

Les 14

Agents

Een LLM in een loop met tools — autonoom tot het klaar is

Van tool-calling naar autonome agents

Tool Calling recap (Les 12):

- Tool Calling met description + inputSchema + execute
- stopWhen: stepCountIs(5) — multi-step
- Polderfest chat met 6 tools

Wat een tool-call bot nog NIET kan:

- X Lange ketens (30+ stappen)
- X Eigen plan opstellen + bijstellen
- X Tools die andere tools triggeren
- X Dynamisch model wisselen per stap

Vandaag: agents. Eén stap verder qua autonomie.

Planning

Vandaag — 180 minuten

Welkom + Terugblik	10 min
Theorie: wat is een agent?	30 min
LIVE DEMO 1 — Setup research-agent	25 min
LIVE DEMO 2 — ToolLoopAgent + stopWhen	25 min
Pauze	15 min
LIVE DEMO 3 — prepareStep	25 min
LIVE DEMO 4 — done-tool + custom stop	20 min
Wanneer agent vs tool-call vs workflow?	10 min
Lesopdracht + Huiswerk	15 min
Vragen + Afsluiting	5 min

Format: Demo-driven. Code-fragmenten in slides + live in editor.

Theorie

Wat is een agent?

"Agents are LLMs that use tools in a loop to accomplish tasks."

— AI SDK docs

LLM

Beslist iedere stap wat de volgende actie is

Tools

Uitbreidingen: search, read, write, anything

Loop

Beheert berichten + stop-condities

Verschil met Les 12:

Les 12: tot 5 stappen, korte taken. Les 14: 20-50+ stappen, langere autonome workflows.

Anatomie van de loop

- 1 Stuur messages naar LLM
- 2 LLM: text-only? → stop. tool-call? → ga door
- 3 Execute tool(s)
- 4 Voeg tool-result toe aan messages
- 5 Check stopWhen
- 6 Niet gestopt? volgende iteratie

Loop stopt wanneer:

- Model geeft text in plaats van tool-call (natural finish)
- stopWhen-conditie voldaan
- Tool zonder execute aangeroepen (done-pattern)
- Tool needs approval

ToolLoopAgent — de v6 abstractie

```
import { ToolLoopAgent, tool, stepCountIs } from "ai";

const researchAgent = new ToolLoopAgent({
  model: "openai/gpt-4o",
  system: "Je bent een research-assistent.",
  tools: { webSearch, readPage, saveReport },
  stopWhen: stepCountIs(30),
});

const result = await researchAgent.generate({
  prompt: "Onderzoek AI in de Nederlandse bouwsector",
});

console.log(result.text); // antwoord
console.log(result.steps); // alle stappen
```

Minder boilerplate

Geen handmatige while-loop

Herbruikbaar

Definieer 1x, gebruik overal

Single config

System, tools, model, stop op één plek

Stop-condities

stepCountIs(N)

Stop na N stappen

Default veiligheid

hasToolCall('done')

Stop bij specifieke tool

Done-pattern

isLoopFinished()

Nooit — model bepaalt zelf

Lange autonomie (risico!)

Custom StopCondition

Eigen callback op steps[]

Kosten / token-budget

```
stopWhen: [ stepCountIs(50), hasToolCall('submit'), customCondition ]  
// stop zodra ÉÉN conditie waar is
```

Theorie

prepareStep — control per stap

Callback die VOOR elke stap draait. Je kunt aanpassen:

- model wisselen (mini → sonnet voor complex reasoning)
- activeTools beperken (fase-gebaseerd)
- messages trimmen (context-budget bewaken)
- toolChoice forceren (bv. 'required' of specifieke tool)

```
prepareStep: async ({ stepNumber, messages }) => {  
  if (stepNumber > 2 && messages.length > 10) {  
    return { model: "anthropic/claude-sonnet-4.5" };  
  }  
  return {}; // niks aanpassen  
},
```

```
// dynamic model: begin goedkoop, switch naar duur voor reasoning
```

Vandaag bouwen we

Research-agent from scratch

Doel: zelfstandig een onderzoeksvraag beantwoorden

Web zoeken → pagina's lezen → rapport schrijven → opslaan in Supabase

Stack:

- Next.js 16 + TypeScript + Tailwind
- AI SDK v6 (ToolLoopAgent)
- Tavily (web search + extract)
- Supabase (rapport opslag)

Tools die we bouwen:

webSearch

Tavily query → 5 resultaten

Read

readPage

URL → fulltext extractie

Read

saveReport

Insert in research_reports

Write

done

No-execute, signaal klaar

Signal

LIVE DEMO 1

Setup research-agent — ~25 min

1

pnpm create next-app + add ai zod @supabase/supabase-js
Setup

2

.env.local met TAVILY + OPENAI + SUPABASE keys
Config

3

Supabase: research_reports tabel + RLS policy
Database

4

lib/tools.ts: webSearch met Tavily fetch
Tools

5

lib/agent.ts: ToolLoopAgent met system + tools + stopWhen
Agent

6

app/api/research/route.ts: POST endpoint met agent.generate()
API

7

curl test in terminal — query → JSON met text + steps
Test

Tavily = gratis 1000 calls/maand. Studenten maken zelf account aan.

ToolLoopAgent + stopWhen variants — ~25 min

- 1 Voeg readPage + saveReport toe als tools
- 2 Test met stepCountIs(10) — agent doet 8-15 stappen
- 3 Switch naar hasToolCall('saveReport') — stop bij opslaan
- 4 Combineer in array: [stepCountIs(20), hasToolCall('saveReport')]
- 5 Test isLoopFinished() — geen limit, alleen natuurlijke stop
- 6 Custom StopCondition: 'stop als 3 pagina's gelezen'

Doel: laten zien dat dezelfde agent met andere stopWhen anders gedrag vertoont

Console log: hoeveel stappen, welke tools — verschil per stop-conditie



Pauze

15 minuten

prepareStep — dynamic control — ~25 min

Use case 1 — Dynamic model

Stap 0-3: gpt-4o-mini (goedkoop). Stap 4+: gpt-4o (sterk reasoning).

Use case 2 — Fase-gebaseerde tools

Stap 0-2: webSearch. Stap 3-5: readPage. Stap 6+: saveReport (required).

Use case 3 — Context-trimming

messages > 12? Keep system + laatste 8. Voorkomt token-explosie.

Live: zien dat console-logs verschillen tussen 'met' en 'zonder' prepareStep

LIVE DEMO 4

Done-tool + custom stop — ~20 min

Pattern 1 — Done-tool (no-execute)

```
done: tool({
  description: "Signaleer dat onderzoek klaar is.",
  inputSchema: z.object({ reportId: z.number(), summary: z.string() }),
  // GEEN execute – stopt de loop
})
```

Pattern 2 — Custom stop op kosten/tokens

```
const costLimit: StopCondition<typeof tools> = ({ steps }) => {
  const tokens = steps.reduce((s, x) => s + (x.usage?.totalTokens ?? 0), 0);
  return tokens > 50_000;
};
```

Pattern 3 — Plan-Act-Reflect (ReAct)

System prompt: '1. plan. 2. uitvoeren met tools. 3. reflecteer.' Geen extra code.

Reflectie

Agent vs tool-call vs workflow

Vraag → 1 tool → antwoord

→ Plain tool-call

Vraag → 2-5 tools → antwoord

→ stopWhen: stepCountIs(5) (Les 12)

Onderzoek → 10-50 tools → rapport

→ Agent (Les 14)

Productie met validatie + audit

→ Workflow (expliciete code)

Wanneer GEEN agent:

Determinisme nodig • Latency-sensitive (< 1s) • Voorspelbare kosten • Eenvoudige flow

Wanneer WEL:

Open-ended taken • Volgorde onbekend • Lange ketens • Async use cases

Lesopdracht + Huiswerk

Lesopdracht (30 min)

- Setup research-agent
- Tavily account + key in .env
- Agent draaien met stepCountIs(10)
- Eerste eigen query stellen
- Logs bekijken — welke stappen

Huiswerk (~2 uur, voor Les 15)

- 4e tool: listReports
- prepareStep — 1 use case
- Custom StopCondition
- AGENT.md — 5 secties
- Bonus: live-step UI / sub-agent

Beoordeling (10 pt — voldoende 6+):

- | | |
|---|------|
| • listReports werkt + agent gebruikt 'm | 2 pt |
| • prepareStep + zichtbaar effect | 2 pt |
| • Custom StopCondition | 2 pt |
| • AGENT.md compleet (5 secties) | 3 pt |
| • End-to-end werkend | 1 pt |

Afsluiting

Vragen?

Vandaag gezien:

- ✓ Agent = LLM + tools + loop
- ✓ ToolLoopAgent met system, tools, stopWhen, prepareStep
- ✓ 4 stop-condities: stepCountIs, hasToolCall, isLoopFinished, custom
- ✓ prepareStep voor dynamic model / tools / context
- ✓ Done-tool pattern + ReAct planning
- ✓ Wanneer wel/niet een agent

Volgende les (Les 15): RAG + Embeddings

- Semantic search op grote datasets met pgvector
- Embedding models — wat zijn vectors
- RAG-pipeline: chunk → embed → retrieve → generate
- Combo: RAG-tool in een agent

Vragen? Feedback?