



Les 15

RAG + Embeddings

AI laten antwoorden op basis van jouw eigen documenten



Terugblik

Het probleem dat we nu oplossen

Lessen 11-14: AI SDK, Tool Calling, Cursor+Vercel, Agents.

Maar AI weet NIET wat in jouw documenten staat:

- Interne kennisbank
- Klantcontracten
- Productdocumentatie
- 200-pagina PDF die vandaag binnenkwam

Twée opties:

X Alle 200 pagina's elke vraag meesturen — niet schaalbaar

✓ RAG — alleen relevante stukjes per vraag

Planning

Vandaag — 180 minuten

Welkom + Terugblik	10 min
Theorie: Wat is een embedding?	20 min
Theorie: RAG pipeline	15 min
Theorie: pgvector + chunking	15 min
LIVE DEMO 1 — pgvector + embed pipeline	25 min
LIVE DEMO 2 — PDF upload + index	20 min
Pauze	15 min
LIVE DEMO 3 — Query pipeline + AI antwoord	30 min
Wanneer RAG wel/niet	10 min
Lesopdracht + Huiswerk	10 min
Vragen + Afsluiting	5 min

Theorie 1

Wat is een embedding?

Tekst in → vector terug. Array van ~1536 nummers.

"De kat zit op de mat" → [0.12, -0.45, 0.88, ...]

"Een poes ligt op het tapijt" → [0.14, -0.41, 0.85, ...]

"Voetbal in Nederland" → [-0.73, 0.21, -0.32, ...]

Eerste twee dichtbij in vector-space. Derde ver weg. Op basis van BETEKENIS.

Welk model:

- text-embedding-3-small – 1536 dim, snel + goedkoop (default)
- text-embedding-3-large – 3072 dim, beter maar duurder
- nomic-embed-text – open-source, lokaal te draaien

Theorie 1

Vector similarity — hoe meet je 'dichtbij'?

Cosine similarity

Hoek tussen vectors (-1 tot 1)

Default voor text

Dot product

Sum van producten

Sneller (genormaliseerd)

Euclidean

Afstand in ruimte

Zelden voor text

Cosine similarity waarden:

1.0

exact gelijke betekenis

0.8-0.95

sterk gerelateerd

0.5-0.8

zwak gerelateerd

< 0.5

meestal niet relevant

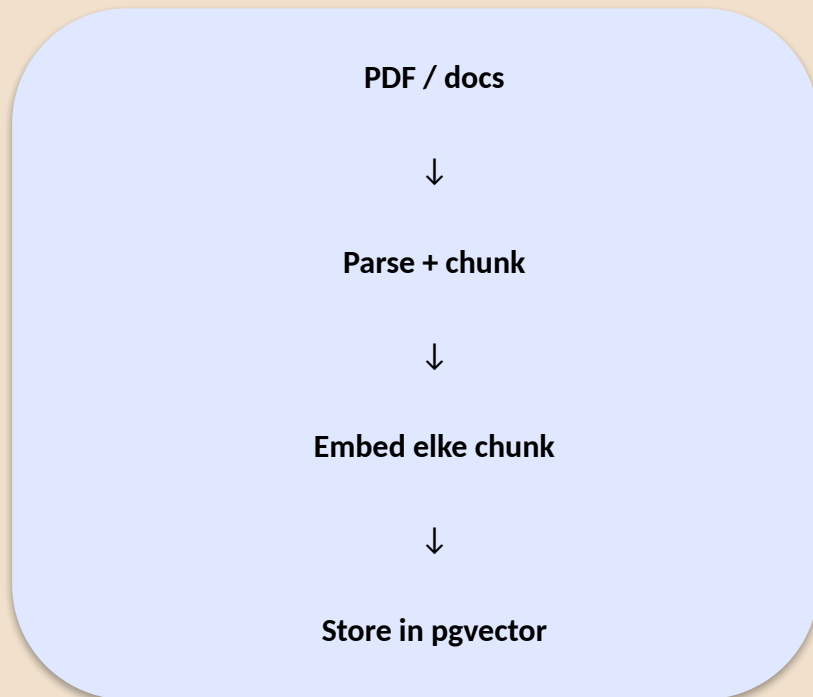
In pgvector: `<=>` is cosine distance (kleinere = similar)

```
SELECT * FROM chunks ORDER BY embedding <=> $1::vector LIMIT 5;
```

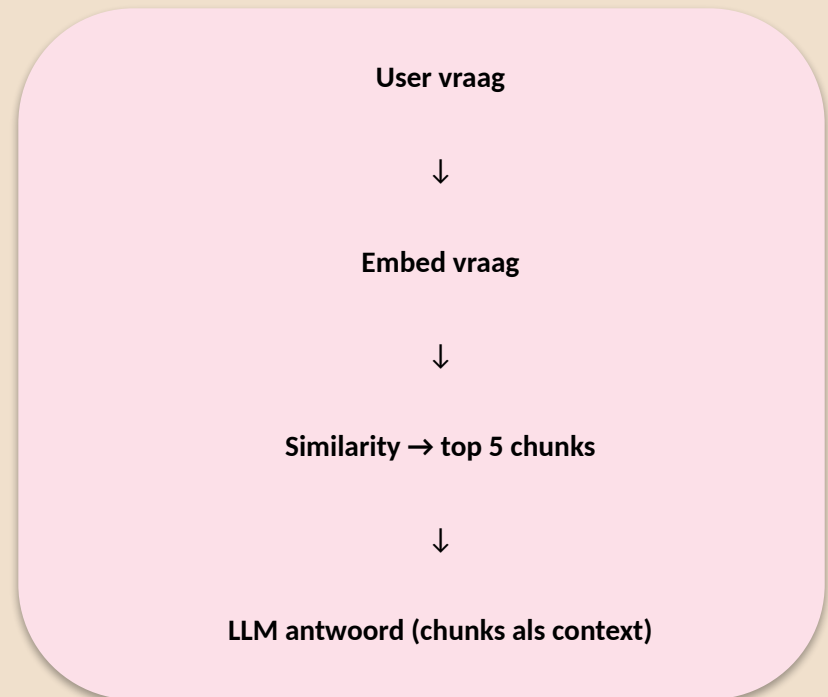
Theorie 2

RAG pipeline — index + query

Index time (eenmalig)



Query time (elke vraag)



Inzicht: LLM ziet nooit hele DB. Alleen ~5 chunks per vraag.

Theorie 3

pgvector — Postgres met vector-support

- Postgres extension — voegt vector datatype toe
- Cosine, euclidean, dot product operators
- Schaalbaar tot ~1M+ vectors per tabel
- Ingebouwd in Supabase — één click activeren

```
create extension if not exists vector;

create table chunks (
  id          bigserial primary key,
  source      text not null,
  page        int,
  content     text not null,
  embedding   vector(1536),
  created_at  timestamp default now()
);

create index on chunks
  using hnsw (embedding vector_cosine_ops);
```

HNSW index = approximate nearest neighbor. 100x sneller bij grote tabellen.

Theorie 3

Chunking — hoe knip je een document op?

Fixed size

500 tokens, 50 overlap

Default, simpelste

Recursive

Splits op para → zin → woord

Behoudt structuur

Semantic

Embed zinnen, group similar

Beste kwaliteit

Wat is een goede chunk:

- ~200-500 tokens (~1000-2500 chars)
- Overlap 10-15% — info aan grenzen niet verliezen
- Houd semantisch geheel — niet midden in zin knippen

Te klein → fragmentatie. Te groot → ruis verdunt signal.

Vandaag bouwen we

PDF Q&A from scratch

Doel: upload PDF, stel er vragen over.

Stack:

- Next.js 16 + TypeScript + Tailwind
- Supabase + pgvector
- AI SDK embedMany + generateText
- OpenAI text-embedding-3-small
- unpdf voor PDF parsing

Twee endpoints:

POST /api/index

PDF upload → chunks → embeddings

Eén keer per doc

POST /api/ask

Vraag → embed → search → answer

Per vraag

Demo flow: setup pgvector + index PDF + query met AI-antwoord.

LIVE DEMO 1

pgvector setup + embed pipeline — ~25 min

1 `pnpm create next-app + add ai @supabase/supabase-js unpdf`

2 Supabase: activeer pgvector extension

3 Schema: chunks tabel met vector(1536) + HNSW index

4 `lib/embeddings.ts`: `embedOne` + `embedBatch` + `chunkText`

5 Test: `console.log` embedding — array van 1536 nummers

`embedMany` = batch embed — sneller dan loop

Eén API-call voor honderden chunks i.p.v. honderden roundtrips.

LIVE DEMO 2

PDF upload + index — ~20 min

```
// app/api/index/route.ts
const formData = await req.formData();
const file = formData.get("file") as File;
const buffer = new Uint8Array(await file.arrayBuffer());
const { text } = await extractText(buffer, { mergePages: true });

const chunks = chunkText(text, 500, 50);
const embeddings = await embedBatch(chunks);

await supabase.from("chunks").insert(
  chunks.map((content, i) => ({
    source: file.name, content, embedding: embeddings[i],
  }))
);
```

Demo:

- Upload polderfest-lineup.pdf via curl
- Wachten 5-10s — response: 30 chunks
- Supabase Table Editor — chunks zichtbaar met embedding



Pauze

15 minuten

LIVE DEMO 3

Query pipeline + AI antwoord — ~20 min

```
// app/api/ask/route.ts
const { question } = await req.json();
const embedding = await embedOne(question);

const { data: chunks } = await supabase.rpc("match_chunks", {
  query_embedding: embedding, match_count: 5,
});

const context = chunks.map((c) => c.content).join("\n\n---\n\n");
const { text } = await generateText({
  model: openai("gpt-4o-mini"),
  prompt: `Beantwoord op basis van deze context:\n${context}\n\nVraag: ${question}`,
});
return Response.json({ answer: text, sources: chunks });
```

Demo: 'Wie zijn de headliners?' → accuraat antwoord + sources!

Reflectie

Wanneer RAG wel, wanneer niet?

WEL RAG

- ✓ Veel docs (>50 pag)
- ✓ Documenten veranderen
- ✓ Semantic search nodig
- ✓ Privacy — data in jouw DB
- ✓ Bronvermelding belangrijk

GEEN RAG

- X Klein doc (<10 pag) — in prompt
- X Exacte data — tool-calls / SQL
- X Structured data — SQL beter
- X 1 keer per dag — overhead
- X Code-base — grep / tree-sitter

Hybrid is vaak best:

- Semantic search (RAG) voor concept-vragen
- Keyword / full-text voor exacte termen
- SQL filter voor metadata (datum, categorie)

Lesopdracht + Huiswerk

Lesopdracht (30 min)

- PDF Q&A app opzetten
- pgvector + schema
- 1 PDF indexed (Tim deelt)
- 3 vragen werkend
- Chunks in Supabase

Huiswerk (~2 uur, voor Les 16)

- Eigen PDF (min 20 pag)
- Chat-UI met streaming
- Simpele chat-UI (vraag/antwoord)
- RAG.md (5 secties)
- Bonus: hybrid / re-ranking

Beoordeling (10 pt — voldoende 6+):

- | | |
|------------------------------------|------|
| • A — UI + indexing + chat werkend | 3 pt |
| • B — RAG.md compleet (4 secties) | 3 pt |
| • C — RAG.md compleet (5 secties) | 3 pt |
| • Fail-case analyse concreet | 1 pt |
| • Chunking-experiment + reflectie | 1 pt |

Afsluiting

Vragen?

Vandaag gezien:

- ✓ Embeddings — tekst als vectors in semantic space
- ✓ Cosine similarity voor 'dichtbij'
- ✓ RAG pipeline — index time vs query time
- ✓ pgvector in Supabase met HNSW index
- ✓ Chunking strategieën (fixed / recursive / semantic)
- ✓ Wanneer RAG wel/niet inzetten

Volgende les (Les 16): MCP — Model Context Protocol

- Externe APIs in Next.js Server Components
- Cursor Composer + Background Agents
- Deploy naar Vercel productie + preview per branch
- Daarna Les 16 (MCP), Les 17 (Externe APIs deep), Les 18 (Auth+RLS)

Vragen? Feedback?