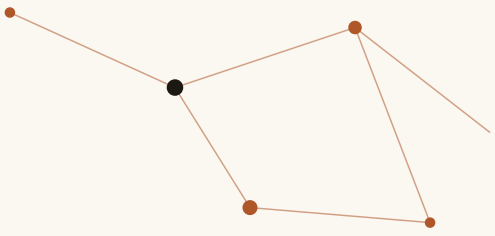
HET PRODUCTIE-AI-HANDBOEK

Van Prototype naar Productie

Cloud-AI, MLOps & LLMOps voor bouwers én beslissers

Tico van Gerner

EDITIE 2026 · NEDERLANDS · NIVEAU INTERMEDIATE



Inhoud

DEEL 6: BORGEN — VEILIGHEID, GOVERNANCE & DE EU-LAAG

DEEL 7: SAMENBRENGEN

APPENDICES

Hoofdstuk 1 — De prototype-naar-productie-kloof

DEEL 1: FUNDAMENTEN & DENKMODELLEN · NIVEAU: INTERMEDIATE

> **Builder** leest het hele hoofdstuk. **Beslisser** kan de codevooruitblik overslaan en focust op "Waarom pilots stranden" en het beslis-kader.

Het verraderlijke aan AI in productie

Een gewone microservice geeft een 200 of een 500. Werkt het, of werkt het niet — je weet het meteen, en als het stuk is, gaat er een pieper af.

Een LLM-systeem geeft een 200. Keurig geformatteerd. En een volledig fout antwoord. Niemand wordt gepiept. Niemand merkt het — tot de klant klaagt of de cijfers een week later wegzakken.

Dat is de kern van dit boek. De sprong van "het werkt in mijn notebook" naar "het draait betrouwbaar voor echte gebruikers" is geen laatste stapje van 10%. Het ís het werk. En het is precies waar de meeste AI-initiatieven stranden.

De kloof, in cijfers

In 2025 publiceerde MIT's Project NANDA het onderzoek *The GenAI Divide: State of AI in Business 2025* (300+ initiatieven, 52 interviews, 153 surveys). De kop die overal werd geciteerd: **95% van de generatieve-AI-pilots levert geen meetbare impact op de winst-en-verliesrekening.** Slechts 5% haalt echte waarde binnen.

Lees de nuance, want die is belangrijker dan de schok: het probleem is zelden het model. Het is **integratie en workflow** — AI die met minimale aanpassing in bestaande processen wordt geduwd. De analyse van ZenML over **1.200 productie-deployments** wijst dezelfde kant op: wat winnaars onderscheidt is niet het nieuwste model, maar **context-engineering, guardrails in de infrastructuur, rigoureuze evaluatie en gewone software-fundamenten.**

Onthoud die zin. Het is de rode draad van dit boek:

> **Engineering en evaluatie verslaan het nieuwste model.**

Waarom pilots stranden

Vier patronen komen telkens terug:

1. **Geen heldere baseline of doel.** "We doen iets met AI" is geen doel. Zonder nulmeting kun je achteraf niet aantonen dat het iets opleverde — en dus sneuvelt het budget.

2. **Data die niet klaar is.** Rommelige, verouderde of slecht-ontsloten kennis is de #1 stille killer. Een RAG-systeem is zo goed als de bron die het ophaalt.
3. **Geen evaluatie.** Omdat output niet-deterministisch is (zelfde vraag, ander antwoord), kun je niet leunen op "geeft het een 200?". Zonder eval weet je simpelweg niet of het werkt — en stuur je blind.
4. **Het demo-naar-dagelijks-werk-gat.** De technologie haalt de demo; de organisatie haalt de gewoonte niet. Change management, niet techniek, is vaak de bottleneck.

Wat de 5% anders doet

De winnaars uit het MIT- en ZenML-materiaal delen een nuchter profiel:

- Ze kiezen een **smalle, goed-gedefinieerde use case** in plaats van een breed "AI-platform".
- Ze halen de hoogste ROI uit **saaie back-office-automatisering** (document-review, support, risk-monitoring) — niet uit flitsende marketing-AI.
- Ze **bouwen niet alles zelf vanaf nul**: externe modellen/partners scoren beter dan in-house-bouwsels die het wiel heruitvinden.
- Ze **meten vanaf dag 1** (baseline + evaluatie), en behandelen het systeem als productiesoftware: geversioneerd, gemonitord, terugdraaibaar.

Beslis-kader: zelf bouwen, kopen, of een partner?

Situatie	Beste route
Standaardprobleem (chatbot, samenvatten, classificatie)	**Kopen / configureren** — gebruik een platform of SaaS; bouw niet na wat al bestaat.
Onderscheidend, kerncompetentie, eigen data	**Zelf bouwen** — maar met productie-discipline vanaf het begin.
Wel ambitie, weinig capaciteit	**Partner / launching-aanpak** — laat bouwen, houd eigenaarschap en kennis.

Vuistregel: bouw alleen zelf wat je écht onderscheidt. De rest koop of configureer je.

Wat "productie-klaar" minimaal betekent

Een vooruitblik op de rest van het boek. Productie-klaar is niet "het draait", maar minimaal:

- **versionering** van prompts én code (elke promptwijziging is een deployment);
- **geautomatiseerde tests + evaluatie** (weet of het werkt vóór het live gaat);
- **observability** (traces, kosten, kwaliteit — niet alleen status 200);
- **guardrails** (filters, PII, fallback) en **rollback** (terug kunnen als het misgaat).

Die vier kom je in Deel 3 t/m 6 in detail tegen.

Wat je in dit boek leert

We bouwen van fundament naar governed production: - **Deel 2** — kies en begrijp je cloud-AI-platform (AWS/Azure/GCP) en de kosten. - **Deel 3** — bouw de LLMOps-levenscyclus (prompts, RAG, agents, serving). - **Deel 4** — bewijs dat het werkt met evaluatie. - **Deel 5** — beheers het in productie (observability, kosten). - **Deel 6** — borg het (veiligheid, governance, EU AI Act, soevereiniteit). - **Deel 7** — breng alles samen in een capstone en een 90-dagen-roadmap.

Aan het eind heb je geen demo. Je hebt een systeem dat je durft te laten draaien.

Bronnen: MIT Project NANDA, "The GenAI Divide: State of AI in Business 2025" (jul 2025); ZenML LLMOps-database (1.200 productie-deployments); Gartner (2026).

Hoofdstuk 2 — Van DevOps naar MLOps, LLMOps en AgentOps

DEEL 1: FUNDAMENTEN & DENKMODELLEN · NIVEAU: INTERMEDIATE

> **Builder**: let op de "minimale toolchain". **Beslisser**: het kader "welke Ops heb je echt nodig" is voor jou geschreven.

Vier lagen, elk met een eigen zwaartepunt

De praktijk van "AI in productie" is geëvolueerd in vier lagen. Ze stapelen op elkaar; je gooit de vorige niet weg.

- **DevOps** — software betrouwbaar bouwen en uitrollen: versiebeheer, CI/CD, automatisering, monitoring. De basis onder alles.
- **MLOps** — de levenscyclus rond *het model*: data prepareren, trainen, versioneren, deployen, **drift monitoren**, hertrainen. Het draaipunt is het modelgewicht.
- **LLMOps** — je begint met een *voorgetraind* model en bouwt eromheen: **prompts, RAG, evaluatie, guardrails, kosten**. Het model is hier het *minst* veranderende onderdeel.
- **AgentOps** — als het systeem zelf beslissingen neemt: tool-gebruik, geheugen, meerstaps-trajecten. Je monitort niet één antwoord, maar een hele reeks acties.

De vier denkmodel-verschuivingen

Wie van MLOps (of DevOps) naar LLMOps gaat, moet vier dingen afleren:

1. **Output is niet-deterministisch.** Dezelfde input geeft niet hetzelfde antwoord. "Klopt de output?" is geen ja/nee meer, maar een score op een schaal. Daarom is evaluatie-infrastructuur (Deel 4) geen luxe.
2. **Prompts zijn code.** Niet het modelgewicht verandert het vaakst, maar je prompt. Elke promptwijziging is een deployment: versioneren, testen, kunnen terugdraaien. Een provider die zijn model stilletjes update, kan je prompt van gisteren breken.
3. **Hertrainen is het laatste redmiddel.** Eerst prompt, dan RAG, dan tools, dan guardrails. Fine-tunen of trainen doe je pas als die niets meer opleveren (zie H14). Dat is een fundamenteel andere economie dan klassieke MLOps.
4. **De testpiramide is platter en breder.** Unit-tests blijven voor de niet-AI-onderdelen, maar de nadruk verschuift naar integratie-, gedrags- en statistische tests op de output.

Beslis-kader: welke "Ops" heb je écht nodig?

De meeste teams die "AI in productie" willen, hebben **geen volledige MLOps met training** nodig. Ze hebben LLMOps op een managed platform nodig.

Jouw situatie	Wat je nodig hebt
Je gebruikt voorgetrainde modellen (Claude, GPT, Gemini) via een platform	**LLMOps** — prompts, RAG, eval, observability, guardrails. Geen GPU-training.
Je traint/finetunet eigen modellen op eigen data	**MLOps + LLMOps** — inclusief experiment-tracking, model-registry, GPU-infra.
Je bouwt autonome agents die tools aanroepen	**LLMOps + AgentOps** — plus trajectorie-evaluatie en strakke guardrails.

Conclusie voor de meeste lezers: investeer eerst in LLMOps-discipline. Klassieke MLOps-zwaarte (GPU-scheduling, training-pipelines) komt pas in beeld als je écht zelf traint.

De minimale LLMOps-toolchain, per fase

Je hoeft niet alles tegelijk te hebben. Dit is de minimale set, gekoppeld aan de delen waar we erop ingaan:

Fase	Functie	Voorbeeldtools	Deel
Bouwen	prompt-management	LangSmith, PromptLayer	3
Bouwen	routing/gateway	LiteLLM, Portkey	2/5
Bouwen	serving	vLLM, BentoML	3
Bewijzen	evaluatie	Ragas, DeepEval, Promptfoo	4
Beheersen	observability	Langfuse, Arize Phoenix	5
Borgen	guardrails	Guardrails AI, NeMo Guardrails, Lakera	6

Waarom dit hoofdstuk ertoe doet

Bijna elke mislukte pilot uit Hoofdstuk 1 is te herleiden tot één van deze verschuivingen niet maken: monitoren alsof een 200 genoeg is, prompts behandelen als losse tekst in plaats van als code, of meteen willen trainen terwijl een betere prompt of schonere RAG het probleem had opgelost. De rest van het boek is in feite deze vier verschuivingen, één voor één, in de praktijk gebracht.

Bronnen: ZenML "MLOps vs LLMOps"; MachineLearningMastery "Roadmap for Mastering LLMOps 2026"; MDPI "Transitioning from MLOps to LLMOps" (2025).

Hoofdstuk 3 — De productie-GenAI-stack op één pagina

DEEL 1: FUNDAMENTEN & DENKMODELLEN · NIVEAU: INTERMEDIATE

> **Builder**: dit is je bouwtekening voor de rest van het boek. **Beslisser**: gebruik de "bouwen vs kopen per laag"-tabel als inkoop- en scopekader.

Eén plaatje om de rest van het boek aan op te hangen

Een productie-GenAI-systeem is geen "model met een API ervoor". Het is een stapel lagen, elk met een eigen taak. Ken je de lagen, dan weet je wat je nog mist — en in welk deel van dit boek je het oplost.

Van gebruiker naar model en terug:

1. **Gateway & routing** — één gecontroleerde ingang naar meerdere modellen; complexiteit-routing, fallbacks, rate-limits, kostentelling. (*Deel 2/5*)
2. **Prompt-laag** — versioneerde prompts en templates; de "code" die je het vaakst wijzigt. (*Deel 3*)
3. **Retrieval / RAG** — je eigen kennis ophalen: chunking, embeddings, vector-database, re-ranking. (*Deel 3*)
4. **Orkestratie & agents** — meerstaps-logica, tool-gebruik, geheugen, MCP. (*Deel 3*)
5. **Model-serving** — het model draaien of aanroepen: batching, caching, latency. (*Deel 3*)
6. **Evaluatie** — meten of de output deugt, offline én online. (*Deel 4*)
7. **Observability** — traces, kosten, kwaliteit; zien wat er gebeurt. (*Deel 5*)
8. **Guardrails & veiligheid** — in/output-filters, PII, jailbreak-verdediging. (*Deel 6*)
9. **Governance & data** — toegang, audit trails, AVG/EU AI Act, dataresidentie. (*Deel 6*)

De onderste twee lagen (8 en 9) zijn geen "extra's achteraf". Ze horen vanaf de eerste regel in het ontwerp — precies wat de 5% uit Hoofdstuk 1 wél doet.

Bouwen vs. kopen — per laag

Je bouwt niet elke laag zelf. Een managed cloud-platform (Deel 2) levert een groot deel kant-en-klaar; jij voegt de onderscheidende lagen toe.

Laag	Meestal kopen/configureren	Meestal zelf bouwen
Gateway/ routing	platform of gateway (LiteLLM/ Portkey)	routing-logica op jouw use case
Prompt-laag	tool voor versionering	de prompts zelf
RAG	vector-DB, embeddings	chunking + retrieval afgestemd op jóuw data
Serving	platform-inferentie	alleen self-hosten bij schaal/soevereiniteit
Evaluatie	framework (Ragas/DeepEval)	jouw golden set + metrics
Observability	platform (Langfuse e.a.)	je dashboards/alerts
Guardrails	guardrail-libraries	beleid en drempels
Governance	platform-controls	je AVG/AI-Act-proces

Vuistregel uit Hoofdstuk 1, nu concreet: **koop de generieke lagen, bouw de onderscheidende** (meestal: je RAG-data, je prompts, je evaluatie).

De kleinst mogelijke productie-stack

Begin je net? Dit is het minimum dat al "productie-klaar" mag heten:

- één **gateway** (ook al is het LiteLLM voor kostentelling en fallback);
- **geversioneerde prompts**;
- **RAG** met één vector-DB als je eigen kennis nodig is;
- een **golden test set + eval** die in CI draait;
- **observability** vanaf dag 1 (achteraf inbouwen is veel duurder);
- minimale **guardrails** (PII-filter, output-validatie) en een **rollback**.

Alles daarboven (multi-model routing, semantische cache, agent-trajectorie-eval, FRIA) voeg je toe wanneer schaal of regelgeving erom vraagt.

Hoe de rest van het boek deze kaart invult

Elk volgend deel pakt één of twee lagen beet en maakt ze productie-klaar. Aan het eind (Deel 7) zet je ze in één referentiearchitectuur en een capstone bij elkaar. Houd dit hoofdstuk bij de hand als kaart — dan verdwaal je niet in de details.

Bronnen: Truefoundry "LLMOps Architecture"; Caylent "LLMOps on AWS reference architecture"; ZenML (1.200 deployments).

Hoofdstuk 4 — Waarom een managed platform, niet een kale API

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

> **Builder**: dit verklaart welke productie-zorgen het platform van je overneemt. **Beslisser**: dit is het "waarom we hiervoor betalen"-verhaal.

Het gat tussen `openai.chat()` en een productiesysteem

Een prototype roept rechtstreeks een model-API aan met een API-sleutel in je code. Dat werkt — tot het de productie in moet. Dan heb je opeens nodig: identiteit en toegang (wie mag wat), netwerk-isolatie (geen verkeer over het open internet), audit logging (wie deed wat, wanneer), beheerde RAG, guardrails, en agent-orkestratie.

Dat is precies wat een **managed AI-platform** levert: een productielaag bovenop modelinferentie. AWS Bedrock, Azure OpenAI / Microsoft Foundry en Google Vertex AI lossen alle drie dezelfde klasse problemen op — ze verschillen in modelcatalogus, ecosysteem-integratie en compliance, niet in hun bestaansreden.

Wat een platform je uit handen neemt

- **Identiteit & autorisatie** — IAM (AWS), Entra ID (Azure), IAM (GCP) i.p.v. een losse sleutel.
- **Netwerk & isolatie** — private endpoints/VPC, je data verlaat je omgeving niet onnodig.
- **Audit & logging** — elke aanroep traceerbaar (CloudTrail, Purview, Cloud Audit Logs).
- **Beheerde RAG** — knowledge bases / search zonder zelf een retrieval-stack te bouwen.
- **Guardrails** — filters voor schadelijke content, PII, jailbreaks (Deel 6).
- **Agent-orkestratie** — agents met geheugen, sessies en tool-gebruik.

De ongemakkelijke waarheid: de keuze is vaak al gemaakt

In de praktijk kiest bijna elke organisatie het platform dat bij haar bestaande cloud past. Zit je data in AWS? Bedrock. Draai je op Microsoft 365 en Azure? Azure OpenAI. Leeft je data in BigQuery? Vertex. Dat is geen luiheid — het scheelt een berg integratie-, identiteits- en compliance-werk.

De volgende drie hoofdstukken nemen elk platform door, daarna (H8) geven we een beslisboom voor wie wél vrij kan kiezen, en hoe je lock-in beperkt.

Bron: MyEngineeringPath "Cloud AI Platforms 2026"; EPCGroup; Reintech (2026).

Hoofdstuk 5 — AWS Bedrock

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

Waar Bedrock voor staat

Bedrock is AWS' "foundation model gateway": één gouverneerde toegang tot de **breedste modelcatalogus** van de drie — Anthropic Claude, Meta Llama, Mistral, Cohere, AI21, Stability en Amazon's eigen Titan/Nova (30+ modellen). Sterkste keuze voor AWS-native organisaties met veel data in S3.

De vier bouwstenen die je moet kennen

- **Model access** — kies per use case een model; vraag toegang aan per model/regio. Multi-model is de norm; je zit niet vast aan één vendor.
- **Bedrock Agents** — agents die taken plannen en tools/Lambda's aanroepen.
- **Knowledge Bases** — beheerde RAG: koppel S3-bronnen, Bedrock doet chunking, embeddings en retrieval.
- **Guardrails** — herbruikbare filters (geweigerde onderwerpen, PII-maskering, hallucinatie-checks) die je over modellen heen legt.

Identiteit, governance, prijs

- **Identiteit/governance**: IAM + CloudTrail (audit) + Macie (gevoelige data). Past naadloos als IAM al je bron-van-waarheid is.
- **Prijs**: per token (input/output), plus aparte posten voor guardrails, knowledge bases en logging. **Provisioned Throughput** voor voorspelbare, hoge volumes. Let op (H9): de modelprijs is niet de platformprijs.

Aandachtspunten in de praktijk

- Modelbeschikbaarheid verschilt per **regio** — check dit vóór je een regio kiest (ook i.v.m. EU-data, Deel 6).
- `boto3`-patronen: vertrouwd als je team AWS kent, maar iets meer boilerplate dan een gespecialiseerde AI-SDK.

Kies Bedrock als…

…je AWS-native bent, meerdere modellen wilt afwegen (vooral Claude), en je data al in AWS staat.

Bron: EPCGroup "Bedrock vs Foundry vs Vertex 2026"; Reintech; AgileSoftLabs.

Hoofdstuk 6 — Azure OpenAI / Microsoft Foundry

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

Waar Azure voor staat

Microsoft Foundry (voorheen Azure AI Studio) is het geünificeerde platform; **Azure OpenAI** levert daarbinnen de diepste integratie met de **GPT-familie** (incl. de nieuwste reasoning-modellen). Sterkste keuze voor Microsoft-centrische organisaties (Entra ID, Microsoft 365, Purview).

Wat het onderscheidt

- **OpenAI-partnerschap** — productiewaardige GPT-toegang met enterprise-SLA's; structureel de eerste die nieuwe OpenAI-modellen krijgt.
- **Foundry Agents** — een agent = model + instructies + tools; Foundry neemt de operationele last (persistentie, governance) over.
- **Identiteit/governance** — Entra ID + Purview + (Agent 365): voor Microsoft-huizen ongeëvenaard strak geïntegreerd.
- **M365-integratie** — copilots in Word/Excel/Teams via Copilot Studio, met respect voor bestaande rechten.

Prijs & governance

Tokengebruik plus commitment-/provisioned-capaciteit en omliggende Azure-diensten. Governance is hier het verkoopargument: identiteit, dataresidentie-controls en auditing zitten in het Microsoft-weefsel (belangrijk voor Deel 6: EU Data Boundary).

Aandachtspunten

- Je zit dichter op het **OpenAI-modellandschap**; minder breed dan Bedrock, maar dieper geïntegreerd.
- Let op de **EU Data Boundary**-nuances (H29): "Flexible Routing" kan pieken buiten de EU verwerken; Anthropic-modellen vallen standaard buiten de boundary.

Kies Azure/Foundry als…

…je een Microsoft-huis bent, GPT-modellen wilt, en governance via Entra/Purview je belangrijkste eis is.

Bron: ESPC; Smartbridge "Azure AI Foundry"; WindowsForum (GPT-5.x in Foundry); Microsoft.

Hoofdstuk 7 — Google Vertex AI / Gemini Enterprise Agent Platform

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

Waar Vertex voor staat

Op Cloud Next 2026 is Vertex AI gaan heten **Gemini Enterprise Agent Platform** (een herpositionering; de facturatie bleef gelijk). Het levert eerste-hands **Gemini**-toegang plus **Model Garden** (200+ modellen, incl. Claude, Llama, Mistral) en de diepste **BigQuery**-integratie. Sterkste keuze voor GCP-native, data-zware en ML-first teams.

De bouwstenen

- **Model Garden + OpenAI-compatibele endpoint** — 200+ modellen; bestaande OpenAI-code porten zonder herschrijven.
- **Agent Builder + Agent Engine** — productie-agents met persistent geheugen, sessies en een beheerde runtime.
- **Vertex AI Search / Vector Search / RAG Engine** — enterprise-search en end-to-end RAG.
- **Media** — Imagen, Veo (video), Chirp (spraak).

Prijs (let op de optelsom)

Per-token voor de modellen, plus **aparte posten** voor Agent Builder-queries, Agent Engine-runtime, sessies, geheugen en search. Eén gebruikersvraag aan een agent kan zo **vier facturabele events** raken. Sinds feb 2026 worden Code Execution, Sessions en Memory Bank apart beprijsd. Ruime gratis tier + $300 startkrediet.

Aandachtspunten

- De SDK is ML-workflow-gericht en integreert sterk met BigQuery — fijn voor data-zware pipelines.
- EU-residentie: regio **europe-west4 (Nederland)** beschikbaar (Deel 6).

Kies Vertex als…

…je op GCP zit, je data in BigQuery leeft, je Gemini wilt, of je multimodale generatie (Imagen/Veo) nodig hebt.

Bron: CloudZero/Finout (Vertex pricing 2026); UI Bakery "Agent Builder 2026"; cloud.google.com.

Hoofdstuk 8 — Platformkeuze zonder lock-in

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

> Dit hoofdstuk is het beslis-kader van Deel 2. Builder: let op de gateway-sectie.

Een eerlijke beslisboom

1. **Waar staat je data en identiteit al?** AWS → Bedrock; Microsoft/M365 → Azure/Foundry; GCP/BigQuery → Vertex. Voor de meeste teams stopt de beslissing hier — en terecht.
2. **Welk model is leidend?** Claude-first → Bedrock of Vertex; GPT-first → Azure; Gemini/multimodaal → Vertex.
3. **Wat is je zwaarste eis?** Breedste modelkeuze → Bedrock; governance/Microsoft-integratie → Foundry; data-zwaar/ML-first → Vertex.
4. **Welke compliance/soevereiniteit?** Zie Deel 6 — dit kan de keuze overrulen.

Lock-in beperken (ook als je één platform kiest)

- **Abstraheer achter een gateway.** Tools als **LiteLLM** of **Portkey** geven je één interface naar alle providers; van model wisselen wordt een config-wijziging, geen herschrijving.
- **Houd prompts, RAG-data en evaluaties platform-onafhankelijk.** Dat zijn jouw assets; laat ze niet vastgroeien aan één SDK.
- **Gebruik open standaarden** waar mogelijk (OpenAI-compatibele endpoints, OpenTelemetry voor observability).

Multi-model gateway in de praktijk

Een gateway levert meteen drie productievoordelen die je sowieso nodig hebt (Deel 5): **kostentelling**, **fallback/failover** bij uitval, en **complexity-routing** (simpele vraag → goedkoop model). Je bouwt 'm dus niet alleen tegen lock-in, maar voor betrouwbaarheid en kosten.

De valkuil

"We doen multi-cloud voor de zekerheid" klinkt verstandig maar verdubbelt vaak de complexiteit zonder echte winst. Kies één hoofdplatform, houd je assets portabel, en zet een gateway ertussen. Dat is 90% van de lock-in-bescherming voor 10% van het werk.

Bron: AgileSoftLabs; MachineLearningMastery (LiteLLM/Portkey); EPCGroup.

Hoofdstuk 9 — Kosten & FinOps voor GenAI

DEEL 2: HET CLOUD-AI-FUNDAMENT · NIVEAU: INTERMEDIATE

> Beslisser: dit is je budget- en verrassingen-voorkomen-hoofdstuk. Builder: de hefbomen zijn concreet en meetbaar.

Tokenprijs ≠ platformkosten

De grootste budgetfout: begroten op de gepubliceerde tokenprijs. De échte rekening bevat verborgen posten, en die verschillen per platform: - **Bedrock** — aparte kosten voor guardrails, knowledge bases, logging, omliggende AWS-diensten. - **Vertex** — datapijplijnen, opslag, evaluatie, retrieval, én per-event agent-kosten (sessies, geheugen). - **Azure** — commitment-/provisioned-capaciteit en omliggende Azure-diensten.

Drie platforms, drie "kostentalen". Wie ze wil vergelijken, normaliseert met **FOCUS** (de FinOps-standaard) naar gelijke eenheden.

De vier hefbomen die echt werken

1. **Modelkeuze** — gebruik het kleinste model dat de kwaliteitseis haalt; schaal pas op als het meetbaar tekortschiet.
2. **Routing** — stuur simpele vragen naar goedkope modellen, reasoning naar dure. Onderbouwd: **RouterBench/LLMRouterBench** laten tot **~32% kostenbesparing zonder kwaliteitsverlies** zien; een IBM-router versloeg GPT-4 én bespaarde ~5 cent per vraag.
3. **Caching** — semantische cache vangt herhaalde/gelijkende vragen. Praktijk (r/LLMDevs): **~40% cache-hit ≈ $3k/maand bespaard**; vendor-claims lopen op tot ~73% (behandel als bovengrens).
4. **Batching & streaming** — batch niet-urgente taken (veel goedkoper per request); stream voor gevoelsmatige snelheid en vroegtijdig afbreken.

De lange-termijn-waarschuwing

Gartner (jan 2026) verwacht dat de **GenAI-kosten per resolutie tegen 2030 boven $3** uitkomen — duurder dan een offshore-agent. Goedkoop-per-interactie is dus niet permanent. Anker je business case op **resolutie en retentie**, niet op "AI is goedkoop".

Minimale FinOps-hygiëne

Meet vanaf dag 1: kosten per request, tokengebruik-trend, kosten per opgeloste taak. Zet per-gebruiker/per-app-limieten en budget-alerts (Deel 5). Zonder meting groeien agent-workflows non-lineair in kosten.

Bron: DigiUsher/Finout (FinOps 2026, FOCUS); RouterBench (Hu et al. 2024) & LLMRouterBench (2026); Gartner (jan 2026); r/LLMDevs.

Hoofdstuk 10 — Promptmanagement & versionering

DEEL 3: BOUWEN — DE LLMOPS-LEVENSCYCLUS · NIVEAU: INTERMEDIATE

> **Builder**: prompts horen uit je code, in een register. **Beslisser**: dit is change-management voor AI-gedrag.

Prompts zijn deployments

In klassieke software verander je code en deploy je. In LLMOps verander je vooral je **prompt** — en elke promptwijziging is net zo goed een deployment. Een herformulering die in een test schoner leek, kan in productie randgevallen breken. Erger: je modelprovider kan het onderliggende model stilletjes updaten, waardoor je prompt van vorige week ineens slechter presteert. Als je prompts los in je code slingeren, zie je dat niet aankomen en kun je niet terug.

Behandel de prompt als een versioneerd artefact

De eenheid die je versioneert is niet alleen de prompttekst, maar **prompt + model + parameters** samen. Leg vast: - welke promptversie, welk model (en versie), welke temperatuur/max-tokens; - wie de wijziging deed en waarom; - de evaluatiescore vóór en na (zie Deel 4).

Zo wordt elke wijziging traceerbaar, testbaar en terugdraaibaar — de drie dingen die een 200-met-fout-antwoord helpen voorkomen.

Werkwijze in de praktijk

- Bewaar prompts **buiten je applicatiecode** (een prompt-register of -bestand met versies). Tools: LangSmith, PromptLayer, of simpel: versiebeheer in Git met een duidelijke structuur.
- **A/B-test prompts op twee assen**: kwaliteit én token-efficiëntie. Een kortere prompt met gelijke kwaliteit is directe kostenbesparing (Deel 9).
- **Koppel elke promptwijziging aan een eval-run** in CI (Deel 4): geen merge zonder dat de scores minstens gelijk blijven.
- **Maak rollback triviaal**: vorige promptversie terugzetten moet één handeling zijn.

Governance-kader

Wie mag prompts wijzigen in productie? Voor klantgerichte of risicovolle systemen: dezelfde discipline als code — review, audit-trail, en een log van wie wat wijzigde. Dit is ook

relevant voor de EU AI Act-documentatieplicht (Deel 6): je moet kunnen aantonen hoe je systeem zich gedraagt en hoe je dat beheert.

De kernfout om te vermijden

"Even snel de prompt aanpassen in productie" zonder versie, test of log is de LLMOps-variant van rechtstreeks in productie coderen. Het werkt — tot het niet meer werkt en niemand weet wat er veranderde.

Bron: MachineLearningMastery (prompts-as-code); DZone "LLMOps Principles"; ZenML.

Hoofdstuk 11 — RAG in productie

DEEL 3: BOUWEN — DE LLMOPS-LEVENSCYCLUS · NIVEAU: INTERMEDIATE

> **Builder**: dit is de meest voorkomende productie-LLM-vorm. **Beslisser**: onthoud "schone, actuele kennisbank met een eigenaar" — dat is de winst of het verlies.

De meeste productie-LLM's zijn RAG-systemen

Niet een kale chatbot, maar Retrieval-Augmented Generation: de gebruiker stelt een vraag, het systeem haalt relevante stukken uit jouw kennis op, en het model antwoordt **gegrond** op die stukken. Bouwen is relatief makkelijk; weten of het klopt is het moeilijke deel (Deel 4).

De pijplijn — en waar hij breekt

1. **Ingest & extractie** — documenten inlezen (PDF's, pagina's). Rommelige extractie = rommelige antwoorden.
2. **Chunking** — in stukken knippen. Te groot: ruis; te klein: context weg. Dit is de meest onderschatte knop.
3. **Embeddings** — stukken naar vectoren; modelkeuze beïnvloedt retrieval-kwaliteit.
4. **Vector-database** — opslaan en doorzoeken (pgvector/Supabase, Pinecone, Redis, Vertex/Bedrock managed).
5. **Retrieval + re-ranking** — de juiste stukken bovenaan; re-ranking verhoogt precisie.
6. **Generatie** — het model antwoordt op basis van de opgehaalde context.

De war-stories uit de praktijk (r/Rag) wijzen telkens naar dezelfde plekken: extractie, chunking, vector-DB-keuze, en — het belangrijkst — **de kennisbank vers houden**. Het terugkerende inzicht: *"een schone, bijgewerkte kennisbank en iemand die hem onderhoudt — dat is het geheim."*

Praktijk

- Begin met een **eenvoudige baseline** en meet retrieval-kwaliteit vóór je gaat optimaliseren.
- **Semantische cache** voor herhaalde vragen (kosten + latency, Deel 9).
- Behandel chunking en re-ranking als **experimenten met meetbare uitkomst** (context precision/recall, Deel 4), niet als onderbuik.

Bouwen vs. kopen

Managed RAG (Bedrock Knowledge Bases, Vertex RAG Engine, Azure AI Search) bespaart je de plumbing en is prima voor standaardgevallen. Zelf bouwen loont als je retrieval echt

onderscheidend moet zijn voor jóuw data. De data-afstemming bouw je sowieso zelf — dat is het stuk dat niemand voor je kan kopen.

Brug naar Deel 4

RAG zonder evaluatie is gokken. De vier RAG-metrics (faithfulness, answer relevancy, context precision, context recall) vertellen je of het probleem in de *retrieval* of de *generatie* zit — onmisbaar om gericht te verbeteren.

Bron: r/Rag "war stories"; r/AI_Agents ("clean KB is the secret"); MachineLearningMastery (RAG-fase); Evidently RAG-guide.

Hoofdstuk 12 — Agents, tools & het Model Context Protocol (MCP)

DEEL 3: BOUWEN — DE LLMOPS-LEVENSCYCLUS · NIVEAU: INTERMEDIATE

> **Builder**: hier komt jouw MCP-edge binnen. **Beslisser**: het kader "wanneer wél/geen agent" voorkomt dure complexiteit.

Wat een agent is (en niet is)

Een agent is een model + instructies + **tools** + **geheugen**, dat een taak analyseert, stappen plant, tools aanroept en bijstuurt op basis van resultaten. Het verschil met een gewone keten: de agent bepaalt zelf de volgorde en wanneer hij klaar is.

Dat is krachtig én gevaarlijk: meer autonomie betekent meer manieren om fout te gaan (loops die je budget opvreten, verkeerde tool-aanroepen). Vandaar **AgentOps** (Deel 4/5): je evalueert niet één antwoord, maar het hele **traject** — koos de agent de juiste tools, in de juiste volgorde, en voltooide hij de taak?

Het Model Context Protocol (MCP)

MCP is de opkomende standaard om tools en data op een uniforme manier aan modellen/agents aan te bieden. In plaats van voor elke integratie maatwerk te bouwen, expose je je systeem via een **MCP-server**; elke MCP-compatibele agent (Claude, en steeds meer andere) kan er dan mee werken.

Waarom dit ertoe doet: het ontkoppelt je applicatie van één AI-leverancier en maakt je systeem "agent-native" — klaar voor hoe teams straks werken. Het zelf bouwen van MCP-servers is op dit moment een schaarse, onderscheidende vaardigheid.

Praktijk

- Begin met **één duidelijke tool** en een nauwe taak; breid pas uit als de evaluatie het rechtvaardigt.
- Bouw **harde grenzen** in: max. aantal stappen, time-outs, kostenplafond per sessie (anders krijg je de "loop die je daghuur opmaakt").
- Log het **volledige traject** (welke tool, welke input/output) — onmisbaar voor debuggen en eval.

Wanneer wél/geen agent?

Situatie	Keuze
Vast, voorspelbaar proces (1–3 stappen)	**Geen agent** — een simpele keten is goedkoper en betrouwbaarder.
Open taak, wisselende stappen, tool-gebruik	**Agent** — met strakke guardrails.
"Het is hip"	**Geen reden** — agentische complexiteit zonder noodzaak is een bekende valkuil.

De nuchtere praktijkles: veel "agentic RAG" is hype; een schone kennisbank en een goede keten verslaan vaak een ingewikkelde agent.

Bron: ideas2it (AgentOps); r/AI_Agents; MachineLearningMastery; Anthropic MCP.

Hoofdstuk 13 — Routing, serving & latency

DEEL 3: BOUWEN — DE LLMOPS-LEVENSCYCLUS · NIVEAU: INTERMEDIATE

> **Builder**: concrete prestatie-hefbomen met echte cijfers. **Beslisser**: self-host vs. managed is een kosten/soevereiniteitskeuze.

Aanroepen of zelf draaien?

Voor de meeste teams: roep het model aan via een platform (Deel 2). Zelf-hosten (vLLM, BentoML, Baseten) wordt interessant bij hoog volume, strakke latency-eisen of soevereiniteit (Deel 6) — maar je neemt dan GPU-beheer erbij, en GPU's zijn whole-or-nothing en 10–50× duurder dan CPU.

De hefbomen die echt werken

- **Batching.** static (offline, voorspelbaar), continuous (real-time chat), micro (sub-100ms). Continuous batching is de standaard voor interactieve apps.
- **KV-cache.** Geheugen is vaak eerder de bottleneck dan compute; KV-cache-optimalisatie is fundamenteel. Praktijk: vLLM met paged attention bracht een opstelling van **11s → 3s latency en ~10× throughput**.
- **Caching.** Semantische cache vangt gelijkende vragen (kosten + latency, Deel 9).
- **Routing & cascades.** Simpele vraag → klein/goedkoop model; alleen complexe vragen naar het grote model. "Routing bespaart meer dan de meeste model-trucs."
- **Streaming.** Verbetert de gevóelde snelheid en laat je slechte antwoorden vroeg afbreken.
- **Fallbacks.** Primary/secondary provider met automatische failover bij uitval (Deel 5).

Volgorde van optimaliseren

Begin niet met exotische trucs. De volgorde die in de praktijk loont: kleinste voldoende model → caching → routing/cascades → batching → pas daarna self-host/quantisatie. Meet bij elke stap (latency, throughput, kosten per request).

Self-host vs. managed

Managed is sneller en goedkoper te starten; self-host loont bij schaal, latency of soevereiniteit. Maak het een **meetbare** beslissing (kosten per 1.000 requests, p95-latency), geen geloofskwestie.

Bron: ZenML "Optimizing LLM Performance and Cost"; Redis LLMOps-guide; vamsikd "LLM Inference Optimization"; r/LLMDevs.

Hoofdstuk 14 — Fine-tunen vs. prompten vs. RAG

DEEL 3: BOUWEN — DE LLMOPS-LEVENSCYCLUS · NIVEAU: INTERMEDIATE

> Dit is het kosten/ROI-beslis-kader van Deel 3. Builder: gebruik de beslisboom voordat je GPU-uren verbrandt.

De goedkoopste oplossing eerst

Een terugkerend denkmodel uit dit boek: het model is het minst veranderende onderdeel, en **hertrainen is het laatste redmiddel**. Klim de ladder van goedkoop naar duur:

1. **Prompt** — de meeste problemen los je op met een betere prompt (instructies, voorbeelden, formaat). Goedkoop, direct, terugdraaibaar.
2. **RAG** — als het probleem *kennis* is (actuele, bedrijfsspecifieke of veranderende informatie), voeg retrieval toe in plaats van te trainen.
3. **Tools/agents** — als het probleem *actie* of *berekening* is, geef het model tools.
4. **Fine-tunen** — pas als prompt + RAG + tools het niet oplossen: voor consistente stijl/format, een nauwe taak, of lagere latency/kosten bij hoog volume.

Wanneer fine-tunen wél loont

- Je hebt een **smalle, repeterende taak** met een vast gewenst formaat.
- Je wilt **latency/kosten** drukken door een kleiner, getuned model high-volume werk te laten doen (vgl. "lokaal model 10× goedkoper" uit de praktijk).
- Prompt-engineering haalt de kwaliteit niet meer omhoog en je hebt genoeg goede voorbeelden.

Wanneer juist niet

- Je kennis verandert vaak → **RAG**, niet fine-tunen (anders moet je blijven hertrainen).
- Je hebt weinig, rommelige voorbeelden → eerst data op orde (Deel 1: data is de #1 killer).
- Je zoekt een snelle fix → fine-tunen is de traagste, duurste route.

Beslismatrix

Probleem	Beste hefboom
"Het antwoordt verkeerd/onhandig"	Prompt
"Het kent onze info niet / info verandert"	RAG
"Het moet iets dóen of berekenen"	Tools/agent
"Vaste stijl/format, of kosten/latency bij schaal"	Fine-tunen

De kosten/ROI-regel: kies de laagste trede die het probleem oplost. Elke trede hoger kost meer geld, tijd en onderhoud.

Bron: ZenML; MachineLearningMastery; Alexander Thamm "LLM Cost Optimization"; r/LLMDevs.

Hoofdstuk 15 — Waarom evaluatie alles is

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: evaluatie is de vaardigheid die je het meeste oplevert. **Beslisser**: dit is de poort die de 5% van de 95% scheidt.

De kern van het hele boek

Herinner je de opening: een LLM geeft een 200, keurig geformatteerd, en tóch een fout antwoord. In klassieke software vertelt een test je of iets werkt: ja of nee. Bij een LLM bestaat dat binaire antwoord niet. Dezelfde vraag geeft vandaag een ander antwoord dan morgen. "Geeft het een 200?" zegt niets over of het *klopt*.

Daarom is evaluatie geen sluitstuk, maar het fundament. Het is letterlijk wat de MIT-data (Hoofdstuk 1) als scheidslijn aanwijst: de 5% die waarde haalt, **meet**; de 95% die strandt, vaart op onderbuikgevoel. Je kunt niet betrouwbaar leveren wat je niet kunt meten.

Van "vibes" naar een getal

Evalueren betekent: outputkwaliteit omzetten in een **score op een schaal**, herhaalbaar en op schaal, zonder elk antwoord met de hand te lezen. Dat doe je met drie bouwstenen, die je in de volgende hoofdstukken uitwerkt:

1. **Golden test set** — een vaste verzameling representatieve inputs met een verwacht resultaat of een beoordelingsrubriek (H15, dit hoofdstuk).
2. **Metrics** — wat meet je precies? Voor RAG: faithfulness, answer relevancy, context precision/recall (H16).
3. **LLM-as-judge** — een model dat outputs scoort volgens jouw rubriek, uitgelijnd op menselijk oordeel (H17).

De golden test set: je belangrijkste bezit

Begin klein en concreet: - Verzamel **20–50 representatieve voorbeelden** (echte vragen, randgevallen, lastige formuleringen). - Leg per voorbeeld vast wat een **goed** antwoord is — exact, of als rubriek ("noemt bron X", "weigert buiten scope"). - **Laat de set groeien vanuit productie**: elke echte fout wordt een nieuw testgeval (de verbeterloop, H25). - Behandel de set als code: versioneer hem, want hij is je maatstaf.

Een golden set van 30 goede voorbeelden is meer waard dan een dashboard vol vage tevredenheidscijfers.

Offline én online

- **Offline** — vóór je live gaat: draai je eval tegen de golden set, in CI (H19). Geen merge als de score zakt.
- **Online** — in productie: scoor een steekproef van echt verkeer (H23), want de echte wereld bevat inputs die je golden set mist.

De testpiramide kantelt

In klassieke software: veel unit-tests, weinig end-to-end. Bij LLM-systemen is de piramide **platter en breder**: unit-tests blijven voor de niet-AI-onderdelen, maar het zwaartepunt verschuift naar integratie-, gedrags- en statistische tests op de output. Je test niet "geeft functie X het juiste label", maar "is het antwoord trouw aan de bron, in 95% van de gevallen".

Begin vandaag

Je hebt geen eval-platform nodig om te starten. Een lijst van 30 voorbeelden, een verwacht resultaat per stuk, en een simpel script dat de score uitrekent — dat is genoeg om vandaag van gokken naar meten te gaan. De tooling (H18) maakt het daarna schaalbaar.

Waarom dit je belangrijkste investering is

Zonder evaluatie kun je niet veilig opschalen, niet aantonen dat een wijziging een verbetering is, en niet voldoen aan de documentatie-eisen van de EU AI Act (Deel 6). Eval is tegelijk je kwaliteitsinstrument, je veiligheidsklep en je bewijslast.

Bron: MachineLearningMastery "Roadmap for Mastering LLMOps 2026"; ZenML "The Evaluation Playbook"; Evidently AI (LLM-evaluation guide); MIT NANDA (2025).

Hoofdstuk 16 — RAG-metrics: meten of het antwoord deugt

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: deze vier metrics vertellen je *waar* het misgaat. **Beslisser**: onthoud dat "klopt het" uiteenvalt in retrieval-kwaliteit én antwoord-kwaliteit.

Vier metrics, twee vragen

De praktijkstandaard voor het evalueren van RAG bestaat uit vier metrics. Ze beantwoorden twee verschillende vragen, en dat onderscheid is goud waard bij het debuggen:

Heeft het systeem de juiste context opgehaald? (retrieval) - **Context precision** — staan de relevante stukken bovenaan, of zit er veel ruis tussen? Lage precision = je haalt te veel irrelevants op (chunking/re-ranking, H11). - **Context recall** — heeft de retrieval *alle* benodigde informatie te pakken gekregen? Lage recall = er ontbreekt iets (embedding/coverage/chunking).

Heeft het model er een goed antwoord van gemaakt? (generatie) - **Faithfulness** — is het antwoord *gegrond* in de opgehaalde context, of verzint het model erbij? Dit is je directe hallucinatie-meter. - **Answer relevancy** — beantwoordt het antwoord daadwerkelijk de vraag, of dwaalt het af?

Waarom dit onderscheid je tijd bespaart

Een gebruiker zegt "het antwoord klopt niet". Zonder metrics ga je gokken. Mét metrics lokaliseer je het:

Symptoom in de cijfers	Waarschijnlijke oorzaak	Waar fixen
Lage **faithfulness**	model verzint, of te veel ruis in context	prompt / minder-maar-betere context
Lage **answer relevancy**	prompt of modelkeuze	prompt / model (H10, H14)
Lage **context precision**	retrieval haalt rommel op	chunking, re-ranking (H11)
Lage **context recall**	retrieval mist informatie	embeddings, chunk-grootte, dekking (H11)

Je weet nu of je je *retrieval* of je *generatie* moet aanpakken — in plaats van blind aan prompts te sleutelen terwijl het probleem in de chunking zat.

Reference-free: zonder perfect antwoord toch meten

Veel van deze metrics zijn **reference-free**: een LLM-as-judge (H17) beoordeelt faithfulness en answer relevancy zonder dat jij voor elke vraag een "perfect antwoord" hoeft te schrijven. Context recall leunt vaak wél op een referentie (wat had opgehaald moeten worden). Dat maakt het haalbaar om op schaal te meten.

Praktijk met Ragas

Ragas is het meest gebruikte framework hiervoor: het rekent precies deze vier metrics uit, kan synthetische testvragen genereren, en werkt reference-free waar mogelijk. Werkwijze: - voed het je golden set (H15) met vraag, opgehaalde context en antwoord; - volg de vier scores over de tijd, niet als momentopname; - zet drempels (bv. faithfulness ≥ 0,9 voor klantgerichte systemen) en koppel ze aan je CI-gate (H19).

Welke metric weegt het zwaarst?

Hangt af van de inzet. Klantgericht of compliance-gevoelig? **Faithfulness eerst** — een verzonnen beleidsregel is erger dan een iets minder vlot antwoord. Interne kennis-zoeker? **Context recall** telt zwaar (niets missen).

Eerlijke waarschuwing (vooruit naar H17)

Deze scores zijn niet absoluut. Ze worden zelf door een model bepaald en kunnen per run wat schommelen. Behandel ze als een betrouwbaar *kompas*, niet als een geijkte meetlat — en lijn ze uit op menselijk oordeel. Dat is precies het onderwerp van het volgende hoofdstuk.

Bron: Ragas (docs); Evidently AI "RAG evaluation"; Atlan & Confident AI (RAG-metrics 2026).

Hoofdstuk 17 — LLM-as-a-judge

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: zo scoor je op schaal zonder elk antwoord te lezen. **Beslisser**: de rechter is zelf een model — vertrouw, maar verifieer.

Het idee

Je kunt niet elk productie-antwoord met de hand beoordelen. De oplossing: laat een *ander* model als beoordelaar optreden. Een **LLM-as-a-judge** krijgt de vraag, het antwoord (en eventueel de context) plus een rubriek, en geeft een score — een cijfer (1–5), een pass/fail, of een vergelijking tussen twee antwoorden. Zo schaal je evaluatie van tientallen handmatige checks naar duizenden geautomatiseerde.

Het is de motor onder de meeste metrics uit Hoofdstuk 16: faithfulness en answer relevancy worden in de praktijk dóór een judge bepaald.

De biases waar je je tegen moet wapenen

Een LLM-rechter is handig maar niet neutraal. Bekende valkuilen:

- **Positie-bias** — bij het vergelijken van twee antwoorden kiest de judge stelselmatig het eerste (of tweede). *Mitigatie:* draai de volgorde om en middel (A vs B én B vs A).
- **Verbosity-bias** — langere antwoorden krijgen onterecht hogere scores. *Mitigatie:* neem beknoptheid in de rubriek op; controleer op lengte.
- **Self-preference / agreeableness** — de judge prefereert zijn eigen stijl of is te meegaand ("ziet er goed uit"). *Mitigatie:* gebruik een sterk, liefst ander model als rechter; dwing kritische criteria af.
- **Format-bias** — nette opmaak verwart "ziet er goed uit" met "is correct". *Mitigatie:* scheid vorm en inhoud in de rubriek.

Uitlijnen op mensen: de stap die niemand mag overslaan

Een judge is pas bruikbaar als hij beslist zoals jouw experts zouden beslissen. Daarom:

1. Laat een mens een **steekproef** (bijv. 50 antwoorden) scoren — je menselijke ground truth.
2. Laat de judge dezelfde steekproef scoren.
3. Meet de **overeenstemming** (agreement). Te laag? Scherp de rubriek aan, voeg voorbeelden toe, of kies een sterker model.
4. Herhaal tot de judge betrouwbaar met mensen meebeweegt — en **herijk periodiek** (modellen en data verschuiven).

Zonder deze uitlijning meet je iets, maar weet je niet wát.

Een goede judge-prompt

- Geef een **expliciete rubriek**: wat is een 5, wat een 1, met criteria.
- Vraag eerst **redenering, dan score** (verhoogt consistentie).
- Voeg **enkele voorbeelden** toe (few-shot) van goed/fout met de gewenste score.
- Gebruik een **capabel model** als rechter; bezuinig hier niet.
- Draai zo **deterministisch** mogelijk (lage temperatuur, vaste prompt-versie — H10).

Waar je het inzet

- **Offline**, in CI, tegen je golden set (H19): geen merge als de judge-score zakt.
- **Online**, op een steekproef van productieverkeer (H23): vang de gevallen die je golden set miste.

Wanneer toch een mens?

Voor hoog-risico of compliance-gevoelige output (medisch, juridisch, financieel): laat de judge schalen, maar houd **menselijke steekproeven** en een escalatiepad. De judge is je zeef, niet je eindrechter.

Eerlijke nuance

Judges schommelen (denk aan de Ragas-wisselvalligheid uit H16). Behandel een enkele score niet als waarheid; kijk naar **trends over een set**, niet naar één getal — en blijf tegen mensen aanijken. Een goed uitgelijnde judge is een van de krachtigste hefbomen in LLMOps; een blind vertrouwde judge is een valse zekerheid.

Bron: Comet/Opik "LLM-as-a-judge"; Evidently AI; Confident AI (DeepEval); Zheng et al. "Judging LLM-as-a-judge" (MT-Bench).

Hoofdstuk 18 — Eval-frameworks in de praktijk

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: kies het juiste gereedschap, bouw niet alles zelf. **Beslisser**: je hoeft er maar één of twee te kiezen — niet alle vier.

Vier frameworks, vier zwaartepunten

De metrics (H16) en de judge (H17) hoef je niet zelf te implementeren; deze frameworks doen het. Ze overlappen, maar elk heeft een thuisbasis.

Ragas — voor RAG-evaluatie. Rekent de vier RAG-metrics uit (faithfulness, answer relevancy, context precision/recall), genereert synthetische testvragen uit je documenten, en werkt grotendeels reference-free. Begin hier als je een RAG-systeem bouwt.

DeepEval — "pytest voor LLM's". Schrijf evals als unit-tests; 50+ kant-en-klare metrics, plus **G-Eval** (eigen criteria via een judge). Draait in je CI als gewone tests, met pass/fail-asserts. Begin hier als je evaluatie in je test-suite en CI-gate (H19) wilt.

Promptfoo — voor prompt- en modelvergelijking. Config-/CLI-gedreven: zet prompts en modellen naast elkaar en zie per testgeval wat wint. Heeft ook red-teaming/security-scans. Begin hier als je prompts itereert of providers vergelijkt (en H8/H10 ondersteunt).

TruLens — voor feedback in je applicatie. "Feedback functions" en de **RAG-triad** (context relevance, groundedness, answer relevance), met instrumentatie in je app-code. Begin hier als je continue feedback dicht op je applicatie wilt.

Een minimale, werkende opzet

Je hoeft niet alles te combineren. Een pragmatische stack voor een RAG-app:

1. **Ragas** op je golden set (H15) om retrieval- en generatie-kwaliteit te meten.
2. **DeepEval** om die checks als asserts in CI te gieten (faithfulness ≥ drempel = groen).
3. **Promptfoo** erbij wanneer je een nieuwe prompt of een ander/goedkoper model overweegt (H14/H9).

Voorbeeld-denkstap (geen volledige code): definieer per testgeval `vraag → context → antwoord`, laat Ragas de scores berekenen, en laat DeepEval falen als een score onder je drempel zakt. Zo wordt "is het beter geworden?" een groen of rood vinkje in plaats van een gevoel.

Hoe dit op H16/H17 aansluit

Deze tools zijn de uitvoering van de vorige twee hoofdstukken: ze gebruiken een **LLM-as-judge** onder de motorkap om de **metrics** te scoren. Alle waarschuwingen van H17 blijven gelden — lijn de judge uit op mensen, kijk naar trends, niet naar één run.

Snelle keuzehulp

Je wilt vooral...	Pak
RAG-kwaliteit meten	**Ragas**
Evals als tests in CI	**DeepEval**
Prompts/modellen vergelijken + security	**Promptfoo**
Feedback in je app/observability-stijl	**TruLens**

De valkuil

Vier frameworks adopteren "voor de zekerheid" levert onderhoudslast en tegenstrijdige cijfers op. Kies er één als basis (meestal Ragas of DeepEval), voeg pas een tweede toe als je een concreet gat voelt. De waarde zit niet in de tool, maar in je **golden set en je drempels** — die zijn van jou, ongeacht het framework.

Bron: Techsy & Atlan (eval-frameworks 2026); QAwerk (Ragas vs DeepEval vs Promptfoo); Confident AI (DeepEval/G-Eval); TruLens docs.

Hoofdstuk 19 — Eval-driven development & CI/CD-gates

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: maak van evaluatie een poort, geen nakomertje. **Beslisser**: dit is je kwaliteitsborging én je bewijslast.

Eval-driven development

In klassieke software ken je test-driven development: eerst de test, dan de code. De LLM-variant is **eval-driven development**: je golden set (H15) is de specificatie. Wat "goed" betekent leg je vast in evalgevallen *voordat* je gaat sleutelen. Elke wijziging — een prompt, een ander model, andere chunking — bewijst zich tegen die set, of gaat niet door.

Het effect: "is het beter geworden?" verandert van onderbuik in een meetbaar verschil. En je voorkomt het klassieke LLM-drama: een promptverbetering voor geval A die stiekem geval B sloopt.

De CI-gate

Bouw evaluatie in je pijplijn, net als unit-tests:

1. Iemand wijzigt een prompt/model/RAG-instelling (een pull request).
2. CI draait de eval-suite (Ragas/DeepEval, H18) tegen de golden set.
3. De scores worden vergeleken met de **baseline** van de hoofdbranch.
4. **Zakt een score onder de drempel → build faalt → geen merge.**

Zo wordt een slechte wijziging tegengehouden vóór productie, niet ontdekt erna.

Drempels: absoluut én relatief

- **Absoluut** — een ondergrens die altijd geldt (bijv. faithfulness ≥ 0,90 voor klantgericht).
- **Relatief** — geen significante daling t.o.v. de baseline (vangt sluipende regressie).

Stem de strengheid af op de inzet: hoog-risico output (Deel 6) krijgt strakkere drempels dan een interne hulp-tool.

De regressieset groeit vanzelf

Elke productiefout wordt een permanent testgeval (de verbeterloop, H25). Zo kan dezelfde fout nooit twee keer ongemerkt terugkomen — je suite wordt sterker met elk incident.

Concreet

- Bewaar **golden set + drempels in je repo** (versiebeheer, net als code).
- Een CI-job draait de evals; **fail de build bij regressie**.
- Log per run **prompt-versie + model + scores** (H10) zodat elke wijziging herleidbaar is.
- Draai de **volledige set** 's nachts of bij release; een **snelle subset** op elke PR.

Omgaan met niet-determinisme

Eén run kan toevallig zakken (scores schommelen, H16/H17). Gate daarom niet op een enkele flaky run: draai kritische gevallen **meerdere keren** en oordeel op het gemiddelde, of bouw tolerantie in.

Kosten en tijd

Evals kosten tokens en tijd. Houd het betaalbaar: een kleine, scherpe subset per PR; de volledige suite minder vaak (nightly/release); cache waar mogelijk.

Tegelijk je audit trail

Deze gate is niet alleen kwaliteit, het is **bewijs**. Voor de EU AI Act (Deel 6) moet je kunnen laten zien hoe je systeem getest en bewaakt wordt. Een geversioneerde golden set, drempels en CI-logs zijn precies die documentatie.

Bron: MachineLearningMastery (eval-pipelines/CI); Confident AI/DeepEval (CI-integratie); ZenML "Evaluation Playbook".

Hoofdstuk 20 — Agents & multi-turn evalueren

DEEL 4: BEWIJZEN — EVALUATIE · NIVEAU: INTERMEDIATE

> **Builder**: evalueer het traject, niet alleen het laatste antwoord. **Beslisser**: nodig zodra je agents draait — niet voor een simpele keten.

Eén antwoord scoren is niet genoeg

Bij een chatbot beoordeel je het antwoord. Bij een **agent** (H12) beoordeel je het hele **traject**: heeft hij de juiste stappen gezet, de juiste tools gekozen, in de juiste volgorde, zonder rondjes te draaien — en is de taak uiteindelijk volbracht? Eén fout antwoord is vervelend; een agent die in een lus je budget opmaakt of de verkeerde tool aanroept, is een ander soort probleem.

Lastiger ook omdat fouten zich **opstapelen**: tien stappen met elk 95% kans op goed, is samen al onder de 60%. En het niet-determinisme uit H15 vermenigvuldigt zich over de stappen.

Wat je meet

- **Taakvoltooiing** — is het doel bereikt? (de uitkomst, niet de vorm)
- **Tool-correctheid** — koos de agent de juiste tool, met de juiste argumenten?
- **Trajectorie-efficiëntie** — geen overbodige stappen, lussen of dubbele aanroepen (kosten + latency).
- **Multi-turn-coherentie** — onthoudt de agent context over meerdere beurten, spreekt hij zichzelf niet tegen?

Hoe je het aanpakt

1. **Log het volledige traject** (elke stap: tool, input, output) — onmisbaar, en je hebt het al voor observability (Deel 5).
2. **Step-level asserts** — controleer per stap of de juiste tool met de juiste argumenten is aangeroepen (DeepEval heeft agent-metrics hiervoor).
3. **Outcome-eval** — beoordeel de eindtoestand met een judge (H17): is het doel objectief gehaald?
4. **Simulatie** — laat een gesimuleerde gebruiker meerdere gesprekken voeren om multi-turn-gedrag op schaal te testen.

Praktijk

- Begin met **taakvoltooiing + tool-correctheid**; dat vangt de meeste echte fouten.
- Combineer met de **harde grenzen** uit H12 (max stappen, kostenplafond) — eval meet kwaliteit, guardrails voorkomen ontsporing.
- Gebruik trace-tools (LangSmith/Langfuse, Deel 5) om trajecten te bekijken én te scoren.

Heb je dit nodig?

Alleen als je echte agents draait. Een vaste keten van 1–3 stappen evalueer je gewoon op het eindantwoord (H16). Trajectorie-eval is krachtig maar kost moeite; zet het in waar de autonomie zit.

Afsluiting van Deel 4

Je hebt nu het complete eval-fundament: waarom (H15), wat (H16), hoe te scoren (H17), waarmee (H18), hoe te bewaken (H19) en hoe agents te beoordelen (H20). In Deel 5 brengen we dit naar productie: observability, online-evaluatie en kostenbeheersing — want eval stopt niet bij de release.

Bron: ideas2it (AgentOps); Confident AI/DeepEval (agentic metrics); LangSmith & Langfuse (trace-based eval).

Hoofdstuk 21 — Waarom klassieke APM liegt

DEEL 5: BEHEERSEN — OBSERVABILITY, MONITORING & KOSTEN · NIVEAU: INTERMEDIATE

> **Builder**: instrumenteer vanaf dag 1. **Beslisser**: "alles groen" op je dashboard zegt niets over of de AI klopt.

Het groene dashboard dat liegt

Klassieke application performance monitoring (Datadog, New Relic, Honeycomb) kijkt naar statuscodes, latency en errors. Voor gewone software is dat genoeg: 500 = stuk, 200 = goed. Voor LLM-systemen is het misleidend. De APM meldt netjes "200, 180 ms, 0 errors" terwijl je chatbot met overtuiging een beleidsregel verzint die niet bestaat. Het systeem is technisch gezond en inhoudelijk fout — en je standaard-monitoring ziet het verschil niet.

Dat is de reden dat **LLM-observability** een eigen categorie werd: je moet niet alleen weten *of* er een antwoord kwam, maar *wat* het was, waarop het gebaseerd was, en of het deugde.

Wat LLM-observability toevoegt

Drie dingen die klassieke APM mist:

1. **Traces, spans & sessies** — de volledige reis van een verzoek: de prompt, het opgehaalde RAG-materiaal, elke tool-aanroep van een agent, het uiteindelijke antwoord, en de samenhang over meerdere beurten. Eén regel "200" wordt een uitklapbaar verhaal.
2. **Kwaliteitssignalen** — online-evaluatiescores op echt verkeer (H23), niet alleen techniek-metrics.
3. **Kosten & tokens per verzoek** — want een stille kostenexplosie is óók een productie-incident (H24).

Instrumenteer vanaf dag 1

De goedkoopste fout is observability "later" inbouwen. Als er iets misgaat in productie, wil je de geschiedenis hebben om te debuggen — niet ontdekken dat je niets hebt gelogd. Een proxy/gateway (H8) geeft je kostentelling in twee minuten; volledige tracing voeg je toe zodra je iets serieus bouwt. Achteraf retrofitten kost veel meer dan vanaf het begin meelopen.

Wat je per verzoek vastlegt

- de **input** (prompt + parameters), de **output**;
- de **opgehaalde context** (RAG) en elke **tool-aanroep** (agents);
- **model + versie**, **prompt-versie** (H10);

- **tokens, latency, kosten**;
- **gebruiker/sessie-id** (voor multi-turn en support);
- een **eval-score** waar je online evalueert (H23).

Met dat pakket kun je elk vreemd antwoord reconstrueren: welk model, welke prompt, welke context, hoeveel het kostte.

Eén investering, vier doelen

Observability is niet alleen monitoring. Diezelfde traces zijn je **debug-gereedschap**, de basis van je **verbeterloop** (H25), je **kostencontrole** (H24), én je **audit trail** voor de EU AI Act (Deel 6: "logging om de traceerbaarheid van resultaten te waarborgen" is een expliciete eis voor hoog-risico-systemen). Je bouwt het één keer en gebruikt het vier keer.

Vooruitblik

In H22 vergelijken we de tools die dit leveren (Langfuse, LangSmith, Arize Phoenix, Helicone, Braintrust, Datadog) met hun prijzen en sterke punten, zodat je een onderbouwde keuze maakt in plaats van de eerste de beste te pakken.

Bron: SigNoz & Firecrawl (LLM-observability 2026); r/mlops ("200 lies"); Getmaxim; DigitalApplied (agent-observability).

Hoofdstuk 22 — Observability-tools vergeleken

DEEL 5: BEHEERSEN — OBSERVABILITY, MONITORING & KOSTEN · NIVEAU: INTERMEDIATE

> **Builder**: kies op framework, hosting en budget. **Beslisser**: er is geen "beste" tool, alleen de juiste default voor jouw situatie.

Het veld is geconsolideerd

In 2026 draait het om een handvol productie-tools. Ze overlappen in tracing, maar elk wint op eigen terrein. Prijzen zijn indicatief — verifieer ze bij aanschaf, ze schuiven.

- **Langfuse** — open source (MIT), gratis self-hosten; cloud gratis tier, daarna ~€29–59/mnd. Framework-agnostisch via OpenTelemetry. Overgenomen door ClickHouse (jan 2026). *Voor*: OSS, *data-eigenaarschap, self-host.*
- **LangSmith** — managed, het strakst geïntegreerd met LangChain/LangGraph; vanaf ~$39/seat. *Voor: teams die al op LangChain bouwen.*
- **Arize Phoenix** — open source, OpenTelemetry-native, met eval-primitieven, drift-detectie en embeddings-analyse; gratis self-host (Arize AX voor enterprise). *Voor: eval-rigor en RAG-drift.*
- **Helicone** — gateway/proxy: één base-URL omzetten → meteen logging, caching, routing en kosteninzicht; gratis tier, vanaf ~$79/mnd, geen opslag op je modelkosten. *Voor: snelste setup + kostenzicht.*
- **Braintrust** — eval-first, met CI-achtige gates; royale gratis tier. *Voor: eval-driven development (H19).*
- **Datadog LLM Observability** — geïntegreerd in hun APM; veruit het duurst. *Voor: teams die al volledig op Datadog draaien.*

Daarnaast: W&B Weave (als je al W&B gebruikt), Evidently en Opik (OSS), Grafana via OpenLIT.

Keuzetabel

Je belangrijkste eis	Pak
OSS / self-host / data in eigen huis	**Langfuse** of **Phoenix**
Je bouwt op LangChain/LangGraph	**LangSmith**
Eval-rigor, RAG-drift, embeddings	**Arize Phoenix**
Snelste installatie + kosten-/routing-controle	**Helicone**
Eval-gedreven met CI-gates	**Braintrust**
Je staat al volledig op Datadog	**Datadog LLM Obs**

Praktische volgorde

- Begin je net? **Helicone** (gateway) geeft je in minuten kosten + logging.
- Wil je vanaf dag 1 volledige tracing + eval-haakjes met data-eigenaarschap? **Langfuse** (self-host).
- Zit je in een RAG-zwaar, eval-serieus team? **Phoenix**.
- Kies bij voorkeur tools die **OpenTelemetry** spreken — dan blijf je portabel en zit je niet vast (vgl. lock-in, H8).

EU/soevereiniteit-noot

Self-hostbare tools (Langfuse, Phoenix, Helicone) houden je traces — inclusief prompts en mogelijk persoonsgegevens — in je eigen omgeving. Dat is relevant voor AVG en de EU AI Act (Deel 6): observability-data is óók data die ergens woont.

De valkuil

Niet de tool maakt het verschil, maar of je hem **vanaf dag 1** gebruikt en of je **kwaliteit** meet, niet alleen techniek. Een dure observability-stack zonder eval-signaal is een mooi dashboard dat nog steeds niet ziet dat de bot onzin verkoopt.

Bron: Confident AI (Top-7 LLM-observability 2026); SigNoz; Firecrawl; Techsy; DigitalApplied (agent-observability).

Hoofdstuk 23 — Online evaluatie & drift

DEEL 5: BEHEERSEN — OBSERVABILITY, MONITORING & KOSTEN · NIVEAU: INTERMEDIATE

> **Builder**: meet ook in productie, niet alleen in CI. **Beslisser**: je golden set veroudert; de echte wereld levert vragen die je niet bedacht.

Offline is niet genoeg

Je CI-gate (H19) test tegen je golden set — een vaste, bekende verzameling. Maar productie stuurt inputs die je golden set mist: nieuwe vragen, rare formuleringen, randgevallen. **Online evaluatie** scoort echt verkeer, zodat je ziet hoe het systeem het buiten je testlab doet.

Het praktijkpatroon: drie lagen

Je kunt niet elk productie-antwoord door een dure LLM-judge halen. De werkende balans:

1. **Goedkope heuristieken op 100% van het verkeer** — regels die bijna niets kosten: formaat-checks, lengte, weigeringen, PII-lekken, verboden woorden, lege/afgekapte antwoorden. Deze vangen de grove fouten meteen.
2. **LLM-as-judge op een steekproef (10–20%)** — de duurdere kwaliteitsscore (faithfulness, relevancy, H16/H17) draai je op een sample, niet op alles. Genoeg om trends te zien, betaalbaar te houden.
3. **Menselijke annotatie, periodiek** — laat experts regelmatig een set beoordelen om je "ground truth" te verversen en je judge opnieuw uit te lijnen (H17). Die gevallen voeden je golden set.

Drift: de stille verslechtering

Je systeem kan langzaam slechter worden zonder dat er iets "stuk" gaat:

- **Input-drift** — gebruikers gaan andere dingen vragen dan bij de bouw.
- **Output-/kwaliteitsdrift** — de modelprovider update stilletjes het model (H10), of je kennisbank veroudert (H11).
- **Retrieval-drift** — nieuwe of veranderde data verschuift wat er opgehaald wordt.

Je detecteert drift door je **online scores, kosten en onderwerpen over de tijd** te volgen — niet door één momentopname. Een dalende faithfulness-trend of een stijgend weigeringspercentage is je vroege waarschuwing.

Alerting

Zet drempels en alerts op wat ertoe doet: kwaliteitsscore zakt onder X, kosten per dag boven Y, latency p95 boven Z, weigeringsratio omhoog. Een alert op een *kwaliteits*daling is precies wat klassieke APM (H21) je niet geeft.

Praktijk

- Heuristieken overal (goedkoop, 100%).
- Judge op een steekproef; sla scores op bij je traces (H22).
- Wekelijks: bekijk de **slechtst scorende N** met een mens.
- Elke bevestigde fout → nieuw golden-case (H25).

De kostenbalans

100% judgen is verleidelijk maar duur en traag. Steekproeven + goedkope heuristieken geven 90% van het inzicht voor een fractie van de kosten. Verhoog de steekproef tijdelijk als je een probleem onderzoekt of net iets nieuws hebt uitgerold.

Brug naar H24 en H25

Online scores tonen je *dat* iets verschuift; je kostendashboard (H24) toont de financiële kant; en de verbeterloop (H25) zet elke gevonden fout om in een blijvende verbetering. Samen maken ze van productie geen black box maar een systeem dat je stuurt.

Bron: MachineLearningMastery (online eval, sampling); Evidently AI (drift-detectie); praktijkpatronen uit r/LLMDevs & ZenML.

Hoofdstuk 24 — Kosten & latency in het wild

DEEL 5: BEHEERSEN — OBSERVABILITY, MONITORING & KOSTEN · NIVEAU: INTERMEDIATE

> **Builder**: concrete hefbomen met echte cijfers. **Beslisser**: kosten groeien non-lineair; zet kleppen vóór het misgaat.

De vier kostenhefbomen, nu in productie

In H9 zag je ze als begrotingsprincipe; hier als productiepraktijk, met cijfers uit het veld:

- **Modelkeuze** — het kleinste model dat de kwaliteitseis haalt. Een lokaal/klein model voor high-volume simpel werk gaf in de praktijk een **~10x kostenreductie** (r/LLMDevs).
- **Routing & cascades** — simpel naar goedkoop, complex naar groot. Benchmarks (RouterBench/LLMRouterBench) tonen **tot ~32% besparing zonder kwaliteitsverlies**. Tradeoff: routing voegt ~5–20 ms latency toe en introduceert een faalmodus.
- **Semantische caching** — herhaalde/gelijkende vragen hergebruiken. Praktijk: **~40% cache-hit ≈ $3k/maand bespaard** (r/LLMDevs); vendor-bovengrens tot ~73% (Redis) — behandel als plafond, niet als belofte.
- **Batching** — niet-urgente taken groeperen; dramatisch goedkoper per request, ten koste van latency.

Latency apart bekeken

- **Streaming** verbetert de *gevoelde* snelheid en laat je slechte antwoorden vroeg afbreken.
- **KV-cache + continuous batching** zijn de motor onder lage latency (vLLM: **11s → 3s, ~10x throughput**).
- **Model cascades** houden de meeste vragen op een snel, klein model.
- Stuur op **p95**, niet op het gemiddelde — de trage staart bepaalt de gebruikerservaring.

Budgetkleppen (anders ontspoort het)

Agent-workflows schalen non-lineair. Bouw in: - **Limieten per gebruiker en per app**. - **Budgetdrempels met automatische alerts**; halt/throttle bij overschrijding. - **Progressieve throttling** bij piekgebruik. - **Kostenplafond per agent-sessie** (H12) tegen de "loop die je daghuur opmaakt".

Kostencontrole-checklist

- Meet kosten per request én **per opgeloste taak** (H21).
- Cache aan; meet je hit-rate.
- Router voor de duidelijke complexiteitssplitsing.
- Batch wat niet realtime hoeft.
- Alerts op dag-spend en p95-latency.

De eerlijke tradeoff

Routing en cascades besparen, maar voegen complexiteit en een faalmodus toe. Voor veiligheids- of compliance-kritische stromen kan één goed-getuned model veiliger zijn dan een slimme router. Optimaliseer waar het volume zit, niet overal.

De lange-termijn-waarschuwing

Gartner (jan 2026): de **GenAI-kosten per resolutie gaan richting >$3 in 2030**. Goedkoop-per-interactie is tijdelijk. Anker je business case op **resolutie en retentie** (waarde), niet op "het model is goedkoop".

Bron: r/LLMDevs; RouterBench & LLMRouterBench; ZenML/vLLM; Redis; Gartner (jan 2026).

Hoofdstuk 25 — De verbeterloop

DEEL 5: BEHEERSEN — OBSERVABILITY, MONITORING & KOSTEN · NIVEAU: INTERMEDIATE

> **Builder**: maak van elke fout een blijvende verbetering. **Beslisser**: dit is "continue monitoring en bijsturing" — ook een AI-Act-eis.

Het vliegwiel dat Deel 4 en 5 verbindt

Observability (Deel 5) vindt de problemen; evaluatie (Deel 4) bewijst of een oplossing werkt. De verbeterloop knoopt ze aan elkaar tot een vliegwiel dat je systeem structureel beter maakt:

1. **Detecteren** — een online-evalscore, heuristiek of gebruikersfeedback markeert een slecht antwoord (H23).
2. **Triëren & annoteren** — een mens bekijkt de trace (H22) en bepaalt wat er precies misging.
3. **Vastleggen als regressie** — het geval wordt een permanent testgeval in je golden set (H19). Dezelfde fout kan nooit meer ongemerkt terugkomen.
4. **Diagnosticeren** — de metrics (H16) wijzen aan of het in *retrieval* of *generatie* zat.
5. **Repareren op de goedkoopste trede** — prompt, RAG of tools (H14), zelden hertrainen.
6. **Bewijzen** — de eval-gate (H19) laat zien dat de fix het oude geval oplost én niets anders breekt.
7. **Uitrollen & bewaken** — terug naar productie, en de loop begint opnieuw.

Waarom dit compounding is

Elke incident maakt je systeem blijvend beter: je golden set groeit, je judge wordt scherper, je faalmodi worden bekend. Na een paar maanden heb je geen verzameling losse fixes, maar een systeem dat aantoonbaar leert. Dit is precies het verschil tussen de 5% en de 95% uit Hoofdstuk 1 — niet één keer goed bouwen, maar een **lus** die fouten omzet in vooruitgang.

Maak het routine

- Een **feedback-knop** voor gebruikers (duim omlaag = gemarkeerde trace).
- **Auto-flag** van laagscorende traces (H23).
- Een **wekelijkse triage** van de slechtst scorende N.
- Een vaste afspraak: elke bevestigde fout → golden-case, dezelfde dag.

Tegelijk je compliance-verhaal

De EU AI Act (Deel 6) vraagt voor hoog-risico-systemen om **continue monitoring en corrigerende maatregelen**. Deze loop ís dat proces: je detecteert, documenteert en verbetert aantoonbaar. Eén mechanisme, twee opbrengsten — betere kwaliteit en aantoonbare zorgvuldigheid.

Afsluiting van Deel 5

Je kunt nu zien wat er in productie gebeurt (H21–H22), het meten op echt verkeer (H23), de kosten sturen (H24) en fouten omzetten in blijvende verbetering (H25). In Deel 6 borgen we het: guardrails, beveiliging, de EU AI Act en dataresidentie — de laag die je systeem veilig en compliant maakt.

Bron: MachineLearningMastery (verbeterloop); ZenML "Evaluation Playbook"; praktijkpatronen r/LLMDevs.

Hoofdstuk 26 — Guardrails & hallucinatie-controle

DEEL 6: BORGEN — VEILIGHEID, GOVERNANCE & DE EU-LAAG · NIVEAU: INTERMEDIATE

> **Builder**: bouw de vangrails in lagen. **Beslisser**: dit is je risicobeheersing — strenger naarmate de inzet hoger is.

Wat guardrails zijn

Guardrails zijn de veiligheidslaag tussen gebruiker, model en buitenwereld, in twee richtingen: - **Input-guardrails** — vóór het model: prompt-injection en jailbreaks weren, PII detecteren, off-topic of verboden verzoeken afvangen. - **Output-guardrails** — ná het model: hallucinaties, PII-lekken, schadelijke of beleidsovertredende output tegenhouden en het formaat afdwingen.

Prompt injection: het #1 agent-risico

Zodra een model tools kan aanroepen (H12), wordt **prompt injection** gevaarlijk: kwaadaardige instructies verstopt in een webpagina, document of invoer die de agent "overneemt". De verdediging is architectuur, geen trucje: - **Least privilege** — alleen de tools/rechten voor díe taak, daarna ingetrokken. - **Vertrouw opgehaalde of gebruikerstekst nooit als instructie** — het is data, geen commando. - **Gevoelige acties achter menselijke goedkeuring** (verwijderen, betalen, contracten). - **Log alles** (H22), zodat je misbruik ziet.

Dezelfde least-privilege-logica als autorisatierollen in SAP: niet iedereen krijgt admin.

Hallucinatie-controle

- **Grounding/faithfulness-checks** (H16): is de output gedekt door de context?
- **Bronvermelding** — laat citeren; ongedekte claims vallen op.
- **"Ik weet het niet"-fallback** — beter een eerlijk gat dan een vloeiende leugen.
- **Tweede-model-check** voor hoog-risico output.

Tooling

- **Platform**: Bedrock Guardrails, Azure/Foundry content filters (+ Purview), Vertex safety.
- **Libraries**: Guardrails AI, NeMo Guardrails, Lakera Guard, Galileo Luna (realtime block).

Gelaagde verdediging

1. Input-filter (injection, PII, scope) → 2. Grounded generatie → 3. Output-validatie (schema, PII-uit, faithfulness) → 4. Menselijke poort voor hoog-risico. Elke laag vangt wat de vorige mist.

Kalibreer op risico

Interne samenvattingstool: licht. Klantgericht of beslissend (mogelijk "hoog-risico", H28): strenge filters, human oversight, volledige logging. Stem de zwaarte af op de impact.

Bron: OWASP LLM Top 10; Guardrails AI / NeMo / Lakera / Galileo; platform-guardrails (Bedrock/Foundry/Vertex).

Hoofdstuk 27 — Beveiliging & toegang

DEEL 6: BORGEN — VEILIGHEID, GOVERNANCE & DE EU-LAAG · NIVEAU: INTERMEDIATE

> **Builder**: behandel je AI-systeem als elk ander productiesysteem met gevoelige data. **Beslisser**: dit is je AVG- en SOC 2-verhaal.

De basis op orde

GenAI verandert niets aan de beveiligingsfundamenten — het voegt nieuwe oppervlakken toe: - **Secrets** — API-sleutels nooit in code; gebruik een secrets manager. Roteer ze. - **Least-privilege toegang** — wie en wat mag welke modellen en databronnen aanroepen. - **Encryptie** — in rust en onderweg; overweeg customer-managed keys voor gevoelige data. - **Audit trails** — elke aanroep en actie herleidbaar (sluit aan op H22). - **Retentie** — hoe lang bewaar je prompts, antwoorden en traces? Minimaliseer.

Identiteitsbewuste retrieval

Een onderschat RAG-risico: een gebruiker krijgt via de AI een antwoord op basis van documenten die hij níet had mogen zien. Bouw **identity-aware retrieval**: het systeem checkt in real time welke bronnen deze gebruiker mag inzien, en haalt alleen díe op. Marketing krijgt zo nooit een antwoord gebaseerd op HR-documenten.

Datapseudonimisering & PII

Voor gevoelige data: **maskeer of pseudonimiseer** (BSN's, namen, bedrijfsgeheimen) vóórdat ze naar het model gaan. Combineer met PII-detectie in je guardrails (H26). Voor RAG geldt: gevoelige velden buiten de embeddings houden of versleutelen.

Leverancier & sub-processors

Weet waar je data heen gaat: traint de provider op jouw input (enterprise-tiers doen dat doorgaans niet), wie zijn de sub-processors, en waar staan ze (H29)? Leg het contractueel vast.

Beveiligingschecklist

- Secrets in een vault, niet in code/ `.env` in productie.
- Least-privilege op modellen, tools én databronnen.
- Identity-aware retrieval voor RAG.
- PII-maskering vóór de prompt.
- Audit trails + retentiebeleid.

- Contract: geen training op jouw data, bekende sub-processors.

AVG-haakjes

De AVG eist een **rechtmatige grondslag**, **dataminimalisatie**, een **verwerkingsregister** en passende **beveiliging** (art. 32). Voor AI met persoonsgegevens is dit geen formaliteit maar de basis — en het overlapt sterk met SOC 2 / ISO 27001 als je die certificering nastreeft.

Bron: AVG art. 5/30/32; identiteitsbewuste retrieval (enterprise RAG-patroon); leverancier-DPA's (o.a. Claude for Work).

Hoofdstuk 28 — Responsible AI & de EU AI Act

DEEL 6: BORGEN — VEILIGHEID, GOVERNANCE & DE EU-LAAG · NIVEAU: INTERMEDIATE

> **Builder**: je eval + logging (Deel 4/5) zijn al het halve compliance-werk. **Beslisser**: classificeer je systeem eerst; dat bepaalt alles.

De tijdlijn — correct

Veel bronnen hebben dit fout. De feiten (primaire tekst + de Digital Omnibus van 7 mei 2026): - **Vanaf 2 aug 2026**: AI-geletterdheid (Art. 4), verboden praktijken (Art. 5) handhaving, én **transparantie (Art. 50)**. - **Zwaarste hoog-risico-verplichtingen → eind 2027** (Annex III vanaf 2 dec 2027; product-geïntegreerd 2 aug 2028). - GenAI die vóór 2 aug 2026 al op de markt was: tot **2 dec 2026** voor machine-leesbare markering. - Boetes tot **€35 mln of 7%** van de wereldwijde omzet.

De vier risiconiveaus

1. **Verboden** (bv. social scoring) — mag niet.
2. **Hoog-risico** (Annex III: o.a. werving, kredietscoring, biometrie) — strengste eisen.
3. **Transparantie-risico** (Art. 50) — informeren dat het AI is, AI-content markeren, deepfakes labelen. **Raakt bijna elke GenAI-gebruiker**, ook open source.
4. **Minimaal** — geen extra eisen.

Hoog-risico-verplichtingen (Art. 9–15)

Risicomanagement, datagovernance, technische documentatie, **logging/record-keeping**, transparantie naar de gebruiker, **human oversight**, en nauwkeurigheid/robuustheid/cybersecurity. Plus: aanbieder (Art. 16) vs. gebruiker/deployer (Art. 26) hebben elk eigen plichten.

Nederland

De **Autoriteit Persoonsgegevens (AP)** is toezichthouder voor zowel AVG als AI Act. Voor persoonsgegevens: een **DPIA**; voor hoog-risico bij de overheid een **FRIA** (grondrechteneffectbeoordeling) + registratie in de EU-database. AI-geletterdheid (Art. 4) is een inspanningsverplichting die je moet kunnen **aantonen** (wie volgde welke training).

Compliance is grotendeels wat je al bouwde

- **Logging** (H22) = de traceerbaarheidseis.

- **Evaluatie + drempels** (H19) = aantoonbare nauwkeurigheid/robuustheid.
- **Human-in-the-loop** (H26) = human oversight.
- **De verbeterloop** (H25) = continue monitoring & bijsturing. Documenteer deze processen en je hebt het leeuwendeel van de hoog-risico-dossiervorming.

Begin met classificeren

Bepaal eerst in welk niveau je systeem valt — dat bepaalt of je een licht transparantie-vinkje nodig hebt of een volledig hoog-risico-dossier. Overclassificeren kost onnodig geld; onderclassificeren is een €35M-risico.

Bron: artificialintelligenceact.eu (Art. 6/16/50, tijdlijn); digital-strategy.ec.europa.eu; Digital Omnibus (7 mei 2026); Autoriteit Persoonsgegevens.

Hoofdstuk 29 — EU-dataresidentie vs. soevereiniteit

DEEL 6: BORGEN — VEILIGHEID, GOVERNANCE & DE EU-LAAG · NIVEAU: INTERMEDIATE

> **Builder**: weet waar je data juridisch woont. **Beslisser**: residentie ≠ soevereiniteit — dat verschil kan je vendorkeuze bepalen.

Het cruciale onderscheid

- **Dataresidentie** = waar je data fysiek staat (bv. "een EU-regio").
- **Datasoevereiniteit** = onder welke jurisdictie die data valt. Een datacentrum in Frankfurt, beheerd door een Amerikaans bedrijf, valt nog steeds onder de **CLOUD Act** — Amerikaanse autoriteiten kunnen toegang afdwingen. Een EU-regio kiezen lost dat niet op.

Voor de meeste organisaties is residentie genoeg (AVG). Voor sterk gereguleerde of gevoelige workloads telt soevereiniteit.

De opties in 2026

- **Hyperscaler-soevereine clouds**: AWS European Sovereign Cloud (Duitse entiteit, jan 2026), Azure EU Data Boundary (let op "Flexible Routing"; Anthropic-modellen standaard buiten de boundary), Vertex regio **europe-west4 (NL)**. Dempen, maar de moedermaatschappij blijft Amerikaans.
- **EU-native**: **Mistral** (model), **Scaleway** (enige met EU **SEAL-3**, hoogste soevereiniteitscertificering; draait ECB digital-euro-infra), **OVHcloud**, **StackIT**, **Aleph Alpha**. Vrij van CLOUD Act-blootstelling.
- **EU-raamcontract**: de Commissie gunde in april 2026 een **€180 mln** soeverein-cloud-contract aan o.a. OVHcloud, StackIT, Scaleway en Proximus/Mistral — een sjabloon voor publieke inkoop.

Beslis-kader per gevoeligheid

Profiel	Aanpak
Standaard B2B, AVG voldoende	EU-regio bij je hyperscaler (residentie)
Gereguleerd (zorg, finance, overheid)	Soevereine cloud of EU-native (geen CLOUD Act)
SAP-huis	**SAP BTP + Mistral** — soevereine AI binnen je bestaande platform
Maximale zekerheid	EU-incorporeerde provider + customer-managed keys + contractuele residentie

Praktisch

- Vraag **contractueel vastgelegde** residentie/soevereiniteit, geen marketingbelofte.
- Gebruik **customer-managed encryption keys** waar mogelijk.
- Documenteer waar elke databron en elke trace (H22) woont — observability-data is óók data.

Afsluiting van Deel 6

Je systeem is nu geborgd: guardrails (H26), beveiliging en toegang (H27), de juridische laag (H28) en de locatie van je data (H29). In Deel 7 brengen we alles samen — een referentiearchitectuur, een capstone en een 90-dagen-roadmap.

Bron: thenextweb (EU €180M); vstorm (Scaleway SEAL-3); gmicloud/exoscale (residentie vs soevereiniteit); ovhcloud (Mistral sovereign); introl (SAP BTP + Mistral).

Hoofdstuk 30 — Referentiearchitectuur end-to-end

DEEL 7: SAMENBRENGEN · NIVEAU: INTERMEDIATE

Alle lagen aan elkaar

In H3 zag je de kaart; nu vullen we hem in tot één werkend systeem. De stroom van een vraag, met tussen haakjes waar je het in dit boek vindt:

1. **Gebruiker** stelt een vraag.
2. **Gateway/routing** (H8) — auth, complexity-routing, fallback, kostentelling.
3. **Input-guardrail** (H26) — injection/PII/scope-check.
4. **Prompt-laag** (H10) — geversioneerde prompt + parameters.
5. **Retrieval/RAG** (H11) — identity-aware ophalen uit de vector-DB; re-ranking.
6. **Orkestratie/agent** (H12) — eventueel tools/MCP, met stap- en kostengrenzen.
7. **Model** op je **cloudplatform** (Deel 2).
8. **Output-guardrail** (H26) — faithfulness, PII-uit, schema.
9. **Antwoord** (streaming, H24).

Dwars erdoorheen: **observability** (H22), **evaluatie** (Deel 4, offline + online), **verbeterloop** (H25) en **governance** (Deel 6: logging, AVG, residentie).

Een uitgewerkt voorbeeld

Een klant-support-assistent: de gateway routeert FAQ's naar een klein model, complexe vragen naar een groot; RAG haalt alleen documenten op die deze klant mag zien; een agent kan één tool aanroepen (orderstatus) achter een goedkeuring; guardrails blokkeren PII en ongegronde claims; Langfuse logt alles; Ragas draait in CI én op 15% van het live verkeer; elke duim-omlaag wordt een golden-case. Dat is geen demo — dat is productie.

Minimaal vs. volledig

- **Minimaal**: gateway + prompt-versionering + RAG + golden-set-eval in CI + observability + basis-guardrails.
- **Volledig**: + multi-model routing, semantische cache, agent-trajectorie-eval, online-eval, FRIA/EU-dossier, soevereine hosting.

Begin minimaal, groei richting volledig naarmate schaal en regelgeving het vragen.

Bron: Truefoundry/Caylent (referentiearchitectuur); ZenML; samenvatting Deel 1–6.

Hoofdstuk 31 — Capstone: bouw, evalueer, deploy & monitor

DEEL 7: SAMENBRENGEN · NIVEAU: INTERMEDIATE

> Het hele boek in één project. Werk de mijlpalen af; aan het eind heb je een governed productiesysteem in het klein.

De opdracht

Bouw een **RAG + agent-app op één cloudplatform**, end-to-end: van use case tot gemonitorde, geëvalueerde, compliant productie.

Mijlpalen (gekoppeld aan de delen)

1. Kies & fundeer (Deel 1–2). Smalle, nuttige use case (back-office, niet flitsend). Platform op je bestaande cloud (H8); auth, gateway, kostentelling.

2. Bouw (Deel 3). RAG op je eigen documenten; één agent-tool of MCP-server met stap-/kostengrenzen; geversioneerde prompts.

3. Bewijs (Deel 4). Golden set van 30+ gevallen; meet met Ragas; CI-gate die faalt bij regressie.

4. Beheers (Deel 5). Observability (traces + kosten) vanaf dag 1; online-eval op een steekproef + alerts; één kostenhefboom (cache of routing).

5. Borg (Deel 6). In/output-guardrails; classificeer onder de EU AI Act; documenteer logging + human oversight; bepaal dataresidentie.

6. Breng samen (H30). Teken je referentiearchitectuur; wijs aan waar elke laag zit.

Succescriteria

- Haalt een afgesproken faithfulness-drempel op de golden set.
- Een brekende promptwijziging wordt door CI tegengehouden.
- Elk vreemd antwoord is reconstrueerbaar via de traces.
- Je weet wat het per opgeloste taak kost.
- Je legt in één alinea uit hoe het voldoet aan transparantie + logging.

Tip

Bouw klein en compleet liever dan groot en half. Een mini-systeem met álle lagen leert je meer dan een indrukwekkende demo zonder eval of guardrails.

Bron: synthese Deel 1–6; capstone-structuur naar ZenML/Google AI-engineering-leerpaden.

Hoofdstuk 32 — Volwassenheidsmodel & 90-dagen-roadmap

DEEL 7: SAMENBRENGEN · NIVEAU: INTERMEDIATE

Waar sta je? Een volwassenheidsmodel

Niveau	Kenmerk
L0 — Prototype	Werkt in een notebook/demo. Geen eval, geen monitoring.
L1 — Live	Draait voor gebruikers, maar "op gevoel". (Hier strandt de 95%.)
L2 — Gemeten	Golden set + evaluatie + observability. Je weet óf het werkt.
L3 — Geborgd	Guardrails, AVG/AI-Act, kostencontrole. Veilig en compliant.
L4 — Zelfverbeterend	De verbeterloop draait; drift wordt gevangen; FinOps is routine.

Het doel van dit boek: van L0/L1 naar L3–L4.

Een 90-dagen-roadmap

- **Weken 1–2 — Fundament.** Denkmodellen (Deel 1), platformkeuze + gateway + kostentelling (Deel 2). Use case gekozen.
- **Weken 3–4 — Bouwen.** RAG op je data, één agent-tool/MCP, geversioneerde prompts (Deel 3).
- **Weken 5–6 — Bewijzen.** Golden set, Ragas/DeepEval, CI-gate (Deel 4). Geen wijziging meer zonder eval.
- **Weken 7–8 — Beheersen.** Observability, online-eval-steekproef, één kostenhefboom, alerts (Deel 5).
- **Weken 9–10 — Borgen.** Guardrails, EU AI Act-classificatie + documentatie, dataresidentie (Deel 6).
- **Weken 11–12 — Samenbrengen.** Referentiearchitectuur, capstone af, verbeterloop ingericht (Deel 7). Je draait op L3+.

De rode draad, voor de laatste keer

De winnaars uit Hoofdstuk 1 onderscheiden zich niet door het nieuwste model, maar door **engineering en evaluatie**: ze meten, ze borgen, en ze laten een lus de boel verbeteren. Doe je dat, dan lever je geen demo op die het goed doet in een vergadering, maar een systeem dat je durft te laten draaien — en dat morgen beter is dan vandaag.

Dat is het verschil tussen prototype en productie. Succes met bouwen.

Verder

Zie de appendices: tool-directory met prijzen (A), eval-prompt-templates (B), productie-checklist (C), woordenlijst (D).

Bron: synthese van het hele boek; MIT NANDA (5%/95%-these); ZenML (1.200 deployments).

Appendix A — Tool-directory (met indicatieve prijzen)

PRIJZEN SCHUIVEN; VERIFIEER BIJ AANSCHAF. STAND: 2026.

Laag	Tools	Indicatie	Het best voor
Cloud-platform	AWS Bedrock · Azure OpenAI/ Foundry · Google Vertex	per token + platformkosten	breedste catalogus / Microsoft-huis / data-zwaar
Gateway/ routing	LiteLLM (OSS) · Portkey · Helicone	gratis–$79/mnd	portabiliteit, kosten, fallback
Vector-DB	pgvector/Supabase · Pinecone · Redis · managed	gratis self-host–usage	RAG-retrieval
Serving	vLLM (OSS) · BentoML · Baseten	self-host/GPU-kosten	latency/throughput, soevereiniteit
Evaluatie	Ragas · DeepEval · Promptfoo · TruLens · Braintrust	gratis–$249/mnd	RAG-eval / CI / prompt-A-B / feedback
Observability	Langfuse · LangSmith · Arize Phoenix · Helicone · Datadog	gratis–enterprise	OSS/self-host / LangChain / eval-rigor / APM
Guardrails	Guardrails AI · NeMo Guardrails · Lakera Guard · Galileo Luna	gratis–enterprise	schema/PII / dialoog / injection / realtime
EU-soeverein	Mistral · Scaleway (SEAL-3) · OVHcloud · StackIT	usage/enterprise	residentie + soevereiniteit

Appendix B — Eval-prompt-templates

PAS AAN OP JE DOMEIN; LIJN ALTIJD UIT OP MENSELIJK OORDEEL (H17).

Faithfulness-judge — Je bent een strenge beoordelaar. Gegeven de CONTEXT en het ANTWOORD: is elke bewering direct ondersteund door de context? Geef eerst je redenering, dan een score 1–5 (5 = volledig gegrond, 1 = grotendeels verzonnen). Markeer ongegronde beweringen.

Answer-relevancy-judge — Gegeven de VRAAG en het ANTWOORD: beantwoordt het de vraag zonder af te dwalen? Redeneer kort, geef dan 1–5. Straf irrelevante uitweidingen af, hoe correct ook.

Generieke rubriek (1–5) — 5 = correct, gegrond, beknopt; 4 = klein gebrek; 3 = bruikbaar met fouten/ruis; 2 = deels fout/irrelevant; 1 = fout/verzonnen/off-topic. Altijd eerst de reden, dan het cijfer.

Tip: vraag JSON terug (reden + score) zodat je het machinaal in CI (H19) verwerkt.

Appendix C — Productie-checklist (go-live readiness)

Bouw — prompts geversioneerd buiten code; RAG (chunking/embeddings/retrieval) gemeten + kennisbank-eigenaar; agents met stap-/kostenplafond, volledig getraceerd.

Bewijzen — golden set (30+) in repo; eval (Ragas/DeepEval) in CI met gate die faalt bij regressie; judge uitgelijnd op mensen.

Beheersen — observability vanaf dag 1 (traces, kosten, kwaliteit); online-eval-steekproef + alerts (kwaliteit, dag-spend, p95); minstens één kostenhefboom (cache/routing).

Borgen — in/output-guardrails (PII, injection, faithfulness); secrets in vault, least-privilege, identity-aware retrieval; EU AI Act-classificatie + documentatie (logging, human oversight); dataresidentie/soevereiniteit bepaald en contractueel vastgelegd.

Loop — feedback-knop + auto-flag laagscorende traces; wekelijkse triage → nieuwe golden-cases.

Appendix D — Woordenlijst

- **MLOps** — beheren van de volledige ML-levenscyclus rond een model.
- **LLMOps** — MLOps voor LLM's: prompts, RAG, evaluatie, guardrails, kosten.
- **AgentOps** — operationele laag voor autonome agents (trajectorie, tools, geheugen).
- **RAG** — Retrieval-Augmented Generation: eigen kennis ophalen en het model erop laten antwoorden.
- **Embeddings / vector-DB** — tekst als vectoren opslaan om op betekenis te zoeken.
- **Faithfulness** — mate waarin een antwoord gegrond is in de context (hallucinatie-meter).
- **LLM-as-a-judge** — een model dat outputs scoort volgens een rubriek.
- **Golden set** — vaste set testgevallen met verwacht resultaat; je maatstaf.
- **Drift** — langzaam verslechteren van prestaties (input-, kwaliteits-, retrieval-drift).
- **Guardrails** — in/output-filters die onveilig of fout gedrag tegenhouden.
- **Prompt injection** — kwaadaardige instructies verstopt in input/data die een agent overneemt.
- **FinOps / FOCUS** — kostenbeheer; FOCUS normaliseert cloudkosten.
- **MCP** — Model Context Protocol: standaard om tools/data aan agents aan te bieden.
- **Residentie vs. soevereiniteit** — waar data fysiek staat vs. onder welke jurisdictie (CLOUD Act).
- **EU AI Act** — Europese AI-wet; transparantie + AI-geletterdheid vanaf 2 aug 2026, zware hoog-risico → eind 2027.
- **DPIA / FRIA** — gegevensbeschermings- resp. grondrechteneffectbeoordeling.