Determinismus vs. Autonomie: Wann sich agentische Orchestrierung in Enterprise-RAG-Systemen wirklich lohnt

Autor: Michael Kettel, rms relationship marketing solutions GmbH | **Stand:** September 2026

MANAGEMENT SUMMARY

Retrieval-Augmented Generation (RAG) hat sich als Standardarchitektur etabliert, um Sprachmodelle mit unternehmenseigenen Datenbeständen zu verbinden. In der Praxis zeigt sich jedoch ein klarer Bruch: Standardisierte Wissensabfragen lassen sich mit optimierten, deterministischen Such-Pipelines schnell, kosteneffizient und verlässlich beantworten. Geht es dagegen um mehrstufige Analysen, verteilte Datenquellen oder dynamische Tool-Aufrufe, stoßen einfache Systeme an funktionale Grenzen.

Die derzeit populäre Antwort der Branche lautet **Agentic RAG** – oft als universeller Nachfolger klassischer Architekturen porträtiert. Für CTOs, IT-Leiter und Enterprise-Architekten führt diese Sichtweise in die Irre. Agentic RAG ist kein automatisches Upgrade, sondern eine spezifische Entwurfsentscheidung für unvorhersehbare Problemstellungen. Autonome Kontrollflüsse helfen dort, wo sich Abfragepfade im Vorfeld nicht festlegen lassen. Gleichzeitig öffnet funktionale Autonomie die Tür für neue Risiken wie Indirect Prompt Injections, treibt Latenzen nach oben und erschwert automatisierte Regressionstests spürbar.

Dieses Whitepaper ordnet den Architekturraum zwischen deterministischem und agentischem Retrieval ein, definiert konkrete Bewertungskriterien jenseits reiner Textqualität und benennt die realen Anforderungen an Security, Governance und operative Kostenkontrolle.

1. Der Architekturraum: Vom deterministischen Ablauf zum autonomen System

Die gängige Gegenüberstellung von „starrem Vektor-RAG“ und „flexiblem Agentic RAG“ greift technisch zu kurz. Reale Enterprise-Systeme bewegen sich auf einem breiten Kontinuum zwischen fest verdrahteten Abläufen und autonomen Agenten:

ABBILDUNG 1: DAS ARCHITEKTUR-KONTINUUM VON RAG-SYSTEMEN

Viele Aufgaben brauchen keine Agenten

Ein Großteil der typischen RAG-Probleme lässt sich mit etablierten, deterministischen Mustern lösen:

- **Präzision bei Fachbegriffen:** Hybrid Search (dichte Vektorsuche kombiniert mit lexikalischem BM25) verhindert, dass exakte Artikelnummern, Normbezeichnungen oder Eigennamen untergehen.
- **Trefferqualität:** Ein nachgelagerter Cross-Encoder sortiert unpassende Chunks vor der Kontext-Injektion verlässlich aus.
- **Zahlen und Fakten:** Ein regelbasierter Router leitet strukturierte Fragen direkt an relationale Datenbanken weiter, statt SQL-Tabellen in Vektoren zu pressen.
- **Dokumentenvergleiche:** Auch der Abgleich zweier Richtlinien erfordert meist keinen Agenten. Ein fester Ablauf zerlegt die Anfrage in zwei vordefinierte Abfragen, filtert nach Versionen und führt die Daten zusammen.

Was macht ein System tatsächlich „agentic“?

Entscheidend ist letztlich der Kontrollfluss. Ein System wird nicht agentisch, weil es mehrere Werkzeuge nutzt oder eine SQL-Abfrage ausführt. Der Unterschied liegt in drei Kernmerkmalen:

- **Der Ablauf steht vorher nicht fest:** Das Modell entscheidet zur Laufzeit anhand von Zwischenergebnissen über den nächsten Schritt.
- **Dynamische Tool-Wahl:** Das System wählt selbstständig aus einem Katalog registrierter Werkzeuge (Vektorspeicher, SQL, APIs) und formuliert die Aufrufparameter eigenständig.
- **Iterative Schleifen:** Das System prüft Zwischenergebnisse selbst. Reichen die Daten nicht aus, startet es einen zweiten Versuch oder reformuliert die Frage.

2. Entscheidungskriterien: Die Vorhersehbarkeit des Kontrollflusses

Der Preis für funktionale Flexibilität ist eine schlechtere Vorhersagbarkeit. Jeder zusätzliche Entscheidungsschritt macht das Gesamtsystem schwerer testbar und meist auch teurer. Der wichtigste Auslöser für Agentic RAG ist selten die reine Komplexität einer Aufgabe. Entscheidend ist vielmehr, ob sich der Kontrollfluss im Vorfeld festlegen lässt. Selbst ein Ablauf mit 15 Einzelschritten gehört in einen deterministischen Workflow, solange die Reihenfolge der Abfragen bekannt ist.

Datenlandschaft	**Vorhersehbarkeit: HOCH** (Bekannte Schritte & Pfade)	**Vorhersehbarkeit: NIEDRIG** (Explorativ / Variabel)
Homogen / Dokumentenzentriert	**Advanced RAG** Hybrid Search (BM25 + Vektor), Metadaten-Filter, Cross-Encoder Reranking.	**Deterministische Workflows** Vordefinierte Fallback-Kaskaden, parallele Subqueries mit fester Konsolidierung.
Heterogen / Multimodal & Transaktional	**Feste Multi-Source DAGs** Parallel ablaufende SQL- und Vektorzapfstellen mit festgelegtem Merge-Schritt.	**★ Agentic RAG** State Graphs, dynamische Werkzeugwahl, mehrstufige Re-Planning Loops.

WANN AGENTISCHE MUSTER HELFEN

- **Unvorhersehbare Analysen:** Bei IT-Incident-Analysen oder regulatorischen Prüfungen über Rechtsräume lässt sich vorab nicht definieren, wie viele Schritte nötig sind.
- **Verkettete Tool-Abhängigkeiten:** Wenn erst die Antwort aus dem CRM darüber entscheidet, welches Folgesystem (Ticket, API, Index) befragt wird.
- **Explorative Recherche:** Wenn Anfragen iterativ angepasst werden müssen, da statische Synonymlisten nicht greifen.

WANN DETERMINISTIK VORZUZIEHEN IST

- **Klar definierte Prozesse:** Sobald ein Ablauf festen Regeln folgt (Schritt 1 bis 4 nacheinander), ist ein starrer Workflow robuster und wartungsärmer.
- **Isolierte Dokumentensammlungen:** Für Handbücher, FAQs oder Dokumentationen reicht sauberes Parsing, Metadaten und Reranking völlig aus.
- **Harte Latenzvorgaben:** In synchronen Web-Chats mit knappen Timeouts verbieten sich mehrstufige Agenten-Schleifen von selbst.

3. Latenz und Kosten im laufenden Betrieb

Agentische Systeme verändern vor allem zwei Dimensionen: Latenz und Kosten. Diese Effekte müssen deshalb bereits beim Architekturentwurf berücksichtigt werden.

Das Latenzprofil

In einer klassischen RAG-Pipeline lässt sich die Antwortzeit gut vorhersagen:

$$T_{ges} = T_{Embedding} + T_{Search} + T_{Rerank} + T_{LLM\text{-}Inferenz}$$

In einem agentischen System multipliziert sich diese Gleichung mit jedem Zwischenschritt:

$$T_{agentic} = \sum_{i=1..N} \left(T_{LLM\text{-}Reasoning,\,i} + T_{Tool\text{-}Execution,\,i} \right) + T_{Synthesis}$$

Jede Korrekturschleife und jeder Tool-Aufruf kostet Zeit. Abhängig von Modellen, Schnittstellen-Latenzen und Denkpausen können Antwortzeiten im Bereich mehrerer Sekunden oder darüber hinaus liegen. Für die Systemarchitektur bedeutet das:

- **Fast-Path und Deep-Path trennen:** Standardabfragen laufen synchron und liefern Ergebnisse unmittelbar. Komplexe Agenten-Workflows werden asynchron ausgeführt und informieren den Nutzer schrittweise per Streaming oder Status-Updates.
- **Tool-Aufrufe parallelisieren:** Unabhängige Abfragen dürfen nicht nacheinander laufen, sondern müssen gleichzeitig an die Zielsysteme übergeben werden.

Kostenkontrolle durch Model-Tiering

Wer für jeden internen Schritt ein Frontier-Modell nutzt, verbrennt unnötig Budget. Bei agentischen Systemen gehört die Aufteilung auf Modellklassen deshalb zum Standard:

Aufgabe	Modellklasse	Typischer Fokus
Klassifikation, Routing & Grading	Small Language Models (SLMs: 8B–14B Parameter)	Hohe Geschwindigkeit, strukturierte Ausgabe, geringe Inferenzkosten.
Query Rewriting & Sub-Queries	Mid-Tier Modelle (~70B Parameter)	Solides Sprachverständnis, mittlerer Durchsatz, robuste Syntax.
Finale Synthese & Argumentation	Frontier-Level LLMs	Komplexe Synthese, Abwägung und Befolgung von Constraints.

Kleine Modelle für Zwischenschritte senken die reinen Inferenzkosten spürbar. Diese Ersparnis darf man jedoch nicht mit sinkenden Gesamtkosten verwechseln: Die höhere Systemkomplexität bindet Engineering-Kapazitäten und verlangt deutlich mehr Monitoring.

4. Groundedness ist nicht gleich Korrektheit

Oft wird behauptet, Verifikationsschleifen würden Halluzinationen zuverlässig verhindern. In der Praxis muss man zwei Begriffe sauber trennen:

- **Groundedness (Texttreue):** Lässt sich die Aussage formal aus den gefundenen Textstellen belegen?
- **Correctness (Inhaltliche Wahrheit):** Stimmt die Aussage in der Realität?

Ein System kann formal perfekt belegt sein und trotzdem inhaltlich falsch antworten:

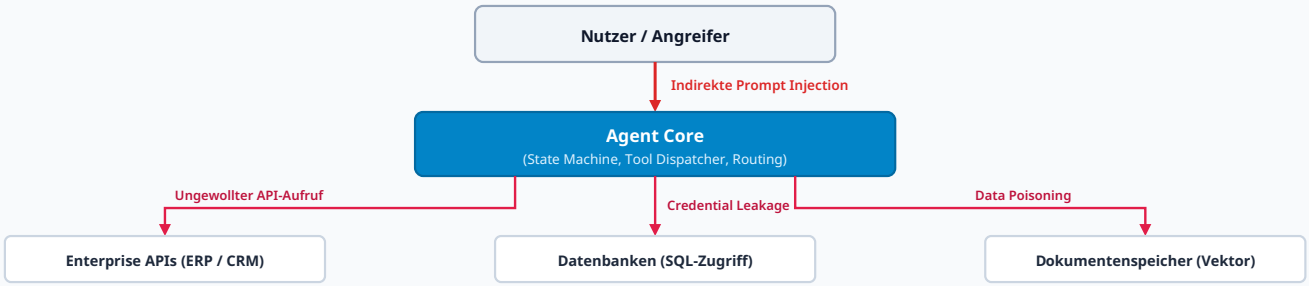
- **Belegte Fehlinformationen:** Wenn veraltete Dokumente im Index liegen oder sich Quellen widersprechen, bewertet der Grader die falsche Antwort dennoch als korrekt belegt.
- **Logische Trugschlüsse:** Modelle ziehen bisweilen falsche Schlüsse aus zwei Fakten, die isoliert betrachtet stimmen. Ein Groundedness-Check bemerkt das selten, weil alle genannten Wörter im Quelltext auftauchen.
- **Bestätigungsfehler (Confirmation Bias):** Nutzt man für die Antwort und deren anschließende Überprüfung dasselbe Modell, neigt es dazu, eigene Fehler im zweiten Schritt durchzuwinken.

> **Praxishinweis:** Automatische Checks filtern offensichtliche Erfindungen heraus. Sie ersetzen aber weder ein sauberes Metadaten-Management für Dokumentenstände noch programmierte Plausibilitätsprüfungen für kritische Daten.

5. Security und Governance: Das veränderte Risikoprofil

Klassische RAG-Systeme lesen Daten nur. Agenten dagegen führen Aktionen aus und greifen aktiv auf Schnittstellen zu. Damit verlagert sich der Sicherheitsschwerpunkt vom reinen Datenschutz hin zur Handlungskontrolle.

ABBILDUNG 2: ANGRIFFS- UND RISIKOVEKTOREN IN AGENTISCHEN SYSTEMEN



Die vier Kernrisiken in der Praxis

- **Indirekte Prompt Injections:** Liest ein Agent ungesicherte Quellen ein – etwa Tickets, Webseiten oder Kundenmails –, können darin versteckte Steuerbefehle stehen. Ein Agent mit Schreibrechten ließe sich so manipulieren, unbefugte Aktionen im Backend anzustoßen.
  Gegenmaßnahme: Schreibrechte und Leserechte strikt trennen. RAG-Agenten sollten Daten prinzipiell nur abrufen. Soll ein System Aktionen auslösen, geschieht das über separate Komponenten – idealerweise mit Freigabe durch den Nutzer (Human-in-the-Loop).
- **Rechteverwaltung (Identity Propagation):** Klassische Suchsysteme filtern Dokumente meist vorab nach Nutzerrechten. Wenn ein Agent jedoch selbstständig Datenbanken oder APIs ansteuert, darf er nicht mit einem globalen Admin-Account arbeiten. Seine Berechtigung muss an die Rechte des jeweiligen Nutzers gekoppelt sein (Token Forwarding).
- **Audit-Trails:** Für regulatorisch sensible Prozesse und interne Compliance-Anforderungen kann ein lückenloser, maschinenlesbarer Audit-Trail erforderlich bzw. organisatorisch geboten sein. Dazu gehören:
  - Der Verlauf der Zustandsübergänge (State Transitions)
  - Alle Tool-Aufrufe samt Parametern und Rückgabewerten
  - Die genutzten Modellversionen und System-Prompts jedes Einzelschritts
- **Die Rolle von Schnittstellen-Standards (wie MCP):** Das Model Context Protocol (MCP) vereinfacht die technische Anbindung von Tools. Ein einheitlicher Standard löst aber weder Berechtigungsfragen noch sorgt er für Datenqualität. Die Endpunkte müssen weiterhin in der eigenen Infrastruktur gegen Missbrauch abgesichert werden.

6. Evaluation: Agentenverhalten messbar machen

Die Qualität agentischer Systeme lässt sich nicht allein mit klassischen Text-Metriken messen. Neben Groundedness und Context Precision müssen Engineering-Teams das Verhalten des Agenten selbst monitoren:

METRIKEN ZUR DATENBASIS

- **Groundedness (Texttreue):** Anteil belegter Aussagen
- **Context Recall & Precision:** Relevanz der Chunks
- **Factual Correctness:** Abgleich gegen Golden Data

METRIKEN ZUM VERHALTEN

- **Task Success Rate:** Anteil gelöster Aufgaben
- **Tool Selection Accuracy:** Trefferquote der Tool-Wahl
- **Tool Calls pro Aufgabe:** Effizienz-Indikator
- **Loop Rate:** Anteil nötiger Korrekturschleifen
- **Failure- & Timeout-Rate:** Abgebrochene Durchläufe

Kosten und Antwortzeiten müssen immer pro erfolgreicher Aufgabe gemessen werden. Betrachtet man nur den Einzelfall, fallen teure Schleifen und abgebrochene Durchläufe aus der Statistik.

Wann sich der Aufwand rechnet

Die zusätzliche Komplexität lohnt sich nur, wenn daraus messbarer geschäftlicher Nutzen entsteht:

$$\Delta\,\textit{Business Value} > \Delta\,\textit{Inferenzkosten} + \Delta\,\textit{Entwicklungs- \& Betriebskosten} + \textit{Latenzkosten} \qquad \text{[Gl. 3: Rentabilitätsbedingung]}$$

Erreicht eine feste Pipeline mit Reranking 88 % Erfolgsquote und ein komplexer Agentengraph 94 %, muss dieser Zuwachs den höheren Wartungsaufwand, die zusätzlichen API-Kosten und die spürbar längere Wartezeit rechtfertigen. Bei kritischen Datenabfragen kann dieser Schritt sinnvoll sein; bei gewöhnlichen Support-Portalen rechnet er sich selten.

7. Der Ausbaupfad in der Praxis

In der Praxis beginnt das Problem meist nicht beim Agenten, sondern bei den Daten. Anstatt sofort ein komplexes System zu bauen, empfiehlt sich ein stufenweises Vorgehen:

STUFE 1: DATENBASIS & ADV. RAG

- Sauberes Parsing von Tabellen und Textstrukturen
- Hybrid Search (Vektor + BM25) und Cross-Encoder Reranking
- Testdatensatz (Golden Dataset) aufbauen & Textqualität messen

→

STUFE 2: FESTE WORKFLOWS

- Strukturierte (SQL) und unstrukturierte Daten deterministisch trennen
- Feste Workflows für bekannte Mehrschritt-Abfragen bauen
- Klare Fallback-Regeln für fehlgeschlagene Suchen definieren

→

STUFE 3: AGENTIC RAG

- State-Graph-Engines für Pfade mit unvorhersehbarem Kontrollfluss
- Dynamische Tool-Wahl für offene Recherchen bereitstellen
- Harte Obergrenzen für Schleifen und Token-Budgets festlegen
- Verhaltens-Metriken kontinuierlich monitoren

Fazit & Ausblick

Agentic RAG ist kein Ersatz für bewährte RAG-Pipelines, sondern eine Erweiterung für spezielle Fälle. Es löst dort Probleme, wo der Ablauf zur Entwurfszeit nicht vorhersehbar ist und ein System flexibel Werkzeuge ausprobieren oder sich selbst korrigieren muss.

Kernbotschaft für Technologieentscheider: Ohne sauberes Data Engineering hilft auch der beste Agent nicht weiter. Schlecht aufbereitete Daten führen in agentischen Schleifen lediglich zu mehr Fehlern, höheren Kosten und langen Antwortzeiten. Erst wenn das Fundament aus Hybrid Search, Metadaten und nachvollziehbaren Schnittstellen stabil steht, lohnt sich der Schritt zu autonomen Komponenten.