Semantische Suche in DEINEN Texten

Kapitel online

Lade eigene Texte oder eine Beispiel-FAQ, verwandle jeden Satz im Browser in einen 512er-Vektor und lass die Bedeutungs-Suche live gegen die Stichwort-Suche antreten — inklusive der ehrlichen Antwort, warum das Modell auf Deutsch schwächelt.

Interaktiv · Kapitel 49 von 50 · ca. 25 Min. · 300 XP

Einfach gesagt: Du wirfst eigene Texte hinein, jeder Satz wird zu einer Zahlenreihe — und die Suche findet Treffer nach Bedeutung, selbst ohne ein gemeinsames Wort.

In „Embeddings in der Praxis“ hast du gesehen, *warum* die Suche „Fahrzeug“ findet, wenn du „Auto“ tippst — und die Kosinus-Ähnlichkeit einmal von Hand nachgerechnet. Das war die Theorie an einem vorbereiteten Handbuch.

Heute wird daraus ein **Werkzeug für DEINE Texte**. Firmen-FAQ, Besprechungsnotizen, Arbeitsanweisungen, Rezeptsammlung — du wirfst sie hier hinein, jeder Satz wird in deinem Browser zu einem **512-Zahlen-Vektor**, und dann lassen wir zwei Suchmaschinen gegeneinander antreten: die **Stichwort-Suche**, die Buchstaben vergleicht, gegen die **Bedeutungs-Suche**, die Richtungen im Raum vergleicht.

Was du hier lernst

- Wie aus einer Textdatei ein **durchsuchbarer Vektor-Index** wird — in vier Klicks
- Was „Indexieren“ wirklich heißt: gespeichert werden **Zahlen, keine Wörter**
- Wo die Stichwort-Suche *null* Treffer liefert und die Bedeutungs-Suche trotzdem das Richtige findet
- Wie dein Suchsatz als **[512]-Tensor** aussieht — die ersten acht Zahlen live
- Und die unbequeme Wahrheit: **warum genau dieses Modell auf Deutsch versagt** — mit dem Beweis

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/470_suche_in_deinen_texten ausprobieren.

Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Deine Texte bleiben auf deinem Gerät

Gleich lädst du womöglich eine echte Firmen-FAQ oder deine privaten Notizen hoch — deshalb vorweg, in aller Klarheit: **Deine Texte verlassen dieses Gerät nicht.** Es gibt in diesem Kapitel keine einzige Zeile Code, die einen Text an einen Server schickt.

- Die Datei wird per `FileReader` direkt im Arbeitsspeicher gelesen — **kein Upload**.
- Das Embedding-Modell (28 MB) liegt **auf diesem Server** und rechnet danach in **deinem** Browser, auf deinem Prozessor bzw. deiner Grafikkarte.
- Nichts wird gespeichert. Tab schließen = alles weg, Vektoren inklusive.

Und genau das ist der Grund, warum dieses Thema in Firmen so heiß ist. Wer seine Wissensdatenbank bei einem Cloud-Suchdienst indexieren lässt, schiebt jedes Dokument — Verträge, Personalakten, Konstruktionsunterlagen — durch fremde Rechenzentren. Ein Embedding-Modell, das lokal läuft, löst das Problem an der Wurzel: Die Vektoren entstehen dort, wo die Dokumente ohnehin liegen. Mehr dazu in „KI auf dem eigenen Rechner“.

Schritt 1: Texte hereinholen

Am Gerät ausprobieren: herrki.de/470_suche_in_deinen_texten#abschnitt-3

Direkt zum Modul

Die Regel ist simpel: **eine Zeile = ein Eintrag**. Jeder Eintrag bekommt gleich seinen eigenen Vektor und wird damit einzeln auffindbar. Drei Wege führen hinein — such dir einen aus.

🎁 Der schnellste Weg: die Beispiel-FAQ

24 Sätze aus dem (erfundenen) Intranet der **Nordwind GmbH** — IT, Fuhrpark, Urlaub, Kantine. Bewusst so geschrieben, dass die richtige Antwort oft *kein einziges Wort* mit der Frage teilt. Es gibt sie zweimal, wortgleich übersetzt:

🇬🇧 Beispiel-FAQ laden (englisch) | 🇩🇪 Dieselbe FAQ auf Deutsch

Fang mit der englischen an. Klingt komisch auf einer deutschen Lernplattform — ist aber der ehrlichste Einstieg: Unser Modell wurde fast nur auf Englisch trainiert und zeigt dort, was Bedeutungs-Suche wirklich kann. Die deutsche Fassung ist danach dein **Gegen-Experiment**, und weiter unten zerlegen wir gemeinsam, woran es scheitert. Beide Dateien enthalten Satz für Satz dasselbe.

✍️ Oder: selbst tippen bzw. hineinkopieren

Ein Absatz oder Satz pro Zeile

```
Die Kaffeemaschine wird freitags entkalkt.
Ersatzteile bestellt die Werkstatt über Lieferant Meyer.
Der Serverraum darf nur zu zweit betreten werden.
```

Übernehmen

📄 Oder: eine Datei ablegen

Textdateien `.txt`, `.md` oder `.csv` — bei CSV nehmen wir automatisch das längste Feld je Zeile, das ist fast immer der eigentliche Satz.

Datei hierher ziehen oder klicken zum Auswählen

.txt · .md · .csv — bleibt auf deinem Gerät

Interaktives Modul – am Gerät ausprobieren

Schritt 2: Das Modell holen und alles vektorisieren

Am Gerät ausprobieren: herrki.de/470_suche_in_deinen_texten#abschnitt-4

Direkt zum Modul

Jetzt kommt das schwere Gerät: der **Universal Sentence Encoder**. Er frisst einen Satz und spuckt **512 Zahlen** aus — die Koordinaten des Satzes im Latent Space (Kapitel 310). Mit **28 MB** ist er das größte Modell dieser Plattform, deshalb laden wir ihn erst auf deinen Klick. Einmal geladen, bleibt er für diese Seite im Speicher.

📦 Modell laden (28 MB — unser größtes, einmalig)

Interaktives Modul – am Gerät ausprobieren

Schritt 3: Das Duell — Buchstaben gegen Bedeutung

Am Gerät ausprobieren: herrki.de/470_suche_in_deinen_texten#abschnitt-5

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

> ⏳ Das Duell schaltet sich frei, sobald Einträge geladen *und* vektorisiert sind — also nach Schritt 1 und Schritt 2 oben.

Runter zum Tensor: was da eigentlich verglichen wird

Am Gerät ausprobieren: herrki.de/470_suche_in_deinen_texten#abschnitt-6

Direkt zum Modul

Deine letzte Suchanfrage ist im Modell kein Satz mehr, sondern eine Zahlenreihe der Form `[512]`. Hier sind ihre **ersten acht** Zahlen — die übrigen 504 sehen genauso unspektakulär aus:

```
// Noch keine Suche gemacht – stell oben eine Frage, dann füllt sich diese Box.
```

Und die ganze Wissensdatenbank ist eine Matrix `[n, 512]`: eine Zeile pro Eintrag. Das Modell verarbeitet die Einträge deshalb auch **stapelweise** — acht Sätze gleichzeitig hinein, ein `[8, 512]`-Block heraus. Genau das siehst du beim Fortschrittsbalken passieren.

Die Suche selbst ist dann nur noch **eine** Formel, für jede Zeile einmal ausgerechnet — die **Kosinus-Ähnlichkeit**:

```
// Zähler: Skalarprodukt. Nenner: die beiden Vektorlängen – sie normieren die Satzlänge weg.
ähnlichkeit(f, s) = (f₁·s₁ + f₂·s₂ + … + f₅₁₂·s₅₁₂) / (‖f‖ · ‖s‖)

// 1.0 = gleiche Richtung (gleiche Bedeutung) · 0.0 = nichts miteinander zu tun
```

Das *an konkreten Zahlen* von Hand nachgerechnet hast du schon — in Kapitel 330, auf drei Achsen zusammengestrichen. Hier läuft dieselbe Rechnung, nur eben 512-mal pro Eintrag. Für 24 Einträge sind das rund 12.000 Multiplikationen pro Suche: der Browser erledigt das schneller, als du die Enter-Taste loslässt. Und weil es *nur* Multiplizieren und Addieren ist (Kapitel 060), skaliert dasselbe Verfahren auf Millionen Dokumente.

Interaktives Modul – am Gerät ausprobieren

Die unbequeme Wahrheit: warum Deutsch hier scheitert

Wenn du beide Beispiel-FAQs durchprobiert hast, ist dir das Gefälle aufgefallen. Das ist **kein Fehler in diesem Kapitel** — das ist die ehrliche Eigenschaft des Modells, und sie lässt sich messen. Wir haben zwölf Frage-Antwort-Paare gebaut, einmal auf Englisch und Satz für Satz identisch auf Deutsch, und gezählt, wie oft die richtige Antwort auf Platz 1 landet:

Sprache	Richtige Antwort auf Platz 1	Urteil
Englisch	9 von 12	brauchbar — genau das erlebst du oben
Deutsch	3 von 12	kaum besser als Würfeln (Zufall wäre 1 von 12)

Und jetzt die Ursache — sie sitzt tiefer, als man denkt: im Tokenizer. Bevor ein Satz zum Vektor wird, zerlegt ihn das Modell in bekannte Wortstücke (Kapitel 040). Dieses Wortstück-Verzeichnis wurde fast nur aus englischem Text gebaut. Was das mit deutschen Wörtern macht:

```
// So zerlegt das Modell die Wörter wirklich (nachgemessen an genau diesem vocab.json):
"password" → _password // 1 Stück – das Modell kennt das Wort
"Passwort" → _Pass w or t // 4 Krümel
"Auto" → _Auto // 1 Stück (steht so auch im Englischen)
"Dienstwagen" → _Di ens t wa gen // 5 Krümel
"Urlaub" → _U r la ub // 4 Krümel
"Kantine" → _K ant ine // 3 Krümel
```

„Dienstwagen“ kommt beim Modell nicht als Wort an, sondern als `Di-ens-t-wa-gen` — Silbenmüll ohne Bedeutung. Deshalb landen *alle* deutschen Sätze fast am selben Punkt im Raum: Ihre Ähnlichkeiten liegen dann bei 70–90 %, aber die Reihenfolge dahinter ist Rauschen. Auf Englisch liegen dieselben Werte bei 20–50 % und **trennen sauber**. Merke: hohe Ähnlichkeitswerte sind kein Gütesiegel — es kommt auf den *Abstand zum zweitbesten* an.

Was man in der Praxis tut: ein **mehrsprachiges** Embedding-Modell nehmen. Die gibt es (auch quelloffen und lokal lauffähig), sie sind nur größer — und für ein Kapitel, das 28 MB in deine Leitung schiebt, war hier Schluss. Die Lehre ist trotzdem die wertvollste des Kapitels: **Bei semantischer Suche entscheidet die Wahl des Embedding-Modells über alles.** Wer ein RAG-System baut und das Modell nicht auf die eigene Sprache und Fachdomäne prüft, baut einen teuren Zufallsgenerator.

Zwei weitere Grenzen, die auch bei perfektem Modell bleiben:

- **Lange Absätze verwässern.** Ein Eintrag bekommt *einen* Vektor. Packst du fünf Themen in eine Zeile, zeigt der Vektor auf den Durchschnitt und damit auf nichts Richtiges. Genau der Chunking-Effekt, den du in Kapitel 330 am Schieberegler gesehen hast — probier es hier ruhig aus, indem du zehn Sätze in *eine* Zeile klebst.
- **Ähnlichkeit ist keine Wahrheit.** Die Suche findet den Eintrag, der am ähnlichsten *klingt* — nicht den, der stimmt. Steht Unsinn in deiner Datei, findet sie den Unsinn zielsicher. Und wenn ein Sprachmodell aus diesen Treffern eine Antwort formuliert (das ist RAG), wird aus dem falschen Treffer eine flüssig formulierte falsche Auskunft: „Warum KI halluziniert“.

Zurück zur Anwendung

Am Anfang stand der Stapel, den fast jeder herumliegen hat: Firmen-FAQ, Besprechungsnotizen, Arbeitsanweisungen, Rezeptsammlung. Jetzt ist daraus ein durchsuchbarer Vektor-Index geworden — in deinem Browser, aus deinen eigenen Sätzen. Du kannst ab sofort selbst einschätzen, wann sich der Aufwand lohnt: Wo Menschen mit anderen Worten fragen, als in deinen Texten stehen, gewinnt die Bedeutungs-Suche; wo es auf exakte Artikelnummern oder Namen ankommt, bleibt die Stichwort-Suche unschlagbar. Und du kennst die unbequeme Seite: Dieses Modell ist auf Englisch stark und auf Deutsch kaum besser als Würfeln — das misst man, statt es zu hoffen.

Zum Abschluss der Reihe verlassen wir die Sprache und wenden dieselbe Vektor-Idee auf Bilder an: In „Style Transfer“ wird der Malstil eines Gemäldes zu 100 Zahlen — und die legst du dann auf dein eigenes Foto.

Zum Weiterlesen

- Cer u. a. 2018, *„Universal Sentence Encoder“* — arxiv.org/abs/1803.11175 — die Arbeit zu genau dem Modell, das hier deine Sätze in 512 Zahlen verwandelt hat.
- Reimers & Gurevych 2019, *„Sentence-BERT“* — arxiv.org/abs/1908.10084 — der Nachfolger-Ansatz, aus dem auch die mehrsprachigen Modelle stammen, die auf Deutsch besser abschneiden.
- TensorFlow.js, Modell *universal-sentence-encoder* — github.com/tensorflow/tfjs-models (github.com/tensorflow/tfjs-models/tree/master/universal-sentence-encoder) — das Modell aus diesem Kapitel samt Schnittstelle für eigene Projekte.
- Johnson u. a. 2017, *„Billion-scale similarity search with GPUs“* (FAISS) — arxiv.org/abs/1702.08734 — wie dieselbe Suche aussieht, wenn es statt 24 Einträgen Milliarden sind.

Kurzer Check – mit Lösung

Lösungen für Lehrkräfte – für Lernende die Online-Version nutzen.

Frage 1: Die Bedeutungs-Suche findet „bicycle cellar“, obwohl du „where do I put my bike“ gefragt hast — kein gemeinsames Wort. Wie geht das?

- Im Modell steckt ein Synonym-Wörterbuch, das „bike“ und „bicycle“ verknüpft.
- **✔ Beide Formulierungen wurden zu Vektoren, die in fast dieselbe Richtung zeigen — verglichen wird Bedeutung, nicht Schreibweise.**
- Die Suche fragt im Hintergrund ein Sprachmodell, das die Frage umformuliert.

Kein Wörterbuch, keine Regel, keine Rückfrage — nur Geometrie. Das Modell hat beim Training Millionen Sätze gesehen, in denen „bike“ und „bicycle“ in denselben Zusammenhängen auftauchen. Deshalb landen sie im Latent Space dicht beieinander, und die Kosinus-Ähnlichkeit misst genau diese Nähe.

Frage 2: Was liegt nach dem „Indexieren“ eigentlich im Speicher deines Browsers?

- Eine sortierte Wortliste, in der zu jedem Wort steht, in welchen Zeilen es vorkommt.
- **✔ Pro Eintrag eine Reihe von 512 Kommazahlen — der Vektor. Die Wörter selbst braucht die Bedeutungs-Suche nicht mehr.**
- Eine komprimierte Kopie des Modells, angepasst an deine Texte.

Genau das ist der Unterschied zur klassischen Suche: Die arbeitet mit einem Wort-Index. Hier wird jeder Eintrag EINMAL durchs Modell geschickt, und übrig bleiben 512 Zahlen. Der Originaltext wird nur noch zum Anzeigen aufbewahrt — gesucht wird ausschließlich in den Zahlen. (Trainiert wurde dabei übrigens nichts: Die Gewichte des Modells bleiben unverändert.)

Frage 3: Warum kannst du hier bedenkenlos eine echte Firmen-FAQ hineinwerfen?

- Weil die Texte verschlüsselt zum Server geschickt und dort sofort wieder gelöscht werden.
- Weil Vektoren anonym sind — aus 512 Zahlen kann niemand mehr einen Text ableiten.
- **✔ Weil die Texte gar nicht erst verschickt werden: Das Modell wurde zu deinen Daten geholt, statt deine Daten zum Modell.**

Vom Server kommt nur das Modell (einmalig, 28 MB) — danach läuft jede Berechnung lokal. Die zweite Antwort ist übrigens ein verbreiteter und gefährlicher Irrtum: Aus Embedding-Vektoren lässt sich der Originaltext teilweise rekonstruieren. Vektor-Datenbanken sind deshalb genauso schutzbedürftig wie die Dokumente selbst.

Das nimmst du mit

- **Indexieren heißt: Zahlen speichern, keine Wörter.** Jeder Satz wird zu einem [512]-Vektor, die ganze Wissensdatenbank zu einer [n, 512]-Matrix.
- **Die Suche ist eine einzige Formel.** Kosinus-Ähnlichkeit — Skalarprodukt geteilt durch die beiden Vektorlängen; 1,0 heißt gleiche Richtung, 0,0 heißt nichts miteinander zu tun.
- **Buchstaben gegen Bedeutung.** Die Stichwort-Suche liefert null Treffer, wenn du andere Wörter benutzt als der Text — die Bedeutungs-Suche findet ihn trotzdem.
- **Das Modell ist englisch stark und auf Deutsch schwach.** 9 von 12 richtigen Erstplatzierungen gegen 3 von 12 — nachgemessen, nicht behauptet.
- **Ähnlichkeit ist keine Wahrheit.** Steht Unsinn in deiner Datei, findet die Suche den Unsinn zielsicher — und ein Sprachmodell formuliert daraus eine flüssige Falschauskunft.

Du hast jetzt ein Werkzeug, kein Demo. Was hier im Browser mit 24 Sätzen läuft, ist bis auf die Größenordnung exakt das, was in Firmen als Vektor-Datenbank betrieben wird — dieselbe Formel, dieselben Vektoren, nur mehr davon. Die Theorie dahinter steht in „Embeddings in der Praxis: Semantische Suche & RAG“, und warum ähnliche Bedeutungen überhaupt nahe beieinander landen, klärt „Latent Space“. Als Nächstes verlassen wir die Sprache und wenden dieselbe Idee auf Bilder an: Im kommenden Kapitel *„Style Transfer“* wird der Stil eines Gemäldes zum Vektor — und lässt sich auf dein Foto übertragen.

So zitierst du dieses Kapitel

HerrKI: Semantische Suche in DEINEN Texten. herrki.de/470_suche_in_deinen_texten, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.