Source: Milad Fakurian on Unsplash

CRAFTING INTELLIGENT AGENTS WITH CONTEXT ENGINEERING

CARLY RICHMOND

ABOUT ME

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

SCAN ME

carlyrichmond.bsky.social

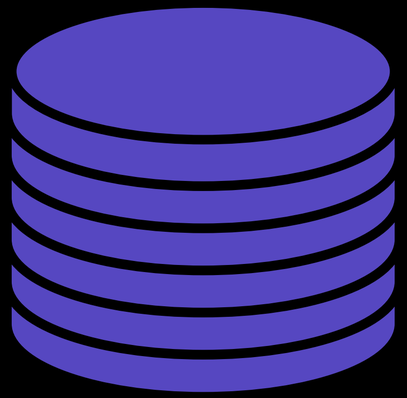
Source: Context window overflow: Breaking the barrier | AWS Security Blog

CONTEXT WINDOW

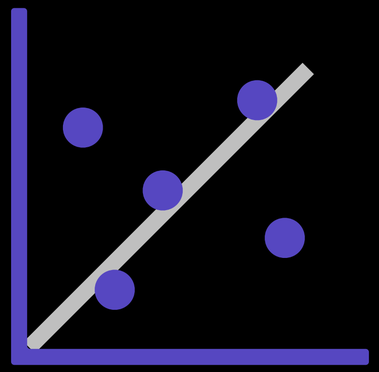
The maximum number of tokens an LLM can process at once.

Source: Braňo on Unsplash

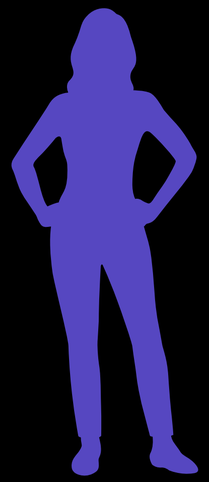
WHY DO LLMS HALLUCINATE?



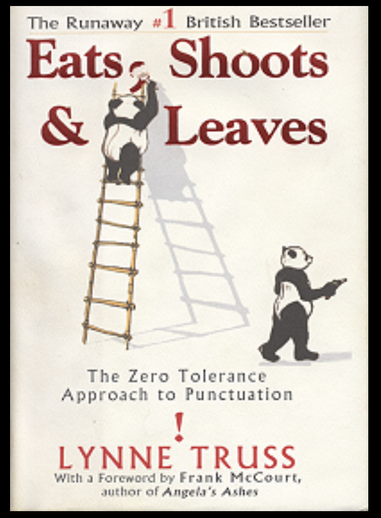
Frozen or Limited Knowledge



Overfitting



Biases



Language Ambiguity

Source: OpenAI | Why Language Models Hallucinate

Why Language Models Hallucinate

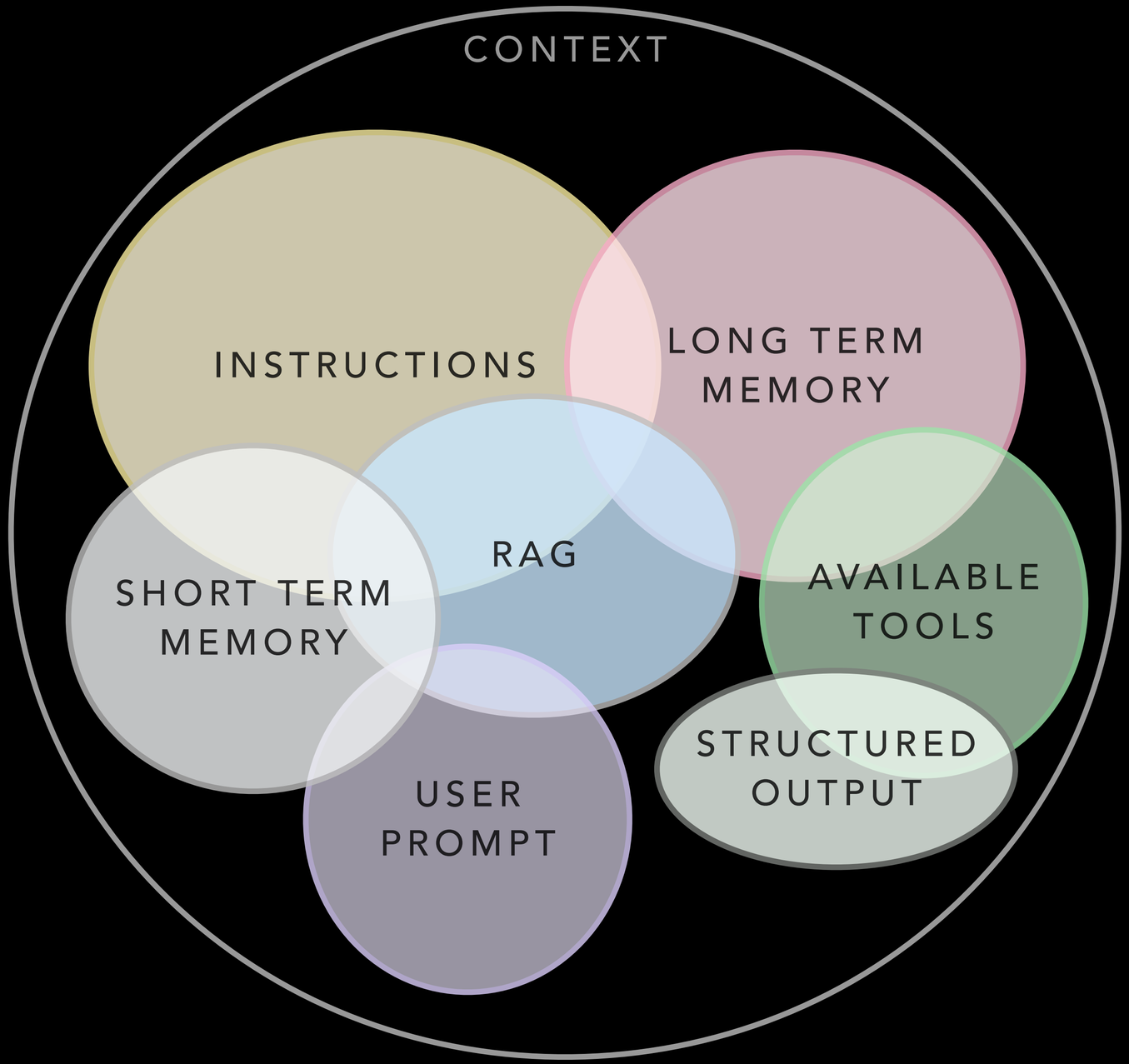
Adam Tauman Kalai* OpenAI
Ofir Nachum OpenAI
Santosh S. Vempala† Georgia Tech
Edwin Zhang OpenAI

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.

Credit: What is Context Engineering | Data Science Dojo



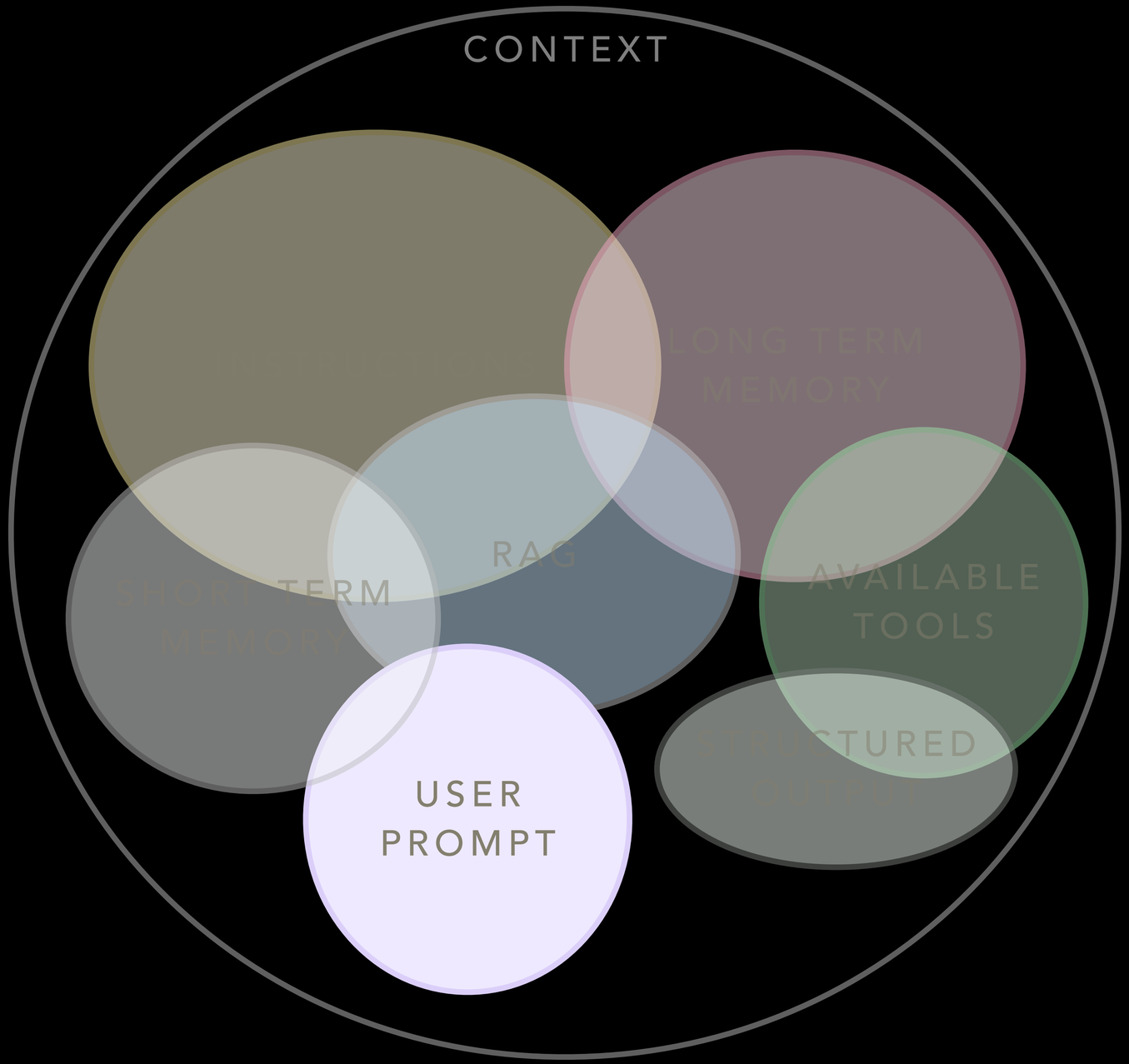
elastic

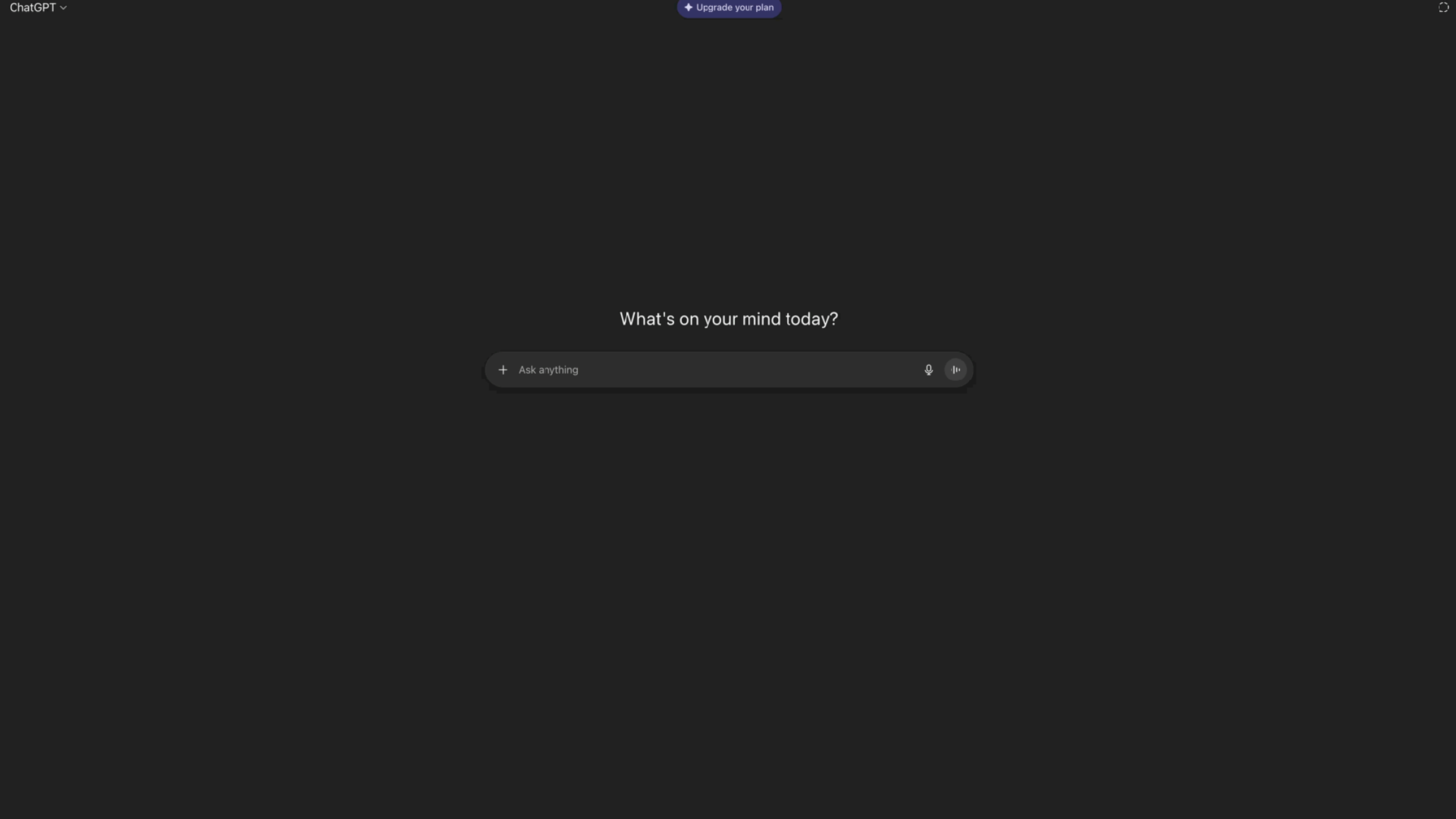
https://www.elastic.co/search-labs/blog/context-engineering-overview

Travel Planner

Where would you like to go?

Credit: What is Context Engineering | Data Science Dojo




ChatGPT
Upgrade your plan
What's on your mind today?
Ask anything

IMAGES

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

PASSING THE USER PROMPT

page.tsx

```tsx
"use client";
import { useChat } from "@ai-sdk/react";
import { FormEvent, KeyboardEvent, useState } from "react";
import Image from 'next/image';
import { Converter } from "showdown";

import Spinner from "./components/spinner";
import { Weather, WeatherProps } from "./components/weather";

import pending from "../../public/multi-cloud.svg";
import { DefaultChatTransport } from "ai";
import pin from "../../public/world-pin.svg";
import { FCDOGuidance, FCDOGuidanceProps } from "./components/fcdo";

export default function Chat() {
  const [input, setInput] = useState("");
  const [lastRequest, setLastRequest] = useState("");
  /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for
  a better user experience */
  const { messages, sendMessage, status, stop, error } = useChat({
    transport: new DefaultChatTransport({
      api: "/api/chat",
    }),
  });
  const markdownConverter = new Converter();

  function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
    event.preventDefault();
    if (input.trim()) {
      sendMessage({
        parts: [{ type: "text", text: input }],
      });
      setLastRequest(input);
      setInput("");
    }
  }

  return (
    <div className="chat__form">
      <div className="chat__messages">
        {
          /* Display all user messages and assistant responses */
          messages.map((m) => (
            <div key={m.id} className="message">
              <div>
                {/* Messages with the role of 'assistant' denote responses from the LLM */}
                <div className="role">
                  {m.role === "user" ? "Me" : "Sorley"}
                </div>
                {/* Tool handling */}
                <div className="tools__summary">
                  {m.parts.map((part, index) => {
                    if (part.type === "text") {
                      { /* User or LLM generated content */}
                      return (
                        <div
                          className="itinerary__div" key={`${m.id}-${index}-text`}
                          dangerouslySetInnerHTML={{ __html: markdownConverter.makeHtml(part.text)}}>
                        </div>
                      );
                    }
                  })}
                </div>
              </div>
            </div>
          ))
        }
      </div>
      {
        /* Spinner and error handling omitted */
      }
      <form
        onSubmit={(event) => {sendUserChat(event)}}>
        <input
          className="search-box__input"
          value={input}
          placeholder="Where would you like to go?"
          disabled={status  == 'ready'}
          onChange={(event) => {
            setInput(event.target.value);
          }}
          onKeyDown={async (event) => {
            if (event.key === "Enter") {
              sendUserChat(event);
            }
          }}
        />
      </form>
    </div>
  );
}
```

https://www.elastic.co/search-labs/blog/ai-agents-ai-sdk-elasticsearch

```
export default function Chat() {
  const [input, setInput] = useState("");
  const [lastRequest, setLastRequest] = useState("");
  /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for a better user experience */
  const { messages, sendMessage, status, stop, error } = useChat({
    transport: new DefaultChatTransport({
      api: "/api/chat",
    }),
  });
  const markdownConverter = new Converter();

  function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
    event.preventDefault();
    if (input.trim()) {
      sendMessage({
        parts: [{ type: "text", text: input }],
      });
      setLastRequest(input);
      setInput("");
    }
  }

  return (
    <div className="chat__form">
      <div className="chat__messages">
        {
          /* Display all user messages and assistant responses */
          messages.map((m) => (
            <div key={m.id} className="message">
```

```
                    }
                  </div>
                </div>
              </div>
            ))
          }
        </div>
        {
          /* Spinner and error handling omitted */

        <form
          onSubmit={(event) => {sendUserChat(event)}}>
          <input
            className="search-box__input"
            value={input}
            placeholder="Where would you like to go?"
            disabled={status !== "ready"}
            onChange={(event) => {
              setInput(event.target.value);
            }}
            onKeyDown={async (event) => {
              if (event.key === "Enter") {
                sendUserChat(event);
              }
            }}
          />
        </form>
      </div>
    );
}
```

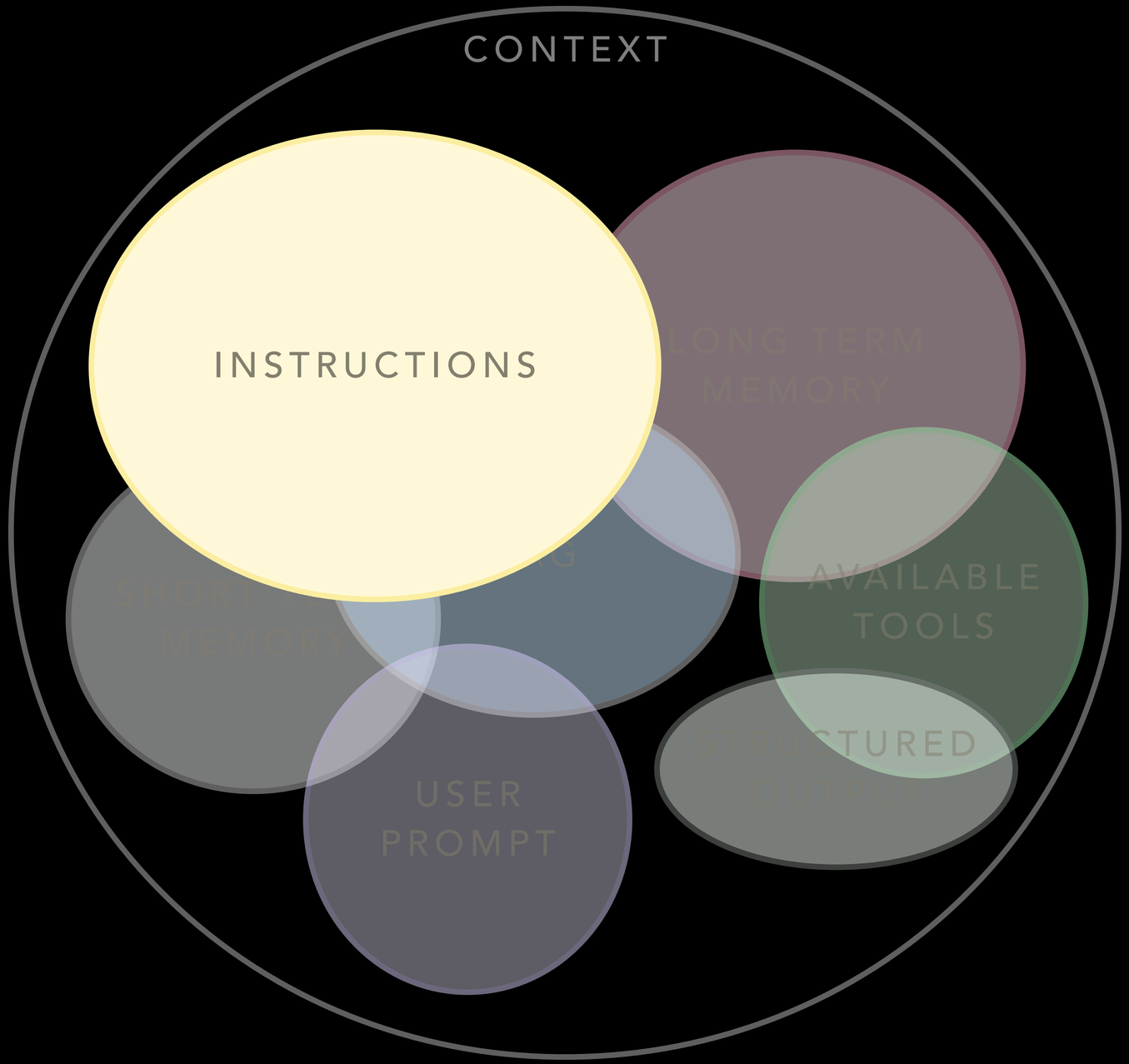
PASSING THE USER PROMPT

page.tsx

```tsx
"use client";
import { useChat } from "@ai-sdk/react";
import { FormEvent, KeyboardEvent, useState } from "react";
import Image from "next/image";
import { Converter } from "showdown";

import Spinner from "./components/spinner";
import { Weather, WeatherProps } from "./components/weather";

import pending from "../../public/multi-cloud.svg";
import { DefaultChatTransport } from "ai";
import pin from "../../public/world-pin.svg";
import { FCDOGuidance, FCDOGuidanceProps } from "./components/fcdo";

export default function Chat() {
 const [input, setInput] = useState("");
 const [lastRequest, setLastRequest] = useState("");
 /* useChat hook helps us handle the input, resulting messages, and also handle the loading and error states for a better user experience */
 const { messages, sendMessage, status, stop, error } = useChat({
   transport: new DefaultChatTransport({
    api: "/api/chat",
   }),
 });
 const markdownConverter = new Converter();

 function sendUserChat(event: FormEvent<HTMLFormElement> | KeyboardEvent<HTMLInputElement>) {
  event.preventDefault();
  if (input.trim()) {
    sendMessage({
      parts: [{ type: "text", text: input }],
    });
    setLastRequest(input);
    setInput("");
  }
 }

 return (
   <div className="chat__form">
     <div className="chat__messages">
       {
         /* Display all user messages and assistant responses */
        messages.map((m) => (
          <div key={m.id} className="message">
            <div>
              {/* Messages with the role of *assistant* denote responses from the LLM */}
              <div className="role">
                {m.role === "user" ? "Me" : "Sorley"}
              </div>
              {/* Tool handling */}
              <div className="tools__summary">
                {m.parts.map((part, index) => {
                  if (part.type === "text") {
                    { /* User or LLM generated content */}
                    return (
                      <div
                        className="itinerary__div" key={`${m.id}-${index}-text`}
                        dangerouslySetInnerHTML={{ __html: markdownConverter.makeHtml(part.text)}}>
                      </div>
                    );
                  }
              </div>
            </div>
          </div>
        ))
      }
     </div>
     {
       /* Spinner and error handling omitted */

     <form
       onSubmit={(event) => {sendUserChat(event)}}>
       <input
         className="search-box__input"
         value={input}
         placeholder="Where would you like to go?"
         disabled={status !== "ready"}
         onChange={(event) => {
           setInput(event.target.value);
         }}
         onKeyDown={async (event) => {
           if (event.key === "Enter") {
             sendUserChat(event);
           }
         }}
       />
     </form>
   </div>
 );
}
```

https://www.elastic.co/search-labs/blog/ai-agents-ai-sdk-elasticsearch

Credit: What is Context Engineering | Data Science Dojo



elastic

https://www.elastic.co/search-labs/blog/context-engineering-overview

carlyrichmond.bsky.social

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 ${DESINATION} 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

```ts
import { openai } from '@ai-sdk/openai';
import { streamText, stepCountIs, convertToModelMessages, UIMessage } from 'ai';
import { NextResponse } from 'next/server';

import { weatherTool } from '@/app/ai/weather.tool';
import { fcdoTool } from '@/app/ai/fcdo.tool';
import { flightTool } from '@/app/ai/flights.tool';

// Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
export const maxDuration = 30;

const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool
};

// Post request handler
export async function POST(req: Request) {
  const { messages }: { messages: UIMessage[] } = await req.json();
  const convertedMessages = convertToModelMessages(messages);

  try {
    const result = streamText({
      model: openai('gpt-4o'),
      system:
      "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." +
      'Use the flight information from tool getFlights only to recommend possible flights in the itinerary.' +
      "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
      "Return an itinerary of sites to see and things to do based on the weather." +
      "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
    // better user experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
    return new NextResponse("Unable to generate a plan. Please try again later!");
  }
}
```

https://www.elastic.co/search-labs/blog/ai-agents-ai-sdk-elasticsearch

```
const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool
};

// Post request handler
export async function POST(req: Request) {
  const { messages }: { messages: UIMessage[] } = await req.json();
  const convertedMessages = convertToModelMessages(messages);

  try {
    const result = streamText({
      model: openai('gpt-4o'),
      system:
      "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." +
      "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
      "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
      "Return an itinerary of sites to see and things to do based on the weather." +
      "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
    // better user experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
```

EXAMPLE SYSTEM PROMPT

route.ts

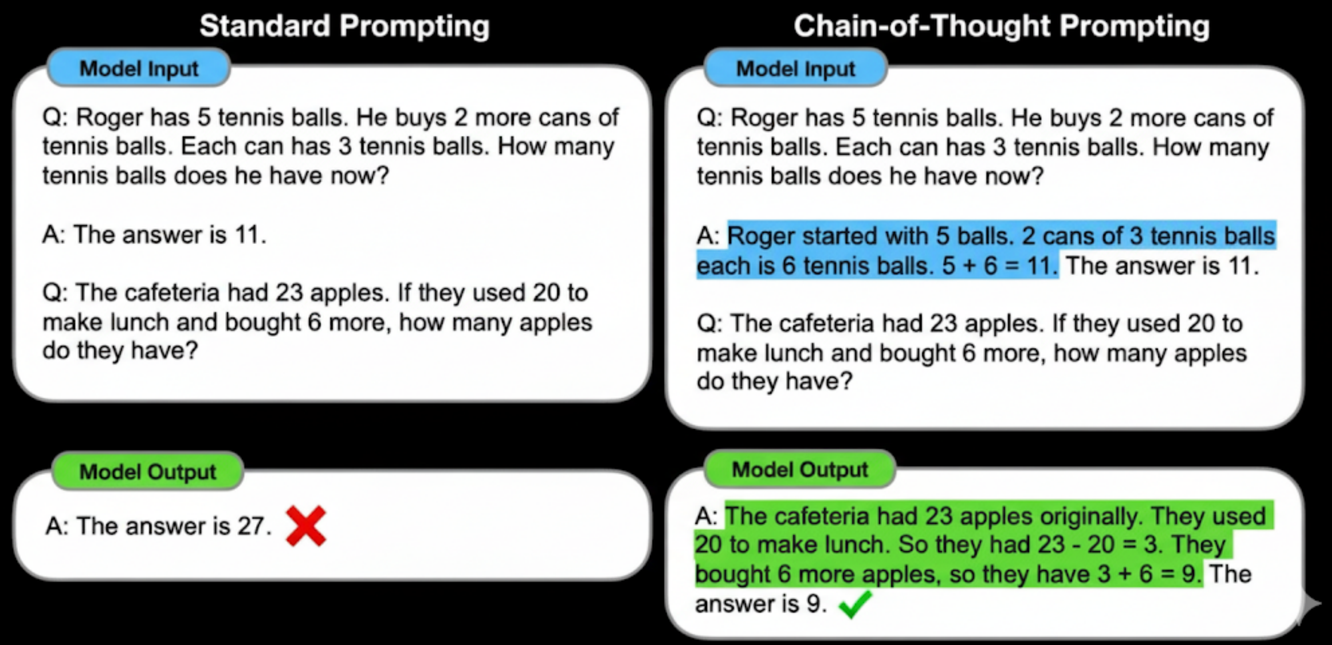
```ts
import { openai } from '@ai-sdk/openai';
import { streamText, stepCountIs, convertToModelMessages, UIMessage } from 'ai';
import { NextResponse } from 'next/server';

import { weatherTool } from '@/app/ai/weather.tool';
import { fcdoTool } from '@/app/ai/fcdo.tool';
import { flightTool } from '@/app/ai/flights.tool';

// Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
export const maxDuration = 30;

const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool
};

// Post request handler
export async function POST(req: Request) {
  const { messages }: { messages: UIMessage[] } = await req.json();
  const convertedMessages = convertToModelMessages(messages);

  try {
    const result = streamText({
      model: openai('gpt-4o'),
      system:
      "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." +
      "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
      "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
      "Return an itinerary of sites to see and things to do based on the weather." +
      "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a
    // better user experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
    return new NextResponse("Unable to generate a plan. Please try again later!");
  }
}
```

https://www.elastic.co/search-labs/blog/ai-agents-ai-sdk-elasticsearch

Source: The Prompt Report: A Systematic Survey of Prompt Engineering Techniques | Schulhoff et al.

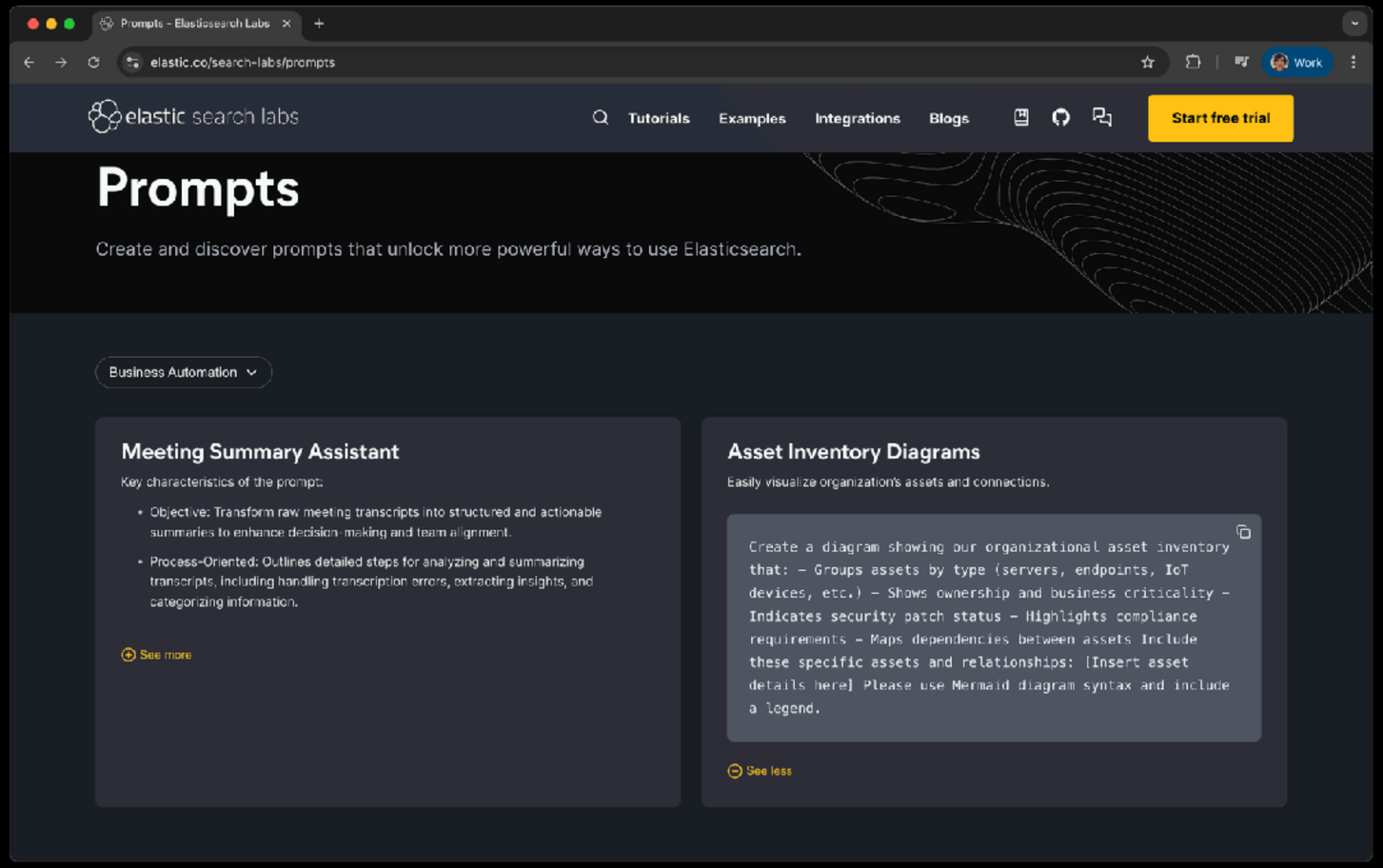
EXAMPLE STRATEGIES

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



HOW TO BUILD A GOOD PROMPT

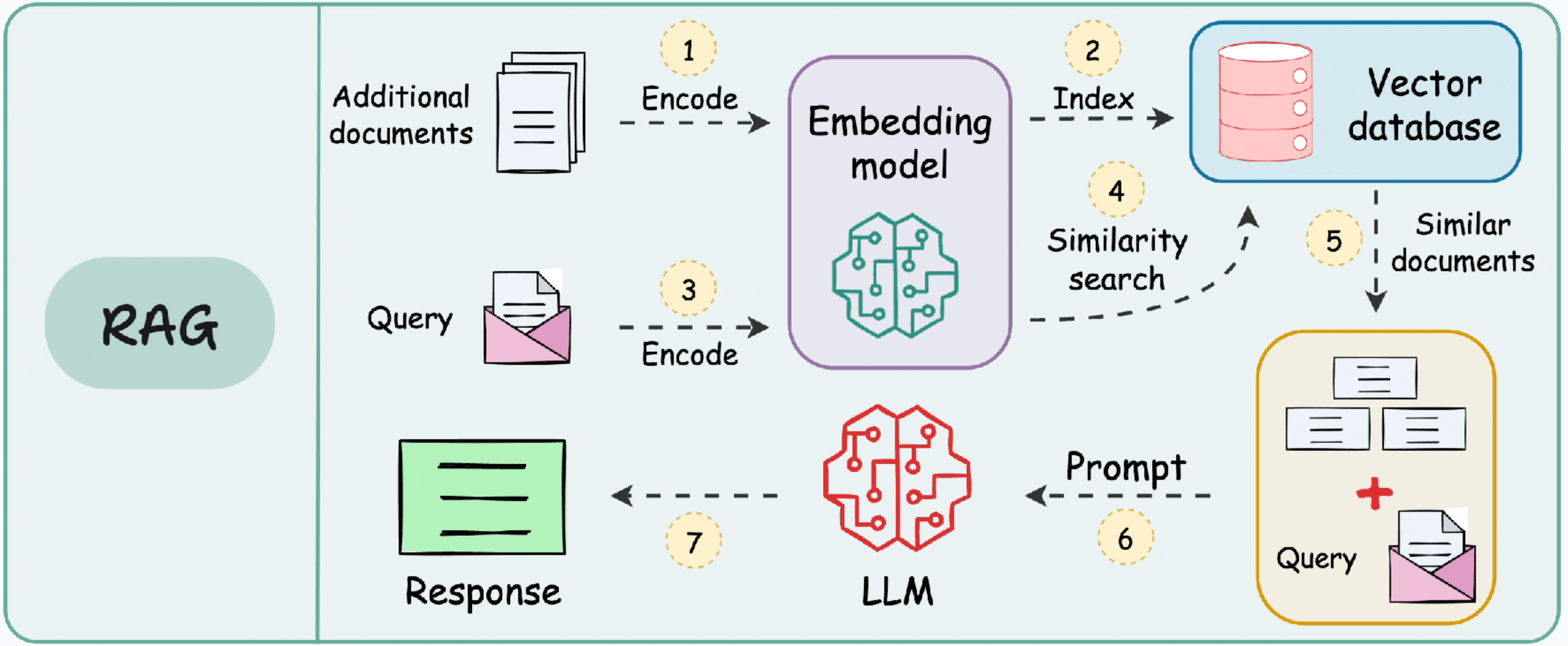
- Task type
- Complexity and ambiguity
- Inputs and outputs
- Data format
- Persona to emulate
- LLM itself

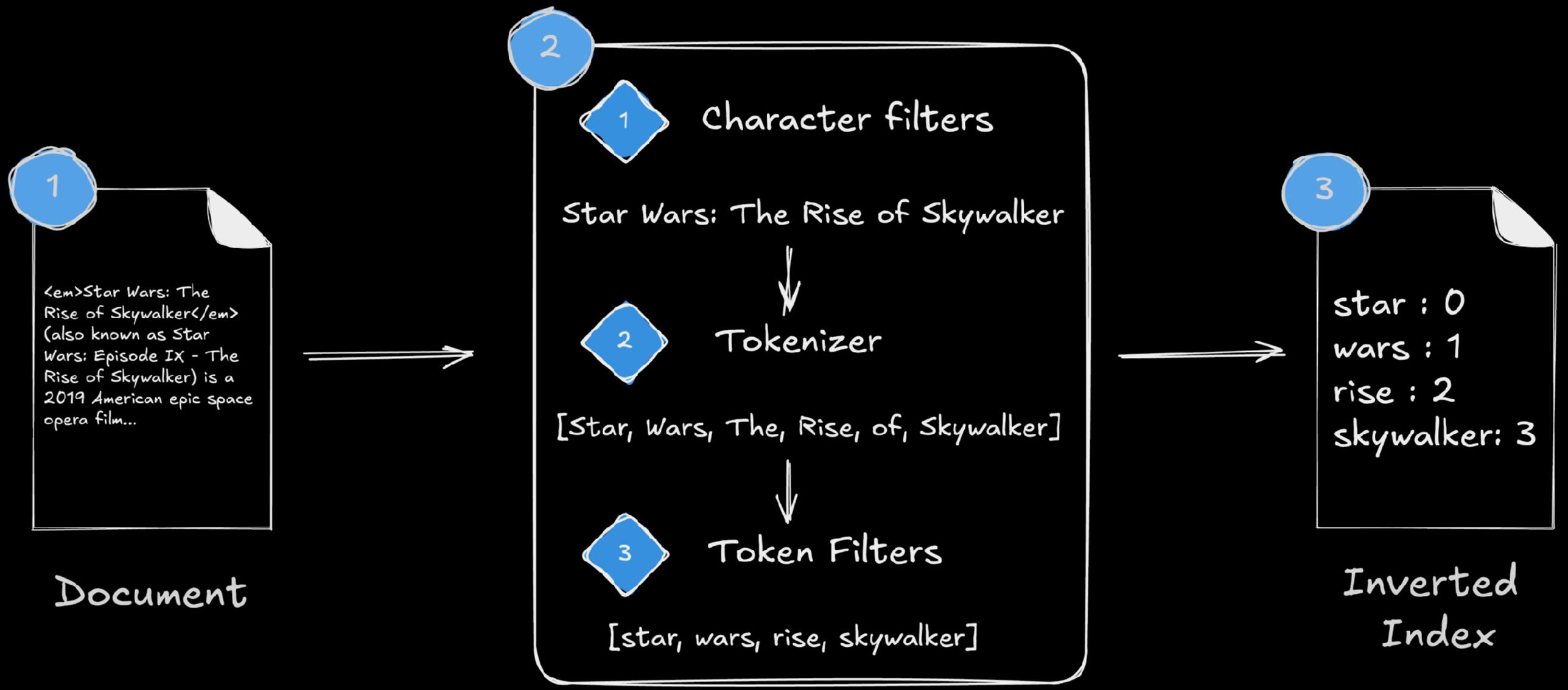


Credit: What is Context Engineering | Data Science Dojo

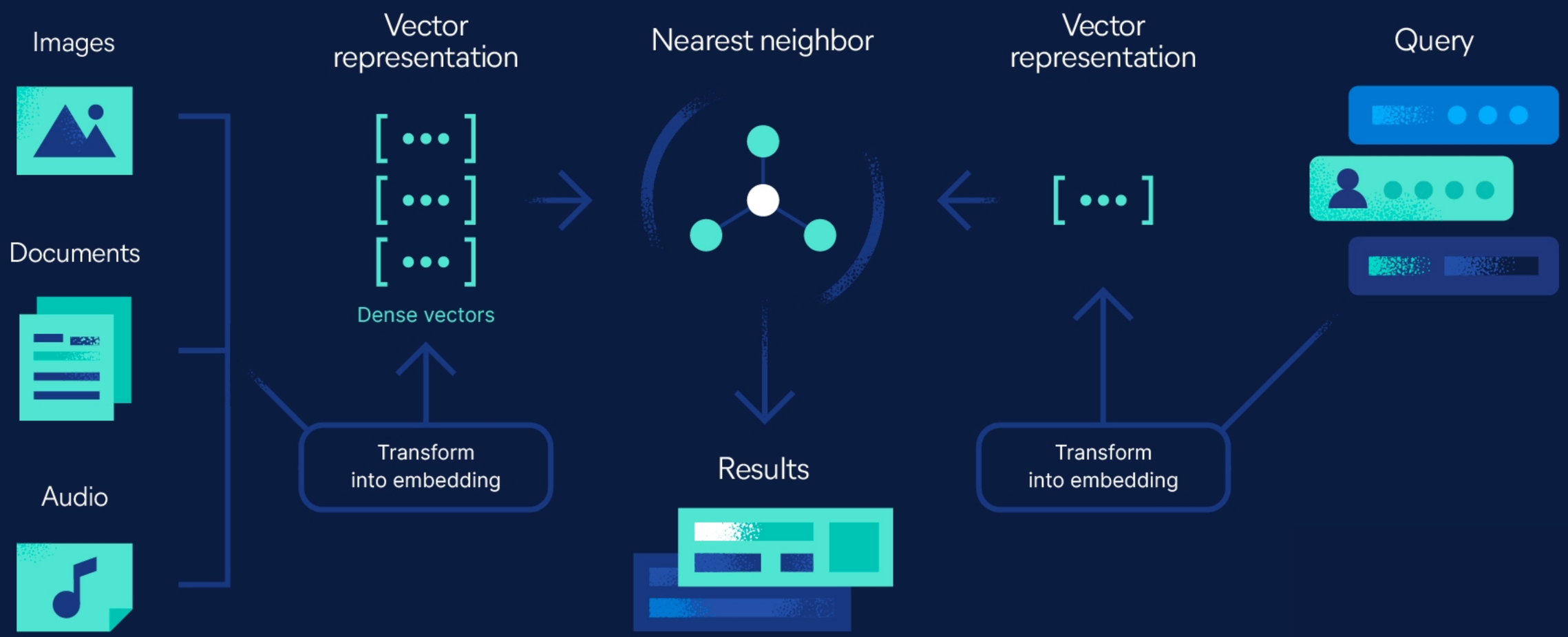
https://www.elastic.co/search-labs/blog/context-engineering-overview

Credit: 5 Chunking Strategies For RAG | Daily Dose of Data Science



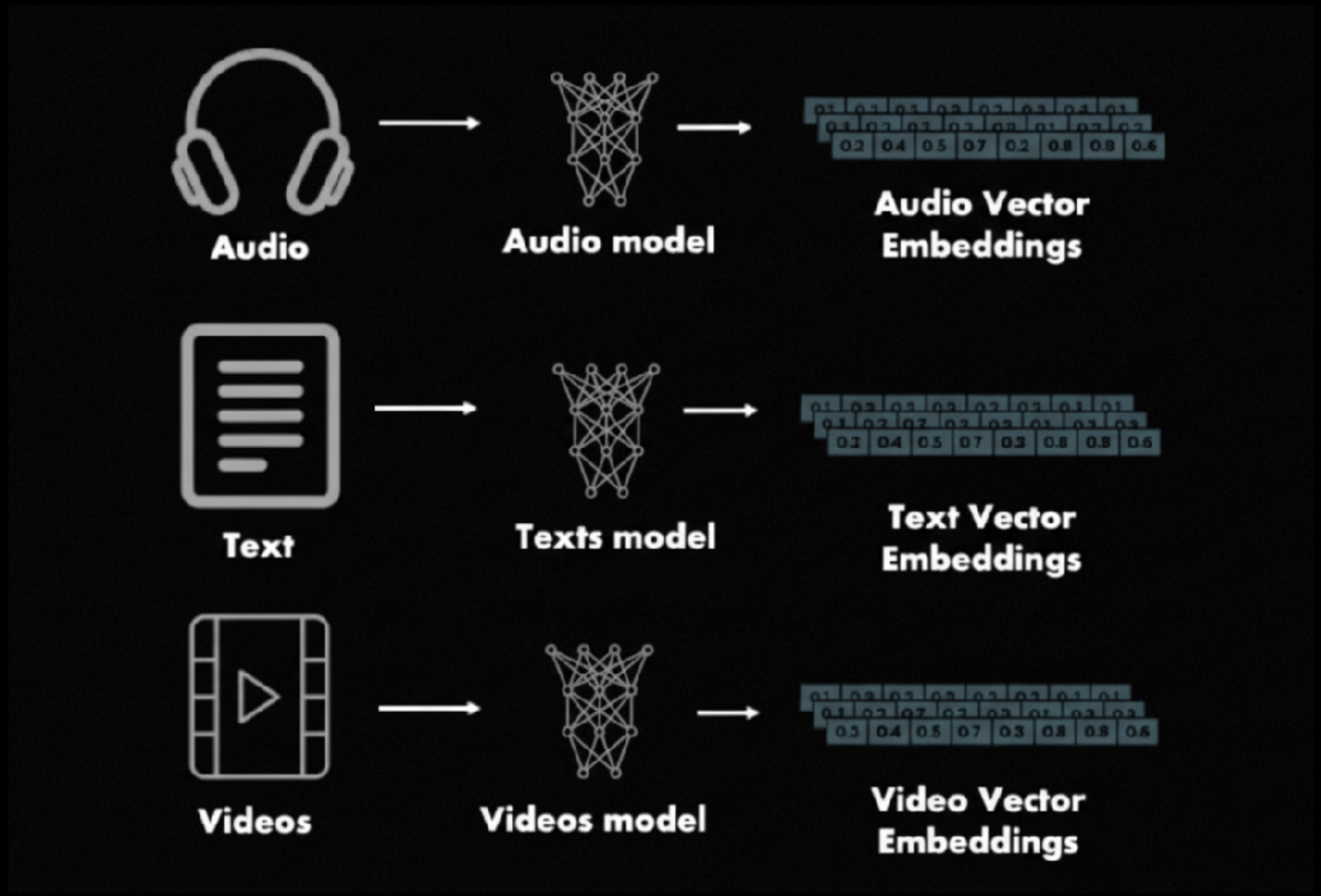

1
Star Wars: The Rise of Skywalker (also known as Star Wars: Episode IX - The Rise of Skywalker) is a 2019 American epic space opera film...
Document
2
1
Character filters
Star Wars: The Rise of Skywalker
2
Tokenizer
[Star, Wars, The, Rise, of, Skywalker]
3
Token Filters
[star, wars, rise, skywalker]
3
star : 0
wars : 1
rise : 2
skywalker: 3
Inverted Index

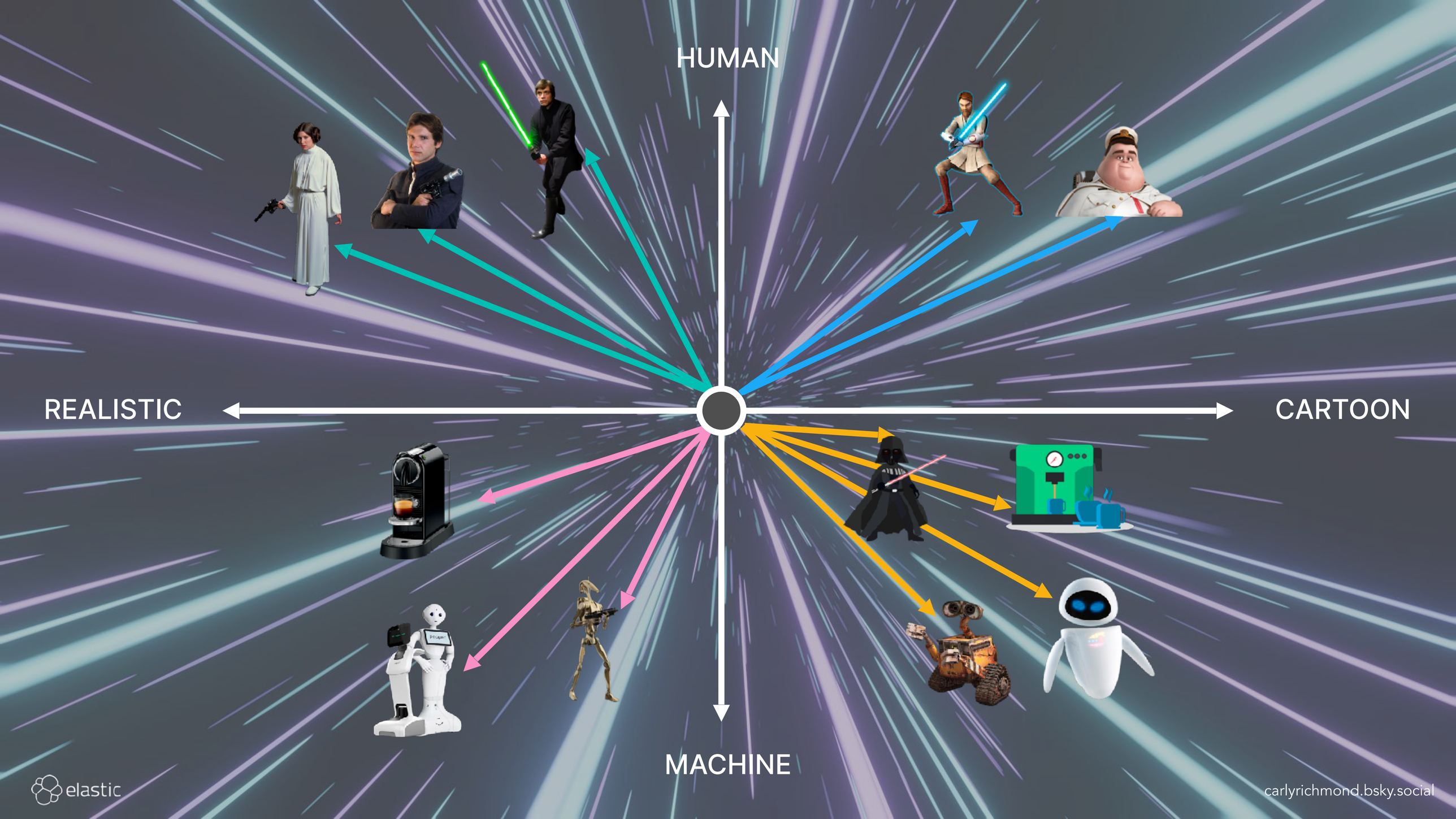
VECTOR SEARCH



Credit: What is Context Engineering | Data Science Dojo

WHERE DO EMBEDDINGS COME FROM?

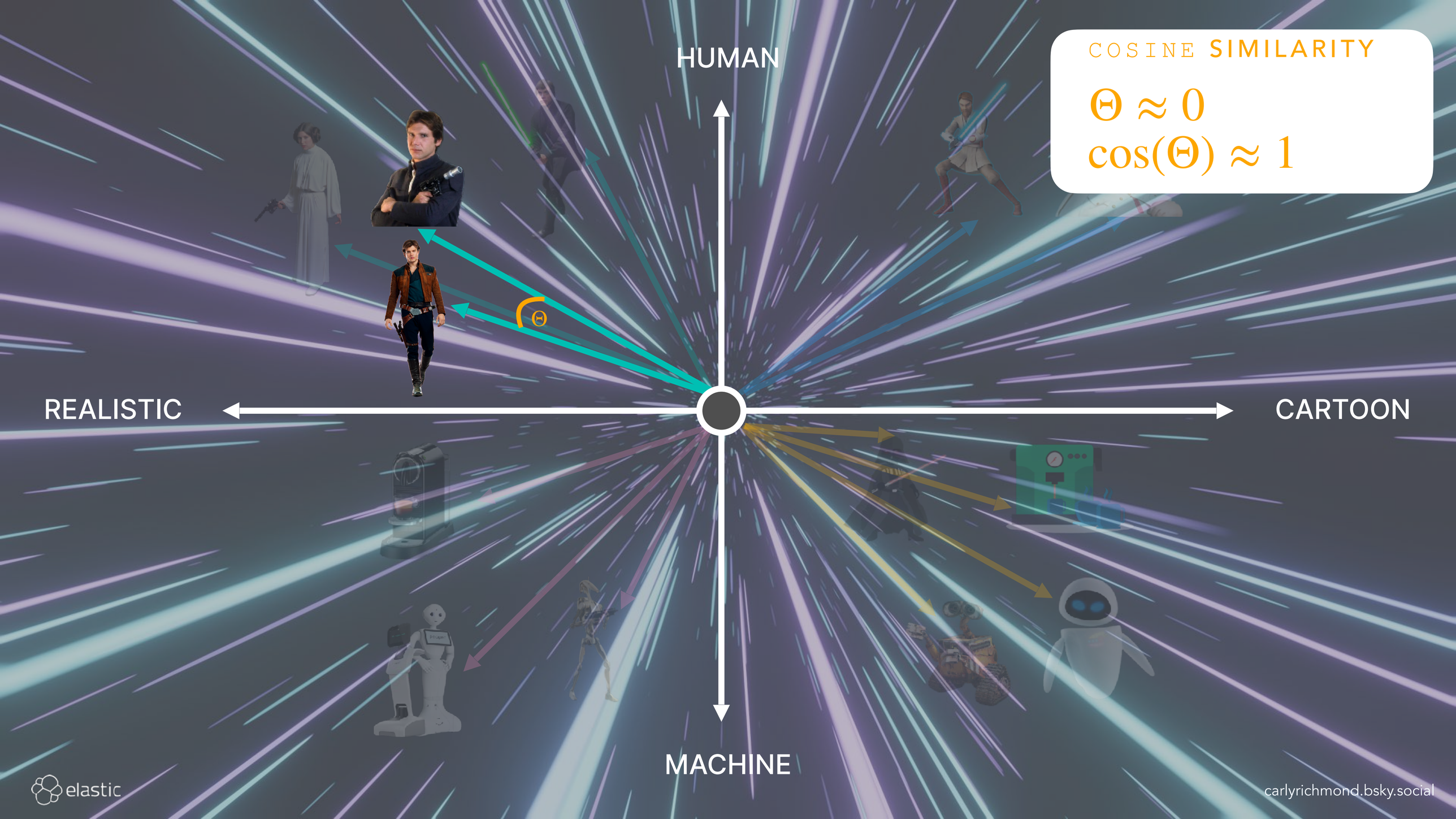




elastic

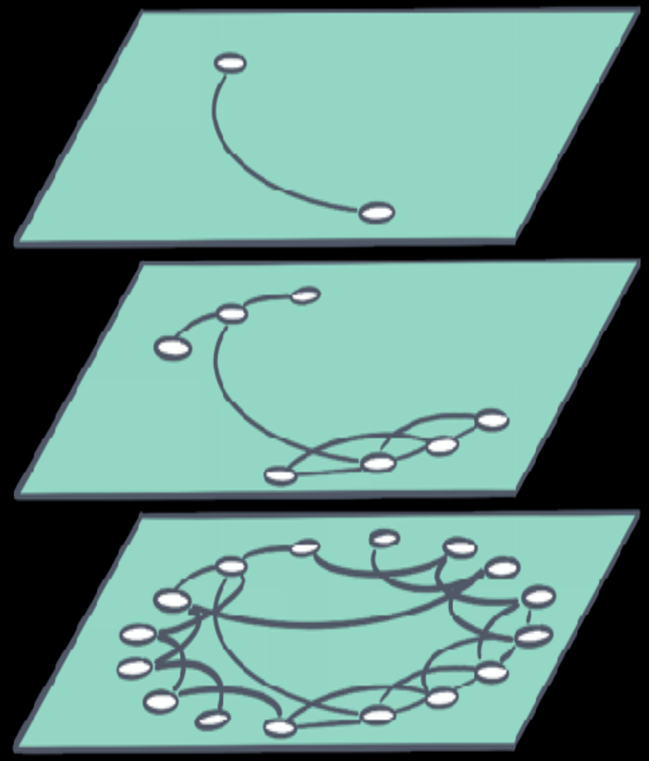
carlyrichmond.bsky.social

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


HUMAN
REALISTIC
CARTOON
MACHINE
Θ
COSINE SIMILARITY
Θ ≈ 0
cos(Θ) ≈ 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

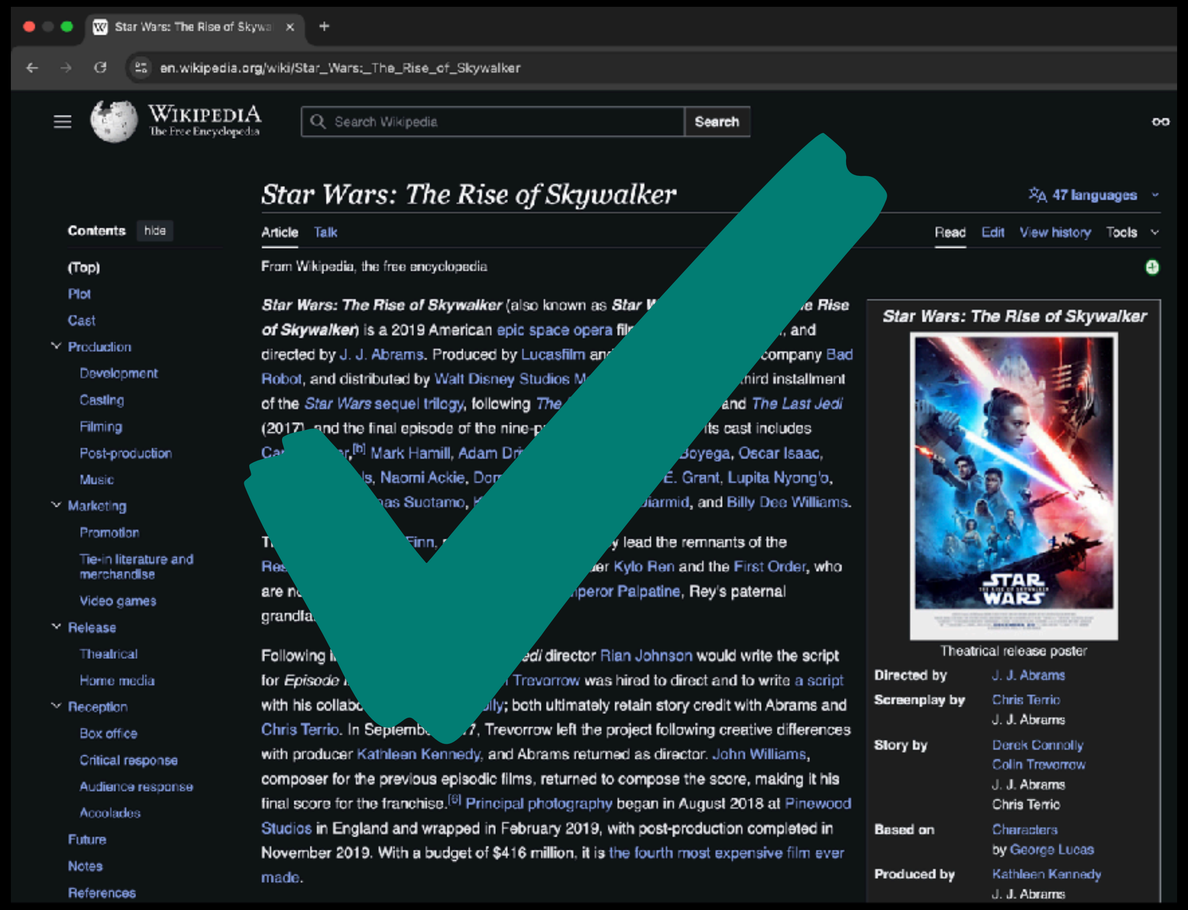


VECTOR SEARCH

```
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],
    }
  }
}
```

Elasticsearch kNN search | Elastic

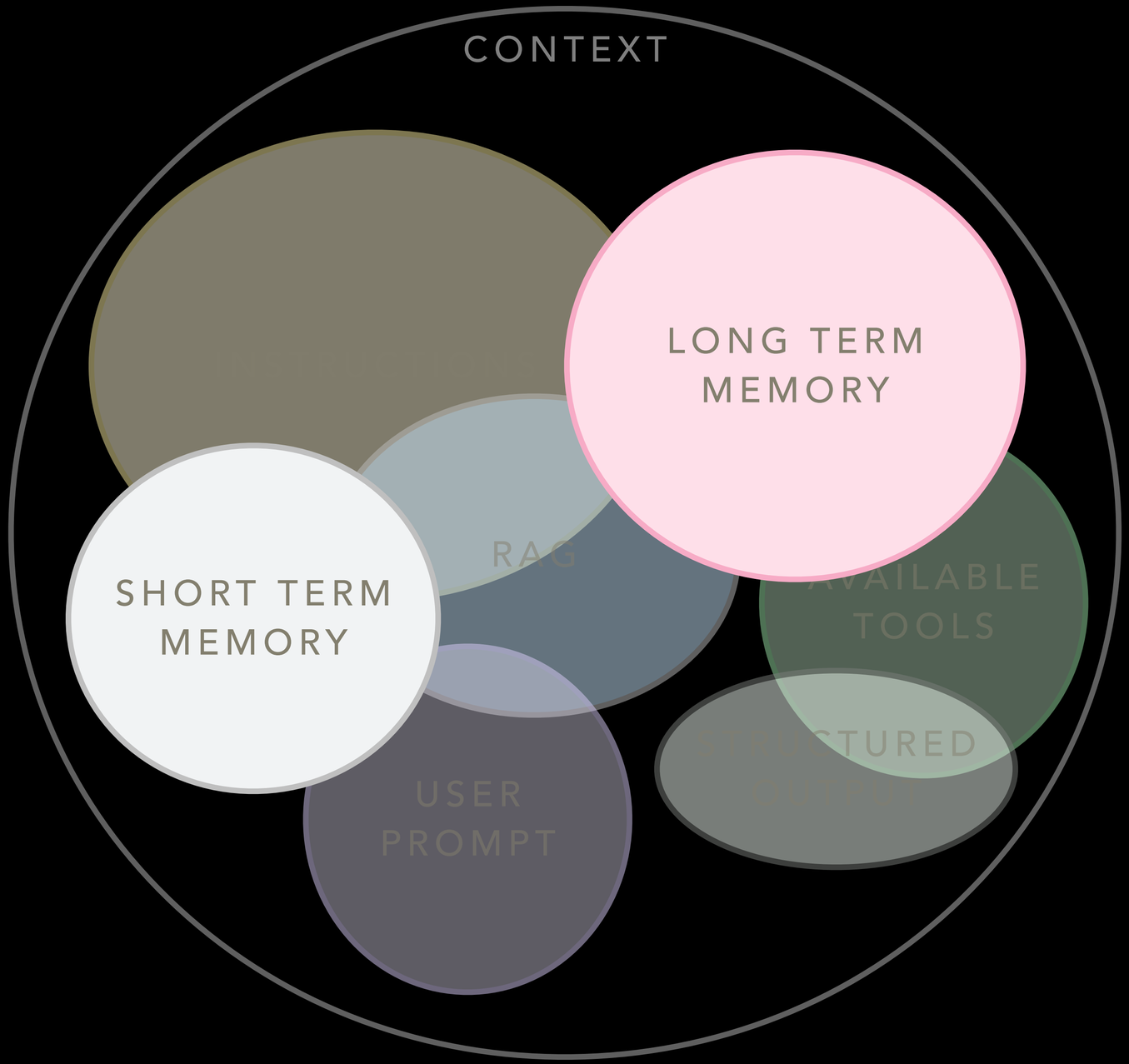
YES!



rise-of-skywalker.json


```
{
  "_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"
  }
}
```

Credit: What is Context Engineering | Data Science Dojo

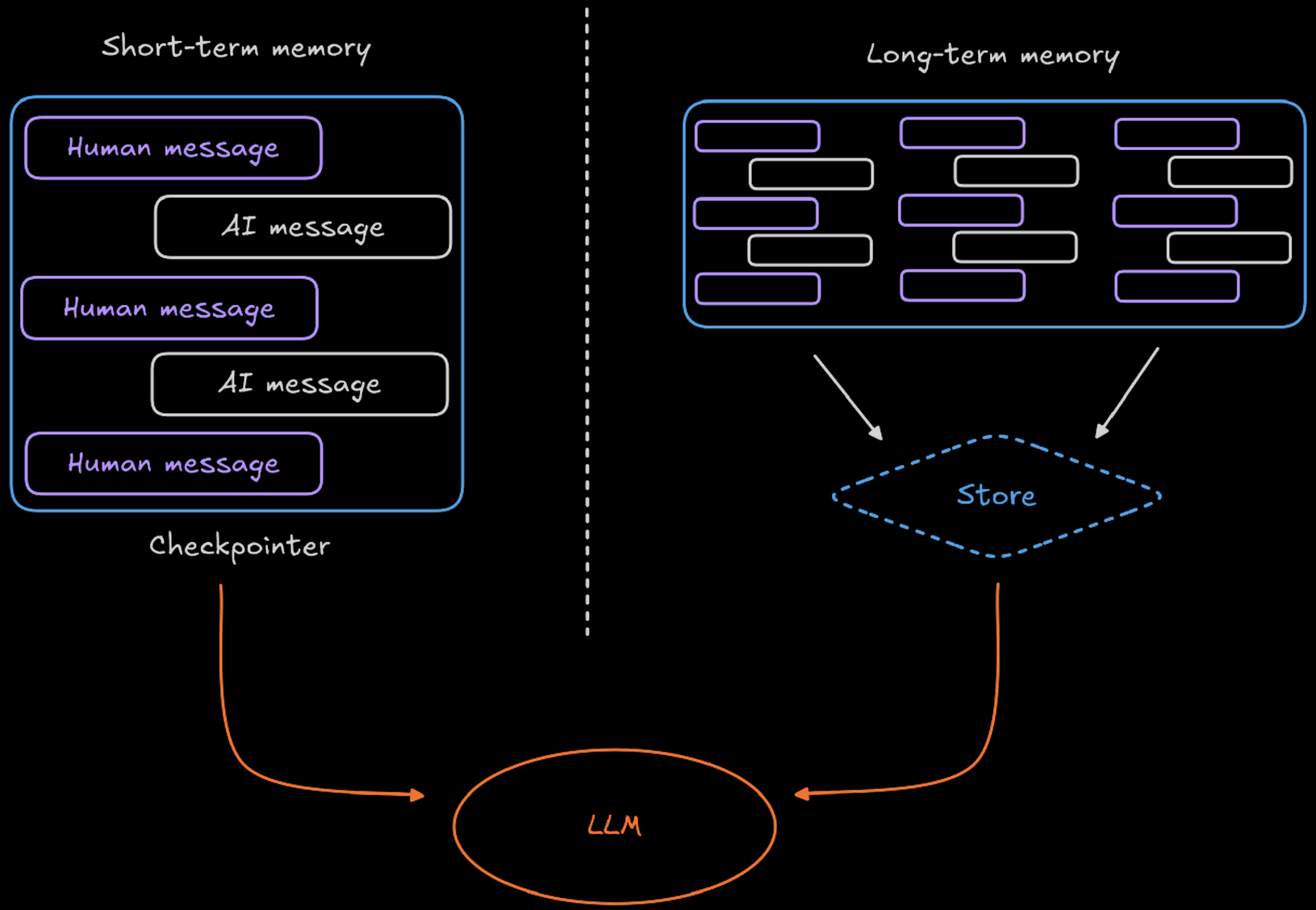


elastic

https://www.elastic.co/search-labs/blog/context-engineering-overview

carlyrichmond.bsky.social

MEMORY


Short-term memory
Human message
AI message
Human message
AI message
Human message
Checkpointer
Long-term memory
Store
LLM

EXAMPLE MEMORY

route.ts

```ts
import { openai } from "@ai-sdk/openai";
import { streamText, stepCountIs, convertToModelMessages } from "ai";
import { NextResponse } from "next/server";

import { weatherTool } from "@/app/ai/weather.tool";
import { fcdoTool } from "@/app/ai/fcdo.tool";
import { flightTool } from "@/app/ai/flights.tool";

import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";

// Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
export const maxDuration = 30;

const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool,
};

// Post request handler
export async function POST(req: Request) {
  const { messages, id } = await req.json();

  // Store current message
  const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
  await persistMessage(messages[lastMessageIndex], id);

  // Get chat history by chat id
  const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
  const allMessages = [...previousMessages, ...messages];

  try {
    const convertedMessages = convertToModelMessages(allMessages);
    const result = streamText({
      model: openai("gpt-4o"),
      system:
        "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." +
        "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
        "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
        "Return an itinerary of sites to see and things to do based on the weather." +
        "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools,
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
    return new NextResponse(
      "Unable to generate a plan. Please try again later!"
    );
  }
}
```

route.ts

```ts
import { openai } from "@ai-sdk/openai";
import { streamText, stepCountIs, convertToModelMessages } from "ai";
import { NextResponse } from "next/server";

import { weatherTool } from "@/app/ai/weather.tool";
import { fcdoTool } from "@/app/ai/fcdo.tool";
import { flightTool } from "@/app/ai/flights.tool";

import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";

// Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
export const maxDuration = 30;

const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool,
};

// Post request handler
export async function POST(req: Request) {
  const { messages, id } = await req.json();

  // Store current message
  const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
  await persistMessage(messages[lastMessageIndex], id);

  // Get chat history by chat id
  const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
  const allMessages = [...previousMessages, ...messages];

  try {
```

```
// Post request handler
export async function POST(req: Request) {
  const { messages, id } = await req.json();

  // Store current message
  const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
  await persistMessage(messages[lastMessageIndex], id);

  // Get chat history by chat id
  const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
  const allMessages = [...previousMessages, ...messages];

  try {
    const convertedMessages = convertToModelMessages(allMessages);
    const result = streamText({
      model: openai("gpt-4o"),
      system:
        "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." +
        "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
        "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
        "Return an itinerary of sites to see and things to do based on the weather." +
        "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools,
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
    return new NextResponse(
      "Unable to generate a plan. Please try again later!"
    );
```

EXAMPLE MEMORY

route.ts

```ts
import { openai } from "@ai-sdk/openai";
import { streamText, stepCountIs, convertToModelMessages } from "ai";
import { NextResponse } from "next/server";

import { weatherTool } from "@/app/ai/weather.tool";
import { fcdoTool } from "@/app/ai/fcdo.tool";
import { flightTool } from "@/app/ai/flights.tool";

import { getSimilarMessages, persistMessage } from "@/app/util/elasticsearch";

// Allow streaming responses up to 30 seconds to address typically longer responses from LLMs
export const maxDuration = 30;

const tools = {
  flights: flightTool,
  weather: weatherTool,
  fcdo: fcdoTool,
};

// Post request handler
export async function POST(req: Request) {
  const { messages, id } = await req.json();

  // Store current message
  const lastMessageIndex = messages.length > 0 ? messages.length - 1 : 0;
  await persistMessage(messages[lastMessageIndex], id);

  // Get chat history by chat id
  const previousMessages = await getSimilarMessages(messages[lastMessageIndex]);
  const allMessages = [...previousMessages, ...messages];

  try {
    const convertedMessages = convertToModelMessages(allMessages);
    const result = streamText({
      model: openai("gpt-4o"),
      system:
        "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." +
        "Use the flight information from tool getFlights only to recommend possible flights in the itinerary." +
        "If there are no flights available generate a sample itinerary and advise them to contact a travel agent." +
        "Return an itinerary of sites to see and things to do based on the weather." +
        "If the FCDO tool warns against travel DO NOT generate an itinerary.",
      messages: convertedMessages,
      stopWhen: stepCountIs(2),
      tools,
    });

    // Return data stream to allow the useChat hook to handle the results as they are streamed through for a better experience
    return result.toUIMessageStreamResponse();
  } catch (e) {
    console.error(e);
    return new NextResponse(
      "Unable to generate a plan. Please try again later!"
    );
  }
}
```

Credit: What is Context Engineering | Data Science Dojo

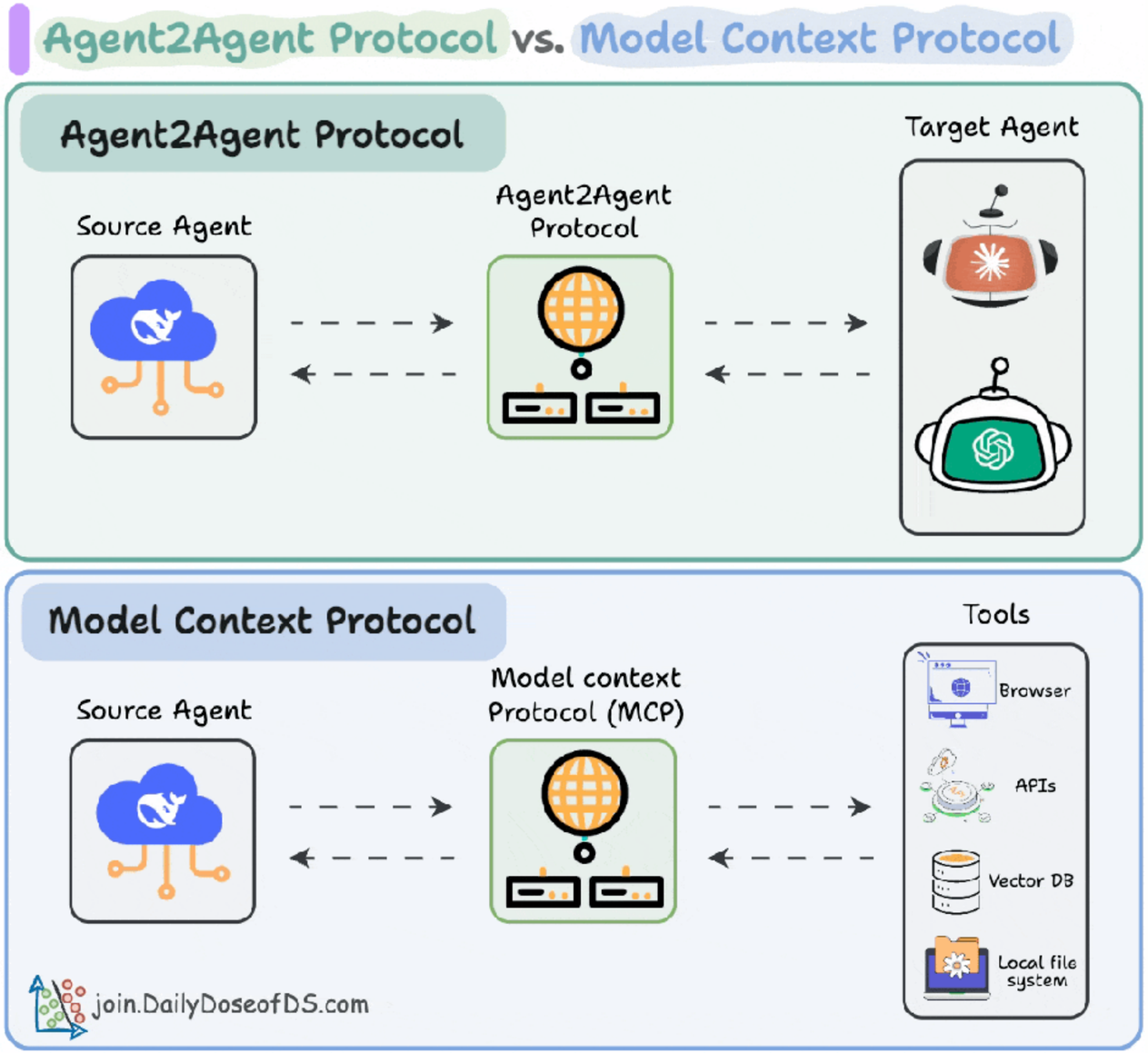
elastic

https://www.elastic.co/search-labs/blog/context-engineering-overview

carlyrichmond.bsky.social

Credit: A Visual Guide to Agent2Agent (A2A) Protocol | Daily Dose of Data Science

MCP AND A2A



EXAMPLE AGENT TOOL

flightTool.ts

```ts
import { tool } from 'ai';
import { z } from 'zod/v3';

import { Client } from '@elastic/elasticsearch';

import { Flight } from '../model/flight.model';

const index: string = "upcoming-flight-data";
const client: Client = new Client({
  node: process.env.ELASTIC_ENDPOINT,
  auth: {
    apiKey: process.env.ELASTIC_API_KEY || '',
  },
});

function extractFlights(response: { hits?: { hits?: { _source: Flight }[] } }): (Flight | undefined)[] {
    if (response.hits && Array.isArray(response.hits.hits)) {
        return response.hits.hits.map((hit: { _source: Flight; }) => hit._source);
    }
    return [];
}

export const flightTool = tool({
  description:
    "Get flight information for a given destination from Elasticsearch, both outbound and return journeys",
  inputSchema: z.object({
    destination: z.string().describe("The destination we are flying to"),
    origin: z.string().describe("The origin we are flying from").default("London"),
  }),
  execute: async function ({ destination, origin }) {
    try {
      const responses = await client.msearch({
        searches: [
          { index: index },
          {
            query: {
              bool: {
                must: [
                  {
                    match: {
                      origin: origin,
                    },
                  },
                  {
                    match: {
                      destination: destination,
                    },
                  },
                ],
              },
            },
          },

          // Return leg omitted
        ],
      });

      if (!responses.responses.length < 2) {
        throw new Error("Unable to obtain flight data");
      }

      return {
        outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight }[] } }),
        inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight }[] } })
      };
    } catch (e) {
      console.error(e);
      return {
        message: "Unable to obtain flight information"
      };
    }
  },
});
```

elastic

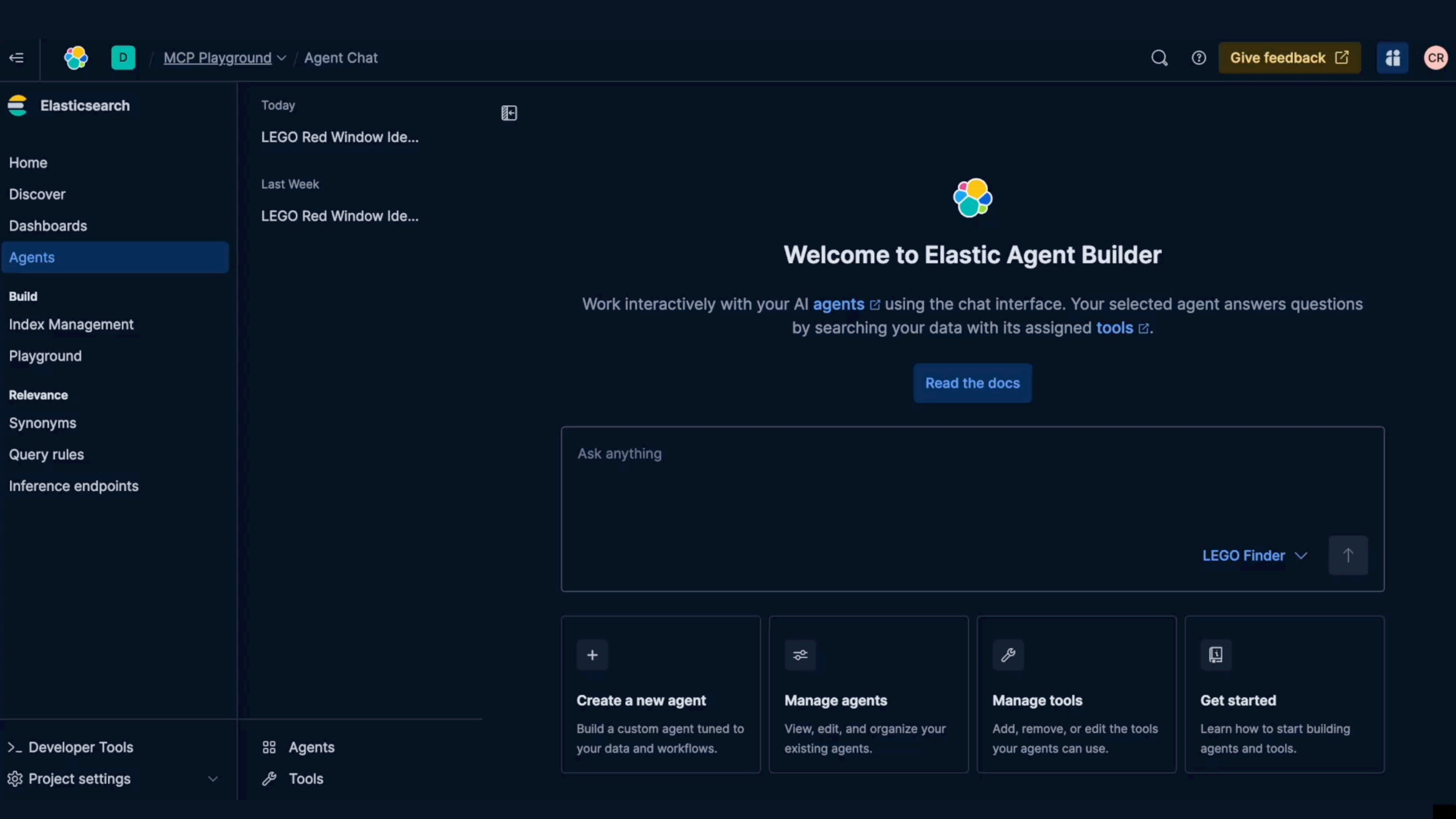
https://www.elastic.co/search-labs/blog/ai-agents-ai-sdk-elasticsearch

```
export const flightTool = tool({
  description:
    "Get flight information for a given destination from Elasticsearch, both outbound and return journeys",
  inputSchema: z.object({
    destination: z.string().describe("The destination we are flying to"),
    origin: z.string().describe("The origin we are flying from").default("London"),
  }),
  execute: async function ({ destination, origin }) {
    try {
      const responses = await client.msearch({
        searches: [
          { index: index },
          {
            query: {
              bool: {
                must: [
                  {
                    match: {
                      origin: origin,
                    },
                  },
                  {
                    match: {
                      destination: destination,
                    },
                  },
                ],
              },
            },
```

```
                    match: {
                      destination: destination,
                    },
                  },
                ],
              },
            },
          },

          // Return leg omitted
        ],
      });

      if (responses.responses.length < 2) {
        throw new Error("Unable to obtain flight data");
      }

      return {
        outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight }[] } }),
        inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight }[] } })
      };
    } catch (e) {
      console.error(e);
      return {
        message: "Unable to obtain flight information"
      };
    }
  },
});
```

flight.tool.ts

```typescript
import { tool } from 'ai';
import { z } from 'zod/v3';

import { Client } from '@elastic/elasticsearch';

import { Flight } from '../model/flight.model';

const index: string = "upcoming-flight-data";
const client: Client = new Client({
  node: process.env.ELASTIC_ENDPOINT,
  auth: {
    apiKey: process.env.ELASTIC_API_KEY || "",
  },
});

function extractFlights(response: { hits?: { hits?: { _source: Flight }[] } }): (Flight | undefined)[] {
    if (response.hits && Array.isArray(response.hits.hits)) {
        return response.hits.hits.map((hit: { _source: Flight; }) => hit._source);
    }
    return [];
}

export const flightTool = tool({
  description:
    "Get flight information for a given destination from Elasticsearch, both outbound and return journeys",
  inputSchema: z.object({
    destination: z.string().describe("The destination we are flying to"),
    origin: z.string().describe("The origin we are flying from").default("London"),
  }),
  execute: async function ({ destination, origin }) {
    try {
      const responses = await client.msearch({
        searches: [
          { index: index },
          {
            query: {
              bool: {
                must: [
                  {
                    match: {
                      origin: origin,
                    },
                  },
                  {
                    match: {
                      destination: destination,
                    },
                  },
                ],
              },
            },
          },

          // Return leg omitted
        ],
      });

      if (responses.responses.length < 2) {
        throw new Error("Unable to obtain flight data");
      }

      return {
        outbound: extractFlights(responses.responses[0] as { hits?: { hits?: { _source: Flight }[] } }),
        inbound: extractFlights(responses.responses[1] as { hits?: { hits?: { _source: Flight }[] } })
      };
    } catch (e) {
      console.error(e);
      return {
        message: "Unable to obtain flight information"
      };
    }
  },
});
```


D
MCP Playground
Agent Chat
Give feedback
CR
Elasticsearch
Home
Discover
Dashboards
Agents
Build
Index Management
Playground
Relevance
Synonyms
Query rules
Inference endpoints
Today
LEGO Red Window Ide...
Last Week
LEGO Red Window Ide...
Developer Tools
Project settings
Agents
Tools
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.

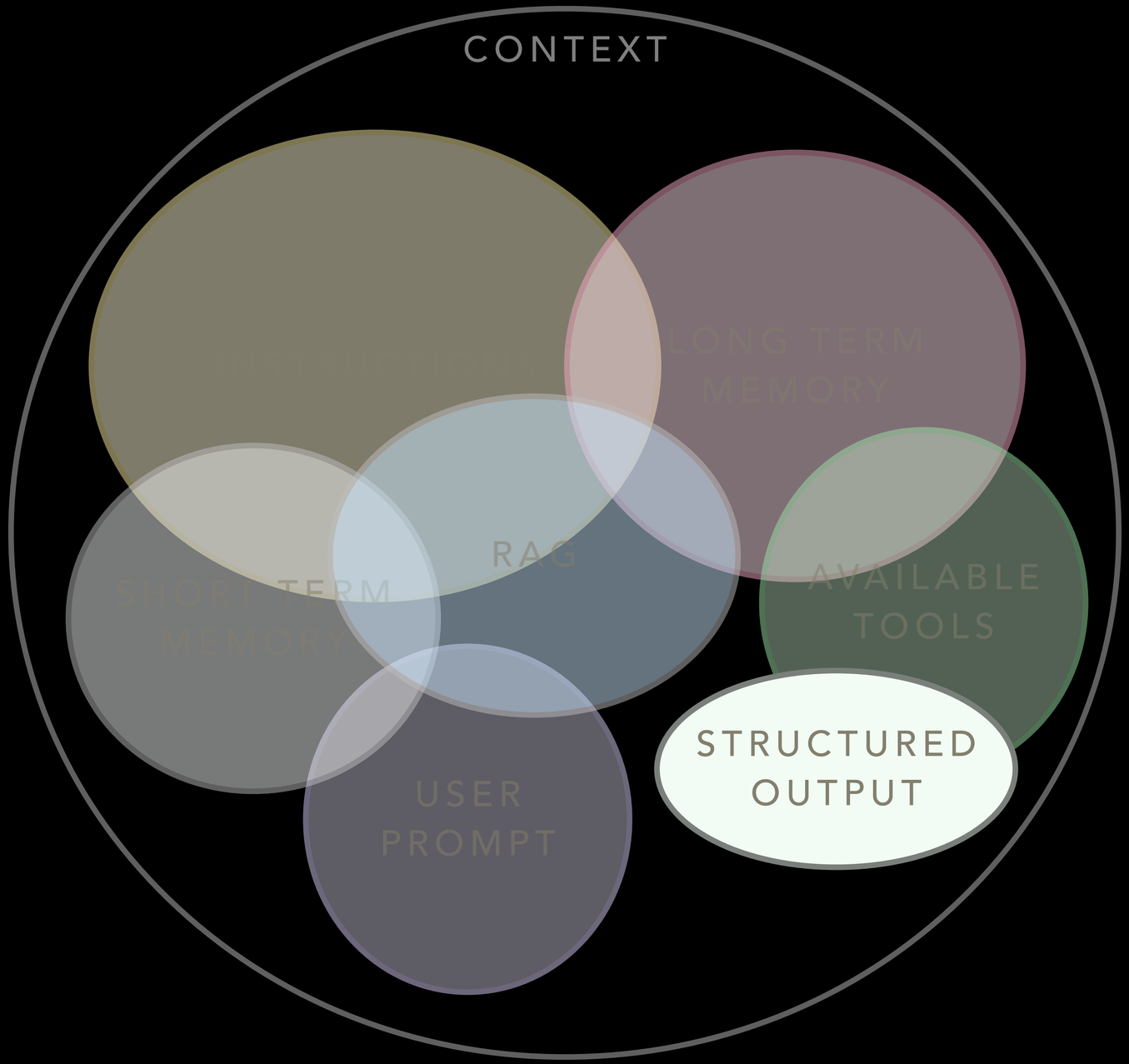
RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation

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

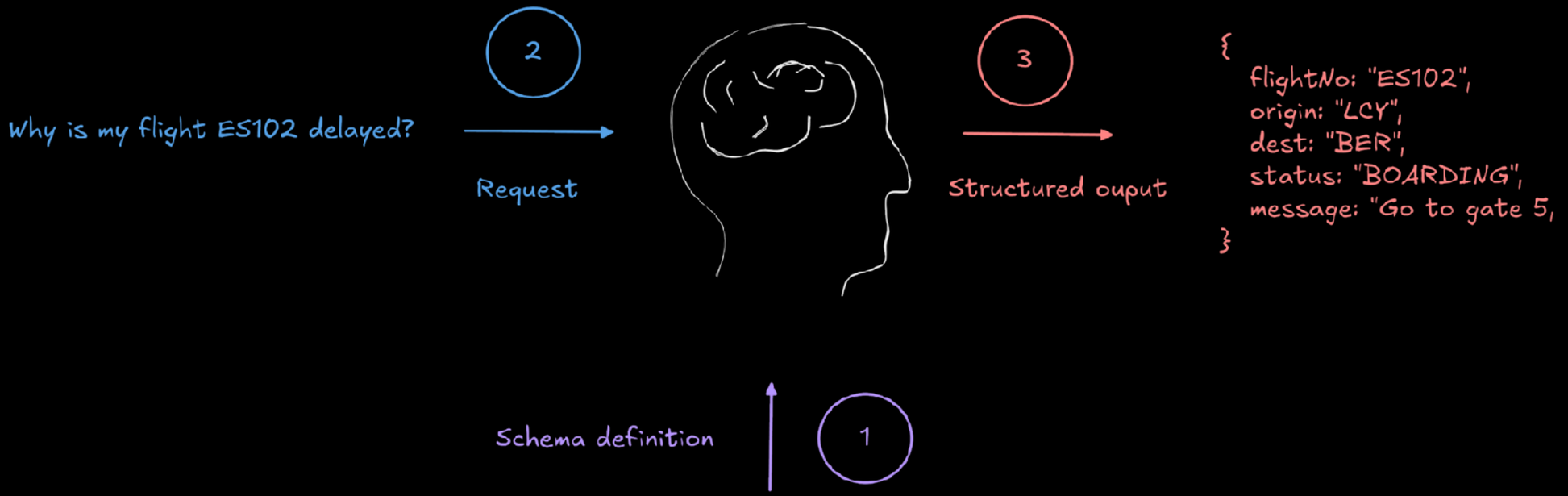
Abstract. Large language models (LLMs) struggle to effectively utilize a growing number of external tools, such as those defined by the Model Context Protocol (MCP)[1], due to prompt bloat and selection complexity. We introduce RAG-MCP, a Retrieval-Augmented Generation framework that overcomes this challenge 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.

Credit: What is Context Engineering | Data Science Dojo



Source: Memory | Langchain

STRUCTURED OUTPUTS




```
{
  flightNo: { type: string, description: "Flight number",
  origin: { type: string, description: "Origin airport 3-letter code",
  dest: { type: string, description: "Destination airport 3-letter code",
  status: { type: enum(["ON TIME", "CANCELLED", "DELAYED", "BOARDING", "DEPARTED"]), description: "Flight status" }
  message: { type: string, description: "Passenger message" }
}
```

EXAMPLE

generate-itinerary.ts

```ts
import { generateObject } from 'ai';
import { z } from 'zod';

const { object } = await generateObject({
  model: 'openai/gpt-4.1',
  schemaName: 'Travel Itinerary',
  schemaDescription: 'Sample travel itinerary for a trip',
  schema: z.object({
    title: z.string(),
    location: z.string(),
    hotel: z.object({
      name: z.string(), roomType: z.string(), amount: z.number(), checkin: z.iso.date(), checkout: z.iso.date()
    }),
    flights: z.array(
      z.object({
        carrier: z.string(), flightNo: z.string().max(8), origin: z.string(), destination: z.string(), date: z.iso.datetime()
      })
    ),
    excursions: z.array(z.object({ name: z.string(), amount: z.string(), date: z.iso.datetime()}))
  }),
  prompt: 'Generate a travel itinerary based on the specified location',
});
```

EXAMPLE TOOL OUTPUT SCHEMA

weather.tool.ts

```ts
import { tool } from 'ai';
import { z } from 'zod/v3';

import { WeatherResponse } from '../model/weather.model';

export const weatherTool = tool({
  description:
  'Display the weather for a holiday location',
  inputSchema: z.object({
    location: z.string().describe('The location to get the weather for')
  }),
  outputSchema: z.object({
    location: z.string().describe('The location to get the weather for'),
    condition: z.string().optional().describe('The current weather condition'),
    condition_image: z.string().optional().describe('An image representing the current weather condition'),
    temperature: z.number().optional().describe('The current temperature in Celsius'),
    feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
    humidity: z.number().optional().describe('The current humidity percentage'),
    message: z.string().optional().describe('An optional message, e.g. for errors')
  }),
  execute: async ({ location }: { location: string}) => {
    // While a historical forecast may be better, this example gets the next 3 days
    const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;

    try {
      const response = await fetch(url);
      const weather : WeatherResponse = await response.json();
      return {
        location: location,
        condition: weather.current.condition.text,
        condition_image: weather.current.condition.icon,
        temperature: Math.round(weather.current.temp_c),
        feels_like_temperature: Math.round(weather.current.feelslike_c),
        humidity: weather.current.humidity
      };
    } catch(e) {
      console.error(e);
      return {
        message: 'Unable to obtain weather information',
        location: location
      };
    }
  }
});
```

weather.tool.ts

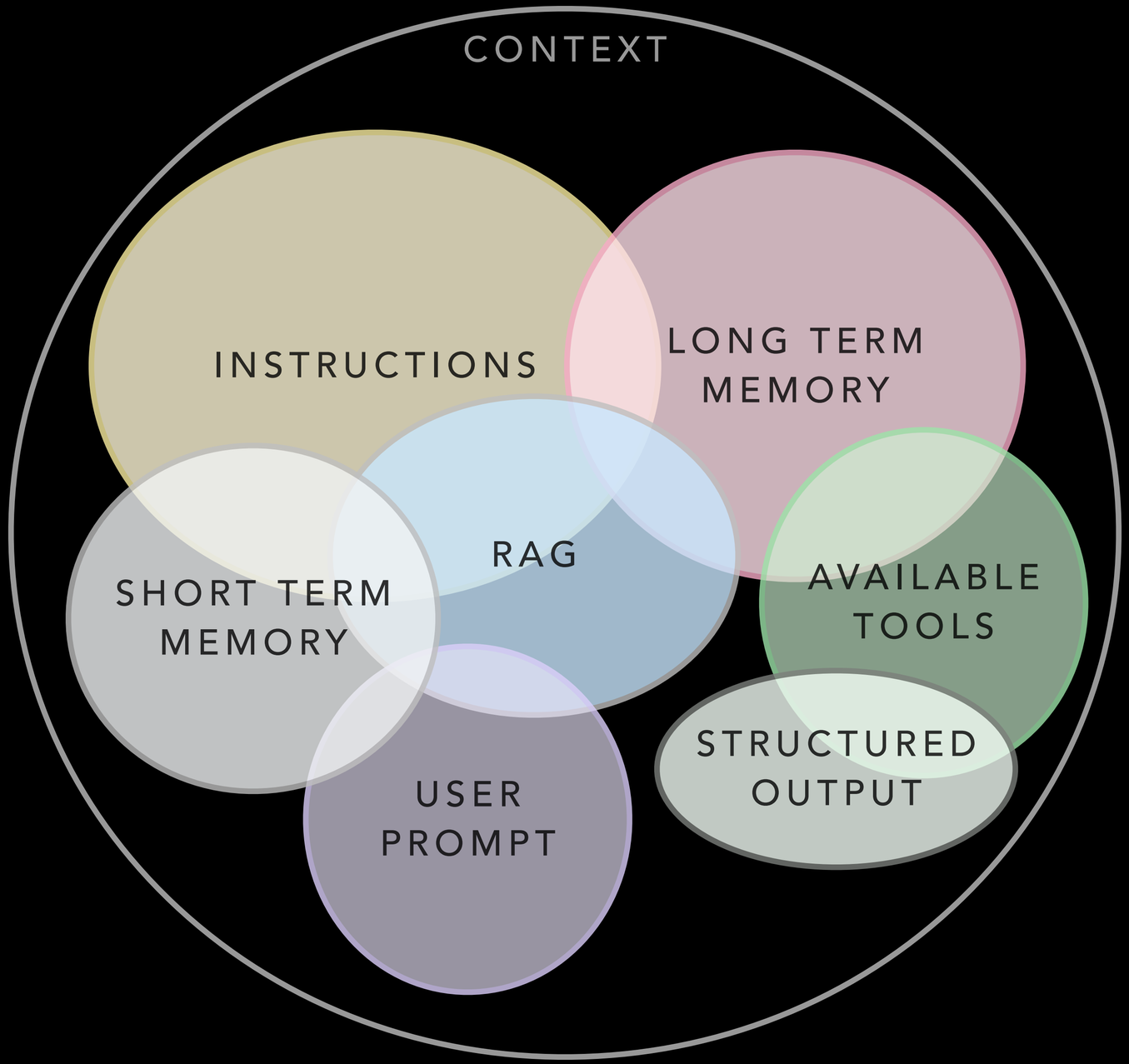
```ts
import { tool } from 'ai';
import { z } from 'zod/v3';

import { WeatherResponse } from '../model/weather.model';

export const weatherTool = tool({
  description:
  'Display the weather for a holiday location',
  inputSchema: z.object({
    location: z.string().describe('The location to get the weather for')
  }),
  outputSchema: z.object({
    location: z.string().describe('The location to get the weather for'),
    condition: z.string().optional().describe('The current weather condition'),
    condition_image: z.string().optional().describe('An image representing the current weather condition'),
    temperature: z.number().optional().describe('The current temperature in Celsius'),
    feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
    humidity: z.number().optional().describe('The current humidity percentage'),
    message: z.string().optional().describe('An optional message, e.g. for errors')
  }),
  execute: async ({ location }: { location: string}) => {
    // While a historical forecast may be better, this example gets the next 3 days
    const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;

    try {
      const response = await fetch(url);
      const weather : WeatherResponse = await response.json();
      return {
        location: location,
```

EXAMPLE TOOL OUTPUT SCHEMA

weather.tool.ts

```ts
import { tool } from 'ai';
import { z } from 'zod/v3';

import { WeatherResponse } from '../model/weather.model';

export const weatherTool = tool({
  description:
  'Display the weather for a holiday location',
  inputSchema: z.object({
    location: z.string().describe('The location to get the weather for')
  }),
  outputSchema: z.object({
    location: z.string().describe('The location to get the weather for'),
    condition: z.string().optional().describe('The current weather condition'),
    condition_image: z.string().optional().describe('An image representing the current weather condition'),
    temperature: z.number().optional().describe('The current temperature in Celsius'),
    feels_like_temperature: z.number().optional().describe('What the temperature feels like in Celsius'),
    humidity: z.number().optional().describe('The current humidity percentage'),
    message: z.string().optional().describe('An optional message, e.g. for errors')
  }),
  execute: async ({ location }: { location: string}) => {
    // While a historical forecast may be better, this example gets the next 3 days
    const url = `https://api.weatherapi.com/v1/forecast.json?q=${location}&days=3&key=${process.env.WEATHER_API_KEY}`;

    try {
      const response = await fetch(url);
      const weather : WeatherResponse = await response.json();
      return {
        location: location,
        condition: weather.current.condition.text,
        condition_image: weather.current.condition.icon,
        temperature: Math.round(weather.current.temp_c),
        feels_like_temperature: Math.round(weather.current.feelslike_c),
        humidity: weather.current.humidity
      };
    } catch(e) {
      console.error(e);
      return {
        message: 'Unable to obtain weather information',
        location: location
      };
    }
  }
});
```

Credit: What is Context Engineering | Data Science Dojo



elastic

https://www.elastic.co/search-labs/blog/context-engineering-overview

Source: Priscilla Du Preez 🇨🇦 on Unsplash

SCAN ME

carlyrichmond.bsky.social

elastic