T2000

Jan Herrmann

4. Dezember 2025

Inhaltsverzeichnis

Abkürzungsverzeichnis

KI künstliche Intelligenz. 3

ML Machine learning. 4

1 Einleitung

1.1 Motivation

1.2 Problemstellung

1.3 Zielsetzung

DEVELOP Test

2 Theoretischer Hintergrund

2.1 Künstliche Intelligenz

Bevor eine präzise Definition von künstlicher Intelligenz möglich ist, muss geklärt werden, welches Ziel ein intelligentes System verfolgen soll. Russell und Norvig zeigen, dass gängige KI-Definitionen in der wissenschaftlichen Literatur entlang zweier zentraler Dimensionen variieren (vgl. [1], Kap. 1.1):

- **Mensch vs. Rationalität** Soll ein System wie ein Mensch denken oder handeln, oder soll es unabhängig vom Menschen ideal rational agieren?
- **Denken vs. Handeln** Soll Intelligenz anhand interner Denkprozesse oder anhand des beobachtbaren Verhaltens beurteilt werden?

Aus diesen beiden Dimensionen ergeben sich vier grundlegende Perspektiven auf KI, die unterschiedliche historische Forschungsrichtungen geprägt haben. Eine Übersicht dieser Einordnung zeigt Tabelle 1

Abgrenzung von KI und ML

Künstliche Intelligenz (KI) umfasst alle Verfahren, die darauf abzielen, intelligentes Verhalten technisch zu realisieren. Dazu gehören sowohl symbolische Ansätze wie Wissensrepräsentation und logisches Schließen als auch datengetriebene Methoden zur Wahrnehmung oder Sprachverarbeitung (vgl. [1], Kap. 1.1).

Kategorie	Beschreibung
Systeme, die wie Menschen denken	Fokus auf Nachbildung menschlicher Denkprozesse, z.B. durch kognitive Modelle oder psychologische Theorien.
Systeme, die wie Menschen handeln	Intelligenz wird anhand menschlich ähnlichen Verhaltens beurteilt, unabhängig vom zugrunde liegenden Denkprozess.
Systeme, die rational denken	Fokus auf logische Schlussfolgerungen und formale Wissensrepräsentation.
Systeme, die rational handeln	Intelligente Agenten handeln zielgerichtet und optimal in ihrer Umgebung.

Tabelle 1: Eigene Darstellung in Anlehnung an [1], Kap. 1.1

Maschinelles Lernen (ML) stellt ein klar abgegrenztes Teilgebiet der KI dar. Russell und Norvig beschreiben es als das Teilfeld der künstlichen Intelligenz, das sich mit Programmen befasst, die aus Erfahrung lernen (vgl. [1], Einleitung zu Teil VI).

Während KI somit als Oberbegriff sämtliche Methoden intelligenter Problemlösung einschließt, konzentriert sich ML ausschließlich auf Verfahren, die Wissen nicht explizit vorgegeben bekommen, sondern selbstständig aus Daten oder Erfahrungen erschließen. ML bildet damit die Grundlage für viele moderne KI-Anwendungen, insbesondere für datengetriebene Systeme wie neuronale Netze oder Large Language Models.

2.2 Vektorstore

Vektor Stores bilden die Grundlage für moderne KI-Anwendungen, die auf semantischer Ähnlichkeitssuche basieren. Sie dienen der Speicherung von Embeddings, also numerischen Repräsentationen von Text- oder Produktdaten, und ermöglichen effiziente Nearest-Neighbor-Abfragen. Damit stellen sie einen zentralen Baustein für Retrieval-Augmented Generation (RAG) und für Systeme dar, die kontextbezogene Informationen an große Sprachmodelle übergeben.

Da der Begriff „Vektorstore“ in der Literatur häufig als Sammelbezeichnung für leichtgewichtige, embeddingbasierte Speichersysteme verwendet wird, ist eine Abgrenzung zu vollwertigen Vektordatenbanken notwendig. Letztere integrieren zusätzliche Mechanismen wie Sharding, interne Partitionierung, Caching oder Replikation und unterscheiden sich damit deutlich in ihrer Komplexität und Skalierbarkeit.

Im Folgenden werden zunächst die grundlegenden Konzepte erläutert, bevor die für Vektordatenbanken zentralen Speicherverfahren systematisch dargestellt werden.

2.2.1 Abgrenzung Vektor-Store und Vektor-Datenbank

Der Begriff *Vektor-Store* wird in der Literatur nicht einheitlich verwendet und dient häufig als Sammelbezeichnung für leichtgewichtige Systeme, die Embeddings speichern und eine grundlegende semantische Ähnlichkeitssuche bereitstellen. Solche Systeme konzentrieren sich in der Regel auf das Einfügen von Embeddings und die Ausführung von k-nächste-Nachbarn-Abfragen, ohne jedoch erweiterte Datenbankfunktionen bereitzustellen.

Unter einer *Vektordatenbank* werden hingegen vollwertige Datenbanksysteme verstanden, die neben der Speicherung und Suche von Embeddings zusätzliche Verwaltungs- und Infrastrukturmechanismen integrieren. Dazu zählen insbesondere Sharding, interne Partitionierung, Caching, Replikation, Indexstrukturen für Approximate Nearest Neighbor (ANN) sowie Metadatenverwaltung und Konsistenzmodelle. Diese Systeme sind auf hohe Skalierbarkeit, Robustheit und Performanz im produktiven Einsatz ausgelegt.

Für die vorliegende Arbeit wird daher folgende Unterscheidung getroffen:

- **Vektor-Store:** leichtgewichtiges System zur Speicherung von Embeddings und zur Durchführung semantischer Ähnlichkeitssuchen.
- **Vektordatenbank:** vollständiges Datenbankmanagementsystem mit skalierbaren Speicher- und Verwaltungsmechanismen.

Diese definitorische Abgrenzung dient der Klarheit und legt die einheitliche Verwendung der Begriffe im weiteren Verlauf der Arbeit fest.

2.2.2 Speicherverfahren für Vektordatenbanken

Die im Folgenden beschriebenen Mechanismen stellen zentrale Bestandteile moderner, skalierbarer Vektordatenbanksysteme dar. Sie werden im produktiven Umfeld zur Optimierung von Leistung, Verfügbarkeit und Robustheit eingesetzt. Für den im Rahmen dieser Arbeit entwickelten Prototypen spielen diese Konzepte jedoch keine operative Rolle, da ein leichtgewichtiges Single-Node-System ohne verteilte Architektur eingesetzt wird. Die Ausführungen dienen daher der theoretischen Einordnung und sollen den technologischen Kontext verdeutlichen, in dem sich Vektordatenbanken typischerweise bewegen.

horizontale Datenpartitionierung über mehrere Maschinen (auch Sharding genannt) bezeichnet ein Verfahren, bei dem eine Datenbank in mehrere logisch getrennte und auf verschiedene physische Knoten verteilte Teilmengen („Shards“) aufgeteilt wird. Durch diese Aufteilung wird das Gesamtdatenset in kleinere, handhabbarere Einheiten zerlegt, was Skalierbarkeit, Lastverteilung und die Verwaltung großer Datenmengen erleichtert (vgl. [2] S. 3).

horizontale Datenpartitionierung innerhalb einer Maschine Die horizontale Datenpartitionierung innerhalb einer Maschine bezeichnet die Aufteilung der in einer einzelnen Datenbankinstanz gespeicherten Vektordaten in mehrere logisch getrennte Teilmengen („Partitionen“). Im Unterschied zum Sharding, das Daten über mehrere physische Knoten verteilt, erfolgt diese Form der Partitionierung ausschließlich innerhalb eines Systems. Ziel ist es, lokale Abfragen effizienter auszuführen, Speicherressourcen besser auszunutzen und parallele Verarbeitung zu ermöglichen. Partitionen können beispielsweise über Wertebereiche (Range-Partitioning), vordefinierte Kategorien (List-Partitioning) oder Hash-Verfahren gebildet werden. Durch diese interne Strukturierung müssen Suchanfragen nur gegen relevante Partitionen ausgeführt werden, was insbesondere bei großen Einbettungsräumen die Latenz der semantischen Ähnlichkeitssuche reduziert (vgl. [2], S. 3f.).

In modernen Vektordatenbanken wird Partitionierung häufig mit Sharding kombiniert:

Während Sharding die Skalierung über mehrere Knoten ermöglicht, optimiert die interne Partitionierung die Datenorganisation innerhalb jedes Shards. Beide Verfahren bilden damit die Grundlage für performante Retrieval-Systeme in Vektor Stores und Vektordatenbanken.

Caching-Mechanismen in Vektordatenbanken beschreiben das Zwischenspeichern häufig oder kürzlich genutzter Daten in besonders schnellen Speichermedien (z.B. RAM), um Zugriffszeiten zu reduzieren und die Last auf der eigentlichen Datenbank zu verringern. Für Vektordatenbanken ist Caching ein zentraler Mechanismus, da semantische Ähnlichkeitssuchen typischerweise rechenintensiv sind und von wiederholten Abfragen profitieren (vgl. [2], S. 4f.).

Im Gegensatz zu klassischen Schlüssel-Wert-Caches (etwa Redis) ist das Caching in VDBs herausfordernder, da Embeddings hochdimensionale Vektoren darstellen und identische Anfragen selten auftreten. Daher kommen allgemeine Cache-Strategien zum Einsatz, die unabhängig vom genauen Vektorinhalt arbeiten, da der Vergleich hochdimensionaler Vektoren zur Bestimmung von Cache-Schlüsseln ungeeignet ist.

Zu den gängigen Verfahren zählen:

- **First-In First-Out (FIFO):** entfernt das älteste Element im Cache; einfach, aber ohne Berücksichtigung der Zugriffshäufigkeit.
- **Least Recently Used (LRU):** löscht den am längsten nicht genutzten Eintrag; gut geeignet für Arbeitslasten mit zeitlicher Lokalität.
- **Most Recently Used (MRU):** entfernt das zuletzt genutzte Element; sinnvoll bei einmaligen Zugriffsmustern.
- **Least Frequently Used (LFU):** bevorzugt das Entfernen seltener genutzter Einträge; vorteilhaft bei stabilen Zugriffshäufigkeiten.

Einige Systeme nutzen zudem **partitioniertes Caching**, bei dem Vektordaten in Gruppen (z.B. nach Kategorien oder Zugriffsmustern) getrennt gecacht werden, um

Ressourcen gezielt zu optimieren. Insgesamt trägt Caching wesentlich zur Reduktion der Abfragelatenz und zur Stabilisierung der Systemlast bei, insbesondere bei wiederkehrenden Ähnlichkeitsanfragen.

Replikation bezeichnet das Anlegen und Verteilen mehrerer Kopien von Vektordaten auf unterschiedliche Knoten eines verteilten Systems, um Ausfallsicherheit, Verfügbarkeit und Leselastverteilung zu erhöhen. Während sie für die semantische Ähnlichkeitssuche nicht unmittelbar leistungsbestimmend ist, stellt sie einen zentralen Mechanismus für die betriebliche Robustheit moderner Vektordatenbanken dar. In der Literatur werden drei grundlegende Replikationsstrategien unterschieden (vgl. [2], S. 5f.).

Leader-Follower-Replikation Bei der Leader-Follower-Replikation übernimmt ein einzelner Knoten die Rolle des Leaders und verarbeitet sämtliche Schreiboperationen. Die erzeugten Updates werden anschließend an mehrere Follower-Knoten repliziert, die ausschließlich Leseanfragen bearbeiten (vgl. [2], S. 5f.).

Dieses Modell ermöglicht eine klare Konsistenzsemantik, da alle Änderungen zentral koordiniert werden. Gleichzeitig entsteht jedoch ein Single Point of Failure, was eine zusätzliche Failover-Mechanik erforderlich macht.

Multi-Leader-Replikation Mehrere Knoten können parallel Schreiboperationen entgegennehmen. Die dabei entstehenden Aktualisierungen werden zwischen den beteiligten Knoten ausgetauscht. Dieses Modell erhöht die Schreibkapazität, erfordert jedoch Strategien zur Auflösung konkurrierender Updates (vgl. [2], S. 5f.).

Leaderless-Replikation Leaderless-Replikation verzichtet vollständig auf die Unterscheidung zwischen Leader- und Follower-Knoten. Jeder Knoten kann Lese- und Schreiboperationen ausführen, wodurch weder zentrale Engpässe noch einzelne Ausfallpunkte entstehen. Konsistenz wird über Quorum-Modelle oder koordinationsbasierte Protokolle sichergestellt. Dieses Modell bietet hohe Fehlertoleranz, bringt aber ggf. erhöhte Komplexität hinsichtlich Konsistenz- und Konfliktbehandlung mit sich (vgl. [2], S. 5f.).

Insgesamt trägt Replikation wesentlich zur Robustheit und Verfügbarkeit verteilter Vektordatenbanksysteme bei, steht jedoch weniger im direkten Zentrum der Ähnlichkeitssuche als vielmehr ihrer infrastrukturellen Betriebsstabilität.

6.4 Ergebnis

6.5 Diskussion

LLM ohne RAG halluziniert → schlecht für PIM-Daten (Faktentreue).

Prompt mit Rollenbeschreibung („Du bist ein Marketing-Experte…“) besser als generischer Prompt.

Techniktexte brauchen präzisere Struktur -> Template hilft.

Marketingtexte profitieren von höherer Kreativität → Temperatur-Einstellungen diskutieren.

Literatur

[1] S. J. Russell und P. Norvig, *Artificial intelligence: a modern approach* (Prentice Hall series in artificial intelligence). Upper Saddle River: Prentice Hall, 1995, 932 S., ISBN: 978-0-13-103805-9 978-0-13-360124-4.

[2] L. Ma u. a., *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge*, 16. Juni 2025. DOI: 10.48550/arXiv.2310.11703. arXiv: 2310.11703[cs]. besucht am 3. Dez. 2025. Adresse: http://arxiv.org/abs/2310.11703.