

Les 13 — Agents

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

Vak:	AI-Assisted Development
Opleiding:	NOVI Hogeschool Utrecht
Klas:	Klas A (3 uur fysiek, demo-driven)
Vervolg op:	Les 12 — Tool Calling
Volgende les:	Les 14 — RAG + embeddings

Leerdoelen

- Definitie en componenten van een agent kennen (LLM + tools + loop)
- ToolLoopAgent class gebruiken in plaats van handmatige loop
- Vier stop-condities kennen: stepCountIs, hasToolCall, isLoopFinished, custom
- prepareStep inzetten voor dynamic model, tools en context
- Done-tool pattern toepassen voor gestructureerde eindantwoorden
- Planning + reflectie patronen herkennen (ReAct, sub-agents)
- Wanneer agent vs tool-call vs workflow inzetten

Inhoud

1. Wat is een agent
2. De agent-loop in detail
3. ToolLoopAgent — de v6 abstractie
4. Stop-condities
5. prepareStep — control per stap
6. Done-tool pattern
7. Planning + reflectie patronen
8. Agent vs tool-call vs workflow
9. Productie-overwegingen
10. Bronnen

1

Wat is een agent

Definitie (AI SDK docs)

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

Drie ingrediënten:

- **LLM** — beslist iedere stap wat de volgende actie is
- **Tools** — uitbreidingen van wat het model kan
- **Loop** — runtime die LLM en tools afwisselt

Vershil met Les 12

In Les 12 deden we maximaal 5 stappen — kort en gecontroleerd. In Les 13 gaan we naar 20-50 stappen, en het model bepaalt zelf welke tools het in welke volgorde gebruikt.

Voorbeelden van agent-taken

- **Research agent** — zoek web, lees pagina's, schrijf rapport
- **Coding agent** — lees code, draai tests, voer fix door
- **Planner** — plan een vakantie op basis van voorkeuren
- **Support triage** — classify ticket, gather context, draft response

2

De agent-loop in detail

Per iteratie:

STAP N:

1. Stuur huidige messages naar LLM
2. LLM antwoordt met:
 - a) text-only? -> loop stopt
 - b) tool-call(s)? -> ga door
3. Execute(s) de tools (parallel mogelijk)
4. Voeg tool-result(s) toe aan messages
5. Check stopWhen-conditie
6. Niet gestopt -> STAP N+1

De loop stopt in vier gevallen:

- **Natural finish** — model genereert text in plaats van tool-call
- **stopWhen voldaan** — bijv. stepCounts(20)
- **Tool zonder execute aangeroepen** — done-pattern
- **Tool needs approval** — agent vraagt user-confirmation (advanced)

Default

stepCounts(20). Veiligheidsgrens tegen runaway loops.

3

ToolLoopAgent — de v6 abstractie

In oudere AI SDK versies schreef je de loop zelf met `streamText`. In v6 is er een dedicated class: **ToolLoopAgent**.

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

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: "Schrijf een rapport over AI in de bouwsector",
});

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

Voordelen

- **Minder boilerplate** — geen handmatige while-loop
- **Herbruikbaar** — definieer 1x, gebruik in API-routes / scripts / queues
- **Single source of config** — system, tools, model, stop op één plek

Streaming vs generate

```
// Eenmalig antwoord
const result = await agent.generate({ prompt });

// Voor chat-UI: streaming
const result = agent.stream({ prompt });
return result.toUIMessageStreamResponse();
```

4

Stop-conditions

stepCountIs(N) — maximum aantal stappen

```
stopWhen: stepCountIs(50) // stop na 50 stappen
```

Bruikbaar als veiligheidsgrens. Default = stepCountIs(20).

hasToolCall(toolName)

```
stopWhen: hasToolCall("saveReport")
```

Stop zodra een bepaalde tool is aangeroepen. Handig voor 'doe het werk en sla op, dan klaar'.

isLoopFinished() — onbeperkt

```
stopWhen: isLoopFinished() // GEEN limit
```

Waarschuwing

Zonder limiet kan agent oneindig doorgaan. Alleen gebruiken met andere safeguards (cost limit, timeout).

Combineren met array

```
stopWhen: [  
  stepCountIs(50),           // max 50 stappen  
  hasToolCall("submit"),     // of submit aangeroepen  
]
```

Custom StopCondition

```
import { StopCondition, ToolSet } from "ai";

const tools = { webSearch, readPage, saveReport } satisfies ToolSet;

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

new ToolLoopAgent({
  tools,
  stopWhen: [stepCountIs(30), budgetExceeded],
});
```

5

prepareStep — control per stap

Async callback die VOOR elke stap draait. Je kunt dynamisch model, tools, messages of toolChoice aanpassen.

Signature

```
prepareStep: async ({
  model,          // huidig model
  stepNumber,     // 0-indexed
  steps,          // alle vorige steps
  messages,       // berichten die naar model gaan
}) => {
  return {
    model?: ...,
    messages?: ...,
    activeTools?: [...],
    toolChoice?: ...,
  };
}
```

Dynamic model selection

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

Tool-filtering per fase

```
prepareStep: async ({ stepNumber }) => {
  if (stepNumber <= 3) {
    return { activeTools: ["webSearch"], toolChoice: "required" };
  }
  if (stepNumber <= 6) {
    return { activeTools: ["readPage"] };
  }
  return { activeTools: ["saveReport", "done"], toolChoice: "required" };
}
```

Context-trimming

```
prepareStep: async ({ messages }) => {
  if (messages.length > 20) {
    return {
      messages: [
        messages[0],           // system
        ...messages.slice(-10), // laatste 10
      ],
    };
  }
  return {};
}
```

6

Done-tool pattern

Soms wil je dat agent ALTIJD via een tool stopt — bijvoorbeeld omdat je het uiteindelijke antwoord gestructureerd wilt.

```
const tools = {
  webSearch, readPage, saveReport,
  done: tool({
    description: "Roep aan wanneer onderzoek klaar is en rapport is opgeslagen.",
    inputSchema: z.object({
      finalReportId: z.number(),
      summary: z.string(),
    }),
    // GEEN execute - signaleert: stop de loop
  }),
};

const agent = new ToolLoopAgent({
  model: "openai/gpt-4o",
  tools,
  toolChoice: "required", // model MOET altijd tool aanroepen
});

const result = await agent.generate({ prompt });
const doneCall = result.staticToolCalls[0];
if (doneCall?.toolName === "done") {
  console.log(doneCall.input.summary);
}
```

Wanneer dit pattern gebruiken

- Je wilt een gestructureerd eindantwoord (geen vrije text)
- Voorkom dat agent tussendoor stopt zonder iets op te slaan
- Combineren met toolChoice: "required" om text-generatie uit te sluiten

7 Planning + reflectie patronen

Plan-Act-Reflect (ReAct)

Een populair pattern uit de literatuur:

1. **Plan** — agent schrijft eerst een plan in text
2. **Act** — agent voert plan uit via tools
3. **Reflect** — agent kijkt terug, vat samen, evalueert

Implementatie: in de system prompt expliciet om deze fasen vragen. Het model doet dit dan vanzelf binnen de loop — geen aparte code nodig.

Sub-agents

Een agent kan een andere agent aanroepen als tool:

```
const summarizer = new ToolLoopAgent({
  model: "openai/gpt-4o-mini",
  system: "Vat &eacute;&eacute;n webpagina samen in 5 bullets.",
  tools: { readPage },
  stopWhen: stepCountIs(3),
});

const summarizePageTool = tool({
  description: "Vat een URL samen in bullets",
  inputSchema: z.object({ url: z.string().url() }),
  execute: async ({ url }) => {
    const r = await summarizer.generate({
      prompt: `Vat samen: ${url}`,
    });
    return { summary: r.text };
  },
});
```

Voordelen sub-agents

- **Token-budget** — sub-agent ziet kleinere context
- **Specialisatie** — sub-agent kan kleiner/goedkoper model gebruiken
- **Modulair** — sub-agent te testen los van hoofdagent

8

Agent vs tool-call vs workflow

Niveau	Wanneer	Voorbeeld
Plain tool-call	1 tool, 1 antwoord	Weer in Utrecht
Multi-step (Les 12)	2-5 tools, 1 doel	Jazz vs rock counts
Agent (Les 13)	10-50 stappen, open-ended	Research rapport
Workflow (code)	Reproduceerbaar, audit	Order intake

Wanneer GEEN agent

- **Determinisme nodig** — finance, juridisch, medische data
- **Latency** — agent gebruikt al snel 30s+
- **Voorspelbare kosten** — agent met 50 stappen kan duur uitlopen
- **Eenvoudige flow** — een if-statement is genoeg

Wanneer WEL

- **Open-ended** — research, planning, debugging
- **Volgorde onbekend** — model bepaalt zelf
- **Lange ketens** — meerdere bronnen, meerdere stappen
- **Async use cases** — backend jobs, niet realtime

Quote AI SDK docs

Agents are flexible and powerful, but non-deterministic. When you need reliable, repeatable outcomes with explicit control flow, use core functions with structured workflow patterns.

Kosten bewaken

Één onbedoelde loop kan tientallen euro's kosten. Altijd:

- `stepCountIs(N)` als safety cap
- Custom `StopCondition` op tokens of dollar-amount
- Logging per step (welke tool, hoeveel tokens)

Latency

Een 20-step agent op gpt-4o duurt 30-60 seconden. Voor user-facing:

- Stream steps naar client (`agent.stream()`)
- Toon progress (welke stap, welke tool)
- Loading state met geschatte tijd

Observability

Log per step:

- Step number
- Tool calls + input
- Tool result (truncated)
- Token usage
- Latency

Essentieel voor debugging — agents zijn niet-deterministisch.

Veiligheid

- Write-tools (DB inserts, mails) altijd met explicit user-intent
- Confirmation UI voor irreversible acties
- Read-only sandbox in dev (open RLS policies)
- Production: tighter RLS + write-tools authenticated

Testing

- **Mock LLM** — test je tools los met fake LLM
- **Snapshot tests** — log volledige step-trace, vergelijk met goedgekeurde versie
- **Eval suites** — set van vragen + verwachte tool-sequences

10 Bronnen

- AI SDK Agents Overview — ai-sdk.dev/docs/agents/overview
- AI SDK Loop Control — ai-sdk.dev/docs/agents/loop-control
- AI SDK Building Agents — ai-sdk.dev/docs/agents/building-agents
- AI SDK Subagents — ai-sdk.dev/docs/agents/subagents
- AI SDK Workflow patterns — ai-sdk.dev/docs/agents/workflows
- Tavily Web Search API — tavily.com
- Anthropic — Building effective agents
- ReAct paper — arxiv.org/abs/2210.03629