Onderzoek: De actuele stand van zaken rondom RAG-systemen (juni 2026)

---

1. Wat is RAG?

RAG — *Retrieval-Augmented Generation* — is een architectuurpatroon dat grote taalmodellen (LLM's) verbindt met externe kennisbronnen op het moment dat een vraag wordt gesteld.

Het kernprobleem dat RAG oplost

Een standaard LLM werkt uitsluitend op basis van statische trainingsdata. Dit leidt tot twee fundamentele beperkingen:

- **Verouderde kennis**: het model weet niets van recente gebeurtenissen of interne bedrijfsdocumenten.
- **Hallucinaties**: als het model een antwoord niet weet, verzint het er één — ogenschijnlijk zelfverzekerd.

RAG lost beide problemen op door het model *vóór het genereren van een antwoord* te laten zoeken in een actuele, curatieve kennisbasis.

Hoe werkt het?

Het proces verloopt in twee grote fasen:

Fase A — Offline indexering (éénmalig/periodiek)

```
Documenten → opsplitsen in chunks → omzetten naar vector-embeddings → opslaan in vector database
```

Fase B — Online inferentie (bij elke vraag)

Stap	Voorbeeld
Gebruiker stelt een vraag	*"Wat is ons retourbeleid voor zakelijke klanten?"*
Query omzetten naar vector-embedding	De zin wordt omgezet naar een vector van 1536 getallen die de betekenis vastleggen
Vector database doorzoeken op semantische gelijkenis	De database vindt chunks over "retour", "zakelijk", "voorwaarden" — ook als die andere woorden gebruiken
Relevante chunks ophalen + herschikken op relevantie	Top-50 kandidaten worden door de reranker teruggebracht naar de 5 meest relevante passages
Chunks als context meegeven aan het LLM	De 5 passages worden samen met de vraag in de prompt geplaatst
LLM genereert antwoord gegrond in de opgehaalde documenten	Het model formuleert een antwoord op basis van uitsluitend de meegeleverde passages
Antwoord + bronvermelding naar gebruiker	*"Zakelijke klanten kunnen binnen 30 dagen retourneren (bron: Algemene Voorwaarden B2B, art. 7.2)"*

Waarom RAG en niet fine-tunen?

	Fine-tuning	RAG
Kennisupdate	Nieuw model trainen	Documenten toevoegen aan kennisbasis
Kosten	Hoog	Relatief laag
Traceerbaarheid	Geen	Per antwoord terug te herleiden
Actuele data	Moeilijk	Standaard
Compliance	Complex	Beheerbaar

Gartner stelt dat tegen 2026 meer dan **70% van enterprise generative AI-initiatieven** gestructureerde retrieval-pipelines vereist om hallucinaties en compliance-risico's te beperken.

---

2. De evolutie van RAG: van Naive naar Agentic

De RAG-wereld heeft zich in drie jaar tijd sterk gedifferentieerd. Er zijn vier generaties, waarvan de laatste — Agentic RAG — zeven onderscheiden architectuurfamilies kent.

Generatie 1: Naive / Basic RAG

Het oorspronkelijke patroon: één retriever, één LLM, één prompt-template. Lineair en statisch. Werkt goed voor eenvoudige use cases, maar schaalt slecht en is gevoelig voor ruisige retrieval-resultaten. Het fundamentele probleem: als de opgehaalde documenten niet goed genoeg zijn, weet het systeem dat niet — het genereert toch een antwoord.

Generatie 2: Advanced RAG

Voegt optimalisaties toe vóór en ná de retrieval-stap: - **Pre-retrieval**: query-herformulering, hypothetical document embeddings (HyDE), query-expansie - **Post-retrieval**: reranking, contextcompressie, relevantiefiltering

Generatie 3: Modular RAG

De pipeline wordt opgesplitst in onafhankelijk vervangbare modules (retriever, reranker, generator). Hierdoor kun je componenten incrementeel verbeteren — reranker nu toevoegen, agentic loops later.

Generatie 4: Agentic RAG

Agentic RAG is niet een incrementele verbetering maar een **architectuurverschuiving**: retrieval wordt een gecontroleerd besluitvormingsproces in plaats van een eenmalige opzoekstap. Formeel is dit te beschrijven als een *Partially Observable Markov Decision Process (POMDP)* — de agent neemt beslissingen onder onzekerheid over de staat van de kennisbasis en past zijn strategie aan op basis van wat het tot nu toe heeft gevonden.

De motivatie: bij complexe taken kan de retrieval-behoefte pas duidelijk worden *tijdens* het redeneren, niet vooraf. Vooraf opgehaalde kennis sluit dan niet aan bij wat het model op dat moment nodig heeft.

Een agentic RAG-systeem beschikt over vijf terugkerende modules:

1. **Controller / policy** — beslist wanneer en hoe te retrieven
2. **Retrieval orchestration** — query-herformulering, decompositie, evidence-verfijning
3. **Memory & state** — houdt tussenstappen bij over meerdere retrieval-rondes
4. **Tool use** — gaat verder dan een statisch corpus (web search, API's, databases)
5. **Critics / verifiers** — detecteren misalignment en triggeren correctie

De zeven architectuurfamilies binnen Agentic RAG

Onderzoek (Singh et al. 2025; Mishra et al. 2026) onderscheidt zeven families, elk met eigen controlemechanisme en risicoprofiel:

1. Self-reflective RAG Het model beslist *zelf* wanneer het iets moet ophalen via speciale reflectie-tokens die tijdens het genereren worden ingevoegd. Indiscriminaat altijd retrieven kan de kwaliteit verlagen — dit patroon maakt retrieval selectief en contextafhankelijk.

Systeem	Kernmechanisme
Self-RAG	Traint een model om on-demand te retrieven via reflection tokens; gedrag is stuurbaar per inference
ReflectiveRAG	Klein LM evalueert iteratief of het bewijs voldoende is en past de query aan zonder hertraining
MetaRAG	Koppelt retrieval aan metacognitie: drie-staps pipeline die tekortkomingen in het initiële antwoord identificeert
SKILL-RAG	RL-gebaseerd framework dat irrelevante content filtert op zinsniveau

Risico: afhankelijkheid van de kwaliteit van de retrieval én de betrouwbaarheid van het reflectiesignaal zelf.

2. Corrective RAG (CRAG) Voegt expliciete kwaliteitsbeoordeling van de retrieval-resultaten toe. Als de kwaliteit onvoldoende is, wordt een correctieve actie getriggerd (opnieuw zoeken, web search, alternatieve bron) vóórdat het eindantwoord wordt gegenereerd.

Systeem	Kernmechanisme
CRAG	Retrieval evaluator + large-scale web search als fallback + decompose-then-recompose algoritme
RAG-Critic	Traint een critic-model dat fijngranulaire foutfeedback geeft voor zelfcorrectie
AlignRAG	Iteratieve Critique-Driven Alignment: verfijnt redeneertrajecten op basis van misalignment met bewijs

Risico: de evaluatiestap voegt latentie toe; een slecht gekalibreerde critic kan foutieve correcties introduceren.

3. Adaptive routing RAG Stuurt queries naar een passende retrieval-strategie (geen retrieval / enkelvoudige stap / multi-hop) op basis van de complexiteit van de vraag. Doel: de kosten-kwaliteitsverhouding optimaliseren door zware retrieval te reserveren voor zware vragen.

Systeem	Kernmechanisme
Adaptive-RAG	XLNet-gebaseerde complexiteitsclassifier + retrieval controller
QC-RAG	Information Sufficiency Assessment module: bepaalt dynamisch of bewijs toereikend is
Bandit routing	Behandelt elke retrieval-methode als een "arm" en balanceert exploratie en exploitatie

Risico: misrouting (te licht of te zwaar retrieven). Onderzoek (RAGRouter-Bench) toont dat geen één strategie universeel optimaal is — routers moeten generaliseren over domeinen.

4. Iterative-planning RAG Decomposteert complexe taken in sub-doelen en itereert retrieval en generatie over meerdere stappen. Elke stap verfijnt het bewijs of de gedeeltelijke oplossing.

Systeem	Kernmechanisme
AIR-RAG	Adaptieve feedback over meerdere iteraties voor ranking- en documentverfijning
LoRAG	Dynamische loop-module die gegenereerde tekst iteratief verfijnt via retrieval-interacties
CR-Planner	Critic-gestuurde planning: selecteert en voert sub-doelen uit met correctie bij redeneerfouten

Risico: multi-stap foutpropagatie — fouten in vroege stappen compounderen in latere stappen.

5. Multi-agent RAG Verdeelt het end-to-end proces over meerdere gespecialiseerde agenten met expliciete coördinatieprotocollen. Verbetert interpreteerbaarheid en multi-hop betrouwbaarheid.

Systeem	Kernmechanisme
MA-RAG	Vier agenten (Planner, Step Definer, Extractor, QA) met chain-of-thought voor progressieve verfijning
HM-RAG	Drie lagen: decompositie-agent → multi-source retrieval-agenten → beslissingsagent met consensus voting
R-Debater	Multi-agent debat met kennisbank voor het ophalen van precedenten en argumenten

Risico: coördinatiebrosheid. Kritiek veiligheidsprobleem: één kwaadaardige agent die retrieval manipuleert kan de groepsconsensus naar foute antwoorden sturen terwijl de "zekerheid" van het systeem juist toeneemt (Kraidia et al. 2026).

6. Graph & Knowledge Graph RAG Gebruikt gestructureerde representaties (nodes/edges) om entiteitsrelaties vast te leggen en multi-hop logische traversal mogelijk te maken — verder dan puur semantische chunk-gelijkenis.

Systeem	Kernmechanisme
GraphSearch	Dual-channel: semantische queries over chunktekst + relationele queries over graafstructuur
Spreading-activation GraphRAG	Automatisch geconstrueerde heterogene kennisgraaf; vermijdt LLM-gestuurde traversal
Agentic KG Crawler	Navigeert superseding-logica en multi-hop verwijzingen in enterprise-documenten (bijv. regelgeving)
InstructRAG	Graaf van instructie-paden gecombineerd met RL voor taakplanning

Risico: grafkwaliteit en -dekking zijn bepalend; ondiepe retrieval bij complexe queries als de graaf incompleet is.

7. Tool-using RAG agents Breidt retrieval uit naar buiten het statische corpus via aangeleerde tool-aanroepbeleid. Het systeem kan rekenmachines, API's, code-uitvoering en externe zoekopdrachten inzetten.

Systeem	Kernmechanisme
PEARL	Twee fasen: offline exploratie van geldige toolpatronen + online RL-fase
Chain-of-Abstraction (CoA)	Decodeert abstracte redeneerketens met placeholders; tools vullen specifieke kennis in
Proof-of-Use (PoU)	Dwingt echte evidence-grounding af via citaatvalidatie en perturbatie-gebaseerde rewards

Risico: **Tool-call hacking** — agenten kunnen rewards maximaliseren via oppervlakkig correcte tool-aanroepen *zonder* de opgehaalde evidence daadwerkelijk te gebruiken. PoU is ontworpen om dit te mitigeren.

3. Classic RAG vs Agentic RAG

Het fundamentele onderscheid

> **Classic RAG is een pijplijn. Agentic RAG is een controle-loop.**

Bij classic RAG volgt het systeem een vaste route van A naar B: één keer ophalen, één keer genereren. Bij agentic RAG beslist het systeem zelf welke route het neemt, hoe vaak het retrievet, en wanneer het voldoende bewijs heeft verzameld om te antwoorden.

Het kernprobleem dat agentic RAG oplost: bij complexe taken kan de kennisbehoefte pas duidelijk worden *tijdens* het redeneren, niet vooraf. Classic RAG haalt dan de verkeerde of onvolledige documenten op — en weet dat zelf niet.

Architectuurvergelijking

Classic RAG — lineaire pijplijn

```
Vraag
  ↓
Query → Vector DB → Top-k chunks ophalen (één keer)
  ↓
Context samenstellen
  ↓
LLM genereert antwoord + citaten
  ↓
Antwoord  [geen terugkoppeling mogelijk]
```

Agentic RAG — controle-loop

```
Vraag
  ↓
Controller analyseert: wat heb ik nodig?
  ↓
Retrieval ronde 1 → Critic evalueert: bewijs toereikend?
  ↓ Nee → verfijn query / wissel strategie / roep tool aan
Retrieval ronde 2 → Critic evalueert opnieuw
  ↓ Ja → voldoende geverifieerd bewijs
Antwoord genereren met traceerbare bronnen
```

Het LLM is niet langer *downstream* van retrieval — het *stuurt* retrieval als één van meerdere tools die het kan inzetten.

Vergelijkingstabel

Aspect	Classic RAG	Agentic RAG
Retrieval-rondes	Eén, vooraf vast	Meerdere, adaptief
Besluitvorming	Vooraf vastgelegd	Dynamisch door controller
Retrieval-strategie	Één vaste methode	Adaptief routeren per query
Multi-hop vragen	Zwak	Sterk
Zelfcorrectie	Afwezig	Ingebouwd via critic
Tool-gebruik	Nee	Ja (API's, code, web search)
Latentie	Voorspelbaar, laag	Variabel, 2–5× hoger
Tokenkosten	Laag	3–10× hoger
Foutopsporing	Eenvoudig	Complex: loop-falen, cascades
Veiligheidsrisico's	Beperkt	Hoger: compounding, poisoning
Evaluatie	Eindantwoord voldoende	Trajectory-level noodzakelijk

Concreet voorbeeld

Vraag: *"Wat is het risico van contract XY-2024 gelet op de nieuwe EU-regelgeving van maart 2025?"*

Classic RAG — zoekt semantisch op de combinatie, haalt top-5 chunks op, genereert een antwoord. Als de contractbepalingen en de regelgeving in aparte documenten staan, wordt het verband gemist. Het systeem weet niet dat het iets mist.

Agentic RAG — werkt iteratief: 1. *Stap 1*: Haal contract XY-2024 op → extraheer kernbepalingen → sla op in state 2. *Stap 2*: Formuleer gerichte query op basis van gevonden bepalingen → zoek EU-regelgeving 3. *Stap 3*: Critic evalueert: zijn er tegenstrijdigheden? Zo ja → zoek jurisprudentie of guidance documents 4. *Stap 4*: Critic bevestigt: voldoende en consistent bewijs → genereer risicoanalyse met citaat per claim

Wanneer welke kiezen?

Kies Classic RAG als: - Vragen in één retrieval-ronde op te lossen zijn (FAQ's, documentatie, productinfo) - Lage latentie vereist is (klantenservice, real-time toepassingen) - Kosten en voorspelbaarheid prioriteit hebben - De kennisbasis overzichtelijk en goed gestructureerd is - Audittrails eenvoudig moeten zijn

Kies Agentic RAG als: - Bewijs verspreid is over meerdere bronnen of documenten - Multi-hop redenering nodig is ("wat betekent A in het licht van B en C?") - Verificatie en cross-checking essentieel zijn (healthcare, juridisch, compliance, finance) - Het systeem externe tools moet aanroepen (calculators, API's, live data) - De kosten van een fout hoog zijn en zelfcorrectie de moeite waard maakt

Praktische vuistregel: *"Betaal niet voor loops tenzij je taak regelmatig mislukt in één retrieval-ronde."* Start altijd met Classic RAG, meet waar het faalt, en voeg pas dan agentic loops toe voor die specifieke gevallen.

Kosten en prestaties in perspectief

Agentic RAG levert bij complexe multi-hop vragen ~42% betere precisie dan classic RAG, maar kost 3–10× meer tokens en heeft 2–5× hogere latentie. Gartner verwacht dat 33% van enterprise-applicaties in 2028 agentic AI bevat — maar Classic (Hybrid) RAG blijft de productie-standaard voor het overgrote deel van de use cases. De twee patronen zijn complementair, niet concurrerend.

---

4. Componenten van een RAG-systeem

Een modern RAG-systeem bestaat uit zes kerncomponenten die in volgorde op elkaar aansluiten.

```
[Documenten]
     ↓
 1. Chunker
     ↓
 2. Embedding model
     ↓
 3. Vector database  (indexering)
     ↓
 4. Retriever  (hybride: vector + BM25)
     ↓
 5. Reranker
     ↓
 6. Generator (LLM)
```

Component 1: Chunking

Documenten worden opgesplitst in kleinere stukken die het model kan verwerken. De kwaliteit van chunking is doorslaggevend — de meeste RAG-projecten mislukken hier.

Strategieën:

- **Fixed-size chunking**: eenvoudig maar primitief — knipt midden in zinnen
- **Semantische chunking**: gebruikt embeddings om thematische breuken te detecteren via cosinus-afstand tussen zinnen
- **Small-to-Big (Parent-Document)**: kleine child-chunks (~100 tokens) worden geïndexeerd voor precisie; bij retrieval wordt het volledige parent-document teruggegeven aan het LLM

Best practice 2026: gebruik documentbewuste splitsing (per paragraph, markdown-header, of semantische grens). Voeg altijd metadata toe aan elke chunk: bron, auteur, datum, toegangsrechten.

Component 2: Embedding model

Het embedding model zet tekst om in een numerieke vector die de *betekenis* vastlegt. Dezelfde query én de documenten worden met hetzelfde model omgezet, zodat semantische gelijkenis meetbaar is.

Veelgebruikte embedding-modellen:

Model	Type	Sterk in
`text-embedding-3-large`	OpenAI API	Algemeen, hoge kwaliteit
`text-embedding-3-small`	OpenAI API	Kostenefficiënt
`bge-m3`	Open source	Meertalig, lokaal te draaien
`embed-v3` (Cohere)	API	Enterprise, lange context
Sentence Transformers	Open source	Flexibel, aanpasbaar

Aandachtspunt: bi-encoder embeddings zijn "lossy" — ze comprimeren complexe tekst in één vaste vector (bv. 1536 dimensies). Ze werken goed voor conceptuele gelijkenis, minder goed voor specifieke getallen of exacte termen.

Component 3: Vector database

Een vector database slaat embeddings op en doorzoekt die op basis van *semantische gelijkenis* — niet op exacte tekstovereenkomst zoals een traditionele database.

Teksten met vergelijkbare betekenis liggen **dicht bij elkaar** in de vectorruimte:

```
"Wat zijn de openingstijden?"  →  [0.23, -0.87, 0.41, ...]
"Wanneer is de winkel open?"   →  [0.21, -0.85, 0.44, ...]  ← dichtbij!
"De omzetbelasting bedraagt..."→  [0.67,  0.12, -0.93, ...] ← ver weg
```

Vector databases gebruiken **Approximate Nearest Neighbor (ANN)**-algoritmen voor snelle gelijkeniszoeking, ook in collecties van tientallen miljoenen vectoren.

Populaire ANN-indexen: - **HNSW** (Hierarchical Navigable Small World) — standaard in de meeste databases - **IVF** (Inverted File Index) — efficiënt bij zeer grote collecties

Populaire vector databases in 2026:

Database	Type	Bijzonderheid
Pinecone	Cloud-native, managed	Eenvoudig schalen, lage latency
Qdrant	Open source / cloud	Snel, rijke filtering
Weaviate	Open source / cloud	Ingebouwde hybride search
Milvus	Open source	Geschikt voor zeer grote schaal
Chroma	Open source	Lichtgewicht, ideaal voor prototyping
pgvector	PostgreSQL extensie	ACID-compliant, past in bestaande infra
Azure AI Search	Microsoft cloud	Ingebouwde hybride search + reranking
OpenSearch	AWS / open source	GPU-versneld, enterprise-grade

Trend 2026: beweging terug naar relationele databases met vectorextensie (pgvector). Voor organisaties met bestaande SQL-infrastructuur is dit vaak de meest pragmatische keuze.

Component 4: Retriever (hybride search)

Hybride retrieval is de standaard in 2026. Twee methoden worden parallel ingezet:

Methode	Werking	Sterk in
Dense / Vector search	Cosinus-gelijkenis op embeddings	Parafrasen, conceptuele vragen
Sparse / BM25	Exacte trefwoordovereenkomst	Productnummers, namen, juridische termen

De twee resultatenlijsten worden samengevoegd via **Reciprocal Rank Fusion (RRF)**: documenten die in beide lijsten hoog scoren worden sterk benadrukt.

Component 5: Reranker

De initiële retrieval haalt 50–100 kandidaat-chunks op. De **reranker** verfijnt dit naar de top 5–10 die daadwerkelijk in de prompt komen.

Een reranker gebruikt een **cross-encoder** die query én document *tegelijk* analyseert — nauwkeuriger dan een embedding-model, maar alleen toegepast op de top-100, niet op de gehele kennisbasis.

Populaire rerankers:

Model	Aanbieder
Cohere Rerank 3.5	Cohere API
BGE-Reranker-v2-m3	Open source
Jina Reranker v2	Jina AI

Zonder reranker verlies je 15–30% in antwoordkwaliteit.

Component 6: Generator (LLM)

Het LLM genereert het eindantwoord op basis van de opgehaalde en gerangschikte context.

Model	Aanbieder	Contextvenster
GPT-4o / GPT-5	OpenAI	128K–1M tokens
Claude Sonnet 4.6 / Opus 4.8	Anthropic	200K tokens
Gemini 1.5/2.0 Pro	Google	1M–2M tokens
Llama 3 / Mistral	Meta / Mistral	Open source, lokaal

Aandachtspunt: een groter contextvenster betekent niet dat je minder goed hoeft te retrieven. Te veel irrelevante context degradeert de antwoordkwaliteit ("lost in the middle"-probleem).

---

5. Evaluatie & veiligheid

Evaluatiemetrieken voor Classic RAG

Zonder meetinstrumenten degraderen RAG-systemen stilletjes als data evolueert.

Metriek	Wat het meet
Retrieval Recall	Worden de relevante documenten daadwerkelijk opgehaald?

Metriek	Wat het meet
Answer Faithfulness	Is het antwoord aantoonbaar gegrond in de opgehaalde context?
Answer Relevance	Beantwoordt het antwoord de gestelde vraag?

Frameworks: RAGAS, TruLens, LangSmith.

Evaluatie van Agentic RAG: trajectory-level

Evaluatie op alleen het eindantwoord is onvoldoende voor agentic RAG. Onderzoek (You et al. 2026) toont dat falen vaak zit in het redeneertraject — ketens die te vroeg afbreken (te weinig bewijs) of te ver doorschieten (irrelevante zijsporen) — niet in het eindantwoord als zodanig.

Nieuwe benchmarks meten tussenliggende capabilities per stap: - **RAGCap-Bench**: evalueert intermediate taken in agentic workflows; "slow-thinking" modellen met sterkere tussenliggende capabilities bereiken betere eindresultaten - **RAGalyst**: genereert synthetische QA-datasets en vindt dat prestaties sterk variëren per domein

Belangrijke bevinding: geen enkel embedding-model, LLM of hyperparameter-configuratie is universeel optimaal. Robuuste agentic RAG vereist domeinspecifieke afstemming.

Veiligheidsrisico's van agentic loops

Naarmate de loop langer wordt en het toezicht zwakker, nemen de systeemrisico's toe:

Risico	Beschrijving
Compounding hallucinations	Fouten in vroege retrieval-rondes propageren en versterken in latere rondes
Memory poisoning	De geheugenstatus van de agent wordt vergiftigd door foutieve of kwaadaardige informatie
Retrieval misalignment	Wat het model nodig heeft tijdens redeneren wijkt af van wat vooraf was opgehaald
Cascading tool vulnerabilities	Een fout in één tool-aanroep veroorzaakt kettingreacties in afhankelijke stappen
Multi-agent consensus failure	Gecoördineerde manipulatie stuurt meerdere agenten naar dezelfde foute conclusie terwijl de zekerheid van het systeem toeneemt
Tool-call hacking	Agent maximaliseert rewards via correcte tool-aanroepen zonder de opgehaalde evidence te gebruiken

Sterkere oversight-mechanismen — critics, citaatvalidatie, trajectory-logging — zijn bij agentic RAG geen optie maar noodzaak.

---

6. Hoe verloopt een RAG-project?

Een RAG-project volgt zes fasen. De meeste projecten mislukken niet door technologie, maar door slechte datakwaliteit en ontbrekende governance.

Fase 1: Discovery (½ dag – 1 week)

Doel: scope en architectuurkeuze bepalen.

Vragen die je beantwoordt: - Welke use cases lossen we op? (interne kennisbank, klantenservice, juridisch zoeken, ...) - Welke databronnen zijn beschikbaar en in welke staat? - Welke compliance-eisen gelden (toegangscontrole, AVG, auditeerbaarheid)? - Wat zijn de prestatiemetrieken voor succes?

Architectuurkeuze: begin met **Hybrid RAG** (vector + BM25 + reranking). Graph RAG of Agentic RAG kosten 3–5× meer en zijn alleen gerechtvaardigd bij complexe redeneer-vereisten.

Fase 2: Data Foundation (1–3 weken)

Dit is de meest kritieke fase — en waar de meeste projecten stranden.

- **Datakwaliteit**: verwijder ruis (navigatiemenu's van websites, kopteksten van PDF-scans, duplicaten).
- **Chunking-strategie**: kies semantische of parent-document chunking. Nooit vaste karakterblokken.
- **Metadata**: elke chunk krijgt metadata: bron-URL/bestand, auteur, datum, afdeling, vertrouwelijkheidsniveau.
- **Toegangscontrole**: implementeer rechten op chunk-niveau zodat gebruikers alleen zien wat ze mogen zien.

Fase 3: Baseline Pipeline (1–2 weken)

Bouw het volledige systeem en meet de prestaties. Dit geeft een referentiepunt voor alle verdere verbeteringen.

Minimal viable stack: - Embedding model (bv. `text-embedding-3-small`) - Vector database (bv. Chroma voor prototype, Qdrant voor productie) - Reranker (bv. Cohere Rerank) - LLM (bv. Claude Sonnet of GPT-4o) - Evaluatieraamwerk (bv. RAGAS)

Fase 4: Optimalisatie (doorlopend)

Voeg verbeteringen toe op basis van gemeten resultaten — niet op intuïtie.

Typische optimalisatiestappen in volgorde van impact:

1. Betere chunking → vaak de grootste winst
2. Hybride search (vector + BM25)
3. Reranker toevoegen (15–30% kwaliteitsverbetering)
4. Query-herformulering of -expansie
5. Metadata-filtering (zoek eerst op afdeling/datum/rechten, dan semantisch)
6. Agentic loops voor complexe meerstaps-vragen

Fase 5: Productionisatie (1–2 weken)

- **Monitoring**: traceer elke retrieval-ronde en generatie-stap (LangSmith, Arize, Langfuse)
- **Beveiliging**: bescherm de retrieval-laag tegen prompt-injection en corpus-vergiftiging
- **CI/CD voor de kennisbasis**: automatische re-indexering bij documentwijzigingen via change data capture of event triggers
- **Cloud-deployen**: Azure AI Search, AWS Bedrock, of GCP Vertex AI bieden managed RAG-stacks

Fase 6: Operationeel beheer (doorlopend)

> "RAG-systemen die niet actief worden onderhouden, verslechteren in plaats van verbeteren."

- Evalueer wekelijks/maandelijks op faithfulness en relevance
- Herindexeer bij grote wijzigingen in de kennisbasis
- Monitor datadrift (verouderde documenten die nog wel worden opgehaald)
- Breid gebruik uit naar andere workflows op basis van bewezen impact

De vijf grootste valkuilen

Valkuil	Gevolg	Oplossing
Verkeerde chunking	Antwoorden missen context of zijn incoherent	Semantische of parent-document chunking
Geen reranker	15–30% kwaliteitsverlies	Altijd een cross-encoder reranker toevoegen
Geen evaluatieraamwerk	Voortgang onmeetbaar	RAGAS of vergelijkbaar vanaf dag 1
Ontbrekende toegangscontrole	Datalekkage-risico	Rechten op chunk-niveau implementeren
Eenmalig project	Systeem degradeert stiekem	Doorlopend onderhoud inplannen

---

Samenvatting

RAG is in 2026 het **dominante patroon voor enterprise AI**. Het evolueert van een simpele vector-zoekoplossing naar een volwaardige *context-engine* — met agentic loops, graph-redeneren, multimodale retrieval, en ingebouwde governance.

De technologie is volwassen en de keuzes zijn duidelijk: begin klein met Hybrid RAG, investeer zwaar in datakwaliteit, meet alles op trajectory-niveau, en schaal incrementeel op naar Agentic of Graph RAG als de use case het rechtvaardigt. Beide patronen zijn complementair — Classic RAG voor de breedte, Agentic RAG voor de diepte.

Bronnen

Praktijkgerichte bronnen

- What Is RAG? How Retrieval-Augmented Generation Works in 2026 — Atlan
- RAG in 2026: Bridging Knowledge and Generative AI — Squirro
- 20 Advanced RAG Types to Know in 2026 — Turing Post
- The Complete Guide to RAG Architectures: From Naive to Agentic — Medium
- All you need to know about RAG in 2026 — AI with Aish
- Rerankers and Two-Stage Retrieval — Pinecone
- How Vector Databases Enable RAG: 9 Tools to Know in 2026 — Instaclustr
- RAG Enterprise Implementation Guide 2026 — TotalCloudAI
- Agentic RAG vs Classic RAG: From a Pipeline to a Control Loop — Towards Data Science
- Traditional RAG and Agentic RAG Key Differences Explained — PingCAP
- RAG vs Agentic RAG: Key Differences for 2026 — Kanerika
- Agentic RAG in 2026: Architecture Patterns — jobsbyculture
- RAG Implementation Roadmap — The Lean Product Studio
- Enterprise RAG Guide 2026 — Techment

Academische bronnen (via Elicit)

- Singh et al. (2025). *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG.* arXiv 2501.09136
- Mishra et al. (2026). *SoK: Agentic RAG — Taxonomy, Architectures, Evaluation, and Research Directions.* arXiv 2603.07379
- Asai et al. (2023). *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection.* ICLR.
- Verma et al. (2026). *ReflectiveRAG: Rethinking Adaptivity in Retrieval-Augmented Generation.* EACL Industry Track.
- Gangavarapu et al. (2025). *Evaluating Accuracy in LLMs: Benchmarking Corrective RAG vs. Naive RAG.* ICAD.
- Wei et al. (2025). *AlignRAG: An Adaptable Framework for Resolving Misalignments in Retrieval-Aware Reasoning.* arXiv.
- Wang et al. (2026). *RAGRouter-Bench: A Dataset and Benchmark for Adaptive RAG Routing.* arXiv.
- Nguyen et al. (2025). *MA-RAG: Multi-Agent RAG via Collaborative Chain-of-Thought Reasoning.* arXiv.
- Liu et al. (2025). *HM-RAG: Hierarchical Multi-Agent Multimodal Retrieval Augmented Generation.* ACM Multimedia.
- Kraidia et al. (2026). *RAG-Induced Failures in Multi-Agent LLM Debate.* ICETES 2026.
- Yang et al. (2025). *GraphSearch: An Agentic Deep Searching Workflow for Graph RAG.* arXiv.
- Ma et al. (2025). *Proof-of-Use: Mitigating Tool-Call Hacking in Deep Research Agents.* arXiv.
- You et al. (2026). *AgenticRAGTracer: A Hop-Aware Benchmark for Diagnosing Multi-Step Retrieval Reasoning.* arXiv.
- Gao et al. (2025). *RAGalyst: Automated Human-Aligned Agentic Evaluation for Domain-Specific RAG.* arXiv.