Index und der Hierarchical Navigable Small World (HNSW)-Index. Der IVF-Index unterteilt die Vektoren zwecks Performancesteigerung in mehrere Listen (Y. Wang, 2022, S. 1). Dafür berechnet der Algorithmus eine Anzahl von Zentroiden (Mittelpunkt einer Menge von Vektoren; Parameter `lists`) und findet die Cluster (eingegrenzte Bereiche im Vektorraum), welche diesem Zentroiden am nächsten sind, siehe Abbildung 2.3a. Bei der Suche nach den k-nächsten Vektoren wird der Suchraum auf eine Teilmenge der Listen (Parameter `probes`) eingegrenzt, anstatt den Abstand zu allen Vektoren zu berechnen, wodurch die Anzahl der Berechnungen reduziert wird (Hoppe & Humm, 2020, S. 67). Die Effektivität des Indexes hängt von den zwei Parametern ab, welche in *pgvector* zu zwei Zeitpunkten gewählt werden.

Definition 3 Bei **Parametern von Indexierungsalgorithmen** wird zwischen **statischen** und **dynamischen** unterschieden. Statische Parameter sind zum Zeitpunkt der Indexerstellung zu wählen und lassen sich anschließend nicht anpassen. Dynamische Parameter sind jederzeit einstell- und anpassbar, beispielsweise vor einer neuen Suchanfrage.

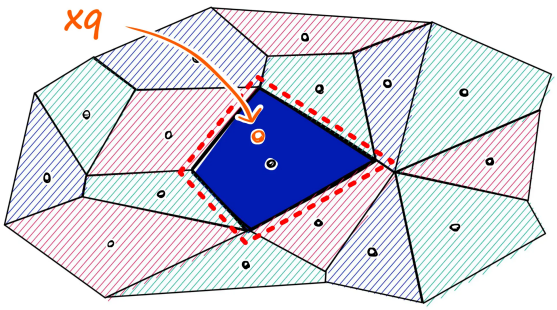
Code-Beispiel 2.1: Erstellung von Index in *pgvector* unter Nutzung von IVF.

```
CREATE INDEX ON items USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 2000); # "lists" ist ein statischer Parameter
SET ivfflat.probes = 100; # und "probes" ist dynamisch.
```

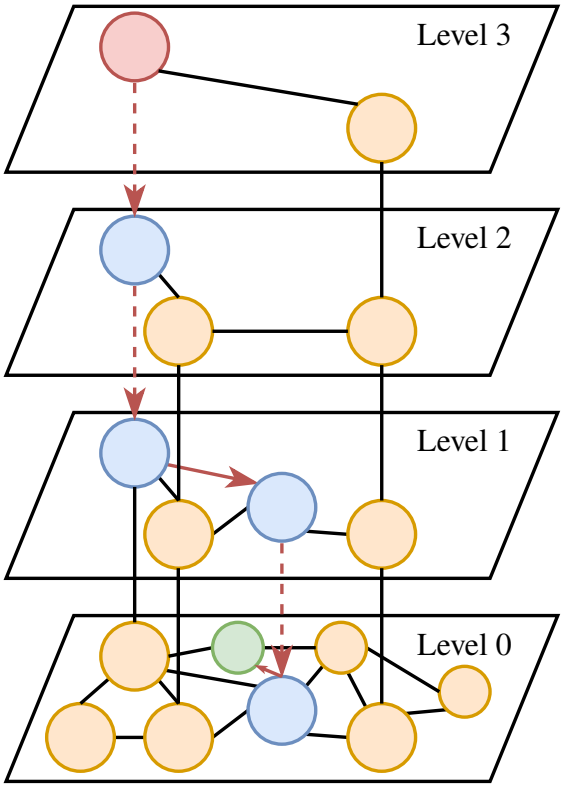
Mit den in Code-Beispiel 2.1 genannten Parametern würde *pgvector* bei einem Datensatz von sechs Millionen Vektoren 2.000 Cluster mit jeweils etwa 3.000 Vektoren erzeugen. Bei einer Abfrage durchsucht *pgvector* nur 100 dieser Cluster und führt so 300.000 Vergleiche

durch, um die nächsten Nachbarn des Suchvektors zu finden. Das entspricht nur 5 % des Rechenaufwands im Vergleich zur vollständigen Durchsuchung aller Vektoren (Brute-Force-Methode).

Der Navigable Small World (NSW)-Index ist eine Technik zur Durchführung von ANN, bei der eine Menge von Datenpunkten in einen Graphen transformiert wird. In diesem Graph zeigen Knoten die Datenpunkte an, die nur unter Erfüllung bestimmter geometrischer Abstandsanforderungen miteinander verbunden sind. Der zugrunde liegende Greedy-Algorithmus startet die Suche mit einem zufällig gewählten Startpunkt (Windsor & Choi, 2023, S. 2 f.). Er berechnet dann den Abstand zwischen dem Abfragepunkt der Suchanfrage und den Nachbarn des Startpunktes, wählt den nächstgelegenen Nachbarn aus und aktualisiert den Startpunkt auf diesen Nachbarn (Sprung). Dieser Vorgang wird fortgesetzt, bis eine festgelegte Abbruchbedingung eintritt, wie z. B. eine maximale Anzahl von Iterationen oder ein Sprung ohne Verbesserung. In der Praxis wird dieser Prozess mit verschiedenen zufälligen Startpunkten wiederholt (Windsor & Choi, 2023, S. 2 f.).



(a) IVF: Jeder Vektor ist in einer Voronoi-Zelle (Abb: Polygon) enthalten. Bei einer Abfrage wird zunächst der Abstand zwischen dem Abfragevektor (Abb: xq) und den geometrischen Schwerpunkten (Abb: kleine kreisförmige Punkte) berechnet und dann der Suchbereich auf die Zelle des nächsten Zentroids beschränkt. Quelle: https://www.pinecone.io/learn/series/faiss/faiss-tutorial/#Partitioning-The-Index, Stand: 21. Oktober 2024.



(b) Die Suche nach NN-Knoten (grün gezeigt) im HNSW-Index startet bei einem Element im obersten Layer (rot gezeigt) bei Laufzeit $O(\log n)$. Abbildung nach Malkov und Yashunin (2020, S. 826).

Abbildung 2.3: Abbildung von IVF- (links) und HNSW-Indexierungsalgorithmus (rechts).

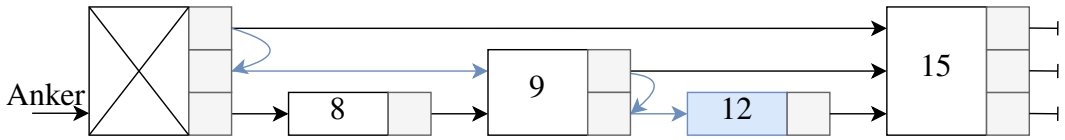


Abbildung 2.4: Suche nach Wert 12 in Skip-Liste mit Blindelement am Anker bei Laufzeit $O(\log n)$.

Code-Beispiel 2.2: Erstellung von Index in *pgvector* unter Nutzung von HNSW.

```
CREATE INDEX ON items USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64); # "m" und "ef" sind statische
    Parameter
SET hnsw.ef_search = 100; # und "ef_search" ist dynamisch.
```

Eine Erweiterung dieser Methode stellt der HNSW dar, der eine mehrschichtige Struktur einführt. Bei N Schichten gilt: $S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots S_N$, wobei S_0 den NSW-Graphen darstellt und alle anderen $S_{>0}$ zu ihm in verschachtelter Teilmengenrelation stehen. N wird als Hyperparameter festgelegt (Windsor & Choi, 2023, S. 3). Die zentrale Idee dieser Struktur besteht darin, dass der Graph von der höchsten Schicht aus durchquert wird, wodurch bei einer Abfrage nach unten navigiert werden kann, siehe Abbildung 2.3b. Dies ermöglicht es, potenziell unnötige Zwischeniterationen des gierigen Prozesses zu überspringen, was die Effizienz der Suche in großen, hochdimensionalen Datenräumen erheblich steigert (Windsor & Choi, 2023, S. 3). So erscheint HNSW als Summe aus NSW und einer Skip-Liste, siehe Abbildung 2.4. Sichtbar ist dies auch in der zeitlichen Komplexität. Hier erreicht HNSW analog zur Skip-Liste eine logarithmische Skalierung der Komplexität (Malkov & Yashunin, 2020, S. 826) und ist damit einer der bisher effizientesten ANN-Algorithmen (W. Li et al., 2020, S. 1478).

HNSW wurde als Python-Bibliothek `hnswlib` implementiert und in mehreren peer-reviewten Studien als bewährtes Werkzeug für das kNN-Problem eingesetzt (vgl. Pan et al., 2024, S. 1593; und W. Li et al., 2020, S. 1478). Für die Erstellung des HNSW-Indexes, siehe Code-Beispiel 2.2, müssen drei Parameter festgelegt werden: 1. Die Verbindungsanzahl pro Ebene (`m`) und 2. die Größe der dynamischen Kandidatenliste (`ef_construction`). Ein größerer Wert für `m` erhöht die Genauigkeit, allerdings bei schlechterer Performance, da mehr Verbindungen überprüft werden müssen.[8] `ef_search` steuert die Größe der dynamischen Liste während der eigentlichen Abfrage.

Die Komprimierungsmethoden Product Quantization (PQ) und Scalar Quantization (SQ) lassen sich auf IVF sowie HNSW anwenden. PQ unterteilt und quantisiert einen Vektor in kleinere Subvektoren, SQ hingegen bildet jedes Skalar eines Vektors einzeln auf einen

[8] Link: https://github.com/nmslib/hnswlib/blob/c1b9b79af3d10c6ee7b5d0afa1ce851ae975254c/ALGO_PARAMS.md#search-parameters (Stand: 21. Oktober 2024).

diskreten Wert ab (J. Wang et al., 2021, S. 2617). Beide erhöhen die Performance auf Kosten der Präzision.

Die Genauigkeit gibt den Anteil der korrekt als Treffer klassifizierten Vektoren (*true positive*) an der Gesamtheit der zurückgegebenen Vektoren (positiv klassifiziert) an (W. Li et al., 2020, S. 1480). Zum Beispiel bedeutet eine Präzision von 1, dass die k berechneten Nachbarn tatsächlich die kNN sind. Präzision ist ein wichtiger Faktor, um Approximationsfehler zu messen und die Indexparameter zu optimieren.

Neben den bereits erwähnten Algorithmen zur Vektorsuche existieren noch weitere Methoden, die jedoch kaum von VDBMS unterstützt und in der wissenschaftlichen Literatur seltener verwendet werden. Im Rahmen des vorliegenden Benchmarks werden diese nicht berücksichtigt. Zu diesen Ansätzen zählen unter anderem Scalable Nearest Neighbors (ScaNN), entwickelt 2022 von Google (Kombinierte Optimierungstechniken), Approximate Nearest Neighbors Oh Yeah (ANNOY), entwickelt 2013 von Spotify (Bäume & Voronoi-Partitionen & MMAP-OnDisk) und Locality-Sensitive Hashing (LSH). Zusammenfassend kann festgestellt werden, dass diese Algorithmen, obwohl sie spezifische Vorzüge bieten, meistens nicht in VDBMS implementiert werden und aus diesem Grund nicht in der vorliegenden Arbeit mit einbezogen werden.

Daher werden IVF-SQ und HNSW als Repräsentanten für geeignete Suchindizes ausgewählt, da sie nach Tabelle 2.3 am vielversprechendsten sind. Zudem wird die Präzision als Vergleichsmetrik aufgenommen.

Struktur	**Index**[a]	**Partitionierung**	**Komplexität**[b]			**Update**[c]
			Konstruktion	Speicher	Abfrage	
Tabelle	LSH	Raum	Mittel	Hoch	Mittel	Ja
	IVF-SQ	Clustering	Mittel	Niedrig	Mittel	R
Baum	ANNOY	Raum	Niedrig	Hoch	Niedrig	R
Graph	HNSW	Annäherung	Niedrig	Hoch	Niedrig	Ja

[a] LSH: Mit großem Abstand langsamer ist als andere Wettbewerber, da es die Datenverteilung nicht ausnutzt; IVF-SQ: Geringere Genauigkeit aufgrund der Kompression; ANNOY: niedrigerer Speicherverbrauch als HNSW, moderate Genauigkeit; HNSW: hohe Genauigkeit

[b] Konstruktion. Mittel. = $O(DN^2)$, Niedrig = $O(D \log N)$; Speicher. Hoch >= $O(DN)$, Niedrig = $O(N \log D)$; Abfrage. Mittel. = $O(DN)$, Niedrig = $O(D \log N)$.

[c] Ja: datenunabhängige Aktualisierungen; R: Aktualisierungen mit Neuausbalancierung.

Tabelle 2.3: Repräsentative Suchindizes, nach Pan et al. (2024, S. 1604) und W. Li et al. (2020, S. 1480 f.). Abwägung zwischen Geschwindigkeit, Genauigkeit und Speicherverbrauch: Auswahl des richtigen Index für einen Anwendungsfall.

3 Anforderungsanalyse

Die Anforderungsanalyse bildet die Grundlage für Vergleich und Bewertung von VDBMS im Kontext von KI. In den vorangegangenen Kapiteln wurden bereits verschiedene VDBMS aufgelistet. In diesem Kapitel wird im Detail erläutert, welche spezifischen VDBMS für die Untersuchung ausgewählt und miteinander verglichen werden. Ein zentrales Ziel ist es, relevante Vergleichsmerkmale zu identifizieren und zu evaluieren, inwieweit die etablierten Metriken traditioneller DBMS auf VDBMS übertragbar sind. Diese Metriken umfassen verhältnisskalierte Kriterien wie Latenz, Durchsatz und Ressourcennutzung. Darüber hinaus existieren auch nominalskalierte Merkmale, etwa die Unterstützung von Indexierungsalgorithmen.

Definition 4 Ein **Benchmarking-System** (hier auch kurz: Benchmark) ist ein methodisches Verfahren, bei dem die Leistungsdaten eines Produktes mit Mitbewerbern verglichen werden, um Verbesserungsmöglichkeiten zu identifizieren und Optimierungen umzusetzen.

Zudem werden spezifische Anforderungen an den Benchmark definiert, die Aspekte wie den verwendeten Datensatz betreffen. Dies ermöglicht letztlich fundierte Empfehlungen für exemplarische Anwendungsszenarien.

3.1 Schlüsselszenarien zum Einsatz von VDBMS

Besonderes Augenmerk bei den Schlüsselszenarien wird auf die Auswahl gelegt, um realistische Anwendungsfälle zu repräsentieren. Dies dient im Kapitel 5.5 dazu, beispielhafte Empfehlungen für den Einsatz von VDBMS zu geben. Diese Szenarien decken unterschiedliche Anforderungen an Latenz, Konsistenz, Replikation und Durchsatz ab, wie sie in praxisrelevanten Anwendungsbereichen von VDBMS, etwa im Bereich der Such- und Empfehlungssysteme sowie der Datenmoderation, eine Rolle spielen.

Szenario 1 Ein Automobilunternehmen möchte seine Confluence-Wissensdomäne im Intranet per Chatbot durchsuchbar machen. Dazu werden alle Artikel in Textchunks zerteilt, per *Embedding-Modell* vektorisiert und in eine Datenbank eingelagert. Da die Artikel bearbeitbar sind, muss die Datenbank neben `Insert` auch `Update` unterstützen. Die **Latenz der**

Suchanfrage hat eine obere Grenze von 100 ms, damit Nutzer:innen nach spätestens einer Sekunde eine Antwort vom Chatbot erhalten.

Szenario 2 Im Hochfrequenzhandel (HFT) handeln Algorithmen automatisch Wertpapiere auf der Grundlage minimaler Preisschwankungen in extrem kurzen Zeitspannen. Der Erfolg des Systems hängt von der Geschwindigkeit ab, mit der Entscheidungen getroffen und Aufträge an die Börse übermittelt werden. Die Algorithmen im HFT verwenden historische Daten, um schnelle Mustererkennung durchzuführen und Kauf- oder Verkaufsentscheidungen zu treffen. Um Ähnlichkeiten in den Preisbewegungen oder Marktparametern zu erkennen und daraus ihre Entscheidungen abzuleiten, muss ein VDBMS eingesetzt werden. Der Anspruch an eine geringe **Latenz bei einer Suchanfrage** ist besonders hoch und liegt bei 3 ms.

Szenario 3 Ein Unternehmen ist im Bereich der medizinischen Bildverarbeitung tätig und entwickelt ein KI-gestütztes System zur präzisen Diagnose von Krankheiten anhand von bildgebenden Verfahren wie MRT- oder CT-Scans. Das System basiert auf Deep-Learning-Algorithmen, die durch die Analyse hochdimensionaler Vektoren aus den Bilddaten trainiert werden. Hierbei spielt die Speicherung und Verwaltung von Vektordaten eine zentrale Rolle, da die kontinuierliche Weiterentwicklung des Systems auf immer umfangreicheren Datensätzen basiert. Ein zentrales Anliegen ist die Sicherstellung einer **hohen Konsistenz** der gespeicherten Daten, da Abweichungen in den Vektorrepräsentationen starke Auswirkungen auf die diagnostischen Ergebnisse haben könnten.

Szenario 4 Ein Videospielentwicklungsunternehmen eines Spiels mit **über 20 Millionen** monatlich aktiven Spieler:innen möchte unstrukturierte Daten moderieren. Die Inhalte wie Mods und individuell eingestellte Level, die von der Spieler:innengemeinschaft erstellt werden, sollen auf schädliche Inhalte überprüft werden. Dazu werden alle Inhalte vektorisiert, in eine Datenbank gespeichert und mit bereits bekannten Schadinhalten verglichen.

3.2 Wahl und Eigenschaften der VDBMS

In diesem Kapitel wird die Menge der zu untersuchenden VDBMS definiert und begründet. Die Analyse umfasst acht auf dem Markt verfügbare VDBMS, die eine signifikante Akzeptanz und Beliebtheit aufweisen, gemessen an der Anzahl der *Stars* auf GitHub (>= 9000 Stars, Ausnahme *Typesense*[9]). Es handelt sich hierbei um *Chroma*, *Milvus*, *OpenSearch*, *pgvector*,

[9] Erst nach Abschluss der Analysen wurde das VDBMS *Typesense* identifiziert. Obwohl *Typesense* nicht in die Vergleichsstudie einbezogen werden konnte, ist es wichtig, die Relevanz und das Potenzial dieses Systems zu betonen. Die Merkmale von *Typesense*, wie die hohe Leistung bei der Volltextsuche, könnten eine Alternative zu *OpenSearch* und *Elasticsearch* darstellen.

Qdrant, *Redis*, *Elasticsearch* und *Weaviate*. Zusätzlich zu den Open-Source-Lösungen wird *Pinecone* als proprietäre Software in die Untersuchung einbezogen. Die Entscheidung, *Pinecone* in die Analyse aufzunehmen, basiert auf dessen fortschrittlichen Funktionen, einem sehr guten Handbuch und der zunehmenden Bedeutung, die proprietäre Lösungen im Bereich der Vektordatenbanken einnehmen. Auf andere proprietäre, Managed-VDBMS wie die von Microsoft oder Google wird mangels öffentlich verfügbarer technischer Informationen nicht weiter eingegangen.

Für den Prototyp, dessen Kern ein algorithmusbasiertes Empfehlungssystem für VDBMS darstellt und welcher in Kapitel 5.4 eingeführt wird, ist die Betrachtung weiterer (nominaler) Eigenschaften notwendig, welche nicht beim Benchmark gemessen werden können. Die VDBMS lassen sich darauf basierend charakterisieren:

Erscheinungsdatum *Chroma* wurde 2022 entwickelt und gehört zu den neuesten VDBMS auf dem Markt. *Pinecone* wurde 2019 veröffentlicht, ebenso wie *Milvus* und *Weaviate*. *Redis* hat seinen Ursprung bereits im Jahr 2009, während *Elasticsearch* erstmals 2010 veröffentlicht wurde. *Qdrant* und *pgvector* stammen beide aus dem Jahr 2021. *OpenSearch* wurde ebenfalls 2021 als Fork von *Elasticsearch* eingeführt.

Besonderheiten *Chroma* zeichnet sich durch sein einfaches Application Programming Interface (API) und die nahtlose Integration mit KI-Modellen aus, während *Milvus* speziell für maschinelles Lernen und hochdimensionale Vektoren konzipiert wurde. Außerdem ist festzustellen, dass *Milvus* als einziges der genannten acht VDBMS On-Disk-Algorithmen oder Algorithmen wie ScaNN und ANNOY unterstützt. *OpenSearch* und *Elasticsearch* sind in erster Linie Such- und Analysewerkzeuge, die durch Erweiterungen Vektordaten unterstützen. *Pgvector* ermöglicht die Speicherung von Vektordaten in einer relationalen Umgebung und integriert diese nahtlos in PostgreSQL. *Qdrant* legt besonderen Wert auf Performance, maschinelles Lernen und KI-Anwendungen. *Redis*, eine In-Memory-Datenbank für Schlüssel-Wert-Paare, bietet durch Module Unterstützung für Vektordaten. *Weaviate* hebt sich durch seine semantische Suche und der Fokussierung auf Ähnlichkeitssuchen ab. *Pinecone* ist ein cloud-natives System, das sich durch extrem schnelle Vektor-Abfragen auszeichnet.

Qdrant bietet verschiedene Möglichkeiten zur Optimierung, um den spezifischen Anforderungen verschiedener Anwendungsfälle gerecht zu werden. So kann je nach Anwendungsfall das richtige Gleichgewicht zwischen Speichernutzung, Suchgeschwindigkeit und Präzision gefunden werden.[10] Aus diesem Grund wird für die entsprechenden Lastprofile eine latenzorientierte Konfiguration namens `qdrant-m-*-ef-*` und eine durchsatzorientierte Konfiguration namens `qdrant-rps-m-*-ef-*` beim Benchmark untersucht. Dies zeigt auf, dass

[10] Link: https://github.com/qdrant/landing_page/blob/75aebaa12583fef00f46ee1fdabe5b26979eb346/qdrant-landing/content/documentation/guides/optimize.md#maximizing-throughput (Stand: 21. Oktober 2024).

eine entsprechende Anforderungsanalyse bei den Anwendungsszenarien für eine passende Empfehlung nötig ist.

Entwicklungsort *Chroma*, *Redis*, *Elasticsearch*, *OpenSearch*, *pgvector* und *Pinecone* wurden in den USA entwickelt, während *Milvus* von der chinesischen Firma Zilliz stammt. *Weaviate* wurde in den Niederlanden entwickelt, während *Qdrant* aus Deutschland stammt.

Open-Source und Self-Host Die meisten der genannten VDBMS sind Open-Source und bieten Self-Host an. Dazu gehören *Chroma*, *Milvus*, *OpenSearch*, *pgvector*, *Qdrant*, *Redis*, *Elasticsearch* und *Weaviate*. *Pinecone* hingegen ist proprietär und bietet keine Self-Hosted-Version an, da es nur als cloud-native Lösung verfügbar ist.

Skalierbarkeit Alle genannten Systeme außer *Chroma* sind für Skalierbarkeit ausgelegt.

Transaktionen Nur einige der Systeme unterstützen Transaktionen im klassischen Sinne. *Pgvector*, das auf der RDBMS PostgreSQL basiert, erbt die vollständige Unterstützung für Transaktionen mit allen Eigenschaften von Atomicity Consistency Isolation Durability (ACID). *Redis* unterstützt ACID-Transaktionen in begrenztem Umfang. *OpenSearch* kann Transaktionen teilweise durch die Integration mit anderen Datenbanken unterstützen. Die anderen Systeme unterstützen keine traditionellen Transaktionen, da ihr Hauptfokus auf schnellen Ähnlichkeitssuchen und der Verarbeitung von Vektordaten liegt.

Pre-Filtering Alle genannten Systeme unterstützten Pre-Filtering.

3.3 Vergleichsmetriken für Datenbanken

Um für den Leistungsvergleich von VDBMS evidenzbasierte Metriken zu entwickeln, ist zunächst eine kurze Untersuchung der Metriken herkömmlicher Datenbanken nötig. Die übertragbaren Metriken werden beschrieben und für den Vergleich der VDBMS genutzt.

Im Buch von Schmidt und Brodie (1983) werden die Charakteristika der Leistung von relationalen DBMS vorgestellt (nach Kalay, 1985, S. 286). Diese Kriterien wurden von der Relational Task Group (RTG) entwickelt, die 1979 vom American National Standards Institute (ANSI) beauftragt wurde. Genannt werden unter anderem die Unterstützung von Operatoren der relationalen Algebra (Union, Join, Intersection), die Unterstützung einer *Query Language*, Operationen, um die Datenbank zu ändern, eine Möglichkeit Constraints zu definieren sowie Sicherheit via Überwachung und Kontrolle (S. 13 f.). Außerdem wird auf die Konfiguration

von Indizes und Transaktionen eingegangen (S. 14, 33). Dies sind nominalskalierte Merkmale, die in Kapitel 3.4 betrachtet werden.

Zudem sind die typischen Leistungsmetriken nach Taipalus (2024a, S. 3 f.) folgende:
Die **Latenz** misst, wie schnell ein System eine Abfrage oder Anfrage verarbeitet und antwortet. Der **Durchsatz** gibt an, wie viele Transaktionen pro Sekunde ein DBMS innerhalb eines bestimmten Zeitraums verarbeiten kann. Die **Ressourcennutzung**-Metrik bewertet, wie effizient ein DBMS Systemressourcen verwendet.

Um die Anforderungen der Metriken der VDBMS-Messung zu erfüllen und die individuelle VDBMS-Wahl zu unterstützen, ist es nötig, die beschriebenen Metriken, welche sich in traditionellen Datenbanksystemen bereits bewährt haben, auf typische, spezifische Lastprofile von VDBMS zu überführen. Für die Lastprofile Echtzeit (A), massivparallele Anfragen (B) und Aktualisierungshäufung (C) werden folgende Metriken angewendet, wie sie auch bei Aumüller et al. (2018, S. 10) und bei Qdrant Solutions GmbH (2022) im Benchmark eingesetzt werden:

A) Such-Latenz ist die Zeit, die eine Ähnlichkeitssuchanfrage benötigt, um vollständig bearbeitet zu werden. Dies umfasst sowohl die Wartezeit als auch die tatsächliche Bearbeitungszeit der Anfrage. Ein schnelles Reagieren auf einzelne Anfragen ist typisch für Anwendungen wie Echtzeitsysteme, bei denen eine verzögerungsfreie Reaktion erforderlich ist. Im Benchmark wird dieser Aspekt durch die Simulation einzelner Anfragen über einen einzelnen Prozess gemessen, um eine detaillierte Analyse der Reaktionszeiten unter minimaler Last zu ermöglichen.

B) Durchsatz zeigt die Anzahl der Anfragen an, die die Datenbank pro Sekunde ($\mathrm{r\,s^{-1}}$) verarbeitet. Eine erhöhte Anzahl an Anfragen pro Sekunde wird oft auf Kosten der verlängerten Bearbeitungszeit einzelner Anfragen (d.h. höherer Latenz) realisiert. Dies ist ein typisches Szenario bei Anwendungen, in denen mehrere Benutzer:innen gleichzeitig Suchanfragen stellen. Um dieses Szenario zu simulieren, werden die Client-Anfragen parallel über mehrere Threads an dem Server übermittelt. Diese Methodik ermöglicht eine realistische Bewertung der Systemleistung unter hoher Last.

C) Upload-Latenz ist die Zeit, die für das Hochladen und die Indexierung von Vektoren benötigt wird. Ein schnelles Einlagern von Daten ist wichtig für Anwendungen, die regelmäßig mit neuen Daten oder Aktualisierungen umgehen müssen. Um dieses Szenario zu simulieren, speichert der Client einen Vektor-Datensatz in das System, wobei die benötigte Zeit für den gesamten Vorgang gemessen wird. Dies ermöglicht die Bewertung der Effizienz des Systems

beim Hinzufügen von Daten, was ein wesentlicher Bestandteil der Arbeitslast von VDBMS ist.

Zeitgleich zum Benchmark werden Ressourcenmetriken wie CPU-, Speicher-, Disk- und Netzwerkauslastung aufgezeichnet. Durch die steigende Anzahl paralleler Anfragen werden die Ressourcen ausgelastet und es treten höhere Latenzen auf.

3.4 Weitere nominale Vergleichseigenschaften

In diesem Abschnitt werden zusätzliche nominale Vergleichseigenschaften vorgestellt, die für die fundierte Evaluierung der untersuchten Systeme von Bedeutung sind. Diese Eigenschaften tragen dazu bei, die Vergleichbarkeit der Ergebnisse zu gewährleisten und präzise Rückschlüsse auf die Effizienz der Algorithmen und Hardwarekonfigurationen zu ermöglichen. Da die Anzahl der evaluierten Datensätze, Algorithmen und Hardwarekonfigurationen proportional zur Ausführungszeit des Benchmarks und zur Komplexität der Ergebnisse ist, ist eine Beschränkung auf eine sinnvolle Auswahl ratsam.

Datensatz Bei der Auswahl der Datensätze ist es entscheidend, solche zu wählen, die den Vektordaten der Szenarien entsprechen. Ebenso wichtig ist die Vergleichbarkeit der Ergebnisse mit bereits veröffentlichen Benchmarks, um diese als Referenzsystem zu nutzen. Zu den bekannten Datensätzen zählen unter anderem `glove-100-angular`, `glove-25-angular`, `nytimes-256-angular`, `fashion-mnist-784-euclidean` und `gist-960-euclidean`, welche detailliert auf der Plattform sowie als Überblick im Paper von Aumüller et al. (2018, S. 10) für den Vergleich von ANN-Algorithmen verfügbar sind.[11] Auf der Benchmark-Webseite von Qdrant Solutions GmbH (2022) werden vier Datensätze verwendet: `dbpedia-openai-1M-1536-angular`, `deep-image-96-angular`, `gist-960-euclidean` und `glove-100-angular`.[12] Es existieren weitere Quellen für Datensätze, die hier jedoch nicht näher betrachtet werden.[13]

Da `deep-image-96-angular` und `gist-960-euclidean` Bild-*Embeddings* darstellen, jedoch zwei der vier Szenarien mit Textdaten arbeiten, sind diese Datensätze für die vorliegende Untersuchung ungeeignet. In der verbleibenden Schnittmenge beider Webseiten bleibt somit nur der Datensatz `glove-100-angular` relevant. Dieser Datensatz basiert auf dem *Wort-Embedding-Modell* GloVe aus Kapitel 2.1 und umfasst 1.183.514 Vektoren, von denen 10.000 als Testdaten mit bekannten 100 nächsten Nachbarn gespeichert sind. Die Vektoren haben

[11] Link: https://github.com/erikbern/ann-benchmarks/blob/92394451a46938d0acf664e12751d9d5f6dbe524/README.md#data-sets (Stand: 21. Oktober 2024).
[12] Link: https://qdrant.tech/benchmarks/#single-node-benchmarks (Stand: 21. Oktober 2024).
[13] Link: https://github.com/qdrant/ann-filtering-benchmark-datasets/blob/f66691ee8efe511a2afd8a2b4bfd3bf9e48e54cb/README.md#data (Stand: 21. Oktober 2024).

eine Dimension von 100 und die Abstände zwischen den Vektoren werden mithilfe der Kosinusähnlichkeit (Kapitel 2.3) berechnet. Eine Vektordimension von 100 ist zwar für typische Vektoren der vier Szenarien etwas gering, dennoch bietet dieser Datensatz eine solide Grundlage für erste Vergleiche.

Indexierungsalgorithmus Die Wahl des Indexierungsalgorithmus, IVF-SQ oder HNSW, ist entscheidend für die Effizienz von VDBMS. Für diese Arbeit wird der HNSW-Indexierungsalgorithmus eingesetzt. Dieser Algorithmus wurde ausgewählt, da er sich als ein de-facto Standard in der Literatur etabliert hat und in allen VDBMS implementiert ist. Wie im Kapitel 2.4 beschrieben, erreicht er $O(\log n)$ und ist auch bei steigenden Datenmengen sehr effizient.

Systeme, die sich mit der Speicherung und Abfrage von Vektordaten beschäftigen, wie beispielsweise *Qdrant*, setzen auf HNSW, um performante, annähernd-korrekte Suchen in hochdimensionalen Räumen zu ermöglichen. Die Entscheidung für HNSW basiert auch auf seiner breiten Implementierung und der Nutzung in verschiedenen Benchmarks (Qdrant Solutions GmbH, 2022; Aumüller et al., 2018, S. 9). Dies zeigt, dass HNSW in realen Anwendungsfällen gut erprobt und in wissenschaftlicher Literatur eingesetzt ist, wodurch die Wahl dieses Algorithmus fundiert und technisch sinnvoll ist.

Hardware Hinsichtlich der Hardwarekonfiguration wird sich an der Arbeit von Qdrant Solutions GmbH (2022) orientiert. Diese Entscheidung wurde getroffen, um vergleichbare Leistungswerte zu erzielen und die Ergebnisse der Arbeit mit bestehenden Benchmarkdaten in Beziehung setzen zu können. Das Client-Setup umfasst daher 8 virtuelle zentrale Verarbeitungseinheiten (vCPUs) und 16 GB RAM, während der Server mit 8 vCPUs und 32 GB RAM ausgestattet ist. Diese Hardwarekonfigurationen basieren auf Cloud-Instanzen im Azure-Cloud-Umfeld, `Standard D8ls v5` für den Client und `Standard D8s v3` für den Server. Diese Auswahl stellt sicher, dass die Performance der Experimente realitätsnah ist und gleichzeitig genug Ressourcen zur Verfügung stehen, um auch größere Datenmengen und komplexe Suchen effizient verarbeiten zu können.

3.5 Ergebnisse der Anforderungsanalyse

Die Anforderungsanalyse hat relevante Szenarien und Vergleichsmetriken für VDBMS im Kontext von KI ergeben. Dabei wurden spezifische Anforderungen an die Leistung, Skalierbarkeit, Konsistenz und Anpassbarkeit der Systeme identifiziert. Aus den gewonnenen Erkenntnissen lassen sich wichtige Schlüsse für das Design eines VDBMS-Benchmarks ableiten. Die zentralen Anforderungen und deren Implikationen für das Benchmarkdesign sowie

potenzielle Synergien lassen sich folgendermaßen zusammenfassen:
Die identifizierten Schlüsselszenarien verdeutlichen, dass VDBMS in unterschiedlichen Umgebungen und für verschiedene Zwecke eingesetzt werden, wie z. B. hochperformante Suchanfragen (geringe Latenz) oder skalierbare Systeme mit Replikation. Die Wahl des HNSW-Algorithmus als Benchmark-Algorithmus reflektiert den aktuellen Stand der Technik, zudem wird er in nahezu allen modernen VDBMS unterstützt. Systeme wie *Qdrant* und *Pinecone*, die eine flexible HNSW-Konfiguration für latenz- oder durchsatzorientierte Workloads bieten, ermöglichen es, das System an die jeweilige Priorität anzupassen. Das Benchmarksystem muss die Möglichkeit bieten, Parameter des Algorithmus anzupassen, um die Leistungsfähigkeit für Lastprofile zu optimieren.

Die Anforderungsanalyse zeigt, dass klassische Metriken wie Latenz, Durchsatz und Ressourcennutzung auch für VDBMS von zentraler Bedeutung sind. Die Anforderungen decken sich mit den spezifischen Einsatzszenarien: niedrige Suchlatenz für Echtzeitanwendungen (Szenario 1 und 2), hoher Durchsatz bei massenhaften, parallelen Anfragen (Szenario 4) und niedrige Uploadzeit für dynamische Datenverarbeitung. Das Benchmarksystem muss in der Lage sein, diese präzise zu messen. Es sollte verschiedene Ressourcenmetriken wie CPU-, Speicher- und Netzwerkauslastung integrieren, um die Effizienz und Skalierbarkeit des Systems zu bewerten. Nicht-funktionale Anforderungen wie Verfügbarkeit, Konsistenz und Transaktionssicherheit (Szenario 3) werden je nach Datenbank unterschiedlich stark erfüllt, wobei einige Systeme wie *pgvector* auf ACID-Konformität setzen, während andere auf Vektorverarbeitung spezialisiert sind. Für das in Kapitel 5.4 zu besprechende Empfehlungssystem müssen auch nominale Eigenschaften bei der Wahl der passenden VDBMS berücksichtigt werden.

Ein zentrales Ziel eines Benchmarksystems ist es, reproduzierbare und vergleichbare Ergebnisse zu liefern. Der standardisierte Testdatensatz `glove-100-angular` wird gewählt, um die Vergleichbarkeit mit bestehenden Benchmarks zu gewährleisten. Die vielen arithmetischen Operationen sowie der hohe Speicherbedarf der Indexierung zeigen auf, dass die Hardwarekonfiguration (CPU und RAM) wesentliche Parameter bei der Bewertung der Leistung eines VDBMS darstellt. Um realitätsnahe Ergebnisse zu erzielen, muss das Benchmarksystem auf gängige Hardwareanforderungen wie Cloud-Umgebungen abgestimmt sein. Da sich die Anforderungen und Technologien im Bereich der Vektordatenbanken schnell entwickeln, sollte das Benchmarksystem modular aufgebaut sein, um leicht erweitert werden zu können. Neue VDBMS, Algorithmen, Metriken oder Datensätze sollten ohne großen Aufwand hinzugefügt werden können. Insgesamt zeigt die Anforderungsanalyse, dass die spezifischen Anforderungen von VDBMS je nach Anwendung variieren. Flexibilität in der Konfiguration und die richtige Wahl von Indexierungsalgorithmen sind wesentliche Kriterien für das Design von VDBMS-Benchmarks.

4 Architektur des Testsystems

Bei der Auswahl der besten Technologie zur Durchführung von Benchmarks stehen viele Programmiersprachen wie Go, Java oder Rust zur Verfügung, jedoch sprechen zwei Gründe für eine Entscheidung zugunsten von Python (Qdrant Solutions GmbH, 2022): Erstens wird Python häufig zur Generierung von *Embeddings* verwendet und ist eng mit verschiedenen auf Python basierenden Machine-Learning-Frameworks verknüpft. Zweitens gehören Python-Clients, gemessen an der Popularität auf GitHub, zu den am häufigsten genutzten Clients für viele VDBMS.

Herkömmliche DBMS-Leistungsbenchmarks bestehen in der Regel aus einer Beispieldatenbank und einer Arbeitslast, welche eine Nutzung der Datenbank simuliert. Die Benchmarks messen in der Regel die Effizienz von Abfragen unter Berücksichtigung von Faktoren wie z. B. Gleichzeitigkeit (Taipalus, 2024a, S. 5). Taipalus stellte fest, dass frühere Studien zum Leistungsvergleich von DBMS unerwartet weit generalisierbar seien. Beispielsweise übertreffe *MongoDB Cassandra* in zehn Studien, während *Cassandra Redis* in vier Studien und *Redis* wiederum *MongoDB* in sechs Studien übertreffe, was zu widersprüchlichen Ergebnissen führe (Mo > Ca > Re > Mo). Zudem seien Studien nicht ausreichend detailliert über ihre Testumgebungen gewesen, was Replikationen erschwere. Aus diesen Ergebnissen der Literatur folgt ein verstärkter Fokus auf Reproduzierbarkeit, da ohne vollständige Angaben zu Hardware und Parametern keine Aussagen getroffen werden können (Taipalus, 2024a, S. 9).

Die Architektur des Testsystems (siehe Abbildung 4.1) integriert mehrere Softwarekomponenten und -technologien, um reproduzierbare VDBMS-Benchmarks zu liefern.

4.1 Management- und Benchmarking-System für VDBMS

Für das Management- und Benchmarking-System wurde auf bestehende Open-Source-Software gesetzt, welche per *Fork* in einem dezidierten *Repository* um weitere Funktionen erweitert wurde. Die Software *vector-db-benchmark* von Qdrant Solutions GmbH (2022) auf GitHub ist ein Framework, das entwickelt wurde, um verschiedene Vektordatenbanken zu benchmarken und deren Leistung unter denselben Hardwarebedingungen zu vergleichen.

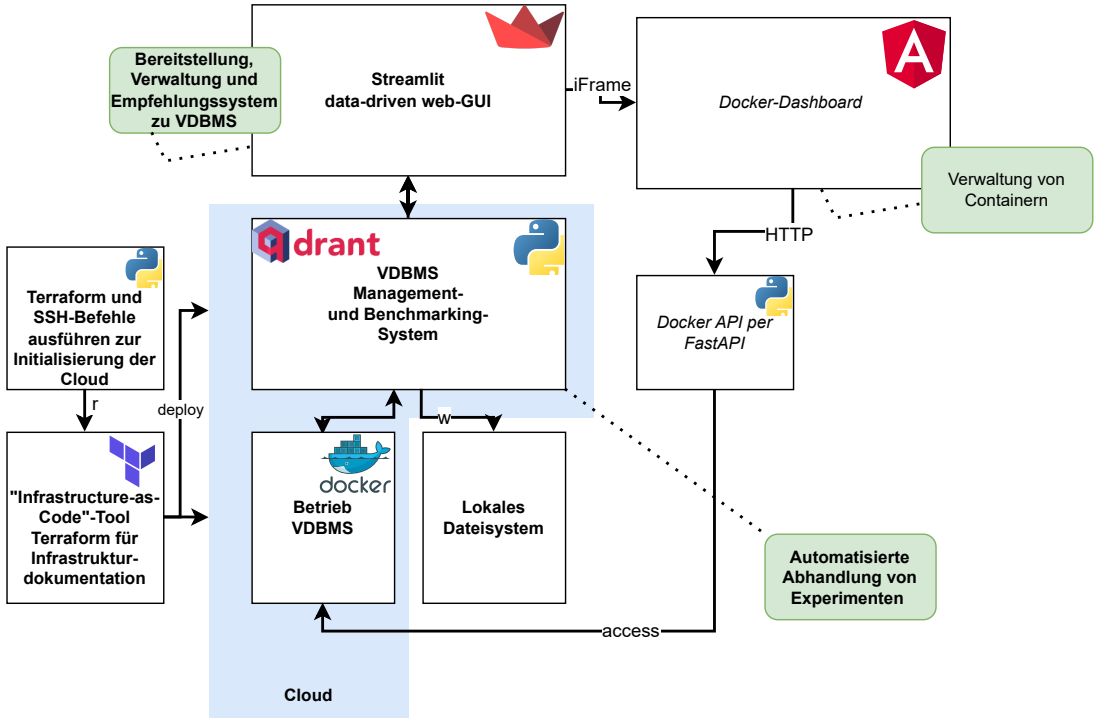


Abbildung 4.1: Architektur der Software zur Verwaltung von VDBMS. Kursivgedruckte Komponenten werden im Anhang D behandelt.

Ziel ist es, Entwickler:innen zu ermöglichen, die beste Datenbank für ihre spezifischen Anforderungen zu finden, sei es hinsichtlich der Geschwindigkeit, Effizienz oder anderer Leistungsmerkmale. Das Framework arbeitet in einer Client-Server-Architektur: Die Vektordatenbanken laufen als Serverinstanzen (unter Verwendung von Docker-Engine), während die Benchmarks vom Client ausgeführt werden. Es unterstützt bereits verschiedene VDBMS. Die Ergebnisse der Benchmarks werden lokal in JavaScript Object Notation (JSON) gespeichert und können anschließend analysiert werden. Dabei lässt sich die Konfiguration der Tests flexibel anpassen, z. B. für Single-Node- oder verteilte Systeme.

Diese *vector-db-benchmark*-Software wurde im Rahmen dieser Arbeit um *Chroma*- und *LanceDB*-Implementation sowie um ein Management-System erweitert. Das Hinzufügen von VDBMS (diese werden in ihrer jeweiligen Manifestation als Python-Client *Engine* genannt) ist durch das von Qdrant Solutions GmbH (2022) passend gewählte Adapter Design Pattern via Interfaces auf einfachem Wege möglich. Die Software enthält für jede implementierte Engine eine Klasse, welche von einer Oberklasse Methoden erbt. In den Unterklassen wird auf Python-Bibliotheken der Engines zurückgegriffen, die meist von den Autor:innen der Datenbanken bereitgestellt werden. Sie ermöglichen die Client-Server-Kommunikation.

Das Management-System umfasst die automatisierte Ausführung, Überwachung und Beendigung der VDBMS-Instanzen über Docker-Engine und sorgt dafür, dass diese Experimente konsistent und effizient ablaufen. Im Kern des Systems steht die Orchestrierung: Experimen-

Code-Beispiel 4.1: Beispielhafte Experimentkonfiguration.

```
{
  "name": "qdrant-m-16-ef-128",
  "engine": "qdrant",
  "connection_params": { "timeout": 30 },
  "collection_params": {},
  "search_params": [
    { "parallel": 1, "config": { "hnsw_ef": 64 } }, { "
       parallel": 100, "config": { "hnsw_ef": 128 } }
  ],
  "upload_params": { "parallel": 16 }
}
```

te mit verschiedenen Kombinationen von Datensätzen, Engines und spezifischen Parametern sequentiell durchzuführen, um die Leistungsfähigkeit der Konfiguration zu untersuchen. Die Parameter können beispielsweise konkrete Konfigurationsdetails eines Algorithmus wie HNSW umfassen.

Definition 5 Ein **Experiment** ist eine zusammenhängende Menge an Benchmark-Ausführungen. Es umfasst ein einmaliges Hochladen des gewählten Datensatzes und mehrere Suchanfragen (Liste in `search_params`).

Experimente werden in JSON definiert, siehe Code-Beispiel 4.1. So wird ein Name vergeben, der aus Engine und den statischen HNSW-Parametern (vgl. Definition 3) besteht. Notiert werden außerdem Parameter der Verbindung, Collection, Suche und des Hochladens des Datensatzes. Bei der Durchführung der Experimente sammelt das System sowohl Vergleichsmetriken als auch Daten der Hardwareressourcen, welche es ermöglichen, die VDBMS miteinander zu vergleichen. Insgesamt stellt das Management- und Benchmarking-System für VDBMS eine automatisierte, datengetriebene Plattform dar. Im Folgenden wird die Nutzung von Docker-Engine als angrenzendes System kurz beschrieben und die Cloud-Umgebung definiert.

Integration Docker-Engine Ein Docker-Host ist ein physischer oder virtueller Server, auf dem Docker-Engine ausgeführt wird. Das Management- und Benchmarking-System greift direkt auf den Docker-Host zu, um VDBMS-Container auszuführen. Dies geschieht mithilfe der Python-Bibliothek *python-on-whales*, die eine Python-Schnittstelle für die Docker-Engine bietet. Die Containerdefinition wird in Docker-Compose-Dateien niedergeschrieben.

4.2 Cloud-Infrastruktur

Die Amazon Web Services (*AWS*) ermöglichen es, den Docker-Host für die VDBMS-Container und die Python-Testclients physisch unabhängig auf Elastic Compute Cloud (*EC2*)-Instanzen zu betreiben. So sorgen sie für Vergleichbarkeit via Cloud und reproduzierbare, belastbare Leistungsdaten und vermeiden hohe Netzwerklatenzen.

Der Benchmark-Client sowie der Server werden dazu auf *AWS EC2*-Instanzen aufgesetzt. Die Cloud-Infrastruktur ermöglicht einen Benchmark unabhängig von der lokalen Hardware, und stellt sicher, dass die Kommunikation zwischen Benchmark-Client und Server im lokalen Netzwerk der Cloud ausgeführt wird. Dadurch können die Messungen ohne Störeinflüsse des Internets erfolgen.

Im vorliegenden Fall wurde eine Virtual Private Cloud (*VPC*) bei *AWS* aufgebaut, um die isolierte Netzwerkumgebung zu schaffen. Zudem wurde eine Security Group erstellt, welche ein- und ausgehenden Verkehr auf definierten Ports beschränkt, um sicherzustellen, dass nur bestimmte Protokolle und Ports verwendet werden. Ein Beispiel hierfür ist der erlaubte Secure Shell (SSH)-Zugriff auf Port 22 von überall aus, während der Zugriff auf andere Dienste wie die Docker-Engine, *Elasticsearch* oder *Redis* nur zwischen den beteiligten Instanzen innerhalb der *VPC* zugelassen ist. Diese granularen Zugriffskontrollen erhöhen die Sicherheit der Infrastruktur.

Neben der Netzwerkkonfiguration wurden wie eingangs erwähnt *EC2*-Instanzen bereitgestellt. Zwei spezifische Instanzen, ein Server und ein Client, wurden konfiguriert, wobei jeder eine eindeutige private IP-Adresse zugewiesen wurde. Beide Instanzen erhielten jeweils eine *Elastic IP*, um sie öffentlich erreichbar und damit per SSH konfigurierbar zu machen. CloudWatch wurde insbesondere für die Überwachung von Server- und Client-Aktivitäten konfiguriert.

Terraform-Integration Der Prozess der Erstellung dieser Cloudumgebung wurde anschließend mittels Terraform und Python automatisiert. Terraform kann die Infrastruktur, das heißt alle *AWS*-Ressourcen, automatisiert verwalten. Dies umfasste sowohl die Initialisierung der Infrastruktur als auch die Implementierung von Anwendungslogik durch das Terraform-Python-SDK und die Python-Integration mit *Paramiko*, einer Bibliothek für SSH-Protokolle.

Die Abbildung 4.2 zeigt den Prozess der Bereitstellung von *AWS*-Ressourcen, wie *EC2*-Instanzen und *VPC*-Security-Groups, aus einer lokalen Entwicklungsumgebung heraus, wobei Terraform verwendet wird, um mittels *Infrastructure-as-Code* die Cloud- und Laufumgebung exakt zu definieren und API-Aufrufe an die *AWS*-Cloud zu senden. Zudem können

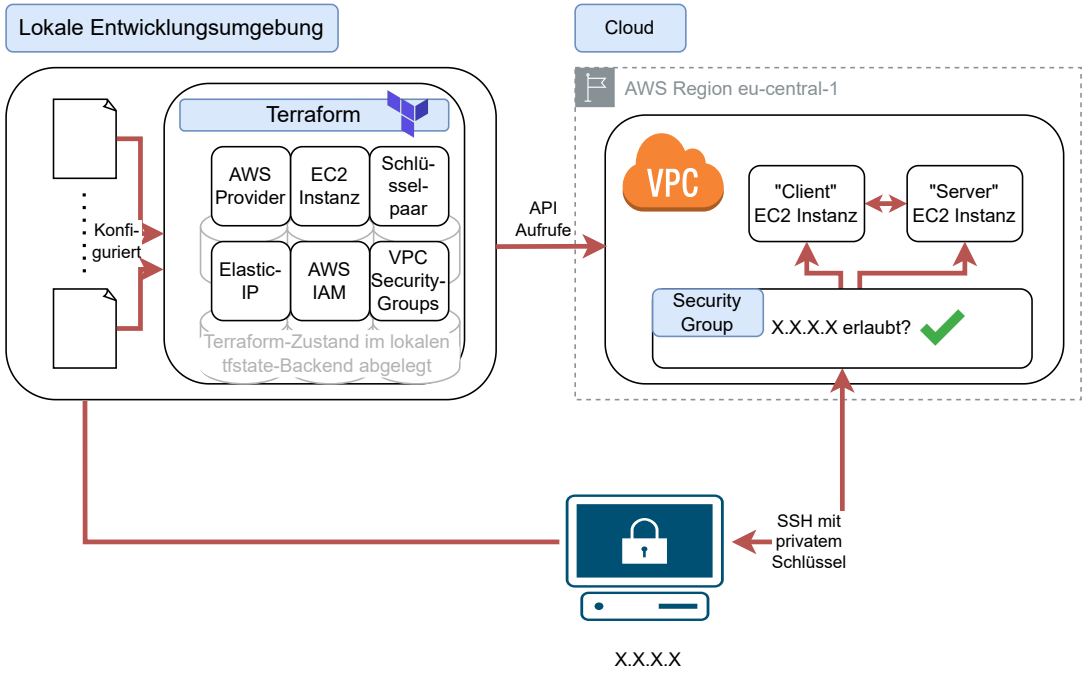


Abbildung 4.2: Architekturdiagramm der Infrastruktur, nach https://medium.com/@sanky43jadhav/deploy-aws-ec2-instance-key-pair-and-security-group-with-terraform-fee3249078f7, Stand: 21. Oktober 2024.

Verbindungen zu *EC2*-Instanzen über SSH hergestellt werden, wenn die entsprechende IP-Adresse in der Security Group erlaubt ist. Diese umfassende Dokumentation dient erneut der Reproduzierbarkeit.

4.3 Rastersuche und Laufzeitanalyse

Neben dem Aufbau der reproduzierbaren Methodik des VDBMS-Leistungsvergleichs ist auch ihre praktische Umsetzung von Relevanz für die Formulierung einer Empfehlung.

Wie in Algorithmus 1 zu sehen ist, gibt es eine Vielzahl an Parametern für den beschriebenen VDBMS-Benchmark. Viele dieser Parameter haben eine (teils unendlich) große Menge an Optionen. Parameter lassen die Anwender:innen die Eigenschaften einer VDBMS wie z. B. bezüglich Indexierung festlegen. Die Wahl der Parameter bzw. die Kombination aus Parametern hat somit entscheidenden Einfluss auf die Leistung der VDBMS. Die vorgestellten VDBMS haben alle ähnliche Parameter für die Indexierung (HNSW), sodass sie zur Betrachtung in diesem Kapitel generalisiert werden können. Um für Anwendungsszenarien aus Kapitel 3.1 geeignete Konfigurationen zu finden, wird im Rahmen dieser Arbeit auf die Rastersuche zurückgegriffen.

Algorithmus 1 : VDBMS-Benchmark; Laufzeitanalyse.

Name : `Experiment(`*E, C, P,* pUpload*, S,* dataset`)`
Eingabe : Liste Engines E, Liste collection_params C, Liste Parallelisierung für Suche P, Upload-Parallelisierung pUpload, Liste search_params S, Datensatz dataset
Nebeneffekte : Experimente ausgeführt und JSON-Ergebnisse gespeichert

```
// Durchlaufe für alle VDBMS
for alle engine ∈ E do
    Starte(engine);
    for alle m, ef ∈ C do                                   // ef ≡ ef_construction
        experimentName ← {engine}-m-{m}-ef-{ef};    // für JSON-Dateinamen
        Upload(experimentName, pUpload, m, ef, dataset);
        for alle parallelLevel ∈ P do
            for alle searchEf ∈ S do
                Suche(experimentName, parallelLevel, searchEf, dataset);
    Beende(engine);
// Die Anzahl der Funktionsaufrufe beträgt: |E| · (2 + |C| · (1 + |P| · |S|))
// Die Anzahl der Konfigurationen beträgt: |E| · |C| · |P| · |S|
```

Definition 6 Die **Rastersuche** (Grid Search) ist ein Konzept der Optimierung, um nach optimalen Parametern zu suchen. Es wird eine umfassende Suche innerhalb einer manuell festgelegten Teilmenge des Parameterraums des Algorithmus durchgeführt, wobei jedes Element dieser Menge eine bestimmte Konfiguration repräsentiert. Die Rastersuche evaluiert systematisch das kartesische Produkt der Mengen der gewählten, endlichen Optionen je Parameter. Die Anzahl der erforderlichen Funktionsaufrufe nimmt exponentiell mit der Dimension des Parameterraums zu (Feurer & Hutter, 2019, S. 7).

Sei die Anzahl der Konfigurationen einer Funktion |Funktion()| und der Betragsoperator $|\cdot|$ auf einer Liste die Anzahl der Elemente der Liste, dann folgt aus dem Algorithmus und der Definition zum kartesischen Produkt

$$|A \times B| = |A| \cdot |B|$$

die Gleichung (4.1).

$$|\texttt{Experiment(}E, C, P, \text{pUpload}, S, \text{dataset}\texttt{)}| = |E| \cdot |C| \cdot |P| \cdot |S| \tag{4.1}$$

Für die Anzahl der Konfigurationen für Benchmark Aufbau 2, siehe Code-Beispiel 4.2, gilt somit die Gleichung (4.2)

Code-Beispiel 4.2: Benchmark Aufbau 2.

```
dataset_name = "glove-100-angular"
normal_engines = ["chroma", "elasticsearch", "milvus", "opensearch-
    single-shard"]
extended_engines = ["pgvector", "qdrant", "qdrant-rps", "redis", "
    weaviate"]
collection_params = [(16, 128), (32, 128), (32, 256), (32, 512), (64,
    256), (64, 512)]
parallel_search = [1, 100]
parallel_upload = 16
normal_search_params = [128, 256, 512]
extended_search_params = [64, 128, 256, 512]
Experiment(normal_engines, collection_params, parallel_search,
    parallel_upload, normal_search_params, dataset_name) # exp1
Experiment(extended_engines, collection_params, parallel_search,
    parallel_upload, extended_search_params, dataset_name) # exp2
```

$$|\text{exp1}| + |\text{exp2}| = 4 \cdot 6 \cdot 2 \cdot 3 + 5 \cdot 6 \cdot 2 \cdot 4 = 384 \tag{4.2}$$

Es ergeben sich 384 untersuchte VDBMS-Konfigurationen (K), wobei die Anzahl der erforderlichen Funktionsaufrufe mindestens ebenso hoch ist. Des Weiteren ergibt sich das Produkt $K \times D$, wobei D die Menge der zu untersuchenden Datensätze bezeichnet. Dies stellt einen weiteren Parameter des Benchmarks dar, der jedoch zur Reduktion des Umfangs dieser Arbeit auf `glove-100-angular` fixiert ist.

5 Evaluation

Zunächst ein Hinweis: Die in diesem Dokument dargestellten Ergebnisse geben die Leistung der verschiedenen VDBMS zu einem bestimmten Zeitpunkt wieder. Zukünftige Versionen dieser Systeme werden voraussichtlich abweichende Resultate zeigen. Ziel dieser Untersuchung war es nicht, endgültig festzustellen, welches VDBMS das beste auf dem Markt ist. Stattdessen bestand das Anliegen darin, eine Reihe von Standard-Benchmarks zu entwickeln, die Entwickler:innen als Grundlage zur Bewertung ihrer Systemen dienen und Nutzer:innen bei der Wahl des für sie am besten geeigneten Systems helfen.

Dazu werden im Folgenden zwei Testaufbauten beschrieben und sowohl miteinander als auch mit einem Referenzbenchmark verglichen. Es folgt ein Vergleich der VDBMS. Dabei hat Qdrant Bestwerte erzielt.

Definition 7 **Aufbau 1** besteht aus zwei t3a.2xlarge *AWS-EC2*-Instanzen. **Aufbau 2** besteht aus zwei m6a.2xlarge *AWS-EC2*-Instanzen.

5.1 Messdaten des ersten Aufbaus

Die Datenwerte für die zu untersuchenden VDBMS aus der automatischen Messung auf dem Aufbau 1 sind in der Tabelle A.1 im Anhang gelistet. Die Metriken Durchsatz, durchschnittliche Latenz und Upload wurden für verschiedene Präzisionsstufen (80 %, 90 %, 95 %) gemessen. Auffällig im Vergleich zu den Daten von Qdrant Solutions GmbH (2022) ist, dass die eigenen Ergebnisse ca. 50 % schlechter sind, trotz identischer Implementierung der Systeme und grundlegender Ähnlichkeit in CPU, RAM und IOPS-Leistung, siehe Tabelle 5.1.

Die Ähnlichkeit zwischen dem Referenzsystem und Aufbau 1 legt nahe, dass die verschiedenen Aufbauten vergleichbare Einsatzmöglichkeiten bieten, jedoch ist ein wesentliches Merkmal von `t3a.2xlarge`, dass sie zur *Burstable*-Familie gehört. Das bedeutet, dass sie bei geringer Auslastung CPU-Credits ansammelt, um diese in Spitzenzeiten zu verwenden. Diese Instanzen sind darauf ausgelegt, kurzfristige Leistungsspitzen zu bewältigen, während sie im Normalbetrieb eine reduzierte CPU-Leistung bieten. Die Azure-Instanzen sowie

	Referenzsystem	Aufbau 1	Aufbau 2
Anbieter	Azure, Standard	AWS	AWS
Instanztyp	D8ls v5 / D8s v3	t3a.2xlarge	m6a.2xlarge
vCPU (Arch)	8x (x64)	8x (x86_64)	8x (x86_64)
CPU-Modell	n.a. / Intel Xeon E5-2673 v3, 2.40GHz	AMD EPYC 7571, 2.20 GHz	AMD EPYC 7R13, 3.60GHz
Arbeitsspeicher	16 GB / 32 GB	32 GB	32 GB
Maximale IOPS	38.000 / 16.000	15.700	40.000
Anbieter-beschreibung	General purpose compute	Multi purpose	Multi purpose

Tabelle 5.1: Vergleich der CPU- und Leistungsparameter von Azure- und *AWS*-Instanzen. Der erste Wert in einer Zelle beschreibt den Client, der zweite, nach dem Schrägstrich, den Server. Falls nur ein Wert angegeben ist, gilt dieser für beide. Datenstand: 21. Oktober 2024.

`m6a.2xlarge` hingegen sind darauf ausgelegt, durchgehend eine konstante Leistung zu erbringen, was sie für Benchmarks mit gleichbleibend hoher Auslastung viel geeigneter und performanter macht.
Aufgrund dessen erweist sich Aufbau 1 sowohl für das in dieser Arbeit angestrebte Empfehlungssystem als auch für den Einsatz in einer produktiven Umgebung eines VDBMS als ungeeignet.

5.2 Messdaten des zweiten Aufbaus

Die automatisierte Messung auf Aufbau 2 hat Messdaten für die zu untersuchenden VDBMS geliefert (siehe Tabelle A.2). Auf Basis dieser Ergebnisse lässt sich feststellen, dass Aufbau 2 dem ersten Aufbau in sämtlichen Aspekten überlegen ist. Sowohl in Bezug auf Durchsatz und durchschnittliche Latenz als auch Upload-Leistung zeigt Aufbau 2 deutlich bessere Resultate. Darüber hinaus übertreffen die Ergebnisse auch jene des Referenzsystems von Azure (siehe Tabelle 5.2). Einzige Ausnahme zu letzterem ist das VDBMS *Elasticsearch* (67 % schlechter) und *Milvus* (73 % schlechter) im Durchsatz-Benchmark sowie *Redis* (7 % schlechter) im Latenz-Benchmark. Positiv ist der Benchmark zu *Qdrant*, hier konnte das VDBMS im Durchschnitt über die drei Kategorien 45 % Leistungswachstum erreichen. Bei *Weaviate* waren es sogar 50 %. Damit hat Weaviate in den im nächsten Kapitel zu besprechenden absoluten Messwerten verglichen mit Qdrant etwas aufgeholt.

VDBMS	Durchsatz	Latenz	Upload
Milvus	**-89 %**	**61 %**	**147 %**
Qdrant	**-57 %**	**59 %**	**1 %**
Redis	**-48 %**	**106 %**	**12 %**
⌀	**-64,6 %**	**75,4 %**	**53,1 %**

(a) Leistungsvergleich zwischen dem Referenzsystem und Aufbau 1.

VDBMS	Durchsatz	Latenz	Upload
Milvus	**-73 %**	-39 %	-16 %
Qdrant	35 %	-40 %	-59 %
Redis	20 %	**7 %**	-52 %
Elastic-search	**-67 %**	-40 %	-40 %
Weaviate	64 %	-40 %	-41 %
⌀	**-4,2 %**	-30,4 %	-41,8 %

(b) Leistungsvergleich zwischen dem Referenzsystem und Aufbau 2.

Tabelle 5.2: Vergleich zwischen dem Referenzsystem und eigenen Aufbauten zu den Leistungsdaten von bis zu fünf VDBMS. In jeder Zelle wird der Durchschnitt über die Präzisionen (80 %, 90 %, 95 %) berechnet. Der Wert wird fettgedruckt, wenn er eine Verschlechterung des Aufbaus im Vergleich zum Referenzsystem darstellt. Beim Durchsatz in Anfragen pro Sekunde ($p = 100$) sind höhere Werte besser, wobei p die Anzahl der parallelen Prozesse angibt. Bei der durchschnittlichen Latenz ($p = 1$) und der Dauer des Hochladens des Datensatzes ($p = 16$) sind niedrigere Werte besser.

5.3 Leistungsvergleich der VDBMS

Im Folgenden werden nach nun erfolgtem Vergleich der zwei Leistungstestaufbauten sowie dem Referenzsystem von Qdrant Solutions GmbH (2022) die Leistungstests genauer betrachtet.

In den durchgeführten Leistungstests wurden die VDBMS hinsichtlich ihres Durchsatzes, ihrer durchschnittlichen Latenz und Upload-Performance untersucht. Die Ergebnisse zeigen Unterschiede in der Leistungsfähigkeit der Systeme. Die Tabellen A.1 und A.2 im Anhang A zeigen die Ergebnisse der Leistungstests von Aufbau 1 und Aufbau 2 für die VDBMS *Chroma*, *Milvus*, *OpenSearch*, *pgvector*, *Qdrant*, *Redis*, *Elasticsearch* und *Weaviate*.

In der folgenden Auswertung werden die VDBMS in den drei Kategorien anhand beider Aufbauten verglichen.

Vergleich des Durchsatzes Im ersten Aufbau zeigt *Qdrant* bei allen Präzisionsstufen die besten Durchsatzwerte. Mit 1081,33 r s^{-1} bei 80 % Präzision und 350,66 r s^{-1} bei 95 % Präzision übertrifft es die anderen Systeme deutlich. *Redis* folgt mit guten Ergebnissen, insbesondere bei 80 % Präzision (469,27 r s^{-1}). Im Gegensatz dazu haben Systeme wie *Chroma* und *OpenSearch* deutliche Probleme, insbesondere bei höheren Präzisionen. Sie erreichen nur geringe Werte oder fallen aus (failed). Auch im zweiten Aufbau ist *Qdrant* das performanteste System, siehe Abbildung 5.1. Es erreicht mit 2966,55 r s^{-1} bei 80 % Präzision den Höchstwert

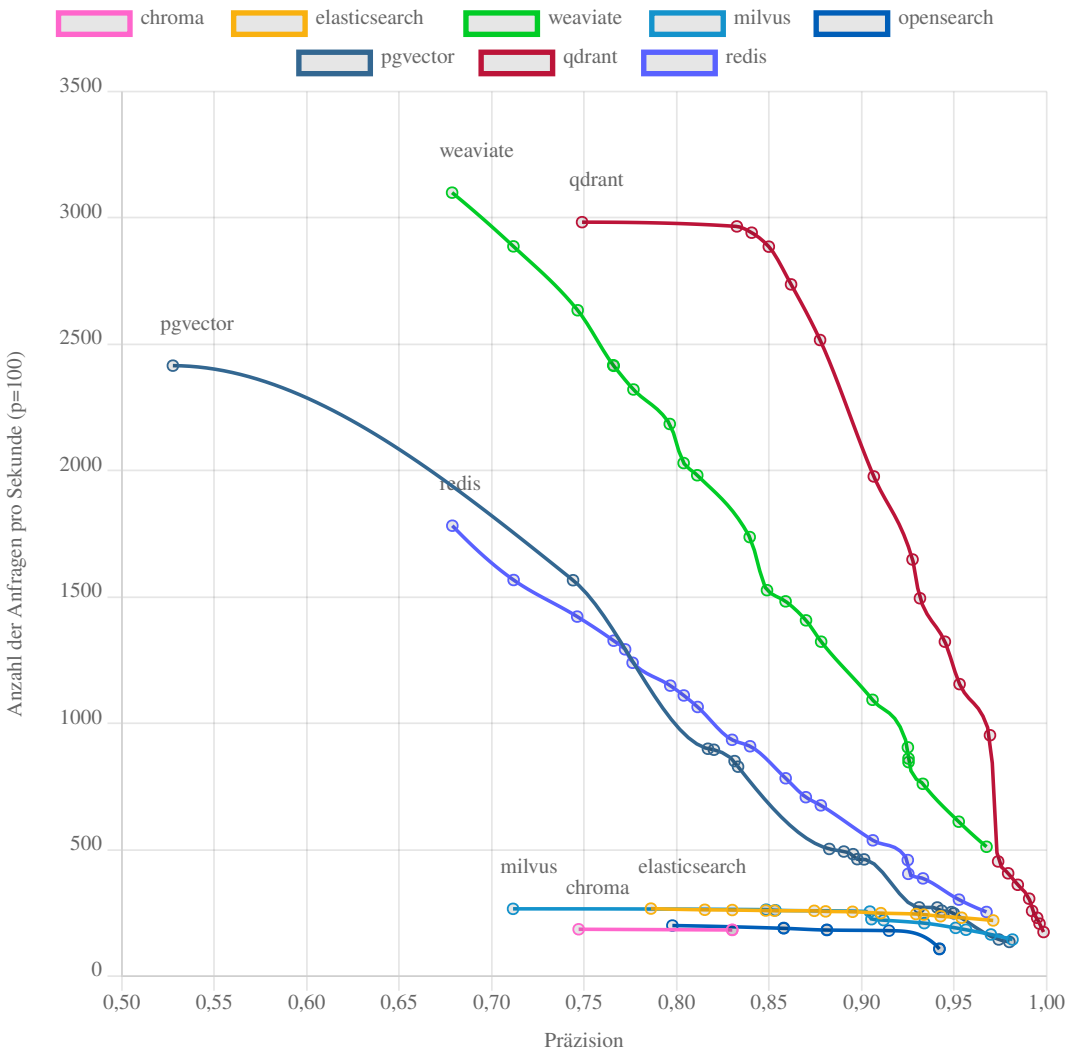


Abbildung 5.1: Vergleich des Durchsatzes von VDBMS auf einer *AWS* m6a.2xlarge Instanz. Die X-Achse zeigt die Präzision der Systeme, während die Y-Achse die Anzahl der Anfragen pro Sekunde der Ähnlichkeitssuche (bei 100 parallelen Prozessen) darstellt. Die Darstellungsform folgt der in Qdrant Solutions GmbH (2022) genutzten Methode, während die gezeigten Daten aus eigenen Messungen stammen.

im gesamten Test. Auch bei 95 % Präzision bleiben die Durchsatzwerte mit 1156,28 $\mathrm{r\,s^{-1}}$ hoch. Im Vergleich dazu zeigt *Redis* ebenfalls solide Ergebnisse, ist allerdings schwächer als *Qdrant* (1111,42 $\mathrm{r\,s^{-1}}$ bei 80 %). Die anderen Systeme, wie *Milvus* und *pgvector*, erzielen moderate Werte, während *Chroma* schwach abschneidet.

Vergleich der durchschnittlichen Latenz Bezüglich der durchschnittlichen Latenz zeigt *Redis* im ersten Aufbau bei 80 % Präzision mit einer Latenz von nur 4,56 ms das beste Ergebnis und bleibt auch bei höheren Präzisionen mit Werten unter 11 ms sehr performant. *Qdrant* zeigt einen niedrigen Latenzwert bei 95 % Präzision mit 9,78 ms (5,88 ms bei 80 %). *OpenSearch* und *Chroma* hingegen haben deutlich höhere Latenzen oder fallen aus. Im zweiten Aufbau

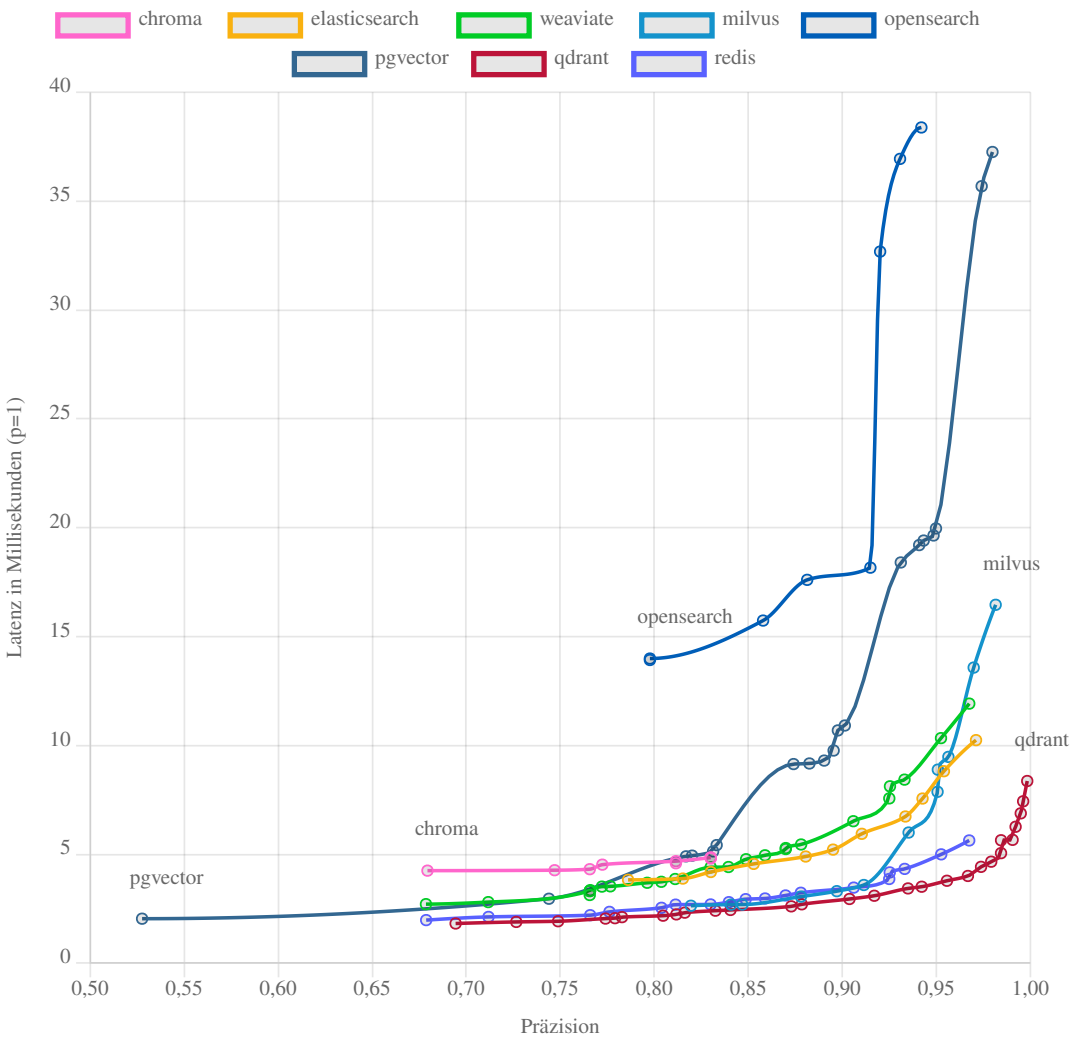


Abbildung 5.2: Vergleich der durchschnittlichen Latenz (in Millisekunden) von VDBMS auf einer *AWS* m6a.2xlarge Instanz. Die X-Achse zeigt die Präzision der Systeme, während die Y-Achse die Latenz der Ähnlichkeitssuche (bei seriellen Anfragen) darstellt. Die Darstellungsform folgt der in Qdrant Solutions GmbH (2022) genutzten Methode, während die gezeigten Daten aus eigenen Messungen stammen.

ist *Qdrant* das schnellste System mit 2,19 ms bei 80 % Präzision, wie in Abbildung 5.2 zu sehen. *Redis* und *Milvus* zeigen bei 80 % Präzision mit 2,55 ms und 2,64 ms ebenfalls gute Werte. Erwartungsgemäß fallen die Latenzen bei allen Systemen im zweiten Aufbau geringer aus.

Vergleich der Upload-Performance Bei der Betrachtung des ersten Aufbaus zeigt sich, dass *Qdrant* in allen Präzisionsstufen den schnellsten Upload mit konstant niedrigen Werten erreicht. Auch beim zweiten Aufbau erzielt es durchweg die kürzesten Upload-Dauern (2,41 min bei 80 %, 90 % und 95 %), unabhängig von der Präzisionsstufe, und ist somit das effizienteste System beim Datenimport (Abbildung 5.3). *Milvus* und *OpenSearch* zeigen hier bei niedrigen Präzisionen gute Ergebnisse (2,43 min und 2,84 min bei 80 % Präzision), je-

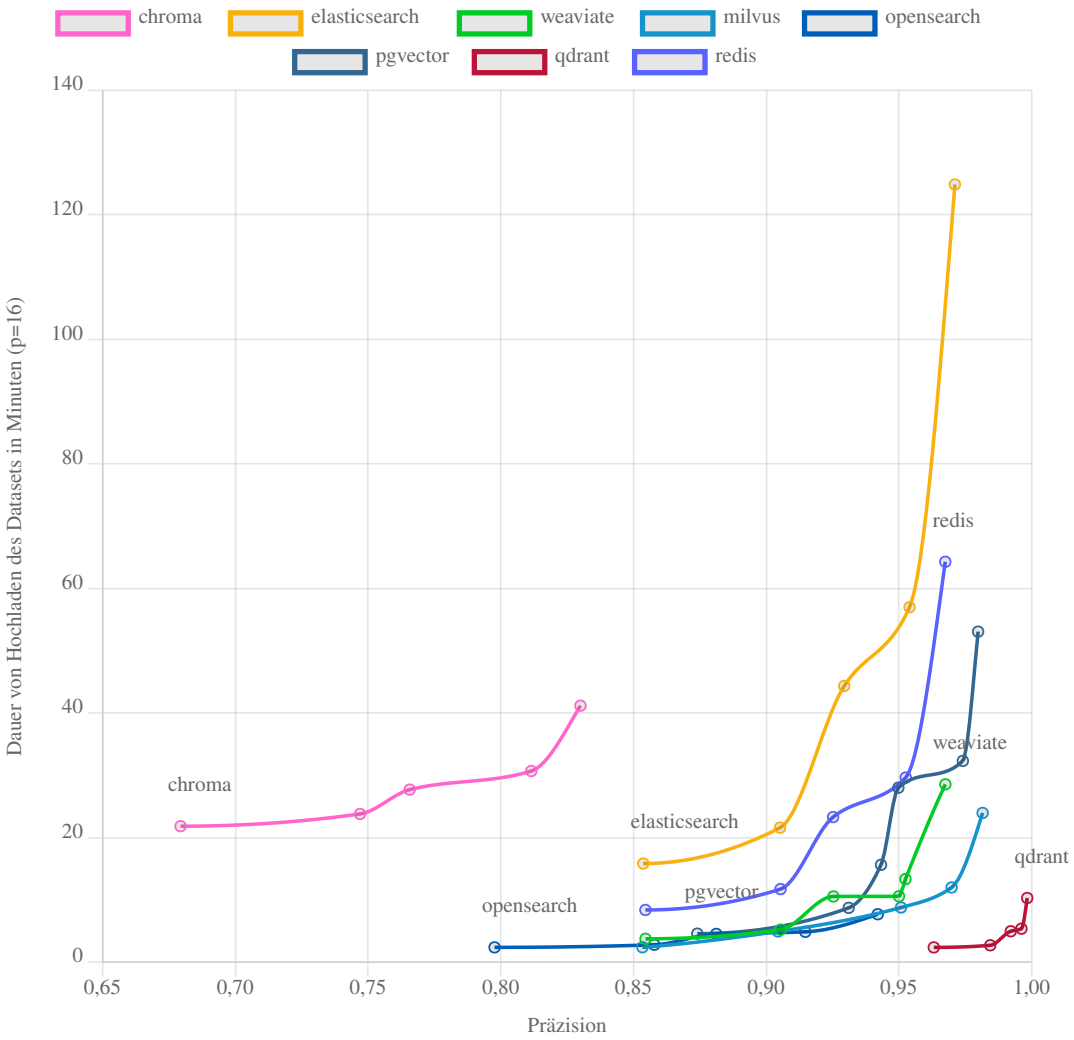


Abbildung 5.3: Vergleich der Ladezeiten (in Minuten) von VDBMS auf einer *AWS* m6a.2xlarge Instanz. Die X-Achse zeigt die Präzision der Systeme, während die Y-Achse die Dauer des Hochladens eines Datensatzes (bei 16 parallelen Prozessen) darstellt. Die Darstellungsform folgt der in Qdrant Solutions GmbH (2022) genutzten Methode, während die gezeigten Daten aus eigenen Messungen stammen.

doch verschlechtern sich die Werte bei höherer Präzision. Die Ergebnisse verdeutlichen, dass *Qdrant* aufgrund seiner konsistenten und sehr niedrigen Upload-Zeiten für Anwendungsfälle, die eine schnelle Datenaufnahme und Indexierung erfordern, gut geeignet ist.

In der vollzogenen Leistungsmessung zeigt sich, dass *Qdrant* in allen drei getesteten Bereichen – Durchsatz, Latenz und Upload – und besonders bei hohen Präzisionen die besten Werte erzielte. Da Durchsatz, Latenz und Upload, wie in Kapitel 3.3 gezeigt, wesentliche Leistungskriterien für VDBMS darstellen und sich sogar antithetisch verhalten, legen die Ergebnisse nahe, dass *Qdrant* eine gute Wahl ist. Dennoch hängt die Wahl des geeigneten Produktes stark von den spezifischen Anforderungen des jeweiligen Anwendungsszenarios ab.

Probleme Für *Chroma* bringt Multithreading bei der Suche keinen großen Gewinn, da es synchron arbeitet und den Zugriff auf Sammlungen serialisiert. Das bedeutet, dass bei 100-facher Parallelisierung 99 Anfragen warten müssen.

Die Analyse der CPU-Auslastung während der Ähnlichkeitssuche ist ein zentraler Aspekt der Bewertung der Systemperformance. Abbildung 5.4 zeigt die CPU-Last bei der Durchführung einer Ähnlichkeitssuche auf dem Server des zweiten Versuchsaufbaus. Dabei lässt sich erkennen, dass die Auslastung über den gesamten Zeitraum relativ konstant unterhalb von 50 % bleibt, was auf eine effiziente Nutzung der Rechenressourcen während der Suchvorgänge hinweist.

Im Vergleich dazu illustriert Abbildung 5.5 die CPU-Last während des Upload-Vorgangs. Hier wird deutlich, dass die Prozessorlast beim Großteil der VDBMS bei ca. 100 % auf allen 6 Kernen stagniert, was auf einen Flaschenhals hindeutet.

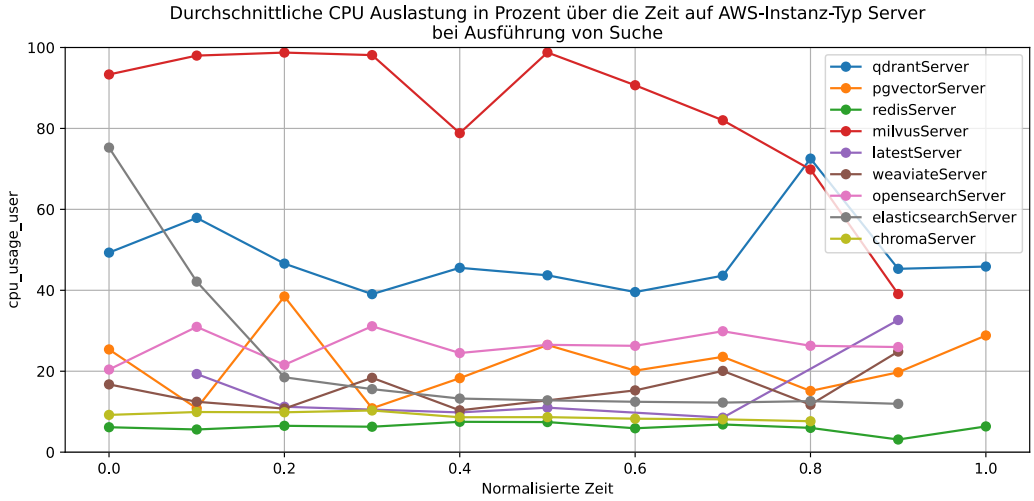


Abbildung 5.4: CPU-Last bei Ähnlichkeitssuche auf Server (Aufbau 2).

5.4 Empfehlungssystem

Mithilfe der ermittelten Ergebnisse aus dem Kapitel 5.2 lässt sich ein Prototyp für ein Empfehlungssystem entwickeln. Um die Präferenzen von Nutzer:innen systematisch zu erfassen und die Eignung von VDBMS anhand der Anforderungen zu bewerten, wird ein Bewertungssystem entwickelt. Im Folgenden wird der Prozess zur Nutzung des Bewertungssystems erläutert:

Zunächst geben die Nutzenden ihre gewünschten technische Spezifikationen an. Zu jedem dieser Kriterien kann die nutzende Person zusätzlich eine Gewichtung vornehmen, die ihre individuellen Prioritäten verdeutlicht. Die Gewichtung wird in Form einer natürlichen

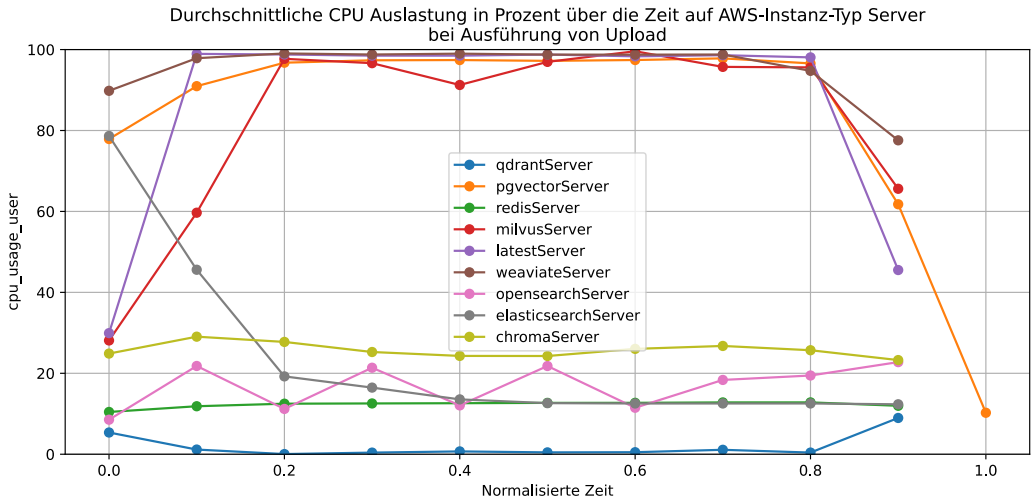


Abbildung 5.5: CPU-Last bei Upload und Indexierung auf Server (Aufbau 2).

Zahl angegeben, wobei höhere Zahlen eine stärkere Relevanz des jeweiligen Kriteriums signalisieren. Der Algorithmus hinter dem Bewertungssystem nutzt diese Informationen, um die Kompatibilität der Produkte mit den Anforderungen zu berechnen. Je detaillierter und umfangreicher die Informationen, desto genauer kann die Bewertung erfolgen. Bei wenig Informationen fallen die Standardwerte für *Checkboxen* mehr ins Gewicht. Die Kompatibilität wird am Ende des Prozesses als Score in Prozent dargestellt.

Die Ergebnisse werden in einer sortierten Liste präsentiert, wobei die Produkte mit dem höchsten Kompatibilitätsscore oben aufgeführt werden. Der Score reflektiert die Übereinstimmung der Produkte mit den eingegebenen Kriterien und ihrer Gewichtung, wobei ein höherer Score auf eine größere Eignung des Produkts hinweist. Durch dieses Verfahren erhalten Nutzer:innen eine transparente und nachvollziehbare Übersicht, die auf objektiven Berechnungen basiert und ihnen ermöglicht, fundierte Entscheidungen zu treffen. Allgemein betrachtete Kriterien sind:

1. die Anzahl der Vektoren,
2. die Dimension dieser,
3. Programmiersprachen von Clientbibliotheken,
4. die Option, *serverless* in die Cloud deployed zu werden,
5. die Option der Open-Source-Entwicklung,
6. die Option der kostenfreien Benutzung,
7. Features wie Transaktionsunterstützung,
8. das maximale Alter des Produktes,
9. benötigte Präzision,
10. maximal tolerierbare durchschnittliche Latenz,
11. maximal tolerierbare Uploadzeit sowie
12. der erforderliche Durchsatz.

Konkret aus den Benchmark-Ergebnissen des zweiten Aufbaus leiten sich die Kriterien 9., 10., 11. und 12. ab. Der Prototyp ist mit *Streamlit* – einem Python-Framework für datengetriebene Bedienoberflächen – entwickelt und alle Kriterien erben von der `RankFactor`-Klasse, welche den Factor-Criteria-Score errechnet. Die Faktoren sind Ganzzahlen, die als Gewichte der Kriterien dienen. Der `total_score` für ein bestimmtes VDBMS wird durch die Summe aller individuellen Scores berechnet. Diese individuellen Scores werden durch verschiedene Ranking-Faktoren (z. B. `ClientsRankFactor`, `LatencyRankFactor`) bestimmt. Jeder dieser Summanden hat einen bestimmten Wert, der auf Basis von Input-Werten (z. B. wünschenswerte Clients, maximal tolerierbare Latenz) und einem Gewicht (factor) berechnet wird. Dieser Wert ist im Interval $[0, 10]$. Die `total_score`-Werte werden nun absteigend sortiert mit ihren VDBMS tabellarisch dargestellt. Die graphische Bedienoberfläche zum Prototyp, welche für die vier Szenarien genutzt wurde, ist in der Abbildung B.1 im Anhang B samt beispielhaften Eingaben zu sehen. Dadurch sind sowohl die Empfehlungen als auch der dahinterliegende Algorithmus nachvollziehbar. Durch den Code des Algorithmus sind die Empfehlungen aus Kapitel 5.5 nicht mehr subjektiv, sondern vollständig nachvollziehbar und reproduzierbar. Ebenso können Nutzer:innen über die App individuelle Hilfe für ein eigenes Szenario erhalten.

5.5 Auswertung der Schlüsselszenarien anhand des Empfehlungssystems

Im Folgenden werden für die Anforderungen der einzelnen Schlüsselszenarien am besten geeignete VDBMS herausgestellt.

Für Szenario 1, ein Automobilunternehmen mit einem Kundenbegehren für Chatbot-Funktionen, ist eine maximale durchschnittliche Latenz von 100 ms bei der gängigen Präzision für RAG-Systeme von 0,80 bei allen VDBMS gegeben. Bei Eingabe in den Empfehlungssystem-Prototypen bei Standardwerten und angegebenen drei Anforderungen (Latenz: 100ms, Präzision: 80 %, Client: Python) erhalten alle getesteten VDBMS einen Score von 72 % und sind damit gleichermaßen empfehlenswert.

Im Szenario 2, dem Hochfrequenzhandel im Finanzmarkt, eignet sich eine latenzoptimierte Konfiguration. Bei Eingabe in das Empfehlungssystem bei Standardwerten und zwei angegebenen Anforderungen (Latenz: 3ms, Präzision: 90 %) gewinnt *Qdrant* mit 72 % Passgenauigkeit und ist damit zu empfehlen. Alle anderen VDBMS liegen auf dem zweiten Platz (63 %).

Im Szenario 3, im Bereich der medizinischen Bildverarbeitung, ist eine hohe Konsistenz der Daten entscheidend, um akkurate Diagnosen zu ermöglichen. Dies ist darin begründet, dass

Fehler, die während einer Aktualisierung der Daten auftreten (z. B. ungültige Vektorformate), im Rahmen einer Transaktion zurückgerollt werden. Es wird verhindert, dass Updates teilweise durchgeführt werden und zu inkonsistenten Datensätzen führen könnten. Um diesen Fall zu simulieren, wird in dem Empfehlungssystem unter Features *ACID Transactions* ausgewählt. Nur pgvector ist geeignet (72 %).
Die Anforderungen an Konsistenz und Transaktionen sind schwer kombinierbar mit hohen Latenz- oder Durchsatzanforderungen. Systeme wie *pgvector*, die auf PostgreSQL basieren, bieten robuste ACID-Transaktionsunterstützung, können jedoch im Vergleich zu In-Memory-Datenbanken wie *Redis* höhere Latenzen aufweisen. Für Anwendungen, bei denen Konsistenz kritischer ist als die Leistung, ist die Wahl eines Systems mit umfassender Transaktionsunterstützung wie *pgvector* sinnvoll.

Im Szenario 4, der Moderation großer Datenmengen bei einer Videospielentwicklungsfirma, eignet sich eine durchsatzorientierte Konfiguration. Zur Simulation dieses Falls wird im Empfehlungssystem die Präzision auf 80 % und die Höhe des erforderlichen Durchsatzes auf $2000\,\mathrm{r\,s^{-1}}$ gesetzt. Dadurch erhalten *Weaviate* sowie *Qdrant* einen Score von 72 % und gewinnen damit das Ranking.

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde die Entwicklung und Anwendung von Vergleichsmetriken zum systematischen Leistungsvergleich sowie zur Bewertung der Eignung von neun VDBMS im Kontext von KI-Anwendungen ausgearbeitet und durchgeführt. Dazu wurden zunächst *Embeddings* als Grundlage von VDB geklärt, um dann das asymptotische Verhalten der Algorithmen des nächsten Nachbarn und der Ähnlichkeitsmessung anzuführen. Neun VDBMS wurden gewählt und vorgestellt und Vergleichsmetriken sowie nominale Eigenschaften festgelegt, basierend auf aktuellen Publikationen. Diese Metriken bildeten die Grundlage für eine Anforderungsanalyse. Anschließend wurde die Architektur des Testsystems beschrieben, welches sich durch Reproduzierbarkeit auszeichnet. Zwei verschiedene Testaufbauten wurden implementiert und die Eigen- mit Fremdmessungen verglichen. Es wurden zahlreiche Daten aus den Experimenten gesammelt, analysiert und bewertet, um die Systeme zu vergleichen und ein „Gewinner-VDBMS“ zu ermitteln. Die Ergebnisse wurden auch im Empfehlungssystem auf Basis der festgelegten Metriken eingesetzt, anhand dessen schließlich prototypisch für die vier eingangs introduzierten Schlüsselszenarien Empfehlungen für geeignete VDBMS ausgesprochen werden konnten.

Im Folgenden werden die wichtigsten Erkenntnisse zusammengefasst, außerdem erfolgt ein Ausblick auf weitere Forschungsfragen, die hier nicht mehr besprochen werden können.

Da für eine fundierte Entscheidung eine systematische (und damit replizierbare) Methodik substanziell ist, wurde in dieser Arbeit der Prototyp eines auf Metriken basierenden Empfehlungssystems entwickelt. Hinsichtlich der VDBMS-Empfehlungen für beispielhafte Szenarien mit verschiedenen Schwerpunkten konnte gezeigt werden, dass eine präzise Analyse von Metriken der VDBMS für die Bewertung der Leistungsfähigkeit in den spezifischen Anwendungsfällen essenziell ist. Insbesondere verdeutlicht die Untersuchung, dass neben klassischen Performance-Metriken wie Latenz und Durchsatz auch spezialisierte Metriken für KI-gestützte Szenarien, z. B. Präzision, von entscheidender Bedeutung sind.

Die Evaluation der in den Experimenten gemessenen Werte zu Durchsatz, durchschnittlicher Latenz und Uploadzeit zeigt, dass *Qdrant* in allen drei Kategorien beim zweiten Aufbau die besten Ergebnisse erzielt und das derzeit performanteste VDBMS ist. *Chroma* und *OpenSearch* hingegen schneiden in vielen Bereichen schwach ab oder fallen aus, insbesondere bei

höheren Präzisionsstufen. *Redis* zeichnet sich durch niedrige Latenz aus und *pgvector* durch Unterstützung für Transaktionen.

Resümierend ist ein VDBMS ein mächtiges Werkzeug zur Verwaltung der hochdimensionalen Vektoren, die in vielen modernen AI- und maschinellen Lernanwendungen auftreten. VDBMS wurden in jüngster Zeit neu- und weiterentwickelt. Die Indexierungsalgorithmen für das kNN-Problem sind aktuelle Forschungsthemen, wie auch andere spezialisierten Mechanismen zur effizienten Speicherung und Abfrage von Vektordaten, um die Leistung und Skalierbarkeit solcher Anwendungen zu gewährleisten. Komplex ist die Durchführung eines Benchmarks vor allem deshalb, weil der Parameterraum sehr groß ist. Da sich viele Parameter (beispielsweise Präzision der Indexierung und Performance der Suche) antithetisch zueinander verhalten, ist auch die Bestleistung der VDBMS mehrdimensional. Dies unterstreicht die zentrale Bedeutung einer individuellen Anforderungsaufnahme sowie der transparenten Darstellung der notwendigen Kompromisse, wie es in vier Fallbeispielen systematisch analysiert und konsequent durchdacht wurde. Die Beispiele verdeutlichen, wie eine sorgfältige Abwägung divergierender Anforderungen dazu beiträgt, ein ausgewogenes Ergebnis zu erzielen und den Projekterfolg nachhaltig zu sichern. Eine klare Kommunikation der resultierenden Entscheidungen ist in der graphischen Bedienoberfläche (Kapitel 5.4) fest verankert.

Die Schlüsselfaktoren für das Design eines VDBMS-Benchmarks sind Flexibilität, Skalierbarkeit und Präzision bei der Messung verschiedener Vergleichsmetriken. Das System muss in der Lage sein, reale Anwendungsszenarien abzubilden, spezifische Anforderungen zu testen und reproduzierbare Ergebnisse zu liefern. Darüber hinaus sollte es modular und erweiterbar sein, um mit den sich schnell entwickelnden Anforderungen im Bereich der Vektordatenbanken Schritt halten zu können.

Im Ausblick auf weitere Arbeiten an dem hier vorgestellten Benchmarking-System sind folgende Punkte erwähnenswert:

Neue VDBMS, welche zwar zum Zeitpunkt der Evaluation dieser Arbeit noch nicht die Bedingung GitHub-*Stars* >= 9000 erfüllten, können in Zukunft mit untersucht werden. Aussichtsreiche Kandidaten sind z. B. *LanceDB* und *Vespa.AI*. Eine Besonderheit von *LanceDB* ist die Integration von *columnar storage*, welche für große Datensätze eine effiziente Speicher- und Abfrageleistung ermöglicht. Wie im Kapitel 3.2 erwähnt, wurde das VDBMS *Typesense* nicht berücksichtigt und könnte in folgenden Arbeiten auch ergänzt werden. Durch die Architektur des Benchmarking-Systems sind solche Erweiterungen einfach möglich.

Weitere graphische Oberflächen für ein niedrigschwelliges Angebot von Experimentiermöglichkeiten mit VDBMS können neuen Nutzer:innen die Einführung in VDBMS, z. B. per einfacher Bereitstellung, Management und Monitoring, ermöglichen. Im Anhang D ist

eine am Rande dieser Arbeit entwickelte Oberfläche zu finden, welche noch um eine Search- und Visualisierungsfunktion, analog zu Sahoo et al. (2023, S. 4), für mehr Interaktion mit der Vektordatenbank erweitert werden könnte.

Neue Technologien, die den VDBMS-Markt verändern könnten, wären neue Indexierungsalgorithmen sowie mehr GPU- und SIMD-Unterstützung. Bei der Benchmark-Software wäre ein Fokus auf GPU-Hardware interessant.

Mehr Optionen im prototypischen Empfehlungssystem könnte MoSCoW („Must have, Should have, Could have, Won't have") bedeuten. Das MoSCoW-Prinzip wird häufig im Projektmanagement verwendet, um Anforderungen in Kategorien zu unterteilen und Prioritäten für Projekte festzulegen. Es könnte in das Empfehlungssystem eingebettet werden, um mit Priorisierung von Anforderungen zur Evaluierung beizutragen. In diesem neuen Kontext könnte MoSCoW folgendermaßen verwendet werden:
Die *Must-have*-Kriterien umfassen unverzichtbare Eigenschaften. Diese Anforderungen sind essenziell, um die Technologie überhaupt in Erwägung zu ziehen. Wenn sie eingestellt sind und bei VDBMS zutreffen, dann wird der Score erhöht (*positive true*). Doch für den Ausschluss von VDBMS, falls ein *Must-have*-Kriterium nicht erfüllt wird (*positive false*), bräuchte es ein Filtersystem. Unter die *Should-have*-Kriterien fallen wünschenswerte, aber nicht zwingend notwendige Funktionen wie die Unterstützung von Clients in weiteren Programmiersprachen oder Kosten-Constraints. *Could-have*-Kriterien beinhalten optionale, wertsteigernde Merkmale wie Cloud-native Funktionen, die zusätzlichen Komfort bieten, aber nicht zwingend erforderlich sind. Mithilfe der *Won't-have*-Funktion können unerwünschte Eigenschaften von VDBMS eingestellt werden. Wenn sie bei *Checkboxen* eingestellt sind und bei VDBMS fehlen, dann wird der Score erhöht (*negative false*). Auch für die *Won't-have*-Funktion bei multivalenten Feldern und Ausschlüssen per *negative true* bräuchte es ein Filtersystem. Alle vier Kriterien könnten aktuell – abgesehen von dem Ausschluss – schon mithilfe der Faktorgewichtung umgesetzt werden. Dafür könnte *Must-have* einem Faktor 10, *Should-have* Faktor 5, *Could-have* Faktor 1 und *Won't-have* Faktor 0 entsprechen.

Als Reaktion auf Boris Power, in dessen Anlehnung der Begriff „Einbettung von allem" eingangs zitiert wurde, verdeutlichte Jeff Harris, ebenfalls Mitarbeiter bei *OpenAI*, am 13.06.2023 im sozialen Netzwerk X die Relevanz der Zugänglichkeit von Vektordatenbankmanagementsystemen für Unternehmen, die nun in der Lage sind, günstig große Mengen an Daten zu analysieren und zu verarbeiten:

„we dropped embeddings prices by 75 % today
now you can embed the whole internet for < $15M" [14]

[14] Link: https://x.com/jeffintime/status/1668702131560542208 (Stand: 21. Oktober 2024).

Glossar

ACID In der Informatik steht ACID für Eigenschaften die Zuverlässigkeit von Transaktionen in Datenbanksystemen gewährleisten. 22

BERT Bidirectional Encoder Representations from Transformers (BERT) ist ein Modell, das bidirektionale Kontextinformationen für Wörter lernt, indem es die Wörter vor und nach einem Wort im Satz betrachtet. 9

CNN Ein Convolutional Neural Network (CNN) ist ein spezialisiertes künstliches neuronales Netzwerk, das insbesondere für die Verarbeitung von Daten mit einer Gitterstruktur, wie Bildern, entwickelt wurde und durch den Einsatz von Faltungsoperationen räumliche Zusammenhänge in den Daten erkennt. 9

Docker-Compose Docker-Compose ist eine auf Docker basierende Technologie zur Orchestrierung von mehreren Docker-Containern. 29

Docker-Engine Docker Engine ist eine Open-Source-Software zur Isolierung von Anwendungen mit Containervirtualisierung. 28–30

GitHub GitHub ist eine webbasierte Plattform für die Versionsverwaltung und gemeinsame Softwareentwicklung, die auf Git basiert und es Entwickler:innen ermöglicht, Code zu hosten, zu verwalten, zu versionieren und kollaborativ an Projekten zu arbeiten. 10, 20, 27, 45

GPT Generative Pretrained Transformer (GPT) ist ein unidirektionales Sprachmodell zur Textgenerierung, das auf einem großen Datensatz vortrainiert wurde und in der Lage ist, flüssigen Text zu generieren. 1

Greedy-Algorithmus Ein Greedy-Algorithmus ist ein Optimierungsverfahren, das durch eine Abfolge sukzessiver Entscheidungen, basierend auf lokalen Informationen, jeweils die zum Zeitpunkt beste Auswahl trifft, um schrittweise eine Lösung zu konstruieren, wobei jedoch keine Garantie besteht, dass die resultierende Lösung global optimal ist. 16

IDC IDC ist ein weltweit führendes Unternehmen für Marktforschung, Beratung und Veranstaltungen in den Bereichen Informationstechnologie, Telekommunikation und Verbrauchertechnologie. 2

KI Ein Bereich der Informatik, der sich mit der Entwicklung von Systemen beschäftigt, die in der Lage sind, Aufgaben zu lösen, für die normalerweise menschliche Intelligenz erforderlich ist. Dies umfasst das Lernen aus Daten, das Erkennen von Mustern, die Sprachverarbeitung, Entscheidungsfindung und Problemlösung. KI-Anwendungen finden sich unter anderem in Spracherkennung, Bildverarbeitung, autonomem Fahren und Empfehlungssystemen. 1

LLaMA Large Language Model Meta AI (LLaMA) ist ein von Meta entwickeltes Sprachmodell für die Verarbeitung natürlicher Sprache. 9

LLM Ein Large Language Model ist ein künstliches neuronales Netzwerk, das darauf trainiert ist, natürliche Sprache zu verstehen und zu generieren, indem es große Mengen an Textdaten analysiert und kontextabhängige Zusammenhänge zwischen Wörtern und Phrasen lernt. 9

RAG Retrieval-Augmented Generation ist ein Ansatz in der natürlichen Sprachverarbeitung, bei dem ein Modell relevante externe Informationen aus einer Datenbank oder einem Dokumentenspeicher abruft, um diese in die Generierung von Antworten oder Texten einzubeziehen und so die Qualität und Genauigkeit der generierten Inhalte zu verbessern. 10

RTG Eine Arbeitsgruppe, bestehend aus führender Persona der Datenbankforschung und -anwendung, die 1979 vom American National Standards Institute (ANSI) beauftragt wurde, Kriterien für relationale Datenbanken zu entwickeln. 22

vCPU Eine virtuelle zentrale Verarbeitungseinheit, die von einem Hypervisor in einer virtualisierten Umgebung erstellt wird. Eine vCPU repräsentiert eine logische CPU, die einem virtuellen Server zugewiesen ist, und ermöglicht es diesem, Rechenleistung zu nutzen, die von physischen CPU-Ressourcen bereitgestellt wird. vCPUs sind entscheidend für die Skalierbarkeit und Effizienz in Cloud-Computing-Umgebungen, da sie es mehreren virtuellen Maschinen ermöglichen, gleichzeitig auf die Hardware-Ressourcen eines physischen Servers zuzugreifen. 25

VDBMS Ein Vektordatenbankmanagementsystem ist ein spezialisiertes Datenbanksystem, das hochdimensionale Vektordaten speichert und abfragt, oft für Anwendungen wie maschinelles Lernen, Ähnlichkeitssuchen oder semantische Abfragen. 1

Abkürzungsverzeichnis

AWS Amazon Web Services 30, 34, 35, 37–39

EC2 Elastic Compute Cloud 30, 31, 34

VPC Virtual Private Cloud 30

ACID Atomicity Consistency Isolation Durability 22, 26, 43, *Glossar:* ACID

ANN Approximierter Nächster Nachbar 15–17, 24

ANNOY Approximate Nearest Neighbors Oh Yeah 18, 21

ANSI American National Standards Institute 22

API Application Programming Interface 21, 30

BERT Bidirectional Encoder Representations from Transformers 9, *Glossar:* BERT

CL Computerlinguistik 5, 7

CNN Convolutional Neural Network 9, *Glossar:* CNN

DBMS Datenbankmanagementsystem 3, 19, 22, 23, 27

GloVe Global Vectors for Word Representation 7, 8, 24

GPT Generative Pretrained Transformer 1, 9, *Glossar:* GPT

HFT Hochfrequenzhandel 20

HNSW Hierarchical Navigable Small World 15–18, 25, 26, 29, 31

IDC International Data Corporation 2, 3, *Glossar:* IDC

IVF Inverted File 15–17, 25

JSON JavaScript Object Notation 28, 29

KI Künstliche Intelligenz 1–3, 5, 7, 8, 10, 19–21, 25, 44, *Glossar:* KI

kNN k-Nächster Nachbar 15, 17, 18, 45

LLaMA Large Language Model Meta AI 9, *Glossar:* LLaMA

LLM Large Language Model 9, 10, 12, 13, *Glossar:* LLM

LSA Linguistic Society of America 6, 8

LSH Locality-Sensitive Hashing 18

NSW Navigable Small World 16, 17

PPMI positive pointwise mutual information 6

PQ Product Quantization 17

RAG Retrieval-Augmented Generation 10, 42, *Glossar:* RAG

RTG Relational Task Group 22, *Glossar:* RTG

ScaNN Scalable Nearest Neighbors 18, 21

SIMD Single Instruction Multiple Data 11, 46

SQ Scalar Quantization 17, 25

SSH Secure Shell 30, 31

vCPU virtuelle zentrale Verarbeitungseinheit 25, *Glossar:* vCPU

VDB Vektordatenbank 1, 44

VDBMS Vektordatenbankmanagementsystem 1–3, 5, 9–12, 14, 18–31, 33–40, 42, 44–46, XIII–XVII, *Glossar:* VDBMS

Literatur

Aumüller, M., Bernhardsson, E., & Faithfull, A. (2018, 15. Juli). ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. http://arxiv.org/pdf/1807.05614

Berghammer, R. (2014). *Mathematik für Informatiker: Grundlegende Begriffe und Strukturen*. Springer Vieweg. https://doi.org/10.1007/978-3-658-06288-0

Burgener, E., & Rydning, J. (2022, 1. Juli). High Data Growth and Modern Applications Drive New Storage Requirements in Digitally Transformed Enterprises (IDC Research, Hrsg.) [White Paper]. Verfügbar 23. Juli 2024 unter https://www.delltechnologies.com/asset/en-my/products/storage/industry-market/h19267-wp-idc-storage-reqs-digital-enterprise.pdf

Di Sarah, Yu, R., & Kapoor, A. (2021, 9. Oktober). Embed Everything: A Method for Efficiently Co-Embedding Multi-Modal Spaces [7 pages, 4 figures]. http://arxiv.org/pdf/2110.04599

Dilocker, E. (2023). Our Mad Journey of Building a Vector Database in Go. *FOSDEM VZW*. https://doi.org/10.5446/62048

Feurer, M., & Hutter, F. (2019). Hyperparameter Optimization. In F. Hutter, L. Kotthoff & J. Vanschoren (Hrsg.), *Automated Machine Learning* (S. 3–33). Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5_1

Girdhar, R., El-Nouby, A., Liu, Z., Singh, M., Alwala, K. V., Joulin, A., & Misra, I. (2023, 9. Mai). ImageBind: One Embedding Space To Bind Them All. http://arxiv.org/pdf/2305.05665

Goldhahn, D., Eckart, T., & Quasthoff, U. (2012). Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages. In N. Calzolari, K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, A. Moreno, J. Odijk & S. Piperidis (Hrsg.), *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)* (S. 759–765). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2012/pdf/327_Paper.pdf

Guo, R., Luan, X., Xiang, L., Yan, X., Yi, X., Luo, J., Cheng, Q., Xu, W., Luo, J., Liu, F., Cao, Z., Qiao, Y., Wang, T., Tang, B., & Xie, C. (2022, 28. Juni). Manu: A Cloud Native Vector Database Management System. http://arxiv.org/pdf/2206.13843

Hamilton, W. L., Leskovec, J., & Jurafsky, D. (2016, 30. Mai). Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change [Association for Computational Linguistics (ACL), 2016.]. http://arxiv.org/pdf/1605.09096

Hocking, J. G., & Young, G. S. (1988). *Topology* (Repr. d. Ausg. Reading, Mass., 1961). Dover Publ.

Hoppe, T., & Humm, B. (2020). *Semantische Suche: Grundlagen und Methoden semantischer Suche von Textdokumenten* [Hoppe, Thomas (VerfasserIn) Humm, Bernhard (VerfasserIn)]. Springer Vieweg. https://doi.org/10.1007/978-3-658-30427-0

Jaeckle, F., Faghri, F., Farhadi, A., Tuzel, O., & Pouransari, H. (2023, 8. März). FastFill: Efficient Compatible Model Update [To appear in The Eleventh International Conference on Learning Representations]. http://arxiv.org/pdf/2303.04766

Jeffrey Pennington, Richard Socher & Christopher D. Manning. (2014). GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. http://www.aclweb.org/anthology/D14-1162

Johnson, J., Douze, M., & Jégou, H. (2017, 28. Februar). Billion-scale similarity search with GPUs. http://arxiv.org/pdf/1702.08734

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (P. Norvig & S. J. Russell, Hrsg.; Second Edition, Pearson International Edition) [Jurafsky, Dan (VerfasserIn) Martin, James H. (VerfasserIn) Norvig, Peter (HerausgeberIn) Russell, Stuart J. (HerausgeberIn) Jurafsky, Dan (VerfasserIn) Martin, James H. (VerfasserIn) Norvig, Peter (HerausgeberIn) Russell, Stuart J. (HerausgeberIn)]. Prentice Hall Pearson Education International.

Kalay, Y. E. (1985). Criteria for relational database systems [PII: 0010448585901319]. *Computer-Aided Design*, *17*(6), 286. https://doi.org/10.1016/0010-4485(85)90131-9

Kenyon-Dean, K., Newell, E., & Cheung, J. C. K. (2020, 12. November). Deconstructing word embedding algorithms [EMNLP 2020, 6 pages. arXiv admin note: substantial text overlap with arXiv:1911.13280]. http://arxiv.org/pdf/2011.07013

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., & Lin, X. (2020). Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, *32*(8), 1475–1488. https://doi.org/10.1109/TKDE.2019.2909204

Li, Y., Yosinski, J., Clune, J., Lipson, H., & Hopcroft, J. (2015, 24. November). Convergent Learning: Do different neural networks learn the same representations? [Published as a conference paper at ICLR 2016]. http://arxiv.org/pdf/1511.07543

Liu, Y., Jun, E., Li, Q., & Heer, J. (2019). Latent Space Cartography: Visual Analysis of Vector Space Embeddings. *Computer Graphics Forum*, *38*(3), 67–78. https://doi.org/10.1111/cgf.13672

Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs [Journal Article]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(4), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

Maryamah, M., Irfani, M. M., Tri Raharjo, E. B., Rahmi, N. A., Ghani, M., & Raharjana, I. K. (2024). Chatbots in Academia: A Retrieval-Augmented Generation Approach for Improved Efficient Information Access. In S. Rasmequan, A. On-Ouen, J. Onpans & S. Srikamdee (Hrsg.). C. Lursinsap (Hrsg.), *KST 2024: The 16th International Conference on Knowledge and Smart Technology : February 28-March 2, 2024, Deevana Plaza Krabi Aonang, Thailand* (S. 259–264). IEEE. https://doi.org/10.1109/KST61284.2024.10499652

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, 16. Januar). Efficient Estimation of Word Representations in Vector Space. http://arxiv.org/pdf/1301.3781

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations [jun,]. In L. Vanderwende, H. Daumé III & K. Kirchhoff (Hrsg.), *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (S. 746–751). Association for Computational Linguistics. https://aclanthology.org/N13-1090

Morris, J. X., Kuleshov, V., Shmatikov, V., & Rush, A. M. (2023, 10. Oktober). Text Embeddings Reveal (Almost) As Much As Text [Accepted at EMNLP 2023]. http://arxiv.org/pdf/2310.06816

Pan, J. J., Wang, J., & Li, G. (2024). Survey of vector database management systems [PII: 864]. *The VLDB Journal*. https://doi.org/10.1007/s00778-024-00864-x

Qdrant Solutions GmbH. (2022). vector-db-benchmark: Framework for benchmarking vector search engines [Commit hash: ea53db4f1ac62babe6d6fd08c86f98cd9211abc2]. Verfügbar 28. August 2024 unter https://github.com/qdrant/vector-db-benchmark

Sahoo, S., Paul, N., Shah, A., Hornback, A., & Chava, S. (2023, 23. März). The Universal NFT Vector Database: A Scaleable Vector Database for NFT Similarity Matching. http://arxiv.org/pdf/2303.12998

Schmidt, J. W., & Brodie, M. L. (Hrsg.). (1983). *Relational Database Systems: Analysis and Comparison* [Schmidt, Joachim W. (editor.) Brodie, Michael L. (editor.)]. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-68847-8

Shen, Y., Xiong, Y., Xia, W., & Soatto, S. (2020, 26. März). Towards Backward-Compatible Representation Learning [Accepted to CVPR 2020 as oral, revised version]. http://arxiv.org/pdf/2003.11942

Singhal, A., & Google, I. (2001). Modern Information Retrieval: A Brief Overview [01]. *IEEE Data Engineering Bulletin*, *24*.

Spencer, N. H. (2014). *Essentials of Multivariate Data Analysis*. CRC Press. http://gbv.eblib.com/patron/FullRecord.aspx?p=1480667

Taipalus, T. (2024a). Database management system performance comparisons: A systematic literature review [PII: S0164121223002674]. *Journal of Systems and Software*, *208*, 111872. https://doi.org/10.1016/j.jss.2023.111872

Taipalus, T. (2024b). Vector database management systems: Fundamental concepts, use-cases, and current challenges [PII: S1389041724000093]. *Cognitive Systems Research*, *85*, 101216. https://doi.org/10.1016/j.cogsys.2024.101216

Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., Yu, K., Yuan, Y., Zou, Y., Long, J., Cai, Y., Li, Z., Zhang, Z., Mo, Y., Gu, J., … Xie, C. (2021). Milvus: A Purpose-Built Vector Data Management System. In G. Li (Hrsg.), *Proceedings of the 2021 International Conference on Management of Data* (S. 2614–2627). Association for Computing Machinery. https://doi.org/10.1145/3448016.3457550

Wang, Y. (2022, 25. Juni). Inverted Semantic-Index for Image Retrieval. http://arxiv.org/pdf/2206.12623v1

Windsor, B., & Choi, K. (2023, 26. März). Thistle: A Vector Database in Rust. http://arxiv.org/pdf/2303.16780

Abbildungsverzeichnis

1.1 Prognose für jährliche Datenmenge aus dem Global DataSphere-Bericht von IDC 3

2.1 Linguistische Nutzung der Vektorräume 8
2.2 Neuartige multimodale Fähigkeiten im Einbettungsraum von IMAGEBIND wie Cross-Modal Retrieval 9
2.3 Abbildung von IVF- und HNSW-Indexierungsalgorithmus 16
2.4 Suche nach Wert 12 in Skip-Liste mit Blindelement am Anker bei Laufzeit $O(\log n)$. 17

4.1 Architektur der Software zur Verwaltung von VDBMS 28
4.2 Architekturdiagramm der Infrastruktur 31

5.1 Vergleich des Durchsatzes von VDBMS auf einer AWS m6a.2xlarge Instanz 37
5.2 Vergleich der durchschnittlichen Latenz von VDBMS auf einer AWS m6a.2xlarge Instanz 38
5.3 Vergleich der Ladezeiten von VDBMS auf einer AWS m6a.2xlarge Instanz . 39
5.4 CPU-Last bei Ähnlichkeitssuche auf Server (Aufbau 2). 40
5.5 CPU-Last bei Upload und Indexierung auf Server (Aufbau 2). 41

B.1 Prototyp eines Empfehlungssystems für VDBMS. XIX
D.1 Prototyp eines Systems für nedrigschwelliges Experimentieren mit VDBMS: Bereitstellung, Management und Monitoring. XXI

Tabellenverzeichnis

2.1 Wort-Wort-Matrix mit Koexistenzen zwischen Wörtern des Wikipedia-Datensatzes der Leipzig Corpora Collection 6
2.2 Neue VDBMS(-Plugins) mit Erscheinungsdatum und Lizenztyp aus den letzten 5 Jahren 10
2.3 Suchindizes: Abwägung zwischen Geschwindigkeit, Genauigkeit und Speicherverbrauch 18

5.1 Vergleich der CPU- und Leistungsparameter von Azure- und AWS-Instanzen 35
5.2 Vergleich zwischen dem Referenzsystem und eigenen Aufbauten zu den Leistungsdaten von bis zu fünf VDBMS 36

A.1 Durchsatz in Anfragen pro Sekunde, durchschnittliche Latenz in Millisekunden und Hochladedauer des Datensatzes in Minuten für evaluierte VDBMS auf Aufbau 1 XIII
A.2 Durchsatz in Anfragen pro Sekunde, durchschnittliche Latenz in Millisekunden und Hochladedauer des Datensatzes in Minuten für evaluierte VDBMS auf Aufbau 2 XIV
A.3 Durchsatz in Anfragen pro Sekunde mit Parallelisierungsgrad 100 für evaluierte VDBMS auf Aufbau 1 XV
A.4 Durchschnittliche Latenz in Millisekunden ohne Parallelisierung für evaluierte VDBMS auf Aufbau 1 XVI
A.5 Hochladedauer des Datensatzes in Minuten mit Parallelisierungsgrad 16 für evaluierte VDBMS auf Aufbau 1 XVII

A Resultate der Experimente: Datentabellen

VDBMS	Präzision in %	Durchsatz EM[a]	Durchsatz FM[b]	Latenz EM[a]	Latenz FM[b]	Upload EM[a]	Upload FM[b]
Chroma	80	89,70[c]	n/a	10,94	n/a	62,01	n/a
	90	failed	n/a	failed	n/a	failed	n/a
	95	failed	n/a	failed	n/a	failed	n/a
Milvus	80	108,59	1640,049	8,64	5,25	5,81	2,13
	90	105,33	1157,5	12.02	6,11	11,13	3,70
	95	89,91	536,59	13,17	10,82	20,83	12,38
OpenSearch	80	79,10	n/a	42,34	n/a	5,83	n/a
	90	73,19	n/a	51,93	n/a	7,23	n/a
	95	64,91	n/a	58,36	n/a	10,77	n/a
pgvector	80	599,54	n/a	7,96	n/a	**5,37**	n/a
	90	313,21	n/a	17,25	n/a	9,65	n/a
	95	101,01	n/a	55,03	n/a	31,30	n/a
Qdrant	80	**1081,33**	2263,66	5,88	3,27	5,96	4,70
	90	**573,29**	1395,88	7,74	4,82	**5,96**	5,31
	95	**350,66**	872,95	**9,78**	7,11	**5,96**	9,52
Redis	80	469,27	912,038	**4,56**	2,11	18,84	17,80
	90	238,42	448,55	**6,80**	3,35	26,92	24,80
	95	134,23	257,66	10,26	5,17	74,67	61,93
⌀	90	**299,53**	**1000,64**	**19,15**	**4,76**	**12,18**	**11,30**

[a] Eigenmessung.

[b] Fremdmessung, Stand 15. Juni 2024 Qdrant Solutions GmbH, 2022.

[c] ohne Parallelisierung.

Tabelle A.1: Durchsatz in Anfragen pro Sekunde ($p = 100$), durchschnittliche Latenz in Millisekunden ($p = 1$) und Hochladedauer des Datensatzes in Minuten ($p = 16$) für evaluierte VDBMS auf Aufbau 1, wobei p die Anzahl der parallelen Prozesse angibt. Der fettgedruckte Wert stellt den besten eigen-gemessenen Wert pro Kategorie dar. Der Eintrag *failed* drückt aus, dass die Präzision nicht erzielt werden konnte.

VDBMS	**Präzision** in %	**Durchsatz** EM[a]	**Latenz** EM[a]	**Upload** EM[a]
Chroma	80	214,13[b]	4,59	30,73
	90	failed	failed	failed
	95	failed	failed	failed
Milvus	80	359,79	2,64	2,43
	90	267,83	3,59	2,43
	95	192,19	7,88	8,82
OpenSearch	80	191,93	15,74	2,84
	90	181,26	18,17	4,58
	95	failed	failed	failed
pgvector	80	900,70	4,91	4,62
	90	462,99	10,92	8,77
	95	146,35	35,68	32,36
Qdrant	80	**2966,55**	**2,19**	**2,41**
	90	**1977,68**	**2,95**	**2,41**
	95	**1156,28**	**3,79**	**2,41**
Redis	80	1111,42	2,55	8,43
	90	538,50	3,47	11,77
	95	303,93	5,00	29,68
Elasticsearch	80	263,81	3,89	15,88
	90	250,50	5,94	21,63
	95	232,38	8,82	57,03
Weaviate	80	2030,68	3,74	3,79
	90	1094,27	6,52	5,26
	95	612,22	10,34	13,38
⌀	90	**681,86**	**7,37**	**8,12**

[a] Eigenmessung.

[b] ohne Parallelisierung

Tabelle A.2: Durchsatz in Anfragen pro Sekunde ($p = 100$), durchschnittliche Latenz in Millisekunden ($p = 1$) und Hochladedauer des Datensatzes in Minuten ($p = 16$) für evaluierte VDBMS auf Aufbau 2, wobei p die Anzahl der parallelen Prozesse angibt. Der fettgedruckte Wert stellt den besten eigen-gemessenen Wert pro Kategorie dar. Der Eintrag *failed* drückt aus, dass die Präzision nicht erzielt werden konnte.

In den Tabellen A.3, A.4 und A.5 werden die Messdaten zu Durchsatz, durchschnittlicher Latenz und Hochladedauer des Datensatzes für evaluierte VDBMS auf Aufbau 1 samt Abweichung zum Referenzsystem und zum arithmetischen Mittel der Eigenmessung dargestellt.

VDBMS	**Präzision** in %	**Durchsatz** EM[a]	**Durchsatz** FM[b]	|Abweichung EM-FM|	Abweichung ⌀
Chroma	80	89,70[c]	n/a	n/a	-81 %
	90	failed	n/a	n/a	n/a
	95	failed	n/a	n/a	n/a
Milvus	80	108,59	1640,049	n/a	(18 % via FM)
	90	105,33	1157,5	n/a	(41 % via FM)
	95	89,91	536,59	n/a	(3 % via FM)
OpenSearch	80	79,10	n/a	n/a	-83 %
	90	73,19	n/a	n/a	-76 %
	95	64,91	n/a	n/a	-60 %
pgvector	80	599,54	n/a	n/a	29 %
	90	313,21	n/a	n/a	5 %
	95	101,01	n/a	n/a	-38 %
Qdrant	80	**1081,33**	2263,66	52 %	133 %
	90	**573,29**	1395,88	59 %	91 %
	95	**350,66**	872,95	60 %	116 %
Redis	80	469,27	912,038	49 %	1 %
	90	238,42	448,55	47 %	-20 %
	95	134,23	257,66	48 %	-18 %
⌀	90	**299,53**	**1000,64**	**52,9 %**	0 %

[a] Eigenmessung.

[b] Fremdmessung, Stand 15. Juni 2024 Qdrant Solutions GmbH, 2022.

[c] ohne Parallelisierung.

Tabelle A.3: Durchsatz in Anfragen pro Sekunde mit Parallelisierungsgrad 100 für evaluierte VDBMS auf Aufbau 1. Der fettgedruckte Wert stellt den besten eigen-gemessenen Wert pro Kategorie dar. Der Eintrag *failed* drückt aus, dass die Präzision nicht erzielt werden konnte.

VDBMS	**Präzision** in %	**Latenz** EM[a]	**Latenz** FM[b]	|Abweichung EM-FM|	Abweichung ⌀
Chroma	80	10,94	n/a	n/a	-18 %
	90	failed	n/a	n/a	n/a
	95	failed	n/a	n/a	n/a
Milvus	80	8,64	5,25	65 %	-35 %
	90	12.02	6,11	97 %	-37 %
	95	13,17	10,82	22 %	-55 %
OpenSearch	80	42,34	n/a	n/a	216 %
	90	51,93	n/a	n/a	171 %
	95	58,36	n/a	n/a	99 %
pgvector	80	7,96	n/a	n/a	-41 %
	90	17,25	n/a	n/a	-10 %
	95	55,03	n/a	n/a	88 %
Qdrant	80	5,88	3,27	80 %	-56 %
	90	7,74	4,82	61 %	-60 %
	95	**9,78**	7,11	38 %	-67 %
Redis	80	**4,56**	2,11	116 %	-66 %
	90	**6,80**	3,35	103 %	-64 %
	95	10,26	5,17	98 %	-65 %
⌀	90	**19,15**	**4,76**	**86,8 %**	0 %

[a] Eigenmessung.

[b] Fremdmessung, Stand 15. Juni 2024 Qdrant Solutions GmbH, 2022.

Tabelle A.4: Durchschnittliche Latenz in Millisekunden ohne Parallelisierung für evaluierte VDBMS auf Aufbau 1. Der fettgedruckte Wert stellt den besten eigen-gemessenen Wert pro Kategorie dar. Der Eintrag *failed* drückt aus, dass die Präzision nicht erzielt werden konnte.

VDBMS	**Präzision** in %	**Upload** EM[a]	**Upload** FM[b]	|Abweichung EM-FM|	Abweichung ⌀
Chroma	80	62,01	n/a	n/a	258 %
	90	failed	n/a	n/a	n/a
	95	failed	n/a	n/a	n/a
Milvus	80	5,81	2,13	173 %	-66 %
	90	11,13	3,70	201 %	-9 %
	95	20,83	12,38	68 %	-27 %
OpenSearch	80	5,83	n/a	n/a	-66 %
	90	7,23	n/a	n/a	-41 %
	95	10,77	n/a	n/a	-62 %
pgvector	80	**5,37**	n/a	n/a	-69 %
	90	9,65	n/a	n/a	-21 %
	95	31,30	n/a	n/a	9 %
Qdrant	80	5,96	4,70	27 %	-66 %
	90	**5,96**	5,31	12 %	-51 %
	95	**5,96**	9,52	37 %	-79 %
Redis	80	18,84	17,80	6 %	9 %
	90	26,92	24,80	8 %	121 %
	95	74,67	61,93	21 %	160 %
⌀	90	**12,18**	**11,30**	**73,80 %**	0 %

[a] Eigenmessung.

[b] Fremdmessung, Stand 15. Juni 2024 Qdrant Solutions GmbH, 2022.

Tabelle A.5: Hochladedauer des Datensatzes in Minuten mit Parallelisierungsgrad 16 für evaluierte VDBMS auf Aufbau 1. Der fettgedruckte Wert stellt den besten eigen-gemessenen Wert pro Kategorie dar. Der Eintrag *failed* drückt aus, dass die Präzision nicht erzielt werden konnte.

B Beispiel: VDBMS-Empfehlungssystem

(a) Seite 1.

(b) Seite 2.

	Name	Score in %
0	Qdrant	100
1	Milvus	100
2	Weaviate	91
7	ChromaDB	91
3	Opensearch	83
4	Redis	83
5	PGVector	75
6	Elasticsearch	58
8	LanceDB	58
9	Pinecone	50

Abbildung B.1: Prototyp eines Empfehlungssystems für VDBMS.

C Arbeit mit Leipzig Corpora Collection

Abruf der Leipziger Korpora-Sammlung (Goldhahn et al., 2012) unter Nutzung eines Import-Skripts. Die Datei enthält einerseits Informationen über die Häufigkeit des Auftretens zweier Wörter in unmittelbarer Nachbarschaft im Korpus und andererseits über die Signifikanz dessen auf Basis von Log-Likelihood.

Code-Beispiel C.1: Starten von mysql-Umgebung.

```
// Lokale Installation von mysql.
wget https://downloads.wortschatz-leipzig.de/corpora/
    deu_wikipedia_2021_1M.tar.gz
// Entpacken der Datei.
mysql --local-infile=1 -u root -p
// Es erscheint ein neuer Prompt: mysql>
```

Code-Beispiel C.2: Extraktion von beispielhaften dünnen Zählvektoren.

```
SOURCE ~/Downloads/deu_wikipedia_2021_1M/deu_wikipedia_2021_1M/
   deu_wikipedia_2021_1M-import.sql;
USE ‘deu_wikipedia_2021_1M‘;
SELECT co_s.freq, w1.word as word1, w2.word as word2
FROM co_s
LEFT JOIN words w1 ON co_s.w1_id=w1.w_id
LEFT JOIN words w2 ON co_s.w2_id=w2.w_id
WHERE w1.word IN ("Buch", "Schule", "deutschen", "Menschen", "Glaube")
   AND w2.word IN ("Jahr", "Welt", "Klassiker", "Tatort")
ORDER BY co_s.freq
DESC;
```

D Ausblick: GUI für Bereitstellung von VDBMS

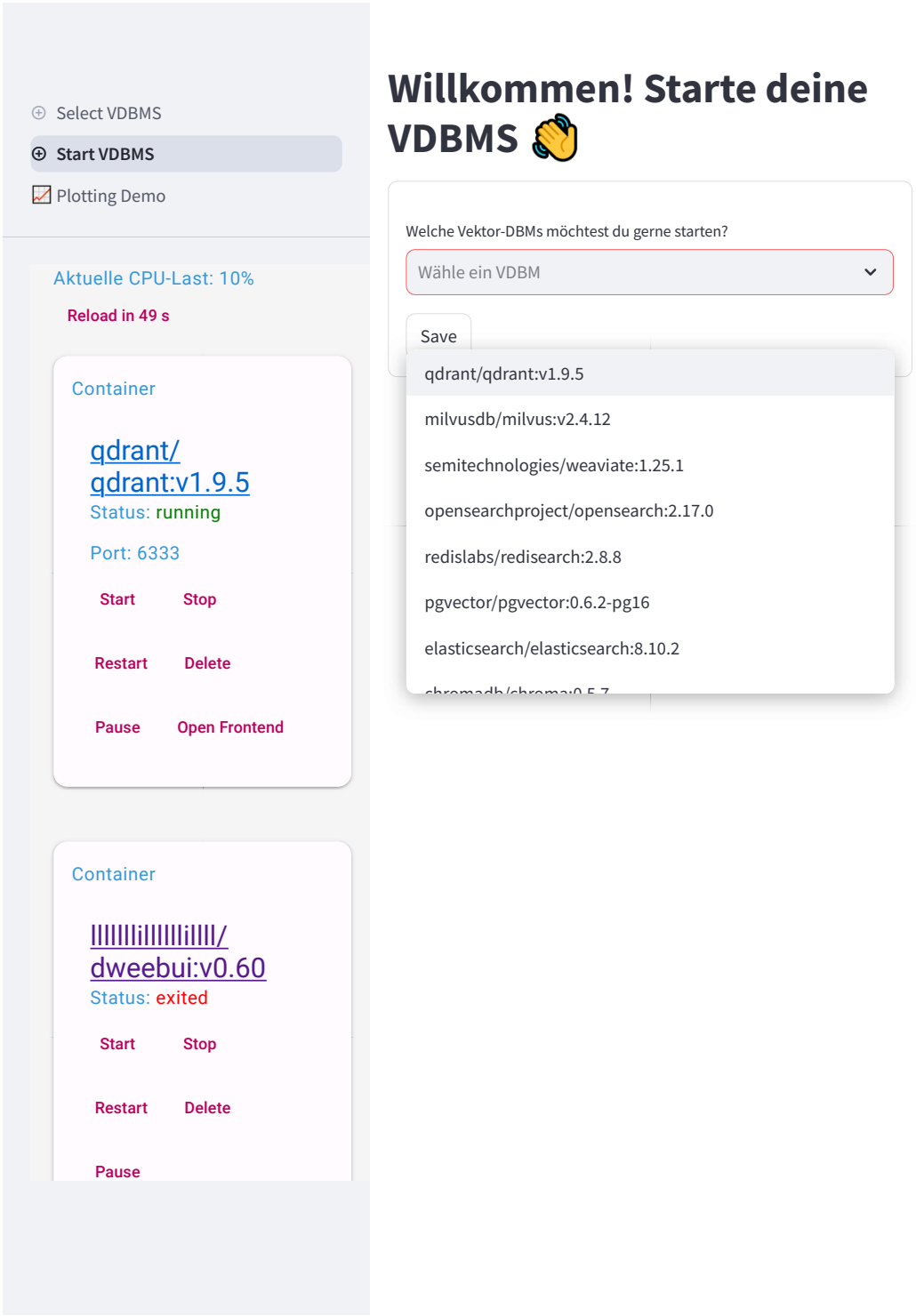


Abbildung D.1: Prototyp eines Systems für nedrigschwelliges Experimentieren mit VDBMS: Bereitstellung, Management und Monitoring.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Leipzig, 22. Oktober 2024

Unterschrift