

CRAFTING INTELLIGENT AGENTS WITH CONTEXT ENGINEERING

CARLY RICHMOND



ABOUT ME

- Developer Advocate Lead @
 elastic
- Frontend Engineer, Speaker & Blogger



SCAN ME



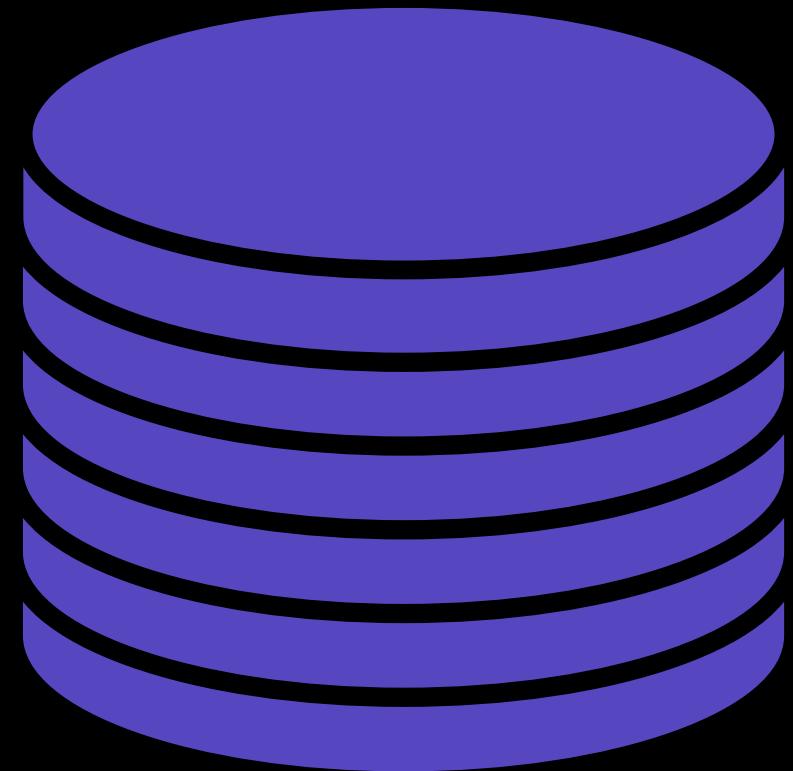
CONTEXT WINDOW

The maximum number of tokens an LLM can process at once.

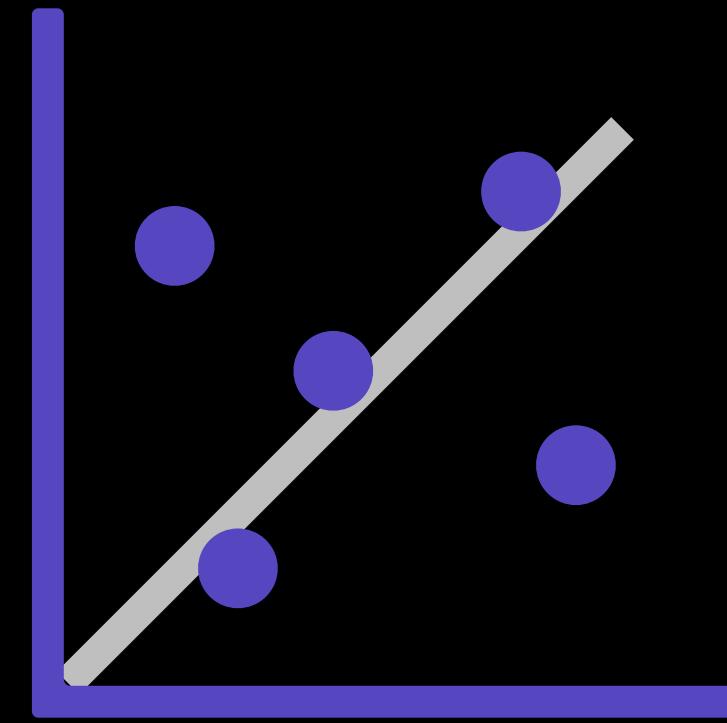




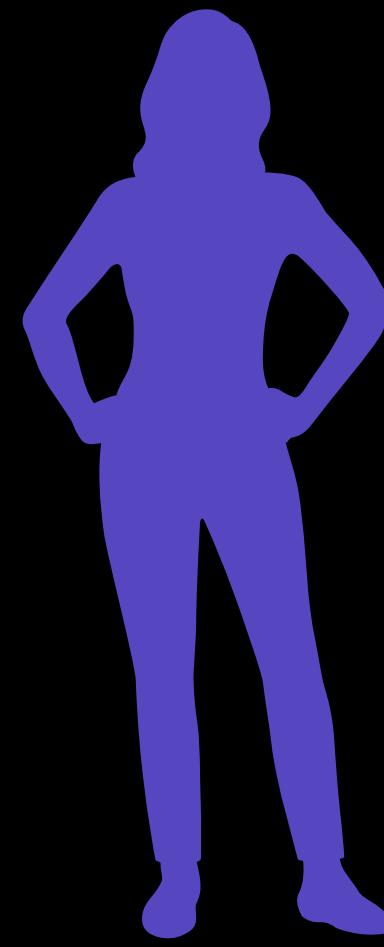
WHY DO LLMS HALLUCINATE?



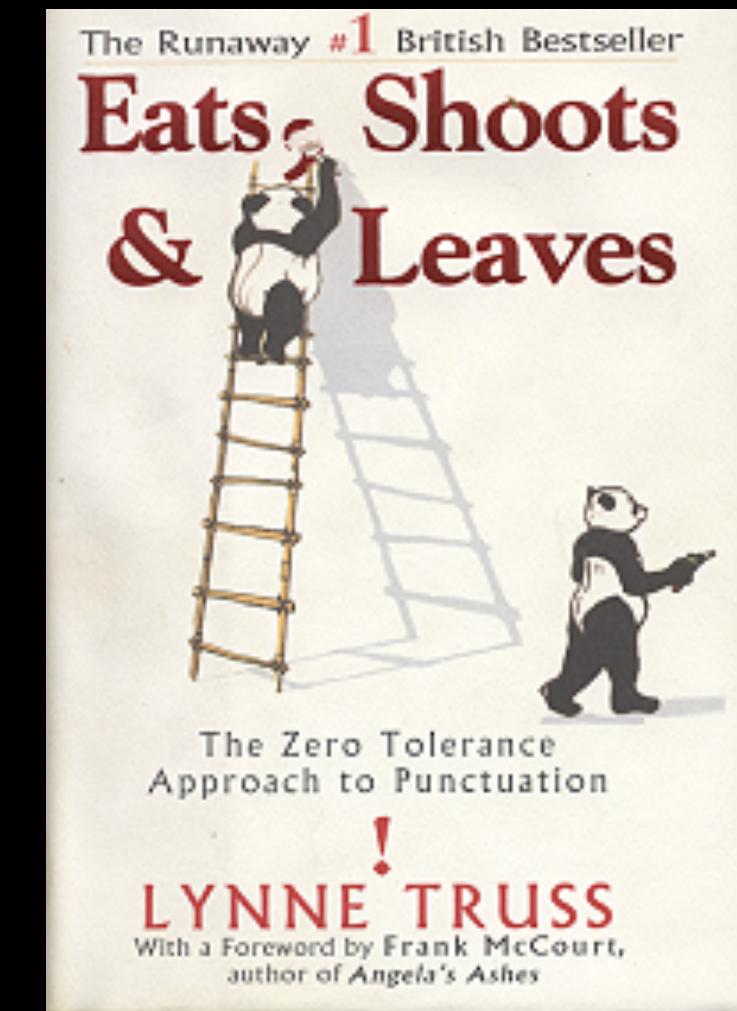
Frozen or Limited
Knowledge



Overfitting



Biases



Language
Ambiguity

Why Language Models Hallucinate

Adam Tauman Kalai*
OpenAI

Ofir Nachum
OpenAI

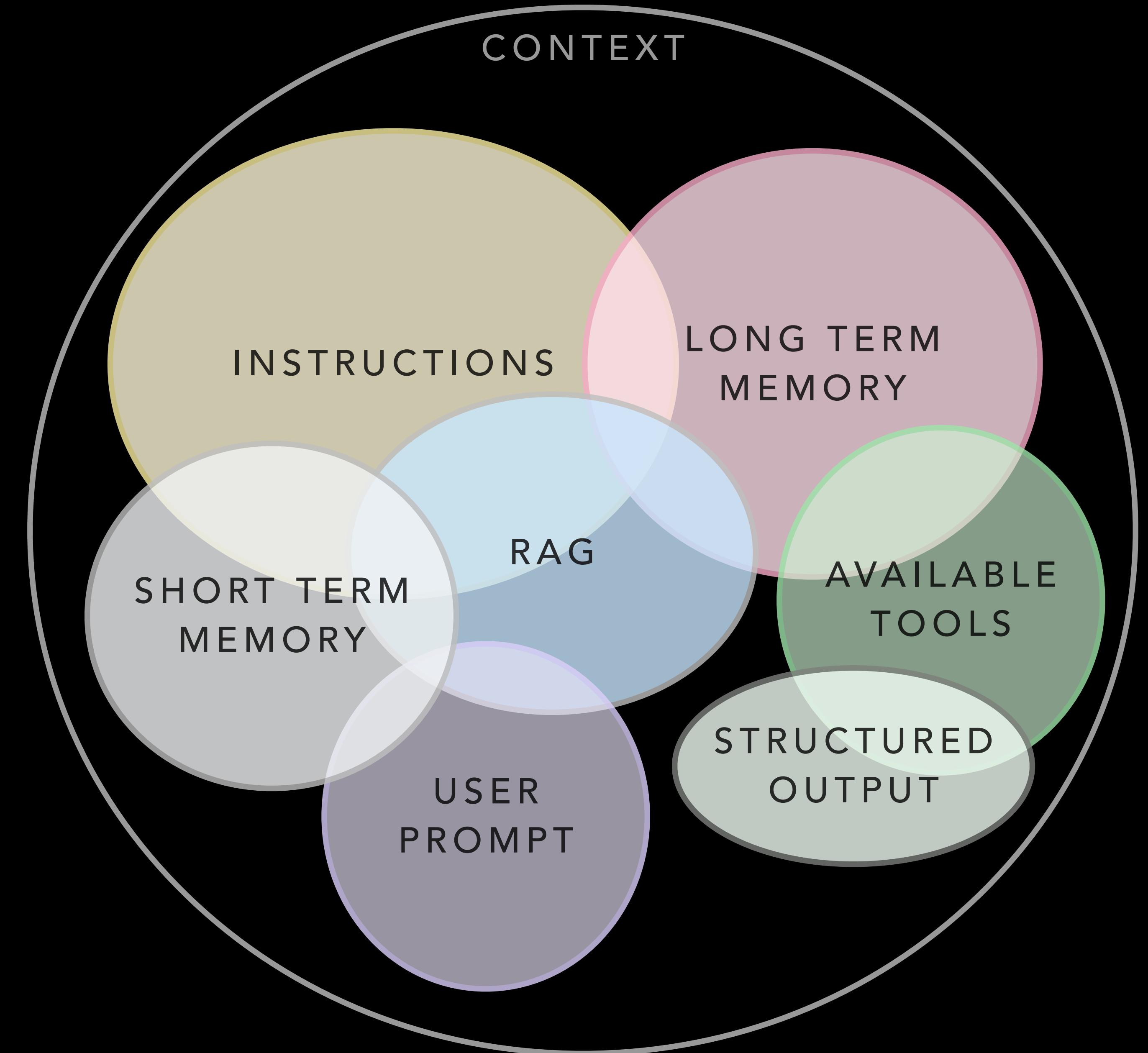
Santosh S. Vempala†
Georgia Tech

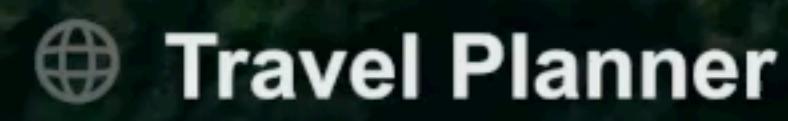
Edwin Zhang
OpenAI

September 4, 2025

Abstract

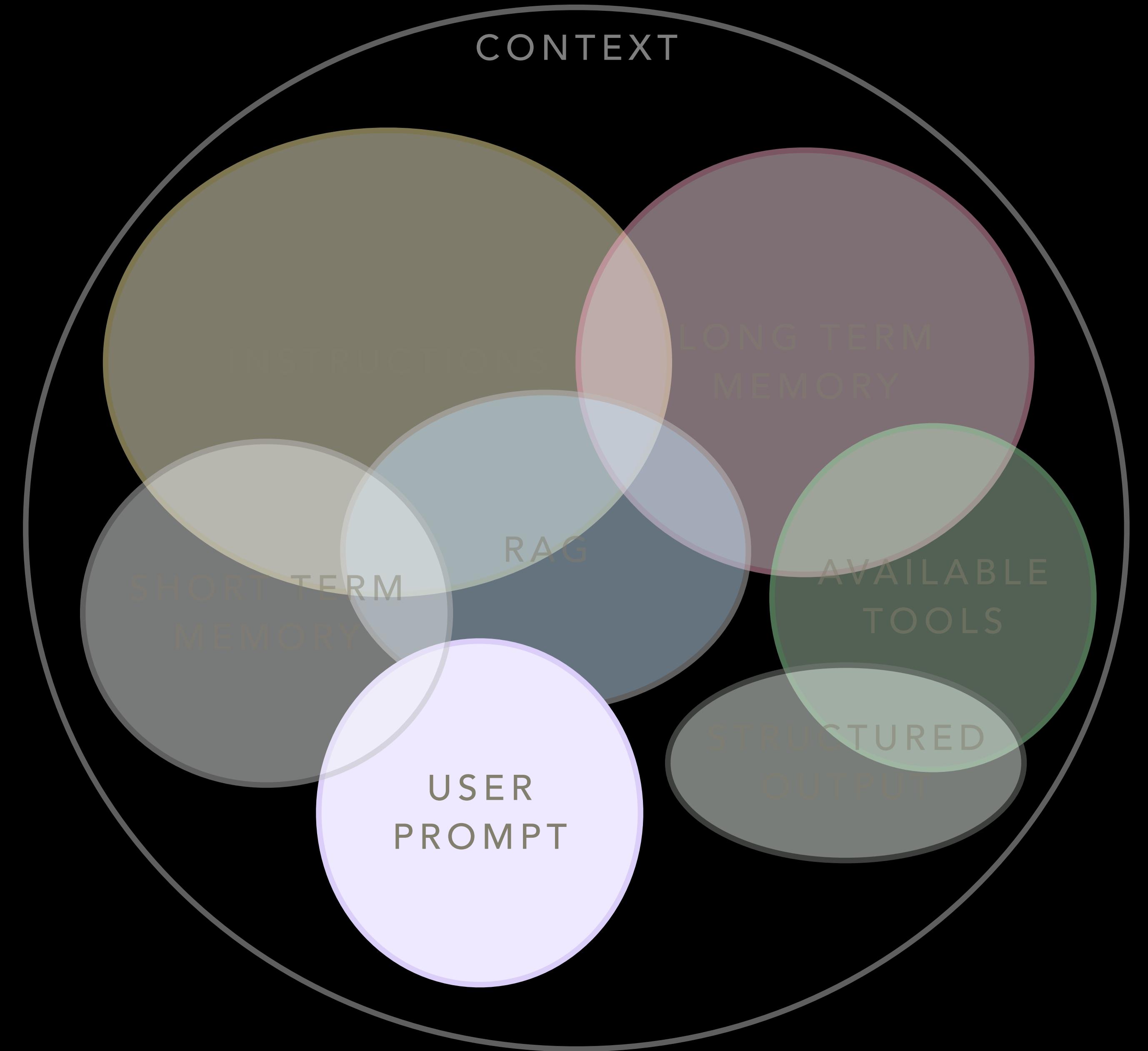
Like students facing hard exam questions, large language models sometimes guess when uncertain, producing plausible yet incorrect statements instead of admitting uncertainty. Such “hallucinations” persist even in state-of-the-art systems and undermine trust. We argue that language models hallucinate because the training and evaluation procedures reward guessing over acknowledging uncertainty and we analyze the statistical causes of hallucinations in the modern training pipeline. Hallucinations need not be mysterious—they originate simply as errors in binary classification. If incorrect statements cannot be distinguished from facts, then hallucinations in pretrained language models will arise through natural statistical pressures. We then argue that hallucinations persist due to the way most evaluations are graded—language models are optimized to be good test-takers, and guessing when uncertain improves test performance. This “epidemic” of penalizing uncertain responses can only be addressed through a socio-technical mitigation: modifying the scoring of existing benchmarks that are misaligned but dominate leaderboards, rather than introducing additional hallucination evaluations. This change may steer the field toward more trustworthy AI systems.





Where would you like to go?







What's on your mind today?

+ Ask anything

Q



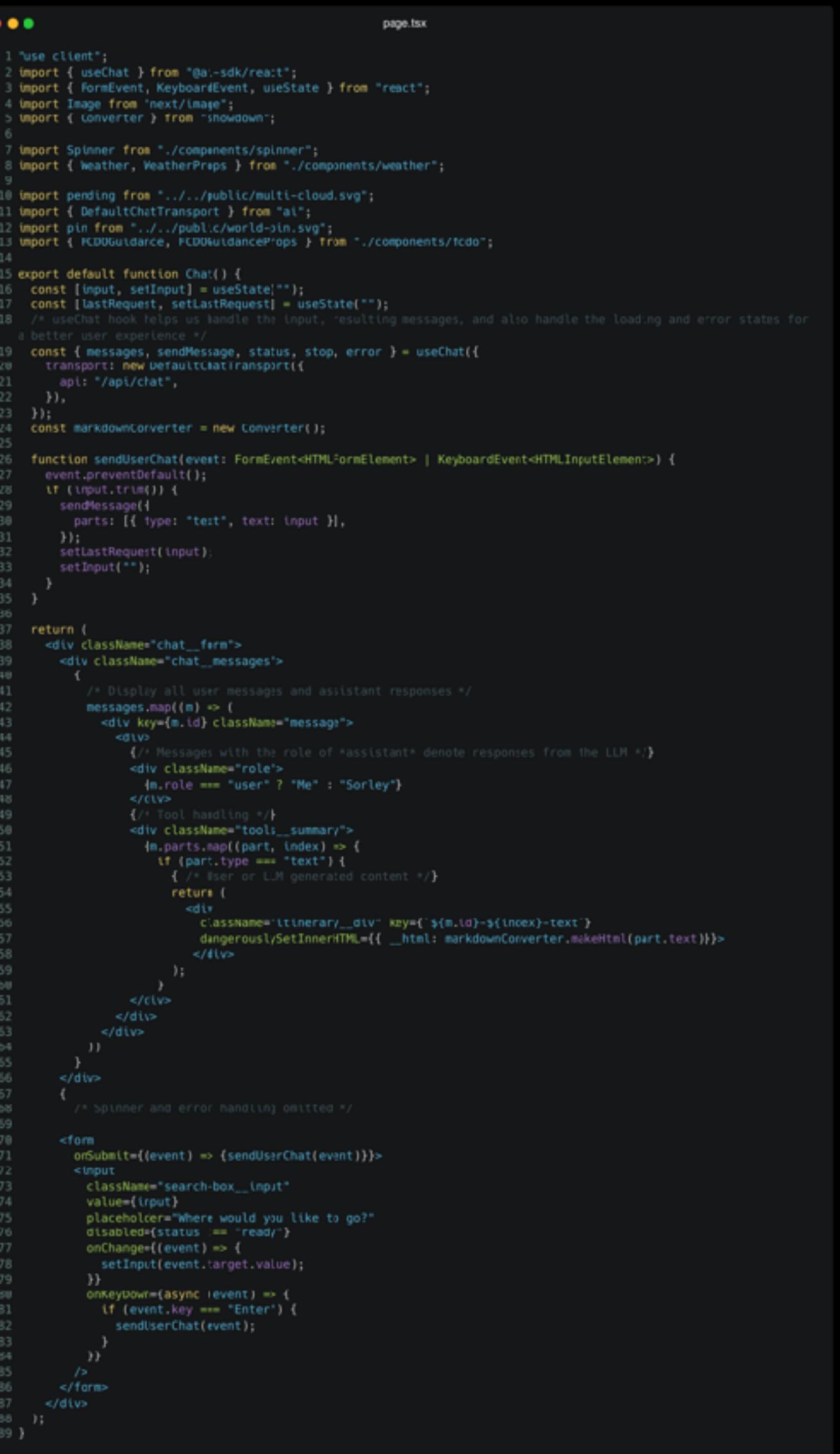
IMAGES

Prompt: red e30 BMW M3 at sunset, photography, real, realistic, 8k, bbs, snow, mountains, photorealistic, ultra detailed, --v 4



 Midjourney
Midjourney

PASSING THE USER PROMPT



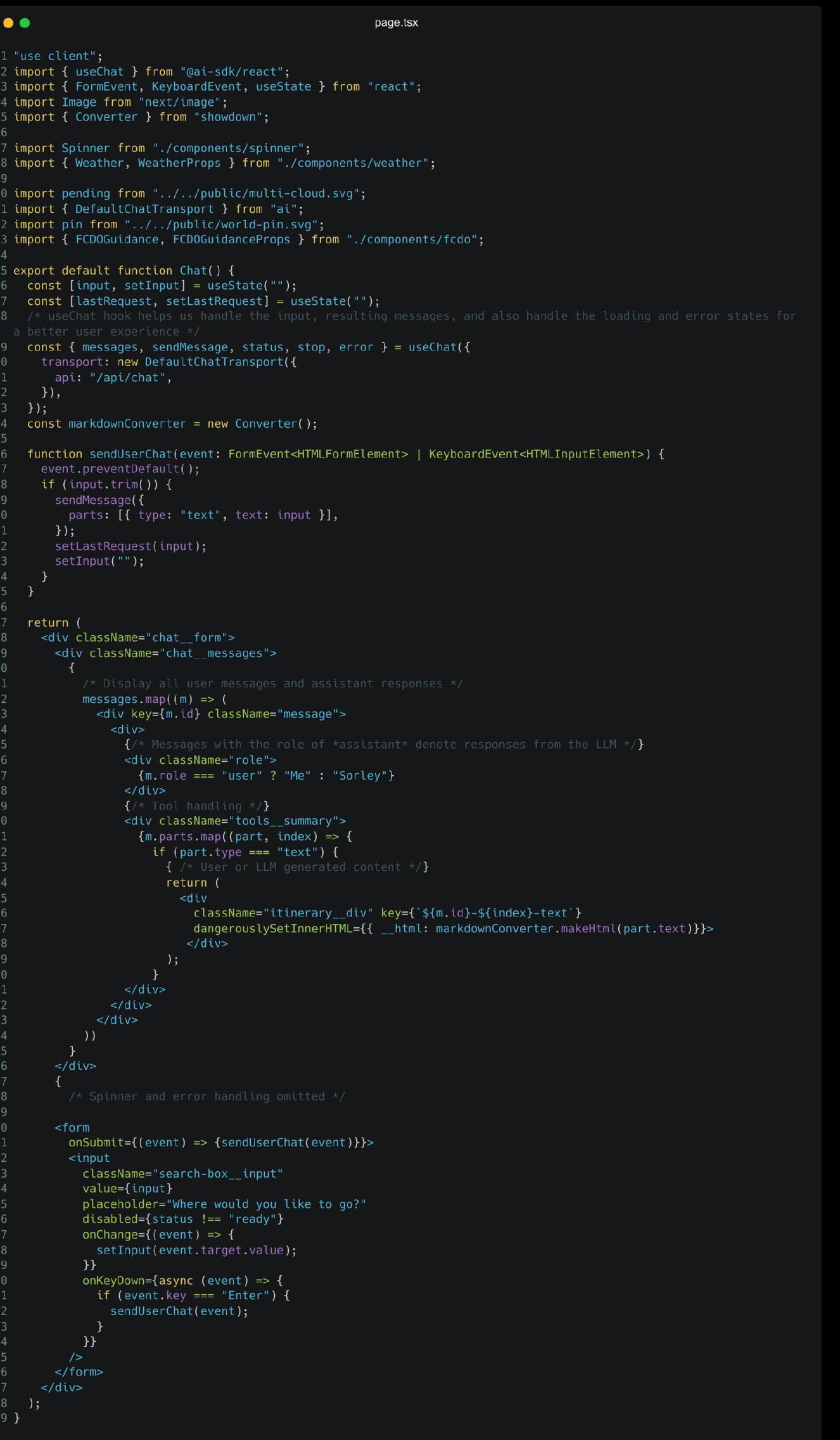
```
page.tsx

1 "use client";
2 import { useChat } from "@elastic/ai-sdk/react";
3 import { FormEvent, KeyboardEvent, useState } from "react";
4 import Image from "next/image";
5 import { Converter } from "showdown";
6
7 import Spinner from "./components/spinner";
8 import { weather, WeatherProps } from "./components/weather";
9
10 import pending from "../../../../public/multi-cloud.svg";
11 import { DefaultChatTransport } from "ai";
12 import pin from "../../../../public/world-pin.svg";
13 import { FCDODGuardce, FCDODGuardceProps } from "./components/fcdod";
14
15 export default function Chat() {
16   const [input, setInput] = useState("");
17   const [lastRequest, setLastRequest] = useState("");
18   /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for
a better user experience */
19   const { messages, sendMessage, status, stop, error } = useChat({
20     transport: new DefaultChatTransport({
21       api: "/api/chat",
22     }),
23   });
24   const markdownConverter = new Converter();
25
26   function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
27     event.preventDefault();
28     if (input.trim()) {
29       sendMessage({
30         parts: [{ type: "text", text: input }],
31       });
32       setLastRequest(input);
33       setInput("");
34     }
35   }
36
37   return (
38     <div className="chat__form">
39       <div className="chat__messages">
40         {
41           /* Display all user messages and assistant responses */
42           messages.map((m) => (
43             <div key={m.id} className="message">
44               <div>
45                 {/* Messages with the role of "assistant" denote responses from the LLM */}
46                 <div className="role">
47                   {m.role === "user" ? "Me" : "Sorley"}
48                 </div>
49                 {/* Tool handling */}
50                 <div className="tools__summary">
51                   {m.parts.map((part, index) => {
52                     if (part.type === "text") {
53                       /* User or LLM generated content */
54                       return (
55                         <div
56                           className="itinerary__div" key={`${m.id}-${index}-text`}
57                           dangerouslySetInnerHTML={{ __html: markdownConverter.makeHtml(part.text) }}>
58                           </div>
59                         );
60                       }
61                     </div>
62                   )}
63                 </div>
64               )
65             </div>
66           )
67           /*
             spinner and error handling omitted */
68
69         <form
70           onSubmit={(event) => {sendUserChat(event)}}
71           >
72             <input
73               className="search-box__input"
74               value={input}
75               placeholder="Where would you like to go?"
76               disabled={status === "read"}
77               onChange={(event) => {
78                 setInput(event.target.value);
79               }}
80               onKeyPress={(async event) => {
81                 if (event.key === "Enter") {
82                   sendMessage(event);
83                 }
84               }}
85             />
86           </form>
87         </div>
88       );
89     );
90   }
91 }
```

```
14
15 export default function Chat() {
16   const [input, setInput] = useState("");
17   const [lastRequest, setLastRequest] = useState("");
18   /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for
19    a better user experience */
20   const { messages, sendMessage, status, stop, error } = useChat({
21     transport: new DefaultChatTransport({
22       api: "/api/chat",
23     }),
24   });
25   const markdownConverter = new Converter();
26
27   function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
28     event.preventDefault();
29     if (input.trim()) {
30       sendMessage({
31         parts: [{ type: "text", text: input }],
32       });
33       setLastRequest(input);
34       setInput("");
35     }
36
37   return (
38     <div className="chat__form">
39       <div className="chat__messages">
40         {
41           /* Display all user messages and assistant responses */
42           messages.map((m) => (
43             <div key={m.id} className="message">
44               <div>
```

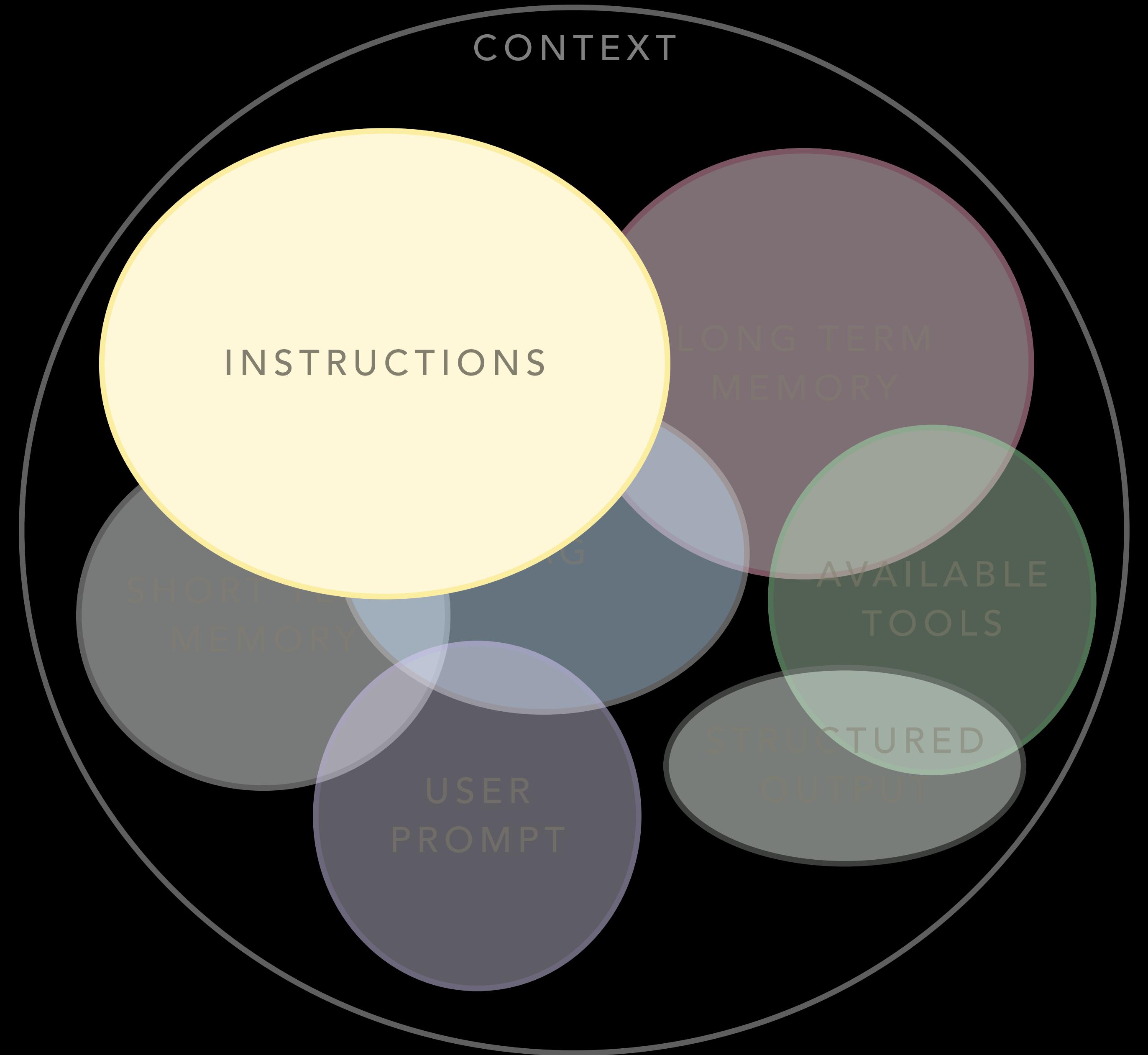
```
60
61      </div>
62      </div>
63      </div>
64    ))
65  }
66 </div>
67 {
68  /* Spinner and error handling omitted */
69
70 <form
71   onSubmit={(event) => {sendUserChat(event)}}
72   <input
73     className="search-box__input"
74     value={input}
75     placeholder="Where would you like to go?"
76     disabled={status !== "ready"}
77     onChange={(event) => {
78       setInput(event.target.value);
79     }}
80     onKeyDown={async (event) => {
81       if (event.key === "Enter") {
82         sendUserChat(event);
83       }
84     }}
85   />
86   </form>
87 </div>
88 );
89 }
```

PASSING THE USER PROMPT



```
page.jsx

1 "use client";
2 import { useChat } from "@ai-sdk/react";
3 import { FormEvent, KeyboardEvent, useState } from "react";
4 import Image from "next/image";
5 import { Converter } from "showdown";
6
7 import Spinner from "./components/spinner";
8 import { Weather, WeatherProps } from "./components/weather";
9
10 import pending from "../../public/multi-cloud.svg";
11 import { DefaultChatTransport } from "ai";
12 import pin from "../../public/world-pin.svg";
13 import { FCDOGuidance, FCDOGuidanceProps } from "./components/fcdog";
14
15 export default function Chat() {
16   const [input, setInput] = useState("");
17   const [lastRequest, setLastRequest] = useState("");
18   /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for
   a better user experience */
19   const { messages, sendMessage, status, stop, error } = useChat({
20     transport: new DefaultChatTransport({
21       api: "/api/chat",
22     }),
23   });
24   const markdownConverter = new Converter();
25
26   function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
27     event.preventDefault();
28     if (input.trim()) {
29       sendMessage({
30         parts: [{ type: "text", text: input }],
31       });
32       setLastRequest(input);
33       setInput("");
34     }
35   }
36
37   return (
38     <div className="chat_form">
39       <div className="chat_messages">
40         {
41           /* Display all user messages and assistant responses */
42           messages.map((m) => (
43             <div key={m.id} className="message">
44               {
45                 /* Messages with the role of *assistant* denote responses from the LLM */
46                 <div className="role">
47                   {m.role === "user" ? "Me" : "Sorley"}
48                 </div>
49                 /* Tool handling */
50                 <div className="tools_summary">
51                   {m.parts.map((part, index) => {
52                     if (part.type === "text") {
53                       /* User or LLM generated content */
54                       return (
55                         <div
56                           className="itinerary_div" key={`${m.id}-${index}-text`}
57                           dangerouslySetInnerHTML={{ __html: markdownConverter.makeHtml(part.text) }}>
58                           </div>
59                         );
60                       }
61                     </div>
62                   )}
63                 </div>
64               )
65             </div>
66           )
67         }
68         /* Spinner and error handling omitted */
69       </div>
70       <form
71         onSubmit={({event}) => {sendUserChat(event)}}
72         ><input
73           className="search_box__input"
74           value={input}
75           placeholder="Where would you like to go?"
76           disabled={status !== "ready"}
77           onChange={({event}) => {
78             setInput(event.target.value);
79           }}
80           onKeyDown={async (event) => {
81             if (event.key === "Enter") {
82               sendUserChat(event);
83             }
84           }}
85         />
86       </form>
87     </div>
88   );
89 }
```



WHAT IS A PROMPT?

Role

"You are a helpful travel agent that returns travel itineraries based on location, the FCDO guidance from the specified tool, the weather captured from the displayWeather tool, and the flight information from tool getFlights."

Directive

"Return a day by day itinerary of sites to see and things to do based on the weather as a textual blurb."

Exemplars

"For example, if you are generating an itinerary for a rainy day in Glasgow, consider recommending indoor activities such as museums over local parks such as Glasgow Green."

Style Instructions

"Also provide the outbound and inbound flight recommendations if available. Present any flight information in the format:

'\${FLIGHT NUMBER}: \${ORIGIN} to
\${DESTINATION} on \${DATE} at \${TIME}, duration
\${HOURS}'"

Additional Information

"If the FCDO tool warns against travel DO NOT generate an itinerary, and simply say 'We do not recommend travelling to this location'."

Output
Formatting

EXAMPLE SYSTEM PROMPT

```
route.ts

1 import { openai } from '@ai-sdk/openai';
2 import { streamText, stepCountIs, convertToModelMessages, UIMessage } from 'ai';
3 import { NextResponse } from 'next/server';
4
5 import { weatherTool } from '@/app/ai/weather.tool';
6 import { fcdoTool } from '@/app/ai/fcdo.tool';
7 import { flightTool } from '@/app/ai/flights.tool';
8
9 // Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
10 export const maxDuration = 30;
11
12 const tools = {
13   flights: flightTool,
14   weather: weatherTool,
15   fcdo: fcdoTool
16 };
17
18 // Post request handler
19 export async function POST(req: Request) {
20   const { messages }: { messages: UIMessage[] } = await req.json();
21   const convertedMessages = convertToModelMessages(messages);
22
23   try {
24     const result = streamText({
25       model: openai('gpt-4o'),
26       system:
27         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
28         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
29         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
30         "Return an itinerary of sites to see and things to do based on the weather." +
31         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
32       messages: convertedMessages,
33       stopWhen: stepCountIs(2),
34       tools
35     });
36
37   // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
38   // better user experience
39   return result.toUIMessageStreamResponse();
40 } catch (e) {
41   console.error(e);
42   return new NextResponse("Unable to generate a plan. Please try again later!");
43 }
44 }
```

```
12 const tools = {
13   flights: flightTool,
14   weather: weatherTool,
15   fcdo: fcdoTool
16 };
17
18 // Post request handler
19 export async function POST(req: Request) {
20   const { messages }: { messages: UIMessage[] } = await req.json();
21   const convertedMessages = convertToModelMessages(messages);
22
23   try {
24     const result = streamText({
25       model: openai('gpt-4o'),
26       system:
27         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
28         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
29         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
30         "Return an itinerary of sites to see and things to do based on the weather." +
31         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
32       messages: convertedMessages,
33       stopWhen: stepCountIs(2),
34       tools
35     });
36
37   // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
38   // better user experience
39   return result.toUIMessageStreamResponse();
40 } catch (e) {
41   console.error(e);
```

EXAMPLE SYSTEM PROMPT



route.ts

```
1 import { openai } from '@ai-sdk/openai';
2 import { streamText, stepCountIs, convertToModelMessages, UIMessage } from 'ai';
3 import { NextResponse } from 'next/server';
4
5 import { weatherTool } from '@/app/ai/weather.tool';
6 import { fcdoTool } from '@/app/ai/fcdo.tool';
7 import { flightTool } from '@/app/ai/flights.tool';
8
9 // Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
10 export const maxDuration = 30;
11
12 const tools = {
13   flights: flightTool,
14   weather: weatherTool,
15   fcdo: fcdoTool
16 };
17
18 // Post request handler
19 export async function POST(req: Request) {
20   const { messages }: { messages: UIMessage[] } = await req.json();
21   const convertedMessages = convertToModelMessages(messages);
22
23   try {
24     const result = streamText({
25       model: openai('gpt-4o'),
26       system:
27         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
28         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
29         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
30         "Return an itinerary of sites to see and things to do based on the weather." +
31         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
32       messages: convertedMessages,
33       stopWhen: stepCountIs(2),
34       tools
35     });
36
37     // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
38     // better user experience
39     return result.toUIMessageStreamResponse();
40   } catch (e) {
41     console.error(e);
42     return new NextResponse("Unable to generate a plan. Please try again later!");
43   }
44 }
```

EXAMPLE STRATEGIES

- Shot-based prompting
- Chain of Thought
- Meta Prompting

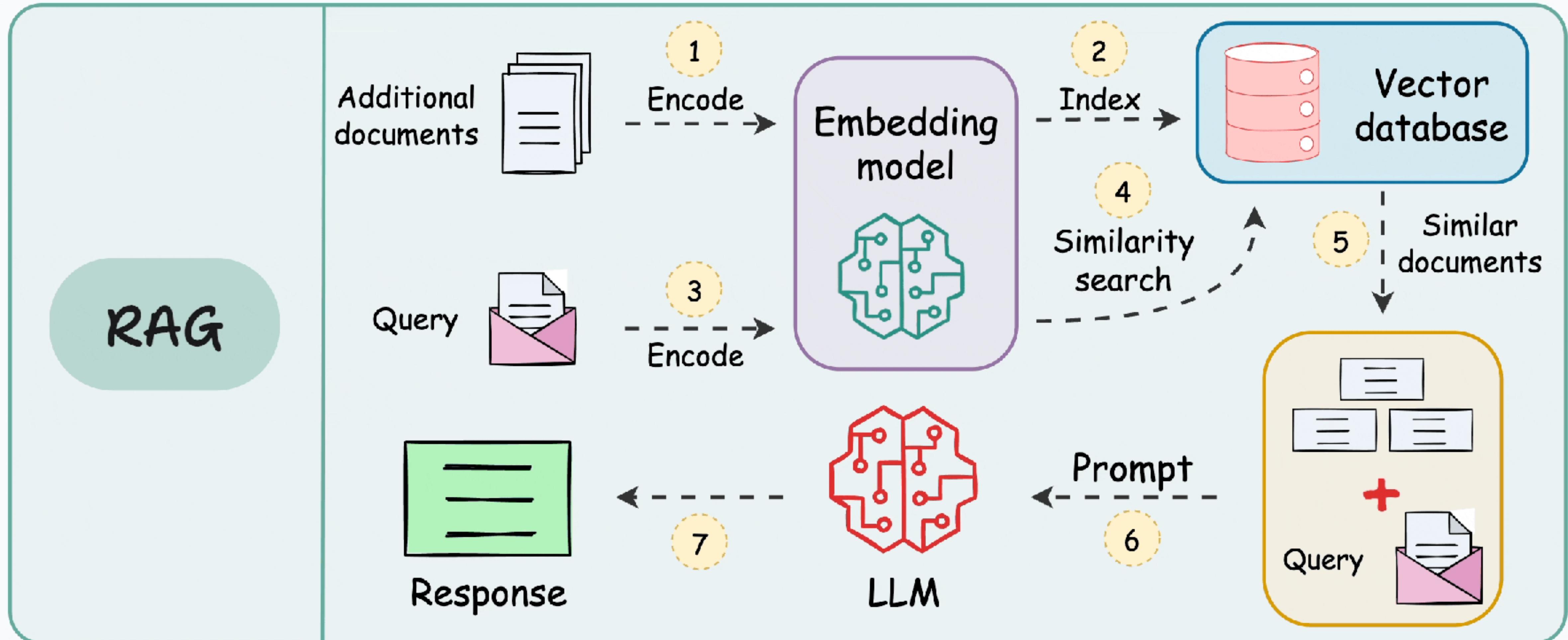
Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. X</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✓</p>

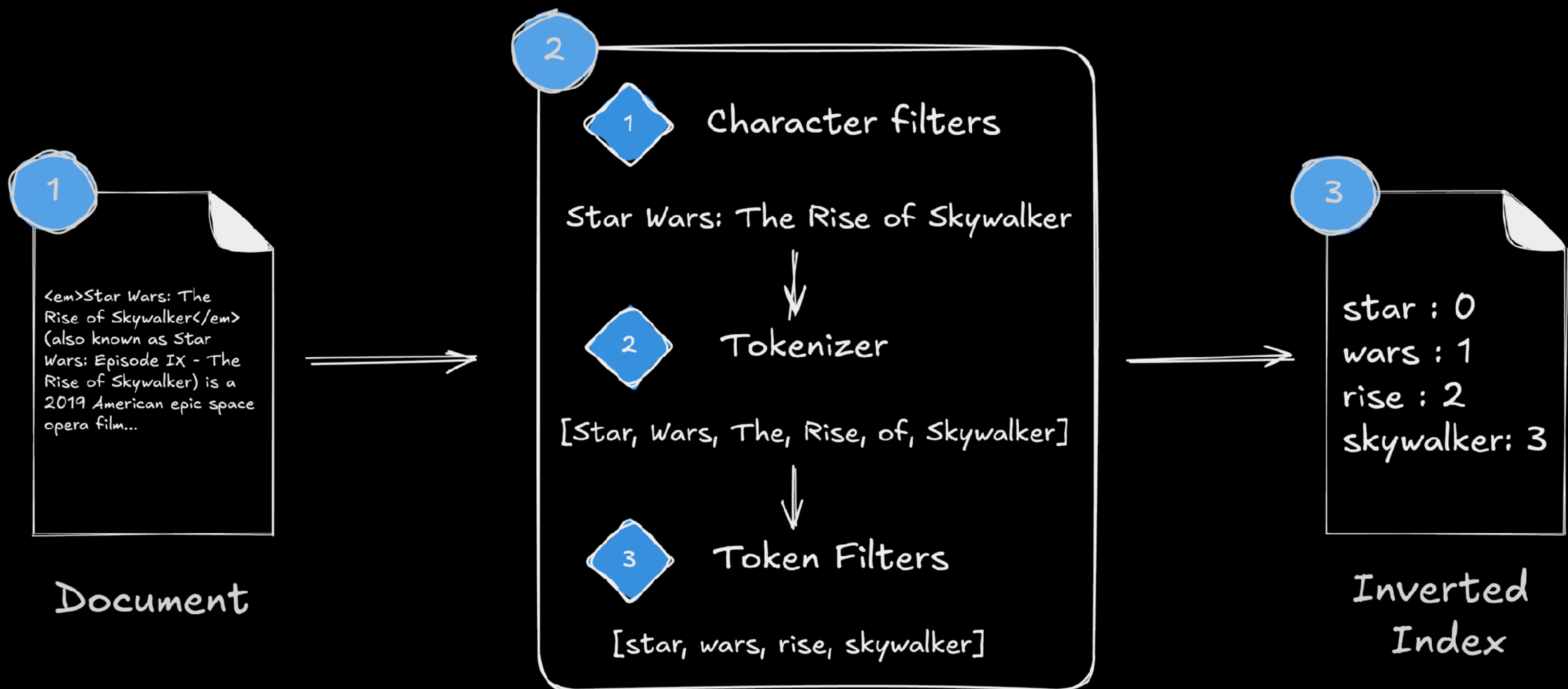
HOW TO BUILD A GOOD PROMPT

- Task type
- Complexity and ambiguity
- Inputs and outputs
- Data format
- Persona to emulate
- LLM itself

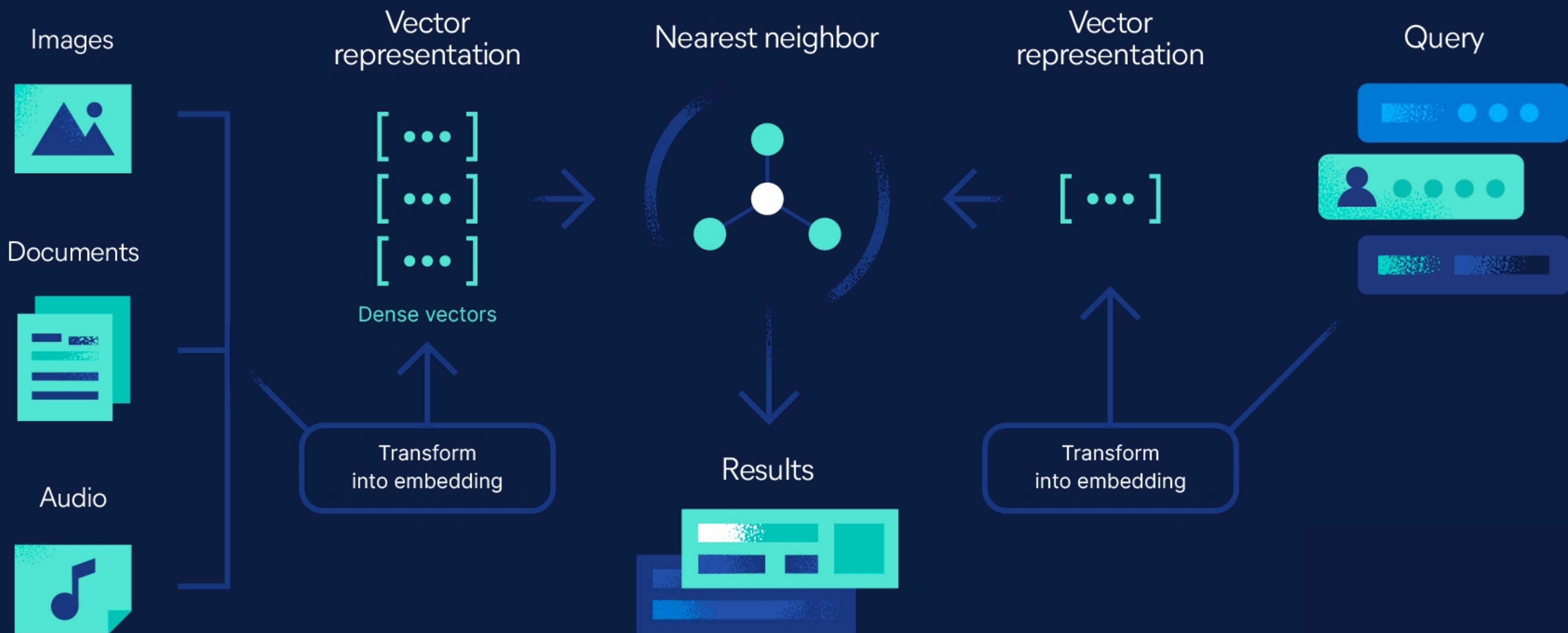
The screenshot shows a web browser window with the title "Prompts - Elasticsearch Labs". The URL in the address bar is "elastic.co/search-labs/prompts". The page features a dark header with the "elastic search labs" logo, a search bar, and navigation links for "Tutorials", "Examples", "Integrations", "Blogs", and a "Start free trial" button. Below the header, there's a decorative graphic of wavy lines. The main content area has a dark background with two prominent cards. The first card, titled "Meeting Summary Assistant", describes its purpose as transforming raw meeting transcripts into structured summaries. It lists key characteristics: "Objective: Transform raw meeting transcripts into structured and actionable summaries to enhance decision-making and team alignment." and "Process-Oriented: Outlines detailed steps for analyzing and summarizing transcripts, including handling transcription errors, extracting insights, and categorizing information." There are "See more" and "See less" buttons. The second card, titled "Asset Inventory Diagrams", describes its purpose as visualizing organization assets and connections. It lists key characteristics: "Easily visualize organization's assets and connections.", "Create a diagram showing our organizational asset inventory that: - Groups assets by type (servers, endpoints, IoT devices, etc.) - Shows ownership and business criticality - Indicates security patch status - Highlights compliance requirements - Maps dependencies between assets. Include these specific assets and relationships: [Insert asset details here] Please use Mermaid diagram syntax and include a legend.", and "See more" and "See less" buttons.



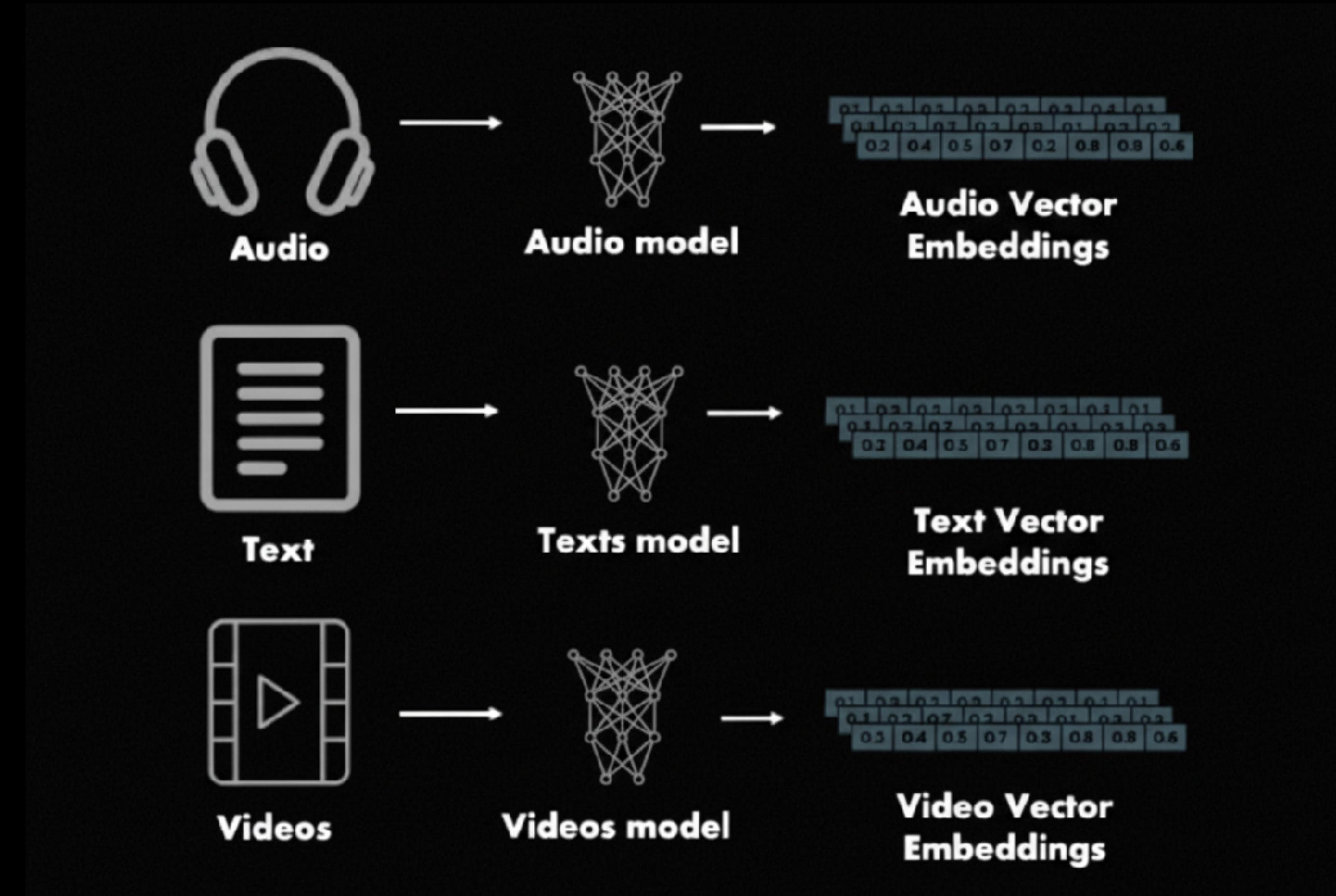




VECTOR SEARCH



WHERE DO EMBEDDINGS COME FROM?



REALISTIC

HUMAN

MACHINE

CARTOON



VECTOR SEARCH RANKS OBJECTS BY SIMILARITY (RELEVANCE) TO THE QUERY



REALISTIC

HUMAN

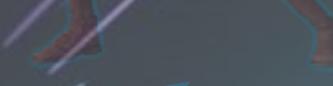
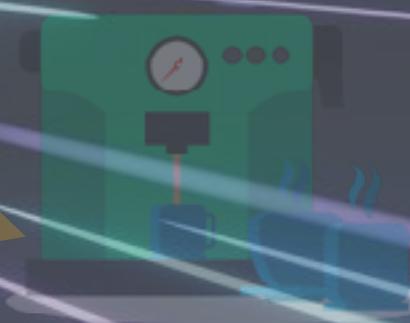
MACHINE

CARTOON

COSINE SIMILARITY

$$\Theta \approx 0$$

$$\cos(\Theta) \approx 1$$



HOW VECTORS ARE INDEXED FOR SEARCH: GRAPHS



Hierarchical Navigable Small Worlds:
a layered approach that simplifies
access to the nearest neighbour



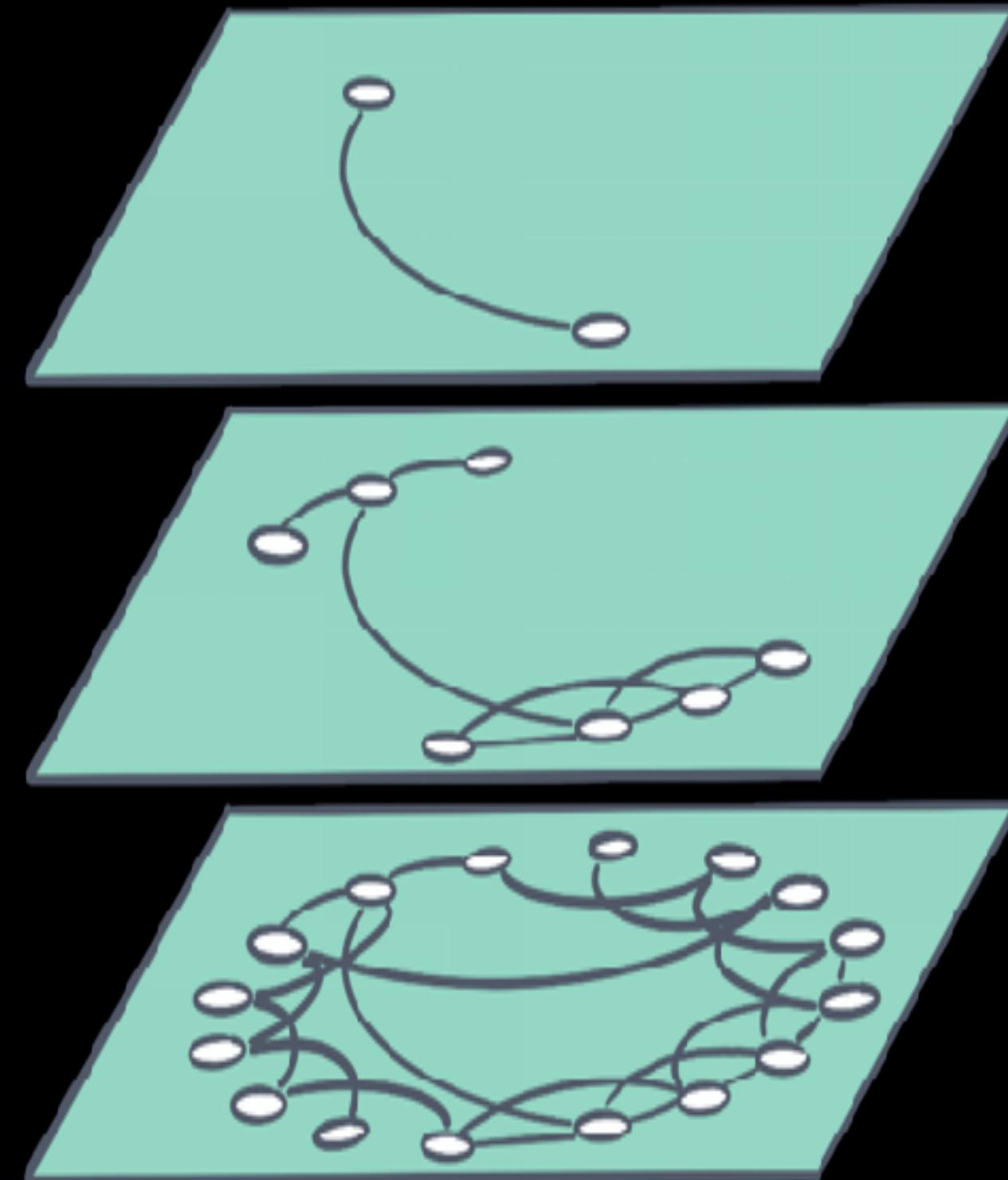
Tiered: from coarse to fine
approximation over a few steps



Balance: Bartering a little accuracy
for a lot of scalability



Speed: Excellent query latency on
large scale indices



VECTOR SEARCH

```
GET vector-search-star-wars-films/_search
{
  "size" : 3,
  "query" : {
    "knn": {
      "field": "text_embedding.predicted_value",
      "k": 10,
      "num_candidates": 100,
      "query_vector": [-0.0078373895958599, -0.215008884668022],
    }
  }
}
```

YES!

Star Wars: The Rise of Skywalker

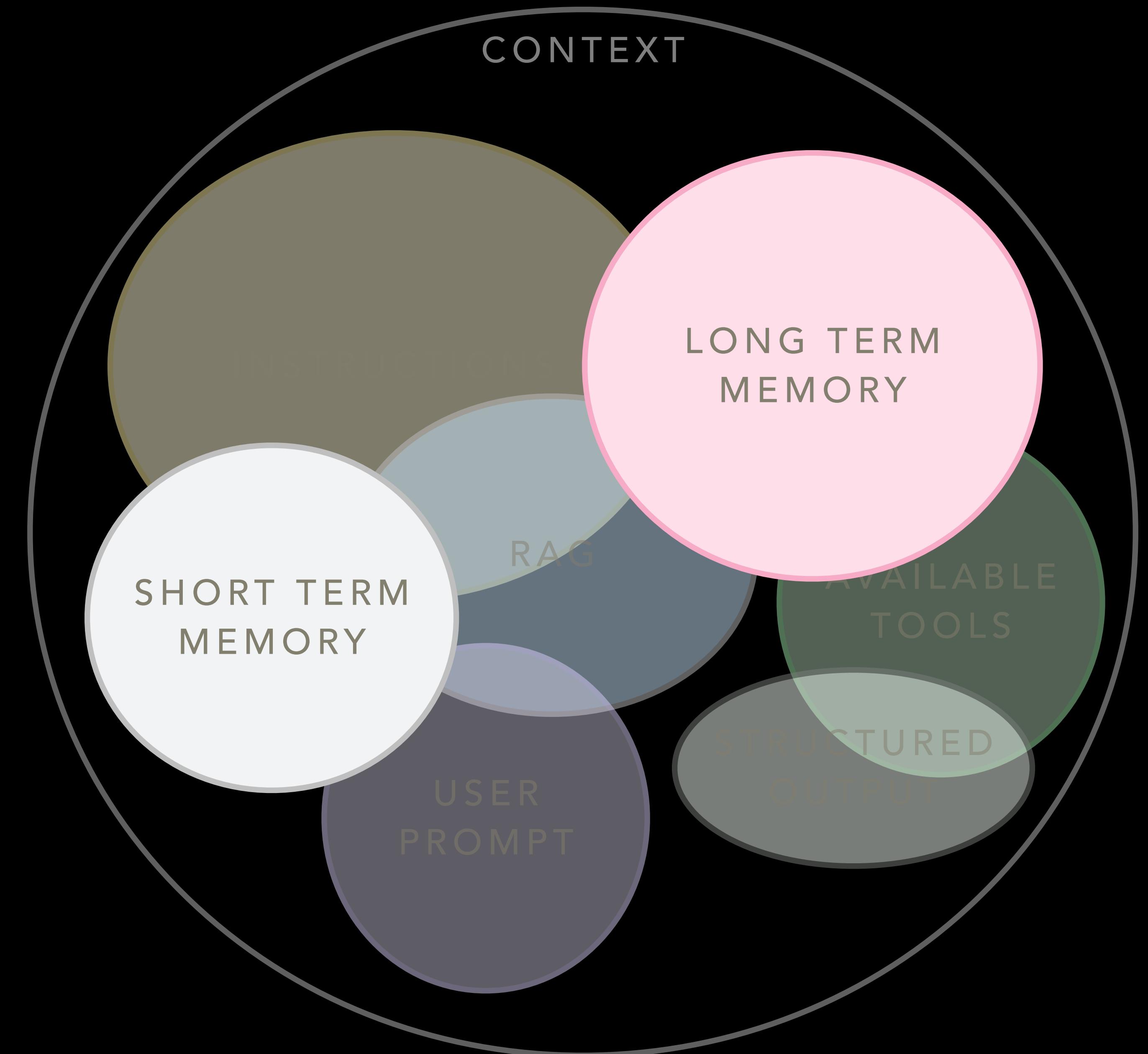
From Wikipedia, the free encyclopedia

Star Wars: The Rise of Skywalker (also known as *Star Wars: Episode IX – The Rise of Skywalker*) is a 2019 American epic space opera film produced, co-written, and directed by J. J. Abrams. Produced by Lucasfilm and distributed by Walt Disney Studios Motion Pictures, it is the third installment of the *Star Wars* sequel trilogy, following *The Force Awakens* (2015) and *The Last Jedi* (2017), and the final episode of the nine-part series. Its cast includes Carrie Fisher, Mark Hamill, Adam Driver, John Boyega, Oscar Isaac, Domhnall Gleeson, Naomi Ackie, Donald Sutherland, E. Grant, Lupita Nyong'o, Thandie Newton, Joonas Suotamo, Kelly Marie Tran, Joonas Suotamo, Keri Russell, and Billy Dee Williams.

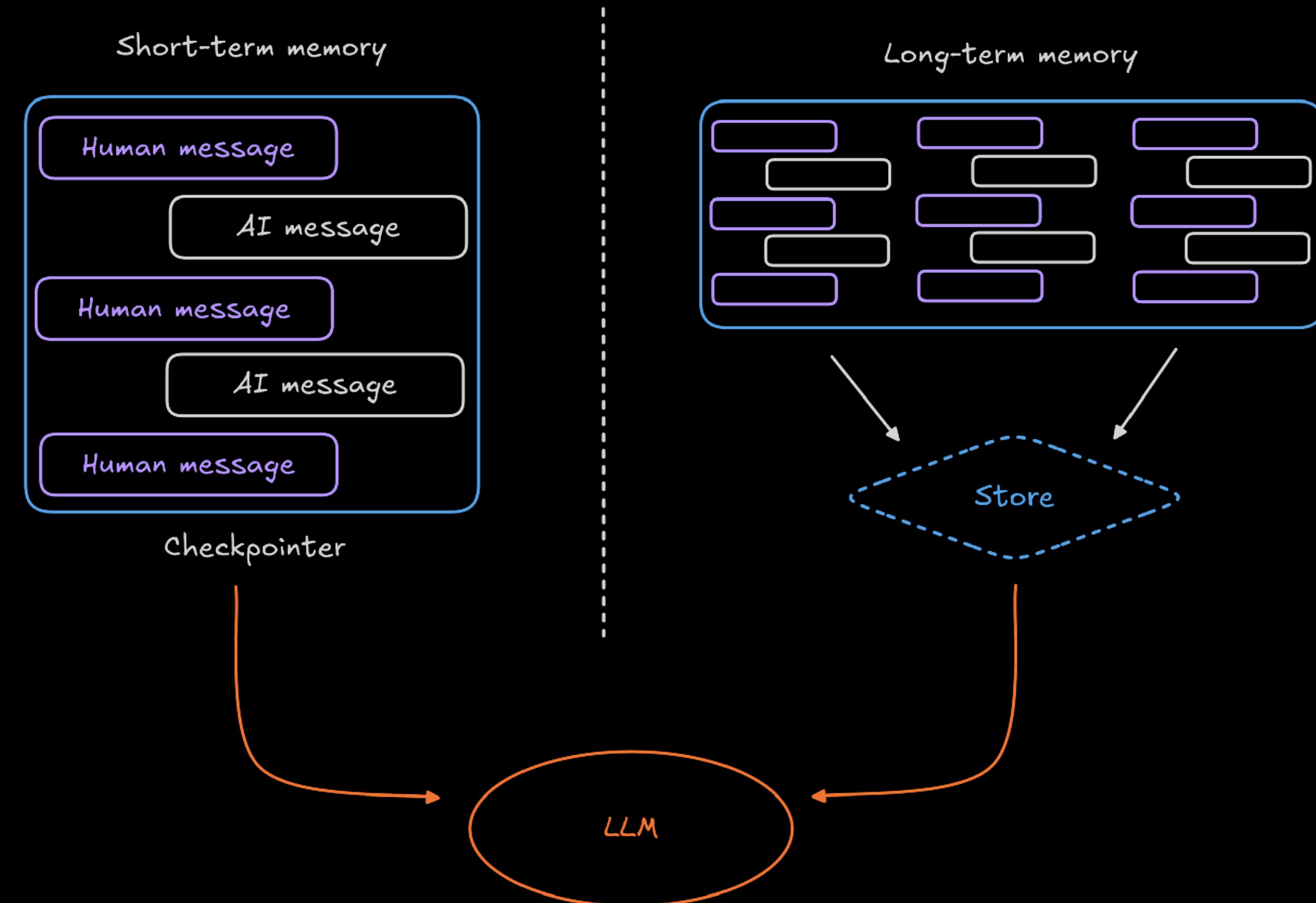
The film follows Rey, Finn, and Poe Dameron as they lead the remnants of the Resistance against the First Order. Kylo Ren and the First Order, who are now led by Emperor Palpatine, Rey's paternal grandfather, are intent on destroying the Resistance.

Following its cancellation, original director Rian Johnson would write the script for *Episode IX*, while Colin Trevorrow was hired to direct and to write a script with his collaborator Michael Waldron; both ultimately retain story credit with Abrams and Chris Terrio. In September 2017, Trevorrow left the project following creative differences with producer Kathleen Kennedy, and Abrams returned as director. John Williams, composer for the previous episodic films, returned to compose the score, making it his final score for the franchise.^[8] Principal photography began in August 2018 at Pinewood Studios in England and wrapped in February 2019, with post-production completed in November 2019. With a budget of \$416 million, it is the fourth most expensive film ever made.

```
{  
  "_index": "vector-search-star-wars-films",  
  "_id": "651447c14fdb8d8600fe12ba",  
  "_score": 0.7053884,  
  "_ignored": ["body_content.keyword"],  
  "_source": {  
    "last_crawled_at": "2023-09-27T15:18:25Z",  
    "text_embedding": {  
      "predicted_value": [-0.007837389595806599, -0.21500888466835022, 0.11815926432609558]  
    },  
    "is_truncated": true,  
    "model_id": "sentence-transformers--msmarco-minilm-l-12-v3"  
  },  
  "additional_urls": [  
    "https://en.wikipedia.org/wiki/Star_Wars:_The_Rise_of_Skywalker"  
  ],  
  "body_content": "Star Wars: The Rise of Skywalker (also known as Star Wars: Episode IX – The Rise of Skywalker) is a 2019 American epic space opera film produced, co-written, and directed by J. J. Abrams.",  
  "domains": ["https://en.wikipedia.org"],  
  "title": "Star Wars: The Rise of Skywalker - Wikipedia",  
  "url": "https://en.wikipedia.org/wiki/Star_Wars:_The_Rise_of_Skywalker"  
}
```



MEMORY



EXAMPLE MEMORY

```
routes

1 import { openai } from "@ai-sdk/openai";
2 import { streamText, stepCountIs, convertToModelMessages } from "ai";
3 import { NextResponse } from "next/server";
4
5 import { weatherTool } from "@/app/ai/weather.tool";
6 import { fcdoTool } from "@/app/ai/fcdo.tool";
7 import { flightTool } from "@/app/ai/flights.tool";
8
9 import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";
10
11 // Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
12 export const maxDuration = 30;
13
14 const tools = {
15   flights: flightTool,
16   weather: weatherTool,
17   fcdo: fcdoTool,
18 };
19
20 // Post request handler
21 export async function POST(req: Request) {
22   const { messages, id } = await req.json();
23
24   // Store current message
25   const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
26   await persistMessage(messages[lastMessageIndex], id);
27
28   // Get chat history by chat id
29   const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
30   const allMessages = [...previousMessages, ...messages];
31
32   try {
33     const convertedMessages = convertToModelMessages(allMessages);
34     const result = streamText({
35       model: openai("gpt-4o"),
36       system:
37         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
38         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
39         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
40         "Return an itinerary of sites to see and things to do based on the weather." +
41         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
42       messages: convertedMessages,
43       stopWhen: stepCountIs(2),
44       tools,
45     });
46
47   // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
48   return result.toUIMessageStreamResponse();
49 } catch (e) {
50   console.error(e);
51   return new NextResponse(
52     "Unable to generate a plan. Please try again later!"
53   );
54 }
55 }
```



```
1 import { openai } from "@ai-sdk/openai";
2 import { streamText, stepCountIs, convertToModelMessages } from "ai";
3 import { NextResponse } from "next/server";
4
5 import { weatherTool } from "@/app/ai/weather.tool";
6 import { fcdoTool } from "@/app/ai/fcdo.tool";
7 import { flightTool } from "@/app/ai/flights.tool";
8
9 import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";
10
11 // Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
12 export const maxDuration = 30;
13
14 const tools = {
15   flights: flightTool,
16   weather: weatherTool,
17   fcdo: fcdoTool,
18 };
19
20 // Post request handler
21 export async function POST(req: Request) {
22   const { messages, id } = await req.json();
23
24   // Store current message
25   const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
26   await persistMessage(messages[lastMessageIndex], id);
27
28   // Get chat history by chat id
29   const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
30   const allMessages = [...previousMessages, ...messages];
31
32   try {
```

```
20 // Post request handler
21 export async function POST(req: Request) {
22   const { messages, id } = await req.json();
23
24   // Store current message
25   const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
26   await persistMessage(messages[lastMessageIndex], id);
27
28   // Get chat history by chat id
29   const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
30   const allMessages = [...previousMessages, ...messages];
31
32   try {
33     const convertedMessages = convertToModelMessages(allMessages);
34     const result = streamText({
35       model: openai("gpt-4o"),
36       system:
37         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
38         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
39         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
40         "Return an itinerary of sites to see and things to do based on the weather." +
41         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
42       messages: convertedMessages,
43       stopWhen: stepCountIs(2),
44       tools,
45     });
46
47   // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
48   return result.toUIMessageStreamResponse();
49 } catch (e) {
50   console.error(e);
51   return new NextResponse(
52     "Unable to generate a plan. Please try again later!"
53   );
}
```

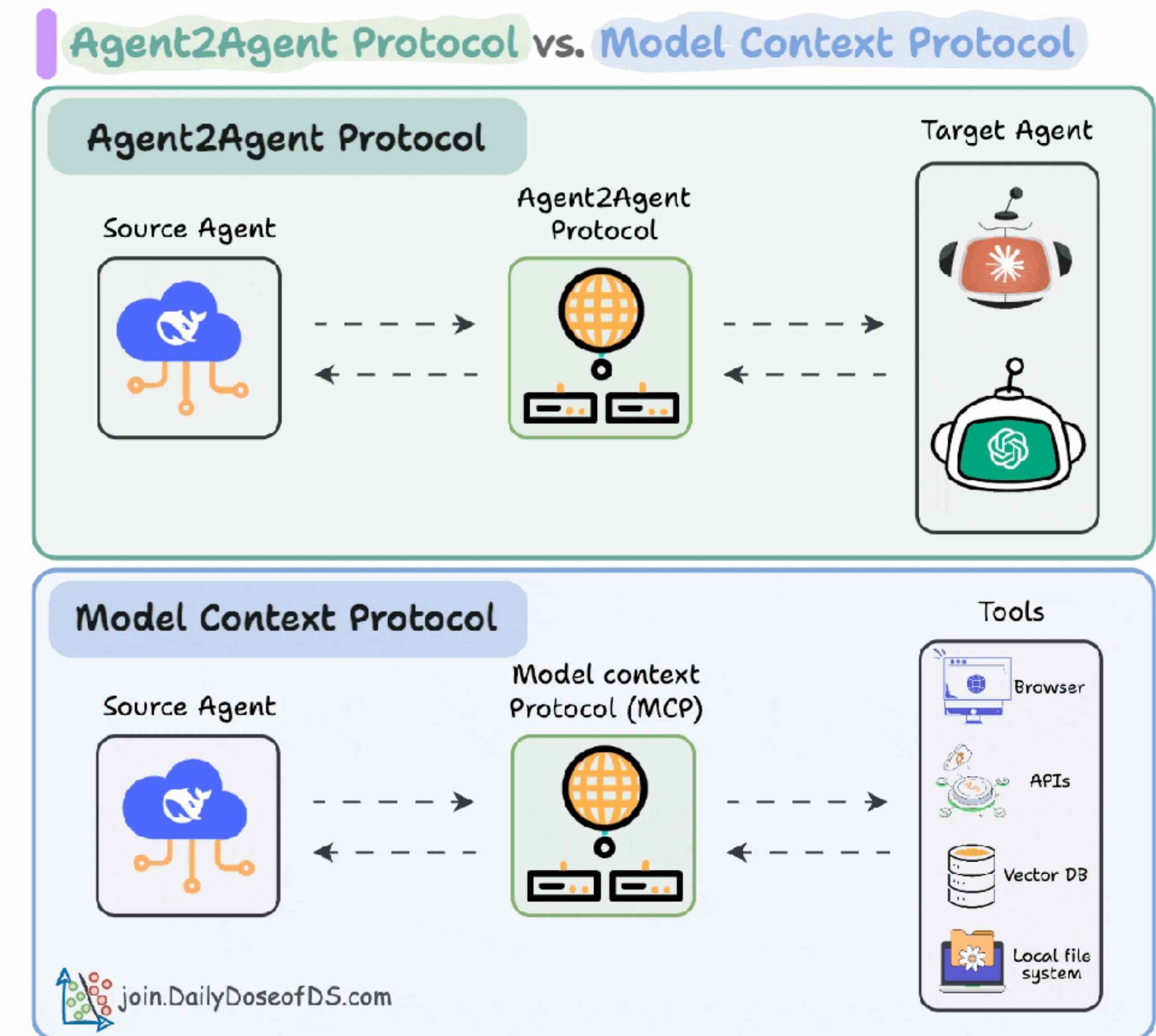
EXAMPLE MEMORY

```
route.ts

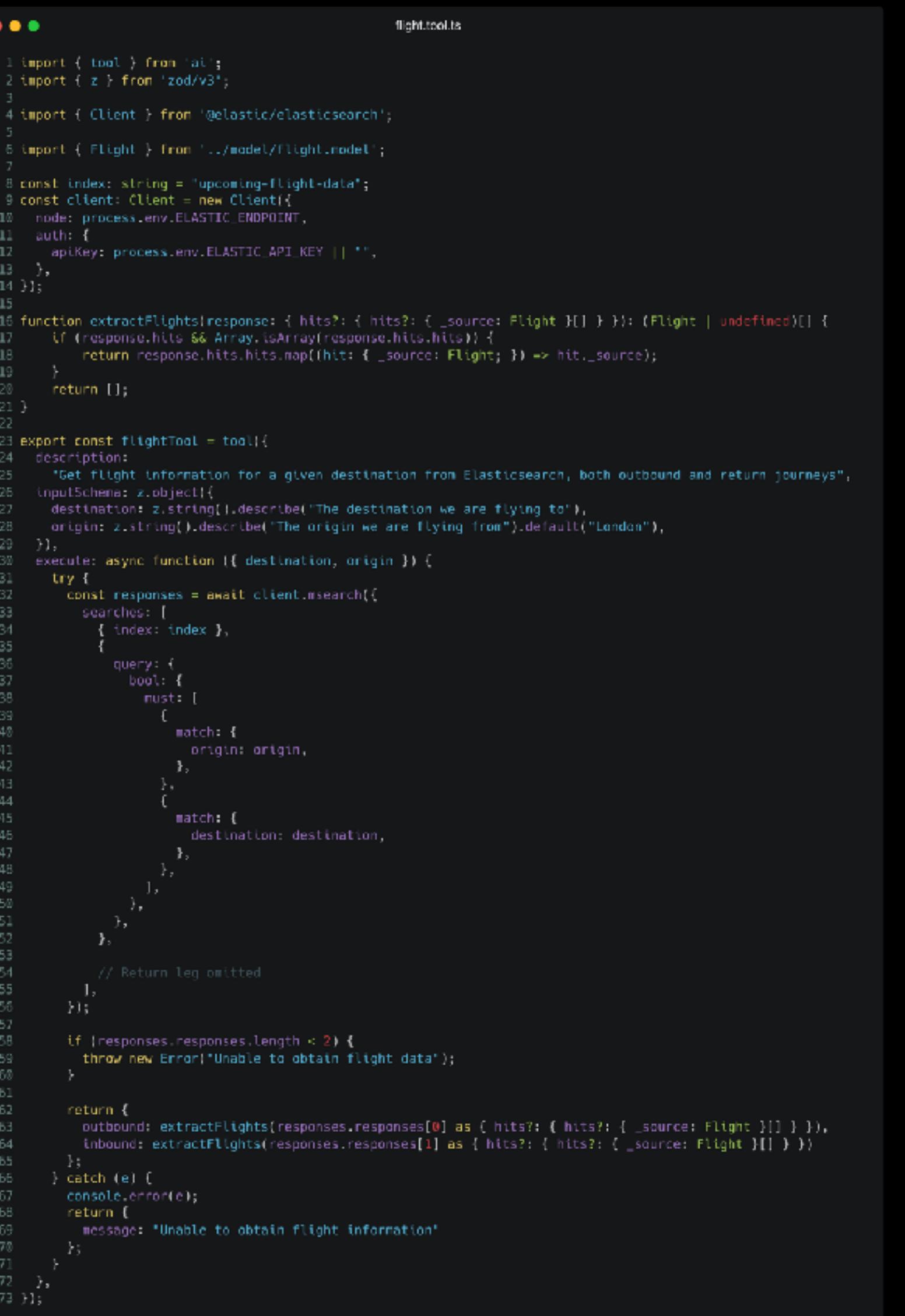
1 import { openai } from "@ai-sdk/openai";
2 import { streamText, stepCountIs, convertToModelMessages } from "ai";
3 import { NextResponse } from "next/server";
4
5 import { weatherTool } from "@/app/ai/weather.tool";
6 import { fcdoTool } from "@/app/ai/fcdo.tool";
7 import { flightTool } from "@/app/ai/flights.tool";
8
9 import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";
10
11 // Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
12 export const maxDuration = 30;
13
14 const tools = {
15   flights: flightTool,
16   weather: weatherTool,
17   fcdo: fcdoTool,
18 };
19
20 // Post request handler
21 export async function POST(req: Request) {
22   const { messages, id } = await req.json();
23
24   // Store current message
25   const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
26   await persistMessage(messages[lastMessageIndex], id);
27
28   // Get chat history by chat id
29   const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
30   const allMessages = [...previousMessages, ...messages];
31
32   try {
33     const convertedMessages = convertToModelMessages(allMessages);
34     const result = streamText({
35       model: openai("gpt-4o"),
36       system:
37         "You are a helpful assistant that returns travel itineraries based on location, the FCDO guidance from the specified tool, and the weather captured from the displayWeather tool." +
38         "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
39         "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
40         "Return an itinerary of sites to see and things to do based on the weather." +
41         "If the FCDO tool warns against travel DO NOT generate an itinerary.",
42       messages: convertedMessages,
43       stopWhen: stepCountIs(2),
44       tools,
45     });
46
47   // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
48   return result.toUIMessageStreamResponse();
49 } catch (e) {
50   console.error(e);
51   return new NextResponse(
52     "Unable to generate a plan. Please try again later!"
53   );
54 }
55 }
```



MCP AND A2A



EXAMPLE AGENT TOOL



```
flighttools
1 import { tool } from 'aot';
2 import { z } from 'zod/v3';
3
4 import { Client } from '@elastic/elasticsearch';
5
6 import { Flight } from '../model/flight.model';
7
8 const index: string = 'upcoming-flight-data';
9 const client: Client = new Client({
10   node: process.env.ELASTIC_ENDPOINT,
11   auth: {
12     apiKey: process.env.ELASTIC_API_KEY || '',
13   },
14 });
15
16 function extractFlights(response: { hits?: { hits?: { _source: Flight[] | null } } } | (Flight | undefined)[]): Flight[] {
17   if (response.hits && Array.isArray(response.hits.hits)) {
18     return response.hits.hits.map(hit: { _source: Flight }) => hit._source;
19   }
20   return [];
21 }
22
23 export const flightTool = tool({
24   description:
25     'Get flight information for a given destination from Elasticsearch, both outbound and return journeys',
26   inputSchema: z.object({
27     destination: z.string().describe('The destination we are flying to'),
28     origin: z.string().describe('The origin we are flying from').default('London'),
29   }),
30   execute: async function ({ destination, origin }) {
31     try {
32       const responses = await client.search({
33         search: [
34           { index: index },
35           {
36             query: {
37               bool: {
38                 must: [
39                   {
40                     match: {
41                       origin: origin,
42                     ...
43                   },
44                   {
45                     match: {
46                       destination: destination,
47                     ...
48                   },
49                   ...
50                 ],
51               },
52             },
53           ],
54           // Return leg omitted
55         ],
56       });
57
58       if (!responses.responses.length < 2) {
59         throw new Error('Unable to obtain flight data');
60       }
61
62       return {
63         outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight[] } } }),
64         inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight[] } } })
65       };
66     } catch (e) {
67       console.error(e);
68       return {
69         message: 'Unable to obtain flight information'
70       };
71     }
72   },
73 });


```

```
23 export const flightTool = tool({
24   description:
25     "Get flight information for a given destination from Elasticsearch, both outbound and return journeys",
26   inputSchema: z.object({
27     destination: z.string().describe("The destination we are flying to"),
28     origin: z.string().describe("The origin we are flying from").default("London"),
29   }),
30   execute: async function ({ destination, origin }) {
31     try {
32       const responses = await client.msearch({
33         searches: [
34           { index: index },
35           {
36             query: {
37               bool: {
38                 must: [
39                   {
40                     match: {
41                       origin: origin,
42                     },
43                   },
44                   {
45                     match: {
46                       destination: destination,
47                     },
48                   },
49                   ],
50                 },
51               },
52             ],
53           }
54         }
55       )
56       return responses
57     } catch (error) {
58       console.error(error)
59       return []
60     }
61   }
62 }
```

```
45     match. {
46         destination: destination,
47         },
48         ],
49         },
50         },
51         },
52         },
53         // Return leg omitted
54     ],
55 });
56 );
57
58 if (responses.responses.length < 2) {
59     throw new Error("Unable to obtain flight data");
60 }
61
62 return {
63     outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight }[] } }),
64     inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight }[] } })
65 };
66 } catch (e) {
67     console.error(e);
68     return {
69         message: "Unable to obtain flight information"
70     };
71 }
72 },
73});
```

```
flight.tool.ts

1 import { tool } from 'ai';
2 import { z } from 'zod/v3';
3
4 import { Client } from '@elastic/elasticsearch';
5
6 import { Flight } from '../model/flight.model';
7
8 const index: string = "upcoming-flight-data";
9 const client: Client = new Client({
10   node: process.env.ELASTIC_ENDPOINT,
11   auth: {
12     apiKey: process.env.ELASTIC_API_KEY || "",
13   },
14 });
15
16 function extractFlights(response: { hits?: { hits?: { _source: Flight }[] } }): (Flight | undefined)[] {
17   if (response.hits && Array.isArray(response.hits.hits)) {
18     return response.hits.hits.map((hit: { _source: Flight }) => hit._source);
19   }
20   return [];
21 }
22
23 export const flightTool = tool({
24   description:
25     "Get flight information for a given destination from Elasticsearch, both outbound and return journeys",
26   inputSchema: z.object({
27     destination: z.string().describe("The destination we are flying to"),
28     origin: z.string().describe("The origin we are flying from").default("London"),
29   }),
30   execute: async function ({ destination, origin }) {
31     try {
32       const responses = await client.msearch({
33         searches: [
34           { index: index },
35           {
36             query: {
37               bool: {
38                 must: [
39                   {
40                     match: {
41                       origin: origin,
42                     },
43                   },
44                   {
45                     match: {
46                       destination: destination,
47                     },
48                   },
49                 ],
50               },
51             },
52           },
53         ],
54         // Return leg omitted
55       ],
56     });
57
58     if (responses.responses.length < 2) {
59       throw new Error("Unable to obtain flight data");
60     }
61
62     return {
63       outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight }[] } }),
64       inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight }[] } })
65     };
66   } catch (e) {
67     console.error(e);
68     return {
69       message: "Unable to obtain flight information"
70     };
71   }
72 },
73});
```

←  D / [MCP Playground](#) / Agent Chat 🔍 ? [Give feedback](#)  CR

Elasticsearch

Home

Discover

Dashboards

Agents

Build

[Index Management](#)

Playground

Relevance

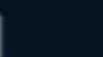
Synonyms

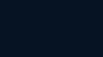
Query rules

Inference endpoints

>_ [Developer Tools](#)

⚙️ [Project settings](#)

Today 
LEGO Red Window Ide...

Last Week 
LEGO Red Window Ide...



Welcome to Elastic Agent Builder

Work interactively with your AI [agents](#) using the chat interface. Your selected agent answers questions by searching your data with its assigned [tools](#).

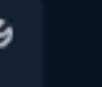
[Read the docs](#)

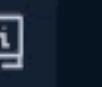
Ask anything 

[LEGO Finder](#) 

 [Create a new agent](#)
Build a custom agent tuned to your data and workflows.

 [Manage agents](#)
View, edit, and organize your existing agents.

 [Manage tools](#)
Add, remove, or edit the tools your agents can use.

 [Get started](#)
Learn how to start building agents and tools.

RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation

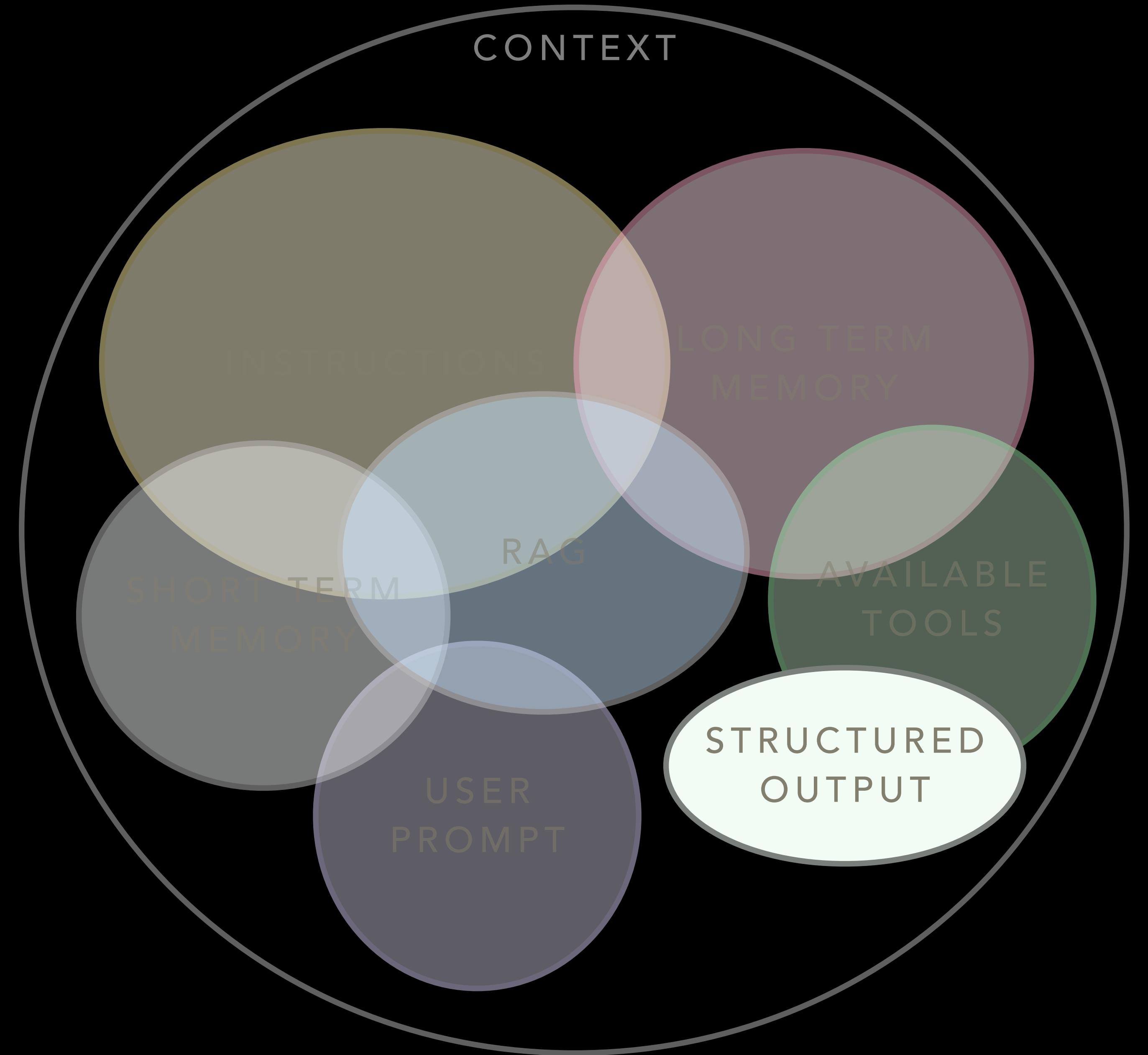
Tiantian Gan^{1,2} and Qiyao Sun^{1,2}

¹ Beijing University of Post and Communications, Beijing, China

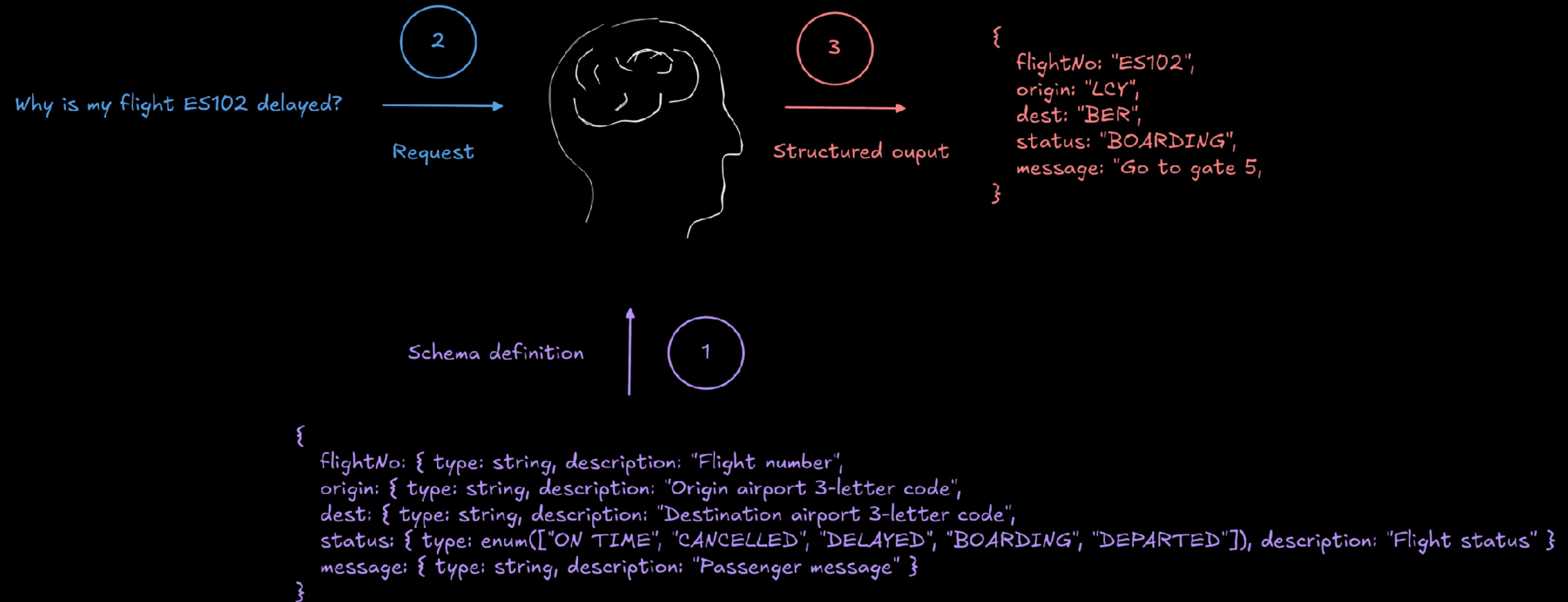
² Queen Mary University of London, London, UK

jp2022213034@qmul.ac.uk, jp2022213402@qmul.ac.uk

Abstract. Large language models (LLMs) struggle to effectively utilize a growing number of external tools, such as those defined by the Model Context Protocol (MCP)[1], due to prompt bloat and selection complexity. We introduce RAG-MCP, a Retrieval-Augmented Generation framework that overcomes this challenge by offloading tool discovery. RAG-MCP uses semantic retrieval to identify the most relevant MCP(s) for a given query from an external index before engaging the LLM. Only the selected tool descriptions are passed to the model, drastically reducing prompt size and simplifying decision-making. Experiments, including an MCP stress test, demonstrate RAG-MCP significantly cuts prompt tokens (e.g., by over 50%) and more than triples tool selection accuracy (43.13% vs 13.62% baseline) on benchmark tasks. RAG-MCP enables scalable and accurate tool integration for LLMs.



STRUCTURED OUTPUTS



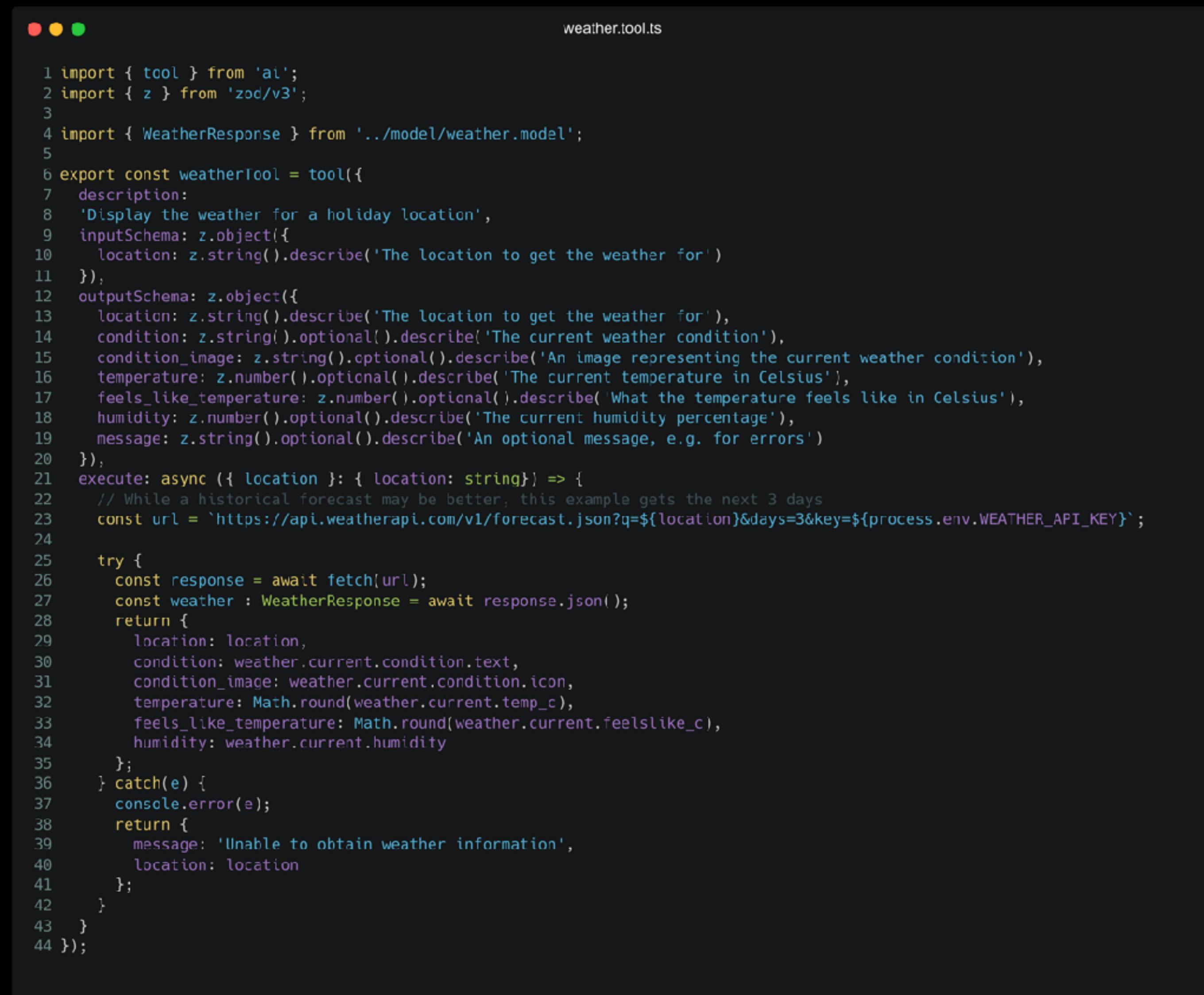
EXAMPLE



generate-itinerary.ts

```
1 import { generateObject } from 'ai';
2 import { z } from 'zod';
3
4 const { object } = await generateObject({
5   model: 'openai/gpt-4.1',
6   schemaName: 'Travel Itinerary',
7   schemaDescription: 'Sample travel itinerary for a trip',
8   schema: z.object({
9     title: z.string(),
10    location: z.string(),
11    hotel: z.object({
12      name: z.string(), roomType: z.string(), amount: z.number(), checkin: z.iso.date(), checkout: z.iso.date()
13    }),
14    flights: z.array(
15      z.object({
16        carrier: z.string(), flightNo: z.string().max(8), origin: z.string(), destination: z.string(), date: z.iso.datetime()
17      })
18    ),
19    excursions: z.array(z.object({ name: z.string(), amount: z.string(), date: z.iso.datetime()}))
20  }),
21  prompt: 'Generate a travel itinerary based on the specified location',
22});
```

EXAMPLE TOOL OUTPUT SCHEMA



```
weather.tool.ts

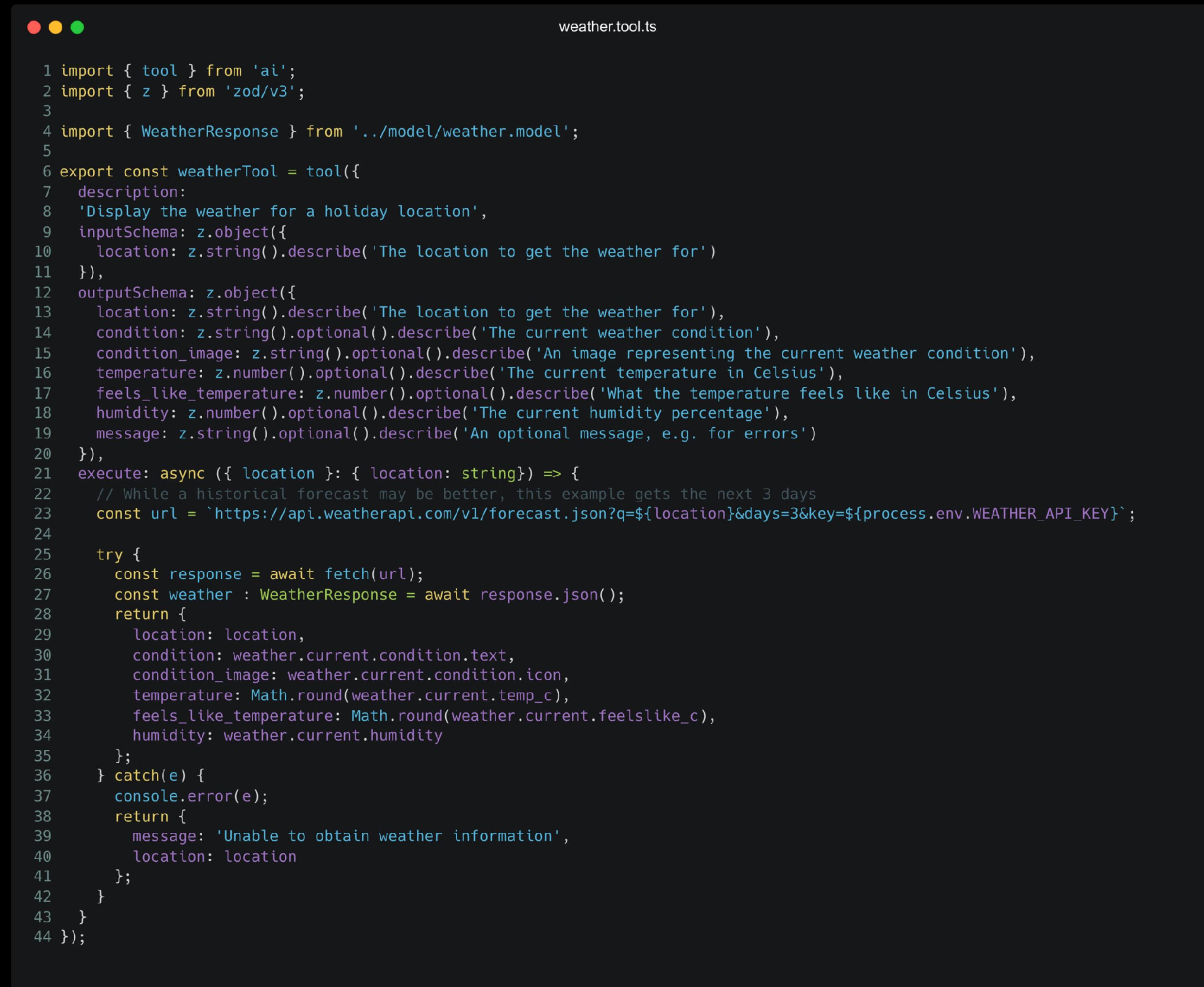
1 import { tool } from 'ai';
2 import { z } from 'zod/v3';
3
4 import { WeatherResponse } from '../model/weather.model';
5
6 export const weatherTool = tool({
7   description:
8     'Display the weather for a holiday location',
9   inputSchema: z.object({
10     location: z.string().describe('The location to get the weather for')
11   }),
12   outputSchema: z.object({
13     location: z.string().describe('The location to get the weather for'),
14     condition: z.string().optional().describe('The current weather condition'),
15     condition_image: z.string().optional().describe('An image representing the current weather condition'),
16     temperature: z.number().optional().describe('The current temperature in Celsius'),
17     feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
18     humidity: z.number().optional().describe('The current humidity percentage'),
19     message: z.string().optional().describe('An optional message, e.g. for errors')
20   },
21   execute: async ({ location }: { location: string }) => {
22     // While a historical forecast may be better, this example gets the next 3 days
23     const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;
24
25     try {
26       const response = await fetch(url);
27       const weather: WeatherResponse = await response.json();
28       return {
29         location: location,
30         condition: weather.current.condition.text,
31         condition_image: weather.current.condition.icon,
32         temperature: Math.round(weather.current.temp_c),
33         feels_like_temperature: Math.round(weather.current.feelslike_c),
34         humidity: weather.current.humidity
35       };
36     } catch (e) {
37       console.error(e);
38       return {
39         message: 'Unable to obtain weather information',
40         location: location
41       };
42     }
43   }
44 });


```



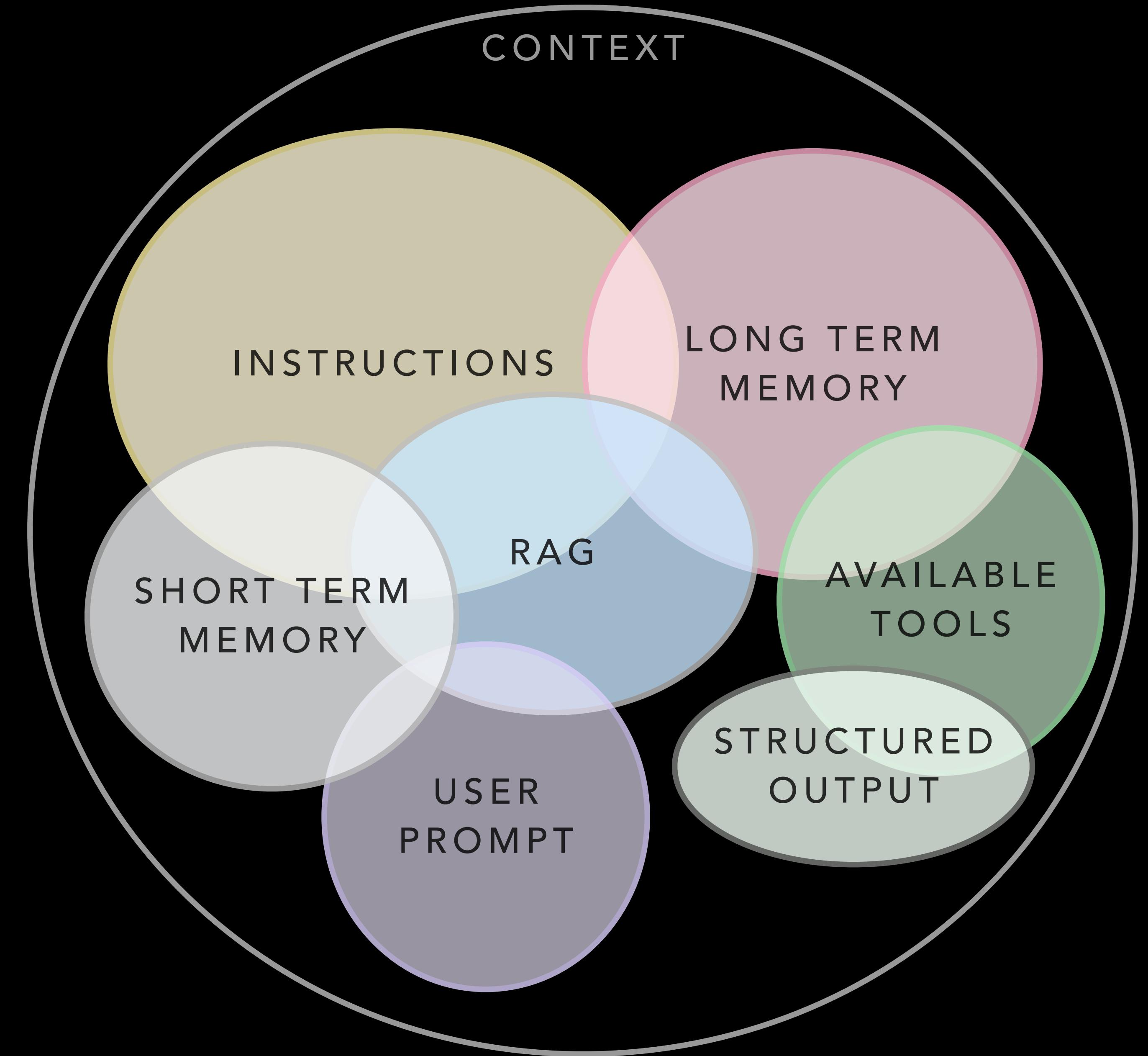
```
1 import { tool } from 'ai';
2 import { z } from 'zod/v3';
3
4 import { WeatherResponse } from '../model/weather.model';
5
6 export const weatherTool = tool({
7   description:
8     'Display the weather for a holiday location',
9   inputSchema: z.object({
10     location: z.string().describe('The location to get the weather for')
11   }),
12   outputSchema: z.object({
13     location: z.string().describe('The location to get the weather for'),
14     condition: z.string().optional().describe('The current weather condition'),
15     condition_image: z.string().optional().describe('An image representing the current weather condition'),
16     temperature: z.number().optional().describe('The current temperature in Celsius'),
17     feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
18     humidity: z.number().optional().describe('The current humidity percentage'),
19     message: z.string().optional().describe('An optional message, e.g. for errors')
20   }),
21   execute: async ({ location }: { location: string }) => {
22     // While a historical forecast may be better, this example gets the next 3 days
23     const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;
24
25     try {
26       const response = await fetch(url);
27       const weather: WeatherResponse = await response.json();
28       return {
29         location: location,
```

EXAMPLE TOOL OUTPUT SCHEMA



The screenshot shows a code editor window with a dark theme. The title bar says "weather.tool.ts". The code is written in TypeScript and defines a tool for getting weather information. It imports tools from 'ai' and 'zod/v3'. The tool has a description and an input schema for a location string. The output schema includes current condition, temperature, feels-like temperature, humidity, and a message. The execute function uses an API endpoint to fetch weather data and returns it according to the schema.

```
1 import { tool } from 'ai';
2 import { z } from 'zod/v3';
3
4 import { WeatherResponse } from '../model/weather.model';
5
6 export const weatherTool = tool({
7   description:
8     'Display the weather for a holiday location',
9   inputSchema: z.object({
10     location: z.string().describe('The location to get the weather for')
11   }),
12   outputSchema: z.object({
13     location: z.string().describe('The location to get the weather for'),
14     condition: z.string().optional().describe('The current weather condition'),
15     condition_image: z.string().optional().describe('An image representing the current weather condition'),
16     temperature: z.number().optional().describe('The current temperature in Celsius'),
17     feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
18     humidity: z.number().optional().describe('The current humidity percentage'),
19     message: z.string().optional().describe('An optional message, e.g. for errors')
20   }),
21   execute: async ({ location }: { location: string }) => {
22     // While a historical forecast may be better, this example gets the next 3 days
23     const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;
24
25     try {
26       const response = await fetch(url);
27       const weather: WeatherResponse = await response.json();
28       return {
29         location: location,
30         condition: weather.current.condition.text,
31         condition_image: weather.current.condition.icon,
32         temperature: Math.round(weather.current.temp_c),
33         feels_like_temperature: Math.round(weather.current.feelslike_c),
34         humidity: weather.current.humidity
35       };
36     } catch(e) {
37       console.error(e);
38       return {
39         message: 'Unable to obtain weather information',
40         location: location
41       };
42     }
43   }
44 });
```





SCAN ME