

Les 15 — RAG + Embeddings

AI laten antwoorden op basis van jouw eigen documenten

Vak:	AI-Assisted Development
Opleiding:	NOVI Hogeschool Utrecht
Klas:	Klas A (3 uur fysiek)
Vervolg op:	Les 14 — Agents
Volgende les:	Les 16 — MCP servers

Leerdoelen

- Begrijpen wat een embedding is en wat 'semantic space' betekent
- Cosine similarity kennen en kunnen toepassen
- RAG pipeline opzetten: index time + query time
- pgvector activeren in Supabase met HNSW index
- Chunking-strategieën kennen en juist kiezen
- Index- en query-endpoints schrijven in Next.js
- RAG-tool integreren in een ToolLoopAgent
- Weten wanneer RAG wel/niet de juiste keuze is

Inhoud

1. Het probleem dat RAG oplost
2. Wat is een embedding?
3. Vector similarity
4. RAG pipeline
5. pgvector in Supabase
6. Chunking strategieën
7. Index pipeline in code
8. Query pipeline in code
9. RAG-tool in een agent
10. Wanneer wel/niet RAG
11. Productie-overwegingen

12. Bronnen

1

Het probleem dat RAG oplost

AI weet veel, maar weet NIET wat in **jouw** documenten staat. Interne kennisbank, klantcontracten, productdocs, dat handboek dat vandaag binnenkwam.

Twee naïeve aanpakken die niet werken:

- **Hele document meesturen** — werkt voor 5 pag, niet voor 500
- **AI zelf laten zoeken** — werkt voor publiek, niet voor private data

RAG in één zin

Haal de **relevante stukjes** uit je documenten op, geef die aan AI als context, AI antwoordt.

2 Wat is een embedding?

Een embedding is een **vector** — array van ~1536 nummers tussen -1 en 1. Tekst erin, vector terug.

```
"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, ...]
```

Vergelijkbare betekenis = vergelijkbare vector. Niet woord-overlap — **betekenis**.

Embedding-modellen

Model	Dim	Cost	Wanneer
text-embedding-3-small	1536	\$0.02/1M	Default
text-embedding-3-large	3072	\$0.13/1M	Betere accuracy
nomic-embed-text	768	Gratis	Lokaal/privacy

In AI SDK

```
import { embed, embedMany } from "ai";
import { openai } from "@ai-sdk/openai";

const model = openai.textEmbeddingModel("text-embedding-3-small");

// Een
{ embedding } = await embed({ model, value: "tekst" });

// Veel tegelijk (sneller)
{ embeddings } = await embedMany({ model, values: ["a", "b"] });
```

3

Vector similarity

Metric	Wat	Wanneer
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
- **0.0** — ongerelateerd

In pgvector

```
-- <=> is cosine distance (kleinere = similar)
SELECT content, 1 - (embedding <=> $1) as similarity
FROM chunks
ORDER BY embedding <=> $1
LIMIT 5;
```

4

RAG pipeline

Twee pipelines: **index time** (eenmalig) en **query time** (per vraag).

Index time

```
PDF / docs
-> Parse + chunk (~500 tokens per chunk)
-> Embed elke chunk
-> Store: chunk_text + embedding in pgvector
```

Query time

```
User vraag
-> Embed vraag (zelfde model als index!)
-> Cosine similarity search -> top-k chunks
-> Geef chunks als context aan LLM
-> LLM antwoordt op basis van context
```

Inzicht

LLM ziet **nooit** de hele DB. Alleen ~5 relevante chunks per vraag. Schaalbaar tot miljoenen documenten.

Context prompt template

```
const prompt = `Beantwoord de vraag op basis van deze context.
Als de context geen antwoord bevat, zeg dat eerlijk.

CONTEXT:
${chunks.map((c) => c.content).join("\n\n--\n\n")}

VRAAG: ${question}

ANTWOORD: `;
```

5

pgvector in Supabase

Postgres extension. 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

Hierarchical Navigable Small Worlds. Approximate nearest neighbor. Tot 100x sneller dan exact search bij grote tabellen. Voor <10k rows kun je het skippen.

Match function (RPC)

```
create or replace function match_chunks(
  query_embedding vector(1536),
  match_count int default 5
)
returns table (
  id bigint, content text, source text, similarity float
)
language sql stable as $$
select chunks.id, chunks.content, chunks.source,
  1 - (chunks.embedding <=> query_embedding) as similarity
from chunks
order by chunks.embedding <=> query_embedding
limit match_count;
$$;
```

6

Chunking strategieën

Strategie	Hoe	Wanneer
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 behouden
- **Semantisch geheel** — niet midden in zin knippen

Fixed size in JS

```
function chunkText(text: string, size = 500, overlap = 50) {  
  const chunks: string[] = [];  
  let i = 0;  
  while (i < text.length) {  
    chunks.push(text.slice(i, i + size));  
    i += size - overlap;  
  }  
  return chunks;  
}
```

7

Index pipeline in code

```
// app/api/index/route.ts
import { extractText } from "unpdf";
import { embedMany } from "ai";

export async function POST(req: Request) {
  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 embedMany({ model, values: chunks });

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

Kosten

200-pag PDF = ~80k tokens = ~160 chunks. Embedding kost ~\$0.0016 (minder dan cent). Indexing is goedkoop + snel.

8

Query pipeline in code

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

const { embedding } = await embed({ model, value: question });

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

const context = chunks.map((c, i) =>
  `[Source ${i+1}: ${c.source}]\n${c.content}`
).join("\n\n---\n\n");

const { text } = await generateText({
  model: openai("gpt-4o-mini"),
  prompt: `Beantwoord op basis van context:\n${context}\n\nVraag: ${question}`,
});

return Response.json({ answer: text, sources: chunks });
```

Tips:

- Geef bronnen mee in output — gebruiker kan verifiëren
- **top-k = 5** is goede default, experimenteer per case
- Stream output (streamText) voor betere UX

9

RAG-tool in een agent

Combo Les 13 (Agents) + Les 15 (RAG):

```
const ragSearch = tool({
  description: "Zoek in geuploade documenten",
  inputSchema: z.object({ query: z.string() }),
  execute: async ({ query }) => {
    const embedding = await embedOne(query);
    const { data } = await supabase.rpc("match_chunks", {
      query_embedding: embedding, match_count: 5,
    });
    return data;
  },
});

const docAgent = new ToolLoopAgent({
  model: openai("gpt-4o-mini"),
  system: "Zoek, lees, eventueel 2e search, antwoord met sources.",
  tools: { ragSearch },
  stopWhen: stepCountIs(10),
});
```

Extra t.o.v. simple RAG:

- **Multi-hop reasoning** — agent kan 2-3 searches doen
- **Query rewriting** — agent verbetert eigen zoek-query
- **Iterative refinement** — eerste resultaat slecht? probeer andere

10 Wanneer wel/niet RAG

WEL RAG

- Veel documenten (>50 pag totaal)
- Documenten veranderen vaak
- Semantic search nodig (niet alleen exact match)
- Privacy: data moet in jouw DB blijven
- Bronvermelding is belangrijk

GEEN RAG

- Klein document (<10 pag) — gewoon in system prompt
- Exacte data (prijzen, IDs) — tool-calls / SQL
- Structured data (tabellen) — SQL beter
- 1 keer per dag bevraagd — overhead niet de moeite
- Code-base — grep / tree-sitter, geen embeddings

Hybrid is vaak best

```
-- Voorbeeld hybrid in Supabase
select content from chunks
where source = 'product-handleiding.pdf' -- metadata filter
and (
  content ilike '%firmware%'           -- keyword
  or embedding <=> $1 < 0.5             -- semantic
)
order by embedding <=> $1 limit 5;
```

11 Productie-overwegingen

Re-ranking

Top-5 van vector search != beste top-5 voor LLM. Re-ranking stap:

1. Vector search -> 20 candidates
2. Re-rank model (Cohere, BGE) -> top 5
3. Aan LLM geven

Cohere rerank-3 = \$1/1k searches.

Evaluation

- 20 vragen + verwachte antwoorden
- Run RAG, vergelijk output
- LLM-as-judge voor kwaliteit
- Track over tijd

Privacy

- Embeddings NIET reversible — geen tekst terughalen uit vector
- Maar: vergelijkbare zinnen -> vergelijkbare vectors (niet 100% anoniem)
- Voor gevoelige data: lokaal embedding model (Ollama)

12 Bronnen

- AI SDK Embeddings — ai-sdk.dev/docs/ai-sdk-core/embeddings
- OpenAI Embeddings — platform.openai.com/docs/guides/embeddings
- pgvector — github.com/pgvector/pgvector
- Supabase pgvector — supabase.com/docs/guides/database/extensions/pgvector
- Supabase vector search guide — supabase.com/docs/guides/ai/vector-columns
- Anthropic Contextual Retrieval — anthropic.com/news/contextual-retrieval
- unpdf — github.com/unjs/unpdf
- Cohere rerank — docs.cohere.com/docs/reranking
- Ragas (RAG evaluation) — docs.ragas.io