// KI-Hub VON STRUKTURAFLOW

KI PRAKTISCH

Internes Wissen für KI-Systeme nutzbar machen – RAG ohne Cloud

Wie Betriebe ihr Unternehmenswissen per RAG in KI-Systeme einbinden - selbst gehostet, DSGVO-konform, ohne Cloud-Zwang. Praxisguide für KMU.

AUTOR
Strukturaflow-Team

VERÖFFENTLICHT
13. Juni 2026

ONLINE LESEN

https://wissen.strukturaflow.it.com/internes-wissen-fuer-ki-systeme-nutzbar-machen-rag-ohne-cloud/

Warum Sprachmodelle Ihren Betrieb nicht kennen — und wie Sie das ändern, ohne ein einziges Dokument in die Cloud hochzuladen.

> **Das Kernproblem in einem Satz**
>
> ChatGPT, Claude und Co. sind auf öffentlichen Daten trainiert. Sie kennen Wikipedia, Gesetzestexte, technische Dokumentationen. Aber sie kennen nicht Ihre Preislisten, Ihre internen Abläufe oder die Entscheidungen aus dem letzten Teammeeting. Das ist kein Fehler — das ist eine strukturelle Eigenschaft. Und es gibt einen Weg, das zu ändern. Ohne Nachtraining. Ohne Cloud-Zwang.

Das eigentliche Problem: Wissen sitzt im Betrieb – nicht im Modell

Ein mittelgroßer Handwerksbetrieb. Zwanzig Mitarbeiter, zwei Jahrzehnte Erfahrung. Das Wissen über Kundenpräferenzen steckt im Kopf des Serviceleiters. Die Angebotskalkulation liegt in Excel-Dateien von 2019. Produktinformationen verteilen sich über Hersteller-PDFs, ausgedruckte Kataloge und eine Preisliste auf dem Netzlaufwerk.

Wenn dieser Betrieb jetzt eine KI einsetzt — etwa einen internen Assistenten oder einen Kunden-Chatbot — dann antwortet das System aus dem allgemeinen Trainingswissen. Es weiß nichts von den bevorzugten Materialien. Kennt nicht die Sonderkonditionen bei Lieferanten. Hat keine Ahnung, welche Produkte auf Lager sind oder welche Rückfragen immer wieder kommen.

Die KI antwortet sprachlich korrekt und freundlich. Aber sie antwortet am Betrieb vorbei.

Das ist nicht das Problem der KI. Das ist das Problem der fehlenden Wissensvermittlung.

Hier setzt RAG an — Retrieval-Augmented Generation. Eine Methode, die Betriebe ohne eigene Entwicklungsabteilung sinnvoll einsetzen können.

Was RAG ist – und was es nicht ist

RAG ist keine neue Art von Sprachmodell. Es ist kein Nachtraining. RAG ist ein Prozess, der **vor** der Antwort des Modells stattfindet.

Das Prinzip: Bevor die KI eine Frage beantwortet, wird aus einer eigenen Wissensdatenbank der relevante Kontext abgerufen. Dieser Kontext wird zusammen mit der Frage an das Sprachmodell übergeben. Das Modell antwortet dann auf Basis dieser konkreten, aktuellen Information.

Ein Vergleich: Sie fragen eine neue Mitarbeiterin nach dem Ablauf einer Kundenreklamation. Sie könnte aus allgemeinem Wissen antworten. Oder Sie geben ihr vorher das interne Handbuch, das genau diesen Fall beschreibt — und sie liest nach, bevor sie antwortet. RAG ist das Nachschlagen. Das Sprachmodell ist die Person, die dann antwortet.

Die drei Schritte von RAG

① Indexierung (einmalig)
Ihre Dokumente werden in kleine Abschnitte zerlegt und in einer Vektordatenbank gespeichert. Jeder Abschnitt wird als Bedeutungsvektor kodiert — eine numerische Darstellung des Inhalts, nicht der einzelnen Wörter.

② Retrieval (bei jeder Anfrage)
Wenn eine Frage gestellt wird, sucht das System in der Datenbank nach den inhaltlich passendsten Abschnitten. Das ist keine Keyword-Suche — es ist eine Bedeutungssuche. Wörter müssen nicht identisch sein, die Bedeutung muss ähnlich sein.

③ Generation
Die gefundenen Abschnitte werden zusammen mit der Frage an das Sprachmodell übergeben. Das Modell antwortet auf Basis dieser Informationen — nicht aus dem Trainingswissen.

Was RAG nicht ist: eine Garantie gegen Fehler, ein Ersatz für gute Datenpflege, oder ein System das sich selbst aufbaut. Garbage in, garbage out gilt hier genauso. Wenn die Dokumente, die Sie indexieren, veraltet oder widersprüchlich sind, wird die KI das nicht heilen.

Warum Cloud hier das falsche Framing ist

Wenn Unternehmer das erste Mal von KI-Systemen hören, denken viele automatisch an Cloud. ChatGPT ist Cloud. Microsoft Copilot ist Cloud. Die meisten Messe-Demos — Cloud.

Das erzeugt zwei Vorurteile:

- „KI ist nur mit Cloud möglich.„
- „Wenn ich KI nutze, verlasse ich die Kontrolle über meine Daten.“

Beides ist falsch. Der gesamte RAG-Stack — Vektordatenbank, Embedding-Modell, Sprachmodell, Orchestrierung — lässt sich selbst hosten. Auf einem eigenen Server im Büro. In einem österreichischen Rechenzentrum. Auf einer Linux-Maschine im Netzwerk.

Das bedeutet: Kein Dokument muss das Gebäude verlassen. Kein API-Call geht an OpenAI oder Google. Kein Datenschutzbeauftragter muss sich über Drittland-Transfers den Kopf zerbrechen.

Selbstgehostete KI ist kein Bastlerprojekt mehr. Die Toollandschaft ist weit genug, dass Betriebe ohne Entwicklungsabteilung produktive Systeme aufbauen können — auf eigener Infrastruktur.

Die Frage ist nicht mehr ob. Die Frage ist: welche Komponenten, in welcher Kombination, für welchen Anwendungsfall.

Die Bausteine: Was Sie brauchen – und was Sie weglassen können

Ein lokales RAG-System besteht aus wenigen Bausteinen, die zusammenspielen. Keiner davon ist zwingend ein Cloud-Dienst.

Baustein 1: Die Dokumente

Das klingt offensichtlich, wird aber oft unterschätzt. Bevor ein technisches System gebaut wird: Welches Wissen soll die KI kennen? Und in welchem Zustand ist dieses Wissen aktuell?

Typische Quellen in einem Betrieb:

- Interne Handbücher und Prozessbeschreibungen
- Produktdatenblätter und technische Dokumentation
- FAQ-Listen aus dem Kundensupport
- Angebots- und Vertragsvorlagen
- Meeting-Protokolle und Entscheidungsdokumentationen
- Marketingtexte, Beschreibungen, Positionierungsdokumente

Ein ehrlicher erster Schritt ist ein Wissens-Audit: Was haben wir? Wo liegt es? Ist es aktuell? Widerspricht sich etwas? Ein schlecht gepflegtes Handbuch, das in die Wissensdatenbank wandert, gibt schlechte Antworten — verlässlicher und schneller als ein Mitarbeiter, der es ignorieren würde.

Baustein 2: Das Embedding-Modell

Ein Embedding-Modell übersetzt Text in Zahlen — hochdimensionale Vektoren, die die Bedeutung eines Textes repräsentieren. Zwei Sätze mit ähnlichem Inhalt landen im Vektorraum nah beieinander, auch wenn kein gemeinsames Wort darin vorkommt.

Beispiel: Der Satz „Wie gehe ich vor wenn ein Kunde eine Rechnung beanstandet?„ findet semantisch den Abschnitt „Ablauf bei Rechnungsreklamation“ — obwohl das Wort Reklamation im Satz nicht vorkommt.

Für deutschsprachige Inhalte: Modelle die explizit mehrsprachig trainiert wurden. **bge-m3** ist das stärkste frei verfügbare Modell für diesen Zweck und läuft lokal via Ollama — ohne Cloud-API-Call, ohne Kosten pro Anfrage.

Baustein 3: Die Vektordatenbank

Hier werden die Vektoren aller Dokument-Chunks gespeichert. Bei einer Anfrage wird hier gesucht — nach den Abschnitten, deren Bedeutungsvektor dem der Frage am nächsten liegt.

Qdrant ist die klare Empfehlung für selbst gehostete Setups: in Rust geschrieben, sehr performant, läuft als Docker-Container, hat eine Web-Oberfläche und unterstützt komplexe Metadaten-Filter. Wer bereits eine PostgreSQL-Datenbank betreibt, kann auch **pgvector** als Extension nutzen — kein neues System, kein neuer Server.

Baustein 4: Das Sprachmodell

Das eigentliche Sprachmodell — das, das antwortet — ist austauschbar. Sie können über eine Cloud-API arbeiten (Anthropic, OpenAI) oder ein lokales Modell betreiben (Llama 3, Mistral, Gemma via Ollama). Die Wahl hängt von Qualitätsanforderungen, Datenschutz und verfügbarer Hardware ab.

Für die meisten Betriebe: Wenn Anfragen keine hochsensiblen internen Daten enthalten und DSGVO-konforme Cloud-Verträge vorhanden sind, ist eine Cloud-API für das Sprachmodell praktikabel. Die Dokumente und die Vektordatenbank bleiben trotzdem lokal — nur die Frage plus die drei relevantesten Abschnitte gehen an die API.

Baustein 5: Die Orchestrierung

Irgendetwas muss die Schritte zusammenhalten: Frage entgegennehmen, Vektordatenbank abfragen, Kontext aufbereiten, Modell aufrufen, Antwort zurückgeben. **n8n** ist hier der natürliche Kandidat für Betriebe, die bereits auf No-Code-Automatisierung setzen. Es hat native Nodes für Vektorspeicher und Embedding-Modelle und lässt sich ohne Programmierkenntnisse konfigurieren.

Drei Szenarien aus der Praxis

Abstrakte Konzepte helfen nur begrenzt. Drei konkrete Anwendungsfälle zeigen, wie unterschiedlich RAG in der Praxis aussehen kann.

Szenario A: Der interne Wissens-Assistent

Ein mittelständisches Planungsbüro mit vierzig Mitarbeitern hat über Jahre eine Wissensbasis aufgebaut: Normen, Richtlinien, interne Projekterfahrungen, Vertragsvorlagen, Checklisten für Bauphasen. Dieses Wissen liegt verteilt auf einem Netzlaufwerk — wer etwas braucht, sucht selbst.

Die Lösung: Ein interner KI-Assistent der auf diese Wissensbasis zugreift. Mitarbeiter stellen Fragen in natürlicher Sprache und bekommen eine strukturierte Antwort, die auf den tatsächlichen Firmen-Dokumenten basiert. Mit Quellenangabe. Ohne Google.

Die gesamte Wissensbasis bleibt auf dem eigenen Server. Kein Dokument verlässt das Netzwerk. Das Sprachmodell wird über eine DPA-konforme API angebunden. Das System wird einmal aufgesetzt, neue Dokumente werden per Workflow automatisch indexiert sobald sie auf dem Laufwerk landen.

Szenario B: Der Kunden-Chatbot mit echtem Produktwissen

Ein Großhändler für Sportartikel betreibt einen Webshop mit dreitausend Produkten. Kundenanfragen kommen täglich: Welches Modell eignet sich für Einsteiger? Was ist der Unterschied zwischen diesen zwei Produkten? Gibt es Zubehör dazu?

Ein Standard-Chatbot würde allgemeine Antworten geben. Ein RAG-gestützter Chatbot zieht bei jeder Frage die tatsächlichen Produktdatenblätter aus der Wissensdatenbank: Materialien, Gewichte, empfohlene Einsatzbereiche, Vergleiche. Er antwortet mit den echten Produktinformationen des Unternehmens, nicht mit allgemeinem Marketingtext.

Die Produktdatenblätter werden in die Vektordatenbank indexiert. Bei Aktualisierungen läuft ein n8n-Workflow der das automatisch nachzieht — konkrete Workflow-Beispiele finden Sie hier. Der Chatbot weiß immer, was gerade im Sortiment ist.

Szenario C: Die interne Support-Wissensbasis für Techniker

Ein Unternehmen im Anlagenbau betreibt eine Servicetechnik-Abteilung. Techniker im Außendienst haben immer wieder dieselben Fragen: Wie war das Vorgehen bei Fehlercode XY bei Anlage Z? Was haben wir beim letzten ähnlichen Einsatz gemacht?

Die Lösung: Eine RAG-gestützte mobile Anwendung, die auf Einsatzberichte, Fehlerprotokolle und Reparaturanweisungen der vergangenen Jahre zugreift. Der Techniker vor Ort tippt eine Frage — und bekommt die Zusammenfassung der relevanten Einsätze mit ähnlichem Fehlerbild. Das interne Erfahrungswissen des Teams wird abrufbar, auch wenn der Kollege der damals dabei war im Urlaub ist.

Wann RAG sinnvoll ist – und wann nicht

RAG löst ein spezifisches Problem. Wenn dieses Problem nicht vorhanden ist, ist RAG der falsche Ansatz.

RAG IST SINNVOLL WENN …	RAG IST NICHT NÖTIG WENN …
Die Wissensbasis mehr als 50–100 Dokumente umfasst	Der Kontext in einen übersichtlichen System-Prompt passt
Informationen sich regelmäßig ändern und aktuell sein müssen	Die Inhalte stabil sind und sich selten aktualisieren
Quellennachweis und Nachvollziehbarkeit wichtig sind	Allgemeines Weltwissen für die Aufgabe ausreicht
Verschiedene Nutzer sehr unterschiedliche Fragen stellen	Immer dieselben Fragen gestellt werden (dann: FAQ-System)
Dokumente durchsuchbar gemacht werden sollen ohne Umweg	Das Budget und der Aufwand unverhältnismäßig wären

Ein Handwerksbetrieb mit zwanzig standardisierten Leistungen und einer dreiseitigen FAQ braucht kein RAG. Ein Ingenieurbüro mit zwanzig Jahren Projektdokumentation, das sein internes Wissen zugänglich machen will, schon.

Der Aufbau Schritt für Schritt

Für Betriebe, die sich konkret mit dem Aufbau eines lokalen RAG-Systems beschäftigen wollen, ist der folgende Ablauf eine bewährte Orientierung.

Schritt 1 — Wissens-Audit durchführen
Welche Dokumente sollen in die Wissensdatenbank? Wo liegen sie aktuell? In welchem Format (PDF, Word, Markdown, Wiki)? Sind sie aktuell und konsistent? Diese Phase wird oft unterschätzt — sie ist die wichtigste. Ein System ist nur so gut wie die Dokumente, auf denen es aufbaut.

Schritt 2 — Format und Qualität vereinheitlichen
Dokumente sollten in einem einheitlichen, maschinenlesbaren Format vorliegen — idealerweise Markdown oder sauber strukturierte PDFs. Gescannte Dokumente ohne Texterkennung, verschachtelte Tabellen-in-Tabellen-PDFs und widersprüchliche Versionen desselben Dokuments sind häufige Probleme, die im Vorfeld bereinigt werden müssen.

Schritt 3 — Technischen Stack auswählen
Für die meisten Betriebe: Qdrant oder pgvector als Vektordatenbank, bge-m3 als lokales Embedding-Modell via Ollama, n8n für die Orchestrierung. Das Sprachmodell je nach Datenschutzanforderung lokal oder via DSGVO-konformer Cloud-API. Keine Entscheidung ist endgültig — der Stack ist modular und austauschbar.

Schritt 4 — Erste Wissensbasis aufbauen und testen
Mit einem kleinen, klar definierten Subset starten — zum Beispiel nur die internen FAQs oder nur die Produktdatenblätter einer Kategorie. Testfragen stellen, Antworten prüfen, Chunk-Größen und Retrieval-Parameter anpassen. Erst wenn die Qualität stimmt, die Wissensbasis erweitern.

Schritt 5 — Aktualisierungs-Workflow einrichten
Ein RAG-System dessen Wissensbasis nicht aktuell gehalten wird, wird zum Risiko. n8n kann so konfiguriert werden, dass neue oder geänderte Dokumente automatisch neu indexiert werden — ausgelöst durch Dateiänderungen, E-Mail-Trigger oder manuelle Auslöser.

Schritt 6 — Berechtigungen und Zugriffskonzept
Nicht alle Mitarbeiter sollen auf alle Dokumente zugreifen können — und die KI auch nicht. Vektordatenbanken wie Qdrant unterstützen Metadaten-Filter: Dokumente können mit Tags versehen werden (z. B. Abteilung, Vertraulichkeitsstufe) und das Retrieval kann so eingeschränkt werden, dass nur freigegebene Inhalte für bestimmte Nutzerrollen abrufbar sind.

Die DSGVO-Frage: Was tatsächlich relevant ist

Datenschutz ist für österreichische und deutsche Betriebe kein optionales Thema — der DSGVO-konforme Einsatz von KI gilt auch für RAG-Systeme. Keine juristische Einschätzung — das ist Aufgabe eines Datenschutzbeauftragten oder Rechtsanwalts. Aber hier die technischen Parameter, die rechtlich relevant sind:

Wo liegen die Dokumente?
Auf eigenem Server: volle Kontrolle. Cloud: Vertrag mit Auftragsverarbeitungsvertrag (AVV) nötig.

Welche Daten verlassen das System?
Beim API-Aufruf ans Sprachmodell gehen nur Frage und die relevanten Chunks raus — nicht die gesamte Wissensbasis. Das ist ein wesentlicher Unterschied zu Systemen, die alle Dokumente in die Cloud hochladen.

Werden personenbezogene Daten indexiert?
Kundendaten, Mitarbeiterdaten in der Wissensbasis sind besonders kritisch. Ein klares Konzept dafür ist Voraussetzung, nicht Kür.

Wo ist der API-Anbieter ansässig?
EU-basiert oder Standardvertragsklauseln (SCCs) für US-Anbieter. Cohere bietet EU-Hosting, bei OpenAI sind SCCs Pflicht.

Self-hosted Komponenten (Qdrant, Ollama, n8n):
Keine Drittland-Problematik, volle Datenkontrolle.

Die gute Nachricht: Ein vollständig selbst gehosteter RAG-Stack — Vektordatenbank, Embedding-Modell und Sprachmodell lokal — ist aus DSGVO-Perspektive der unkomplizierteste Weg. Kein API-Anbieter, kein Drittland-Transfer, keine Auftragsverarbeitungsverträge nötig. Die Ein-

schränkung ist die Qualität lokal betriebener Sprachmodelle, die bei allgemeinen Aufgaben noch hinter den großen Cloud-Modellen liegt — bei spezifischem Domänenwissen durch gutes Retrieval aber näher rückt als Sie denken.

Häufige Fehler beim Aufbau – und wie Sie sie vermeiden

Fehler 1: Die Wissensbasis wird nicht gepflegt

Ein RAG-System ist kein statisches Produkt. Wenn die Dokumente, auf denen es aufbaut, veralten, veraltet das System mit — ähnlich wie Wissen im Kopf statt im System langfristig zum Risiko wird. Betriebe die keine klare Verantwortlichkeit für die Wissensbasis definieren, haben nach sechs Monaten ein System das mit Überzeugung falsche Informationen liefert — weil das Handbuch aus 2022 noch drin ist.

Lösung: Klare Ownership. Wer ist verantwortlich, welche Dokumente in der Wissensbasis aktuell zu halten? Automatische Trigger für Neuindexierung bei Dateiänderungen.

Fehler 2: Zu viele Dokumente auf einmal

Der Impuls, alles auf einmal zu indexieren, ist verständlich — führt aber zu einem System das schwer zu testen und zu optimieren ist. Wenn die Qualität nicht stimmt, wissen Sie nicht wo das Problem liegt: am Embedding-Modell? An der Chunk-Größe? An den Dokumenten selbst?

Lösung: Klein anfangen. Eine klar definierte Dokumentengruppe. Qualität validieren. Dann erweitern.

Fehler 3: Chunk-Größen werden nicht angepasst

Dokumente werden in Abschnitte zerlegt — sogenannte Chunks. Zu kleine Chunks verlieren Kontext, zu große bringen irrelevante Informationen mit. Die ideale Chunk-Größe hängt vom Dokumenttyp ab: technische Handbücher funktionieren mit anderen Größen als Gesprächsprotokolle oder Produktbeschreibungen.

Lösung: Chunk-Parameter pro Dokumenttyp testen. Overlap zwischen Chunks einbauen, damit kein Kontext an Grenzen verloren geht.

Fehler 4: Das Retrieval wird nicht evaluiert

Viele Betriebe prüfen die Ausgabe des Sprachmodells — aber nicht, ob das Retrieval die richtigen Abschnitte findet. Ein Modell kann eine schlechte Grundlage trotzdem zu einem kohärenten Text formen. Das macht Fehler schwerer zu erkennen.

Lösung: Testfragen mit bekannter korrekter Antwort und bekanntem Quelldokument verwenden. Prüfen ob das Retrieval die richtigen Chunks zurückgibt — unabhängig von der generierten Antwort.

Nächste Schritte: Wo Sie konkret anfangen

Der wichtigste Schritt ist nicht technisch. Er ist konzeptuell.

Die entscheidende Frage: Welches Wissen in unserem Unternehmen ist so wertvoll, so oft gebraucht und so schwer zugänglich, dass es sich lohnt, es systematisch verfügbar zu machen?

Wenn Sie diese Frage beantwortet haben, ergibt sich daraus automatisch:

- Welche Dokumente in eine erste Wissensdatenbank gehören
- Welche Nutzergruppe davon profitiert
- Welche Schnittstelle sinnvoll ist (interner Assistent, Kundenchatbot, Techniker-App)

Der technische Stack ist lösbar. Tools wie n8n, Qdrant und Ollama sind gut dokumentiert, aktiv gepflegt und für Betriebe ohne eigene Softwareentwicklung handhabbar — insbesondere wenn jemand den Aufbau begleitet.

Drei konkrete nächste Schritte:

1. **Wissens-Audit durchführen:** Liste erstellen — welche Dokumente gibt es, wo liegen sie, wer pflegt sie, wie aktuell sind sie.
2. **Einen kleinen Anwendungsfall definieren:** Nicht „wir digitalisieren alles„. Sondern: „Wir machen die FAQ-Wissensbasis für den Support durchsuchbar“ oder „Wir bauen einen Assistenten für die Produktdatenblätter der Kategorie X„.
3. **Technische Machbarkeit klären:** Welche Hardware ist vorhanden? Läuft bereits ein Server? Gibt es jemanden, der Docker-Container betreuen kann? Falls nicht: wo kann das eingekauft werden?

Internes Wissen zugänglich zu machen ist keine IT-Aufgabe. Es ist eine Entscheidung über das eigene Unternehmen: Wie viel des Wissens, das Sie aufgebaut haben, soll nur in Köpfen und Laufwerken bleiben — und wie viel davon soll tatsächlich wirken?

NÄCHSTER SCHRITT

Mehr praktische KI-Anleitungen für KMU

Dieser Artikel ist Teil des KI-Hubs von Strukturaflow — einer deutschsprachigen Plattform für den praktischen KI-Einsatz in kleinen und mittleren Unternehmen.

https://wissen.strukturaflow.it.com