Source: Vanburn Gonsalves on Unsplash

CALLING "HOUSE" ON YOUR AI BUZZWORD BINGO CARD

CARLY RICHMOND

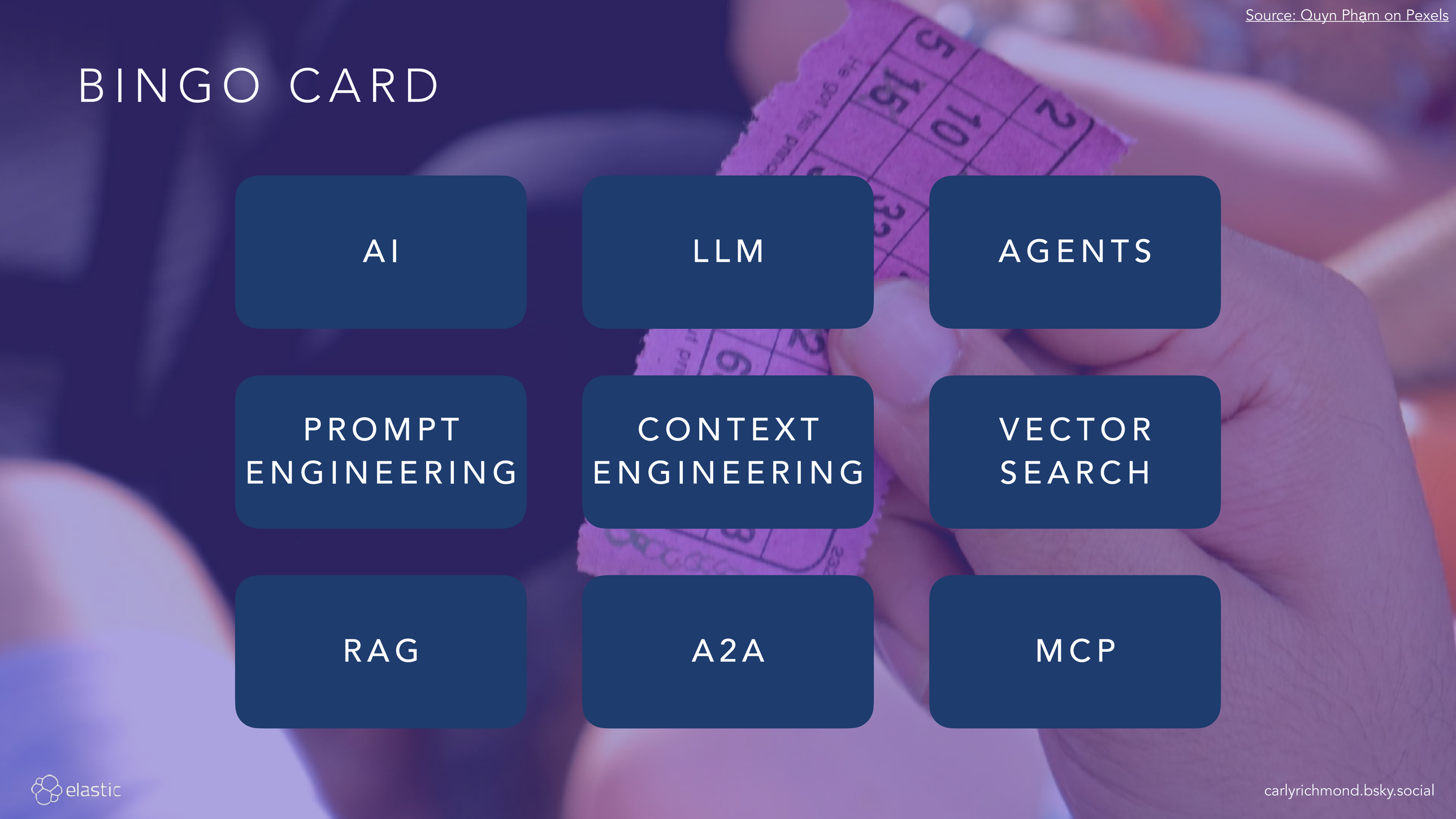
ABOUT ME

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

SCAN ME

carlyrichmond.bsky.social

Source: Quyn Phạm on Pexels



Source: Quyn Phạm on Pexels

BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

"**Artificial intelligence (AI)** is the capability of computational systems to perform tasks typically associated with human intelligence, such as learning, reasoning, problem-solving, perception, and decision-making."

–ARTIFICIAL INTELLIGENCE (AI), WIKIPEDIA

Source: Geoffrey Moffett on Unsplash

A. M. Turing (1950) Computing Machinery and Intelligence. *Mind 49:* 433-460.

COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

1. The Imitation Game

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

"**Artificial General Intelligence (AGI)**—sometimes called **human-level intelligence AI**—is a type of artificial intelligence that would match or surpass human capabilities across virtually all cognitive tasks."

–ARTIFICIAL GENERAL INTELLIGENCE (AGI), WIKIPEDIA

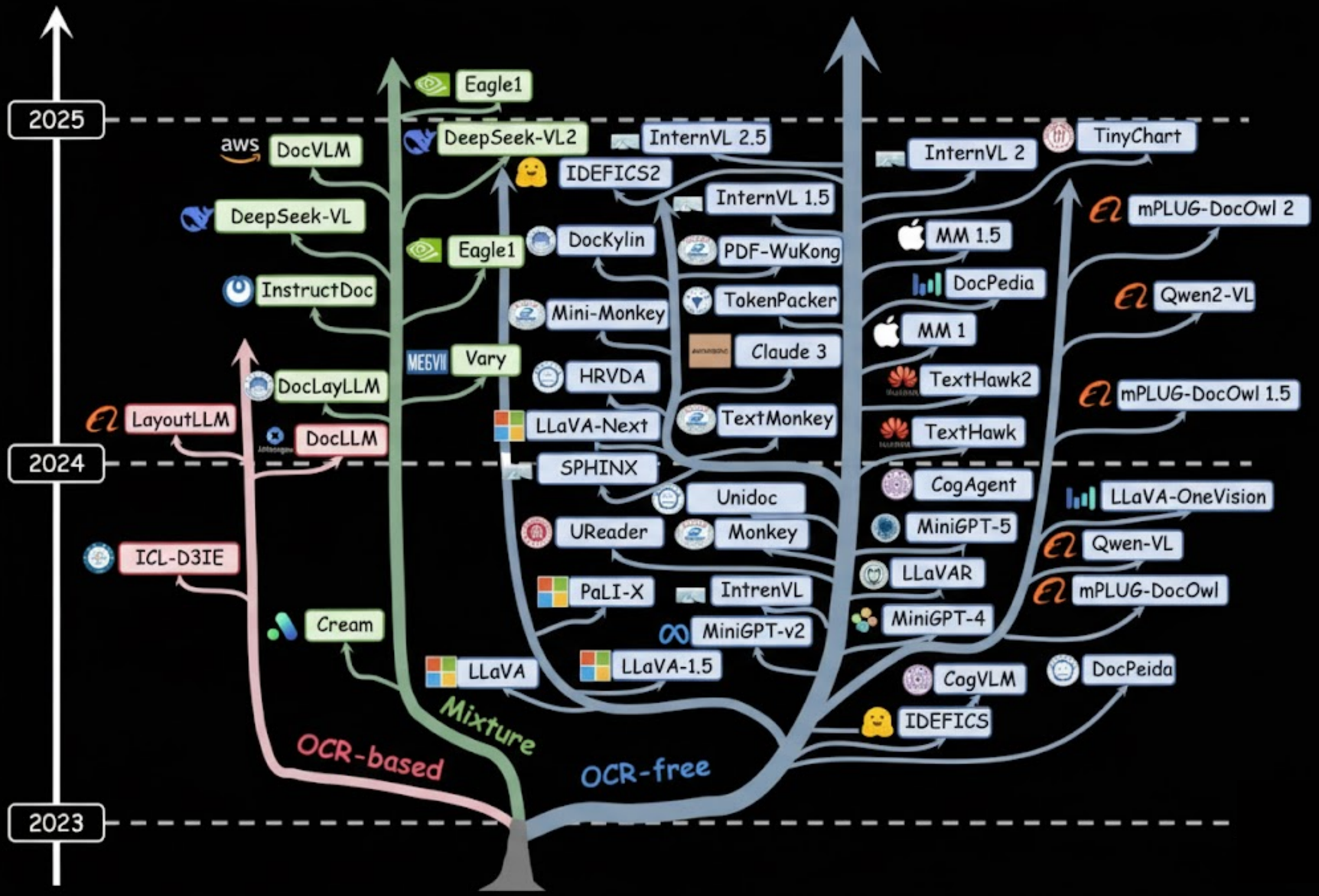
Source: Quyn Phạm on Pexels

BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

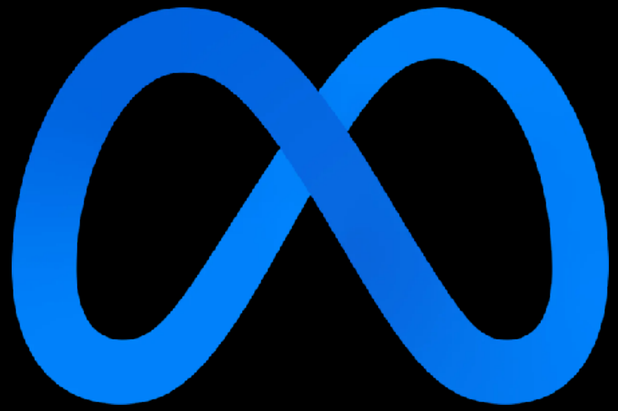
Source: Quyn Phạm on Pexels

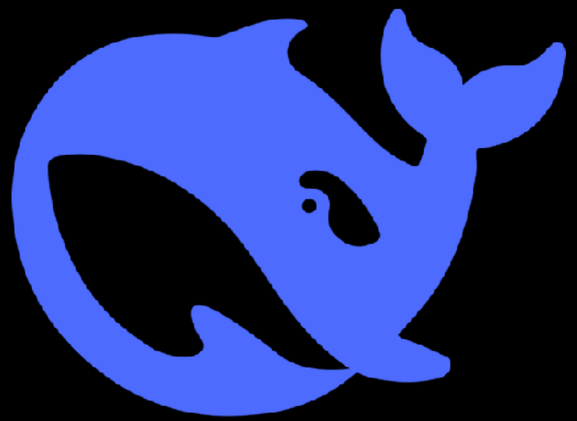
Source: Multimodal Large Language Models for Text-rich Image Understanding: A Comprehensive Review | Fu et al.



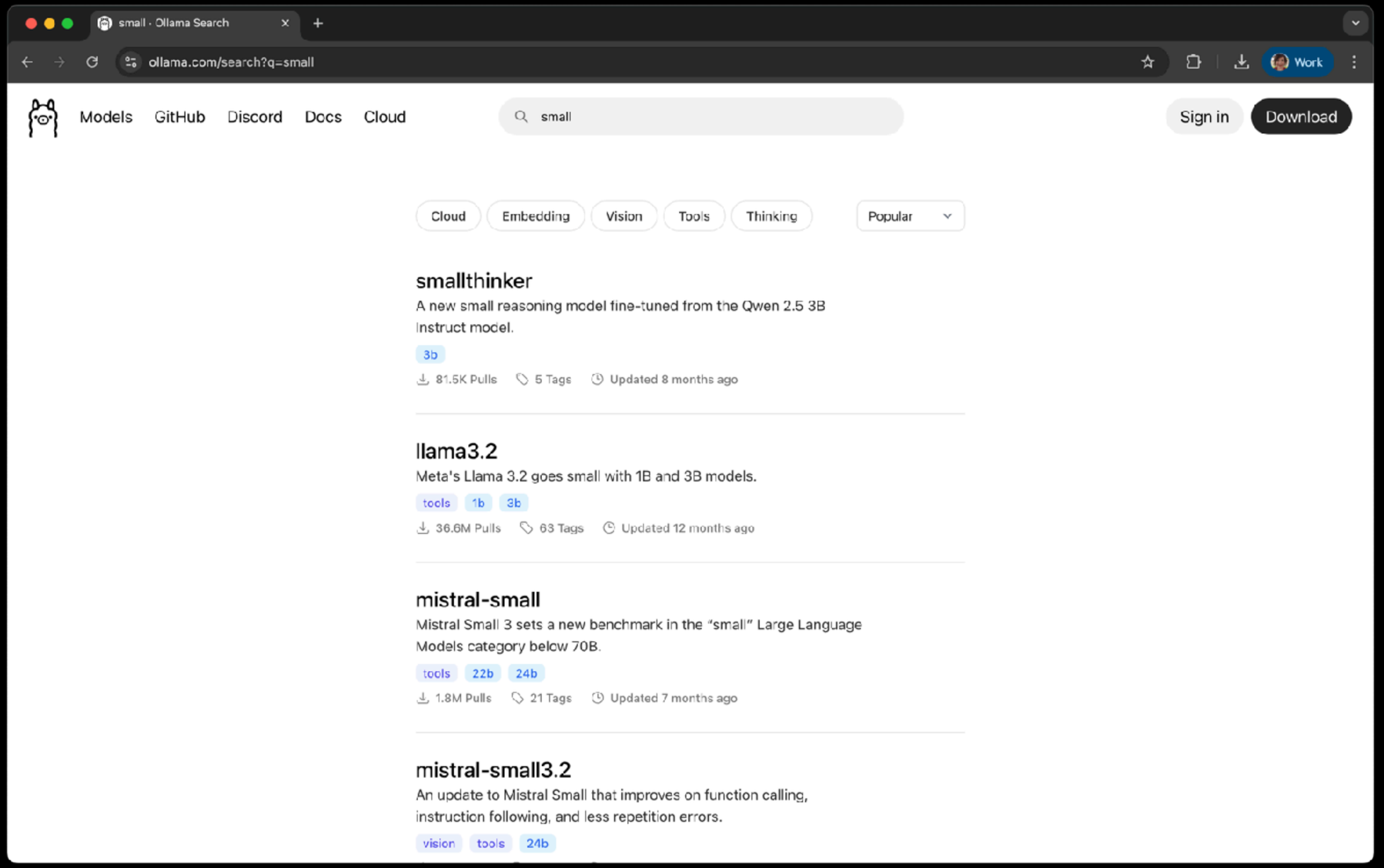
FOUNDATIONAL MODELS

Claude



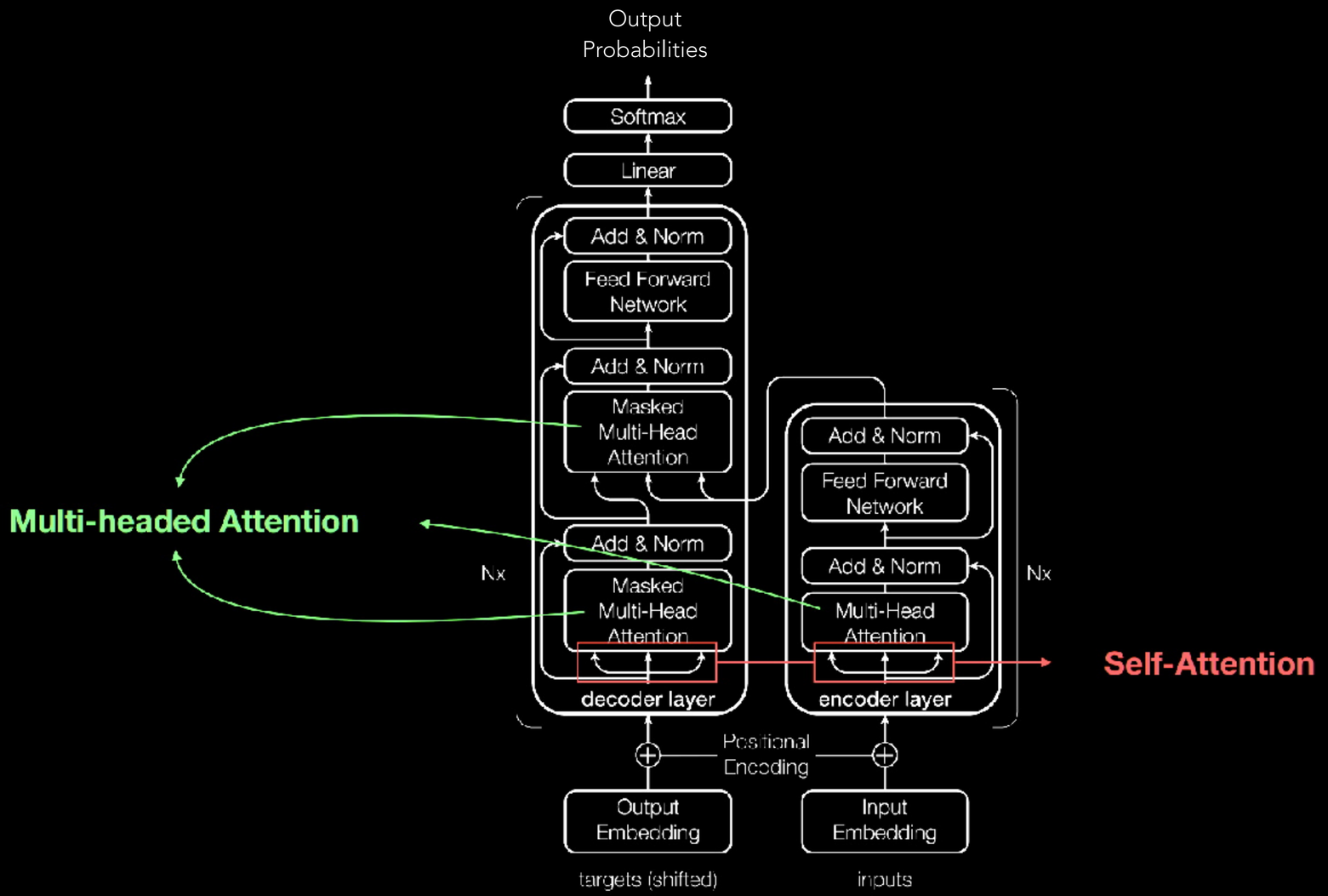


SMALL LANGUAGE MODELS



Credit: What is a Transformer Model | NVIDIA

WHAT IS A TRANSFORMER?



“If only everything in life was as reliable as a…”

“If only everything in life was as reliable as a Volkswagen.”

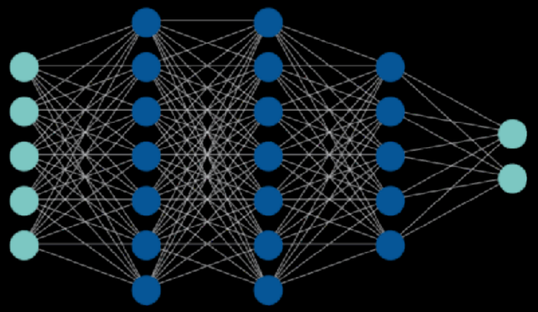
–VOLKSWAGEN UK ADVERTISEMENT CAMPAIGN, 1980S

Source: Geoffrey Moffett on Unsplash

INSIDE AN LLM

If only everything in life was as reliable as a

Input



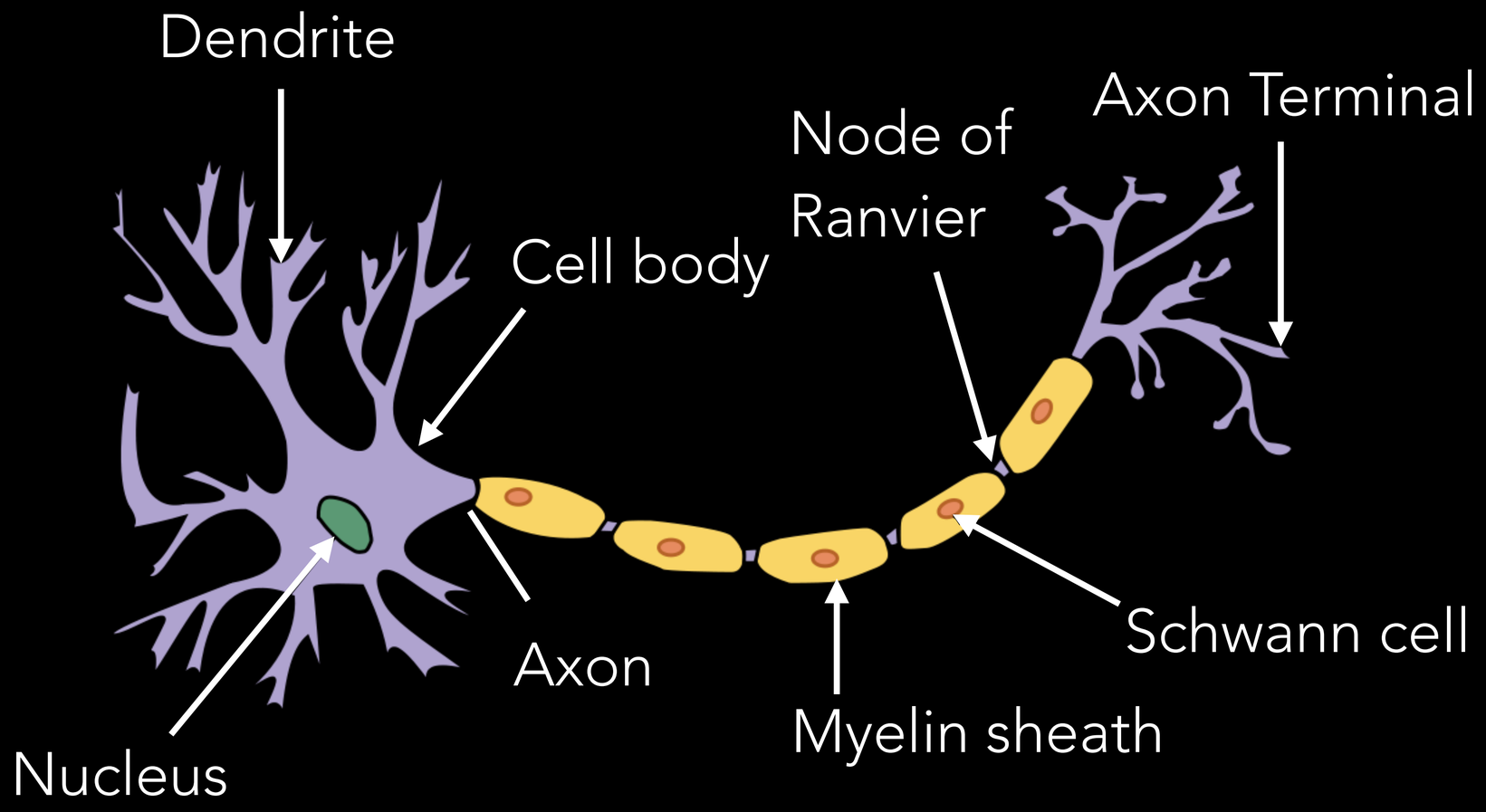
Neural Network (LLM)

	WORD	PROBABILITY
0	Toyota	0.218
1	Dog	0.135
2	Volkswagen	0.892
3	Leaky Teapot	0.0001
4	BMW	0.023

Output

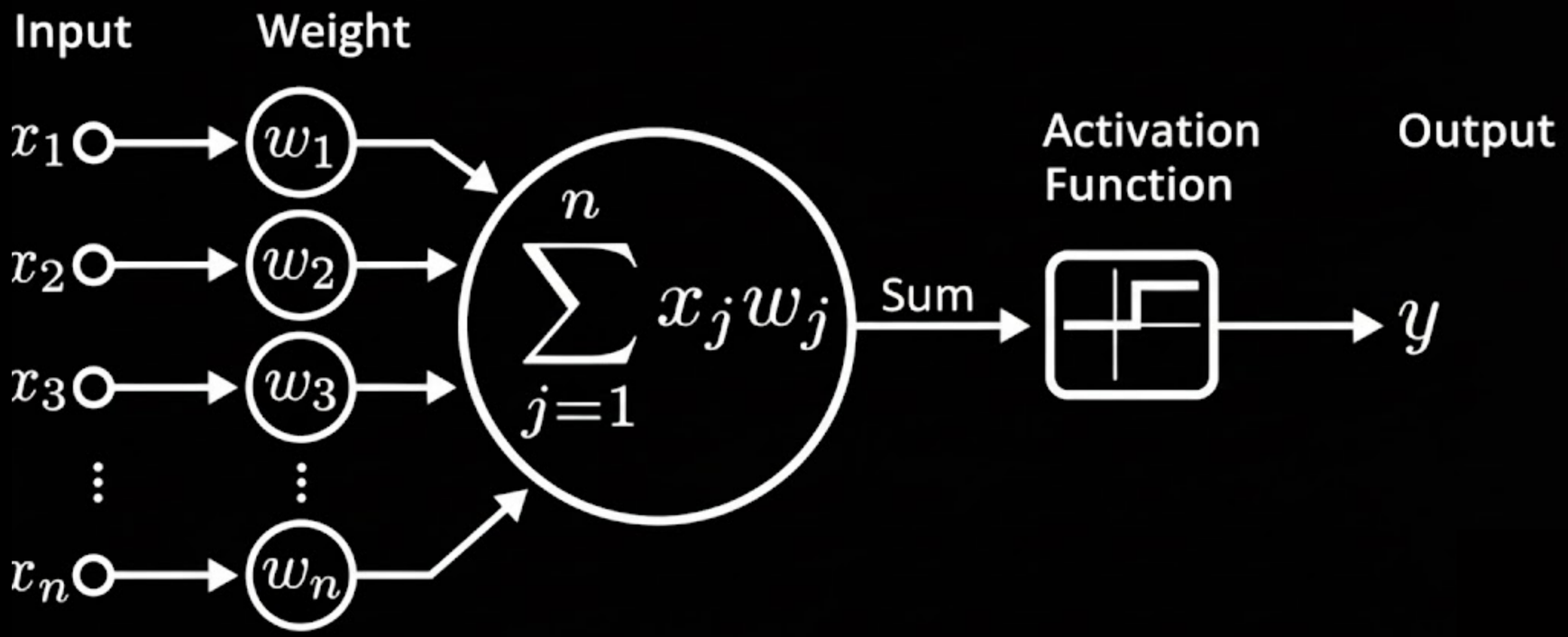
Credit: Understanding Neurons in Deep Learning | Nick McCullum

WHAT IS A NEURON?



Credit: Understanding Neurons in Deep Learning | Nick McCullum

WHAT IS AN ARTIFICIAL NEURON?



Credit: llm-stats.com

PARAMETERS

Model	License	Parameters (B)	Context	Input $/M	Output $/M	MMLU	HumanEval	Multimodal	Knowledge Cutoff
DeepSeek-R1	Open	671	131,072	$0.55	$2.19	-	-	✗	-
DeepSeek-R1-0528	Open	671	131,072	$0.50	$2.15	-	-	✗	-
DeepSeek R1 Zero	Open	671	0	-	-	-	-	✗	-
DeepSeek-V3	Open	671	131,072	$0.27	$1.10	88.5%	-	✗	-
DeepSeek-V3 0324	Open	671	163,840	$0.28	$1.14	-	-	✗	-
DeepSeek-V3.1	Open	671	163,840	$0.27	$1.00	-	-	✗	-
Llama 3.1 405B Instruct	Open	405	128,000	$0.89	$0.89	87.3%	89.0%	✗	-
Llama 4 Maverick	Open	400	1,000,000	$0.17	$0.60	85.5%	-	✓	-
DeepSeek-V2.5	Open	236	8,192	$0.14	$0.28	80.4%	89.0%	✗	-
Qwen3 235B A22B	Open	235	128,000	$0.10	$0.10	87.8%	-	✗	-
Qwen3-235B-A22B-Instruct-2507	Open	235	131,072	$0.15	$0.80	-	-	✗	-
Qwen3-235B-A22B-Thinking-2507	Open	235	256,000	$0.30	$3.00	-	-	✗	-
Pixtral Large	Open	124	128,000	$2.00	$6.00	-	-	✓	-
Mistral Large 2	Open	123	128,000	$2.00	$6.00	84.0%	92.0%	✗	-
GPT OSS 120B	Open	116.8	131,072	$0.09	$0.45	-	-	✗	-
Llama 4 Scout	Open	109	10,000,000	$0.08	$0.30	79.6%	-	✓	-
Llama 3.2 90B Instruct	Open	90	128,000	$0.35	$0.40	86.0%	-	✓	-
Qwen3-Next-80B-A3B-Instruct	Open	80	65,536	$0.15	$1.50	-	-	✗	-
Qwen3-Next-80B-A3B-Thinking	Open	80	65,536	$0.15	$1.50	-	-	✗	-
QvQ-72B-Preview	Open	73.4	0	-	-	-	-	✓	-
Qwen2-VL-72B-Instruct	Open	73.4	0	-	-	-	-	✓	2023-06-30

TEMPERATURE

- A parameter that controls the randomness of the generated text ranging from 0 to 2
- Lower temperatures, like 0.2, lead to more deterministic outputs
- Higher temperatures, up to 2.0, result in more creative and random responses

Source: Quyn Phạm on Pexels

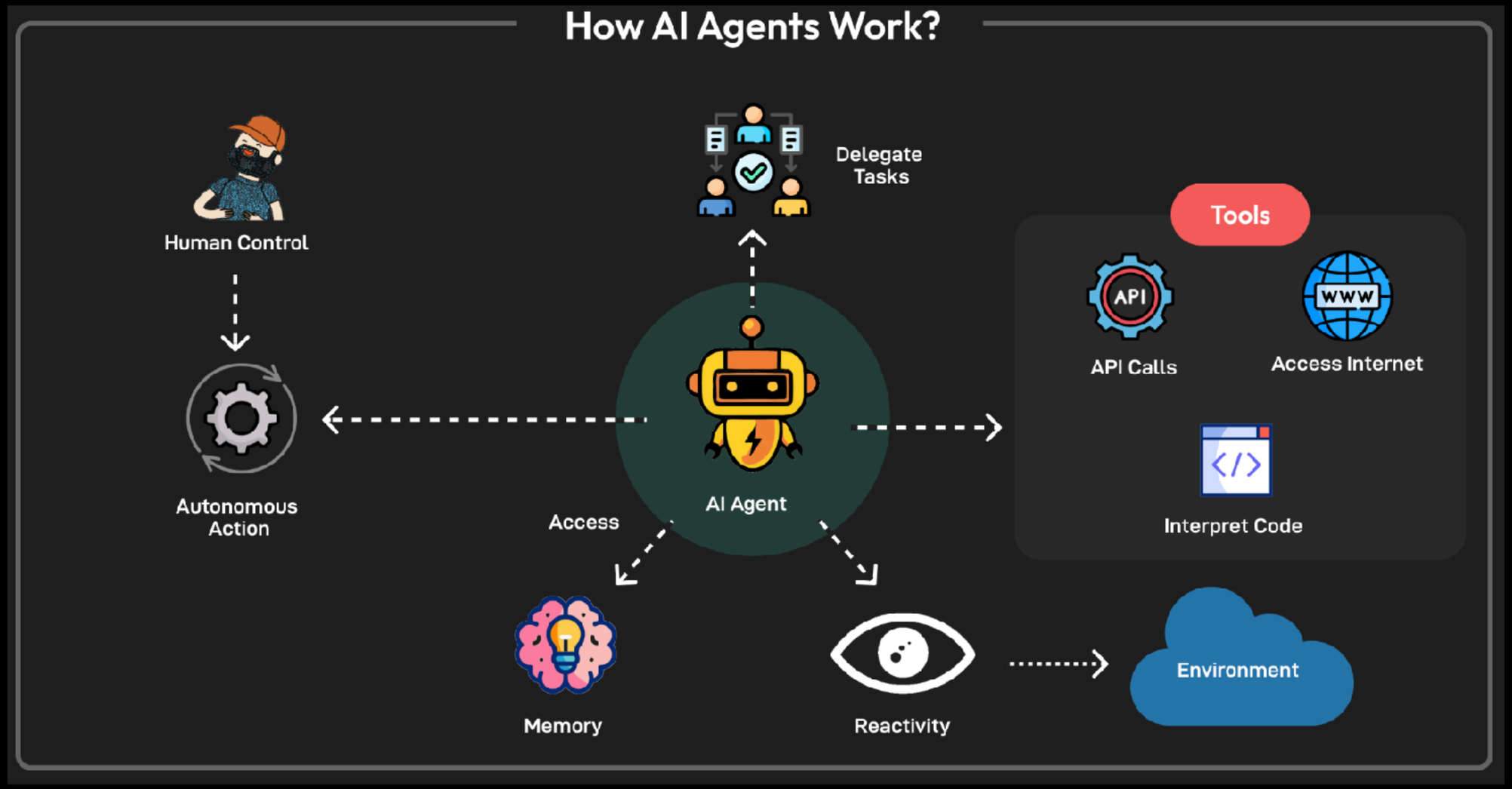
BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

Credit: What is an AI Agent? Byte Byte Go

WHAT IS AN AGENT?



Source: What are AI Agents? A Complete Guide For Beginners | igmGuru

Types of AI Agents

Model-based Reflex Agents

Learning Agents

Goal-based Agents

Utility-based Agents

Simple Reflex Agents

Credit: Incident Summaries using LLMs | Karl Stoney

SIMPLE AGENT

index.ts

```ts
import { Ai } from '@cloudflare/ai'
import { Hono } from 'hono'

export type Bindings = {
  AI: Ai
}

const app = new Hono<{ Bindings: Bindings }>()

app.post('/ml/question', async (c) => {
  const { text } = await c.req.json()
  if (!text) {
    return new Response('No text provided', { status: 400 })
  }

  const ai = new Ai(c.env.AI)
  const prompt: string[] = [
    'My name is Skippr and I help with Software Engineering problems.',
    'I do not ask questions, I only respond to prompts.  I never suggest follows up questions.',
    'My answers are concise and to the point, I do not guess.'
  ]

  const messages = [
    ...prompt.map((item) => ({ role: 'assistant', content: item })),
    { role: 'user', content: text }
  ]
  const output = await ai.run('@cf/meta/llama-2-7b-chat-fp16', {
    messages
  })
  return new Response(JSON.stringify(output))
})
```

Source: Building AI agents with AI SDK and Elastic | Carly Richmond @ Elasticsearch Labs

EXAMPLE AGENT

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!");
  }
}
```

```
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);
```

Source: Building AI agents with AI SDK and Elastic | Carly Richmond @ Elasticsearch Labs

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!");
  }
}
```

Source: Building AI agents with AI SDK and Elastic | Carly Richmond @ Elasticsearch Labs

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"
      };
    }
  },
});
```

```
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,
                    },
                  },
                ],
              },
            },
```

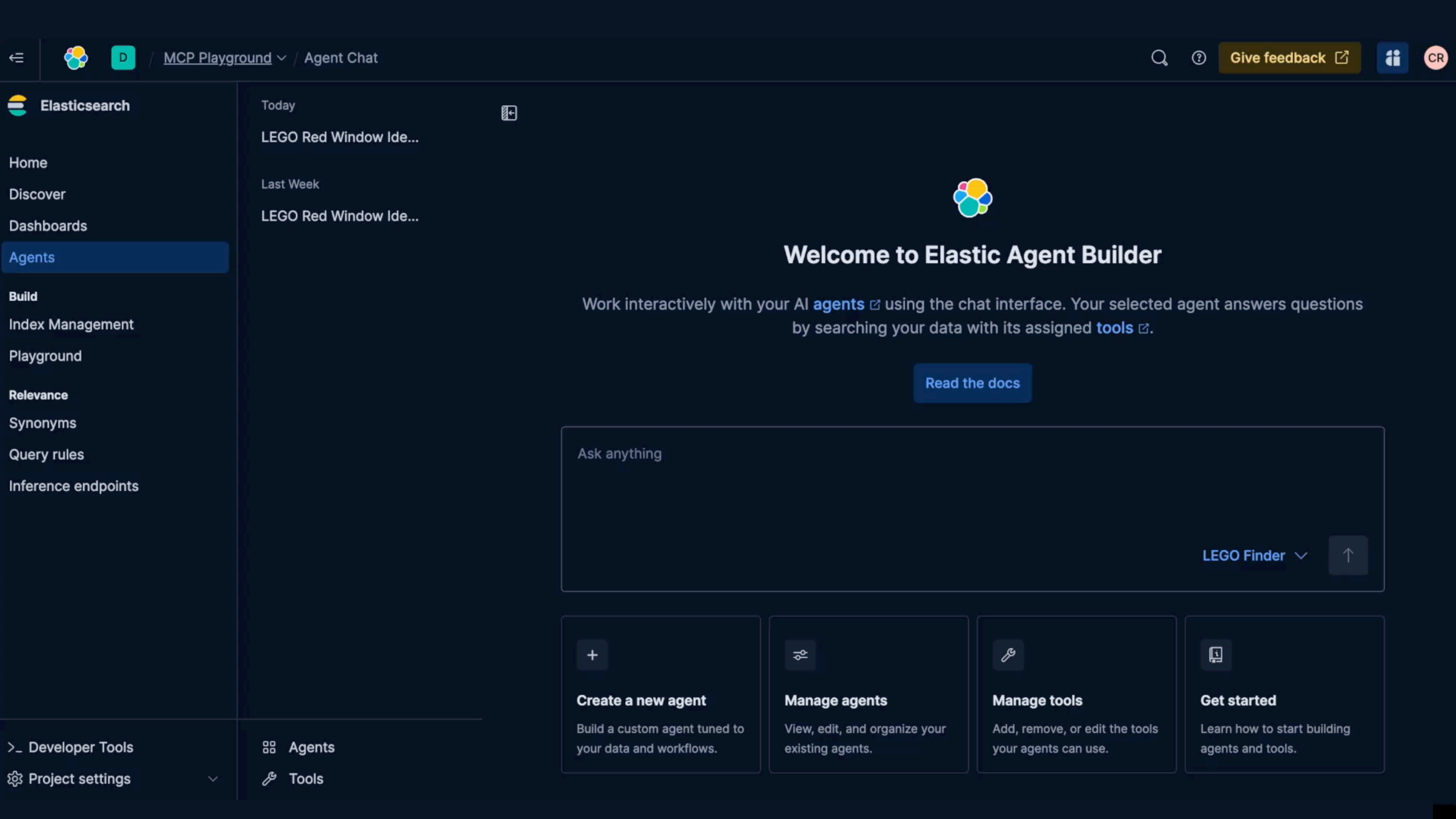
```
                    match: {
                      destination: destination,
                    },
                  },
                ],
              },
            },
          },

          // 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"
      };
    }
  },
});
```

Source: Building AI agents with AI SDK and Elastic | Carly Richmond @ Elasticsearch Labs

EXAMPLE AGENT TOOL

flight.tool.ts

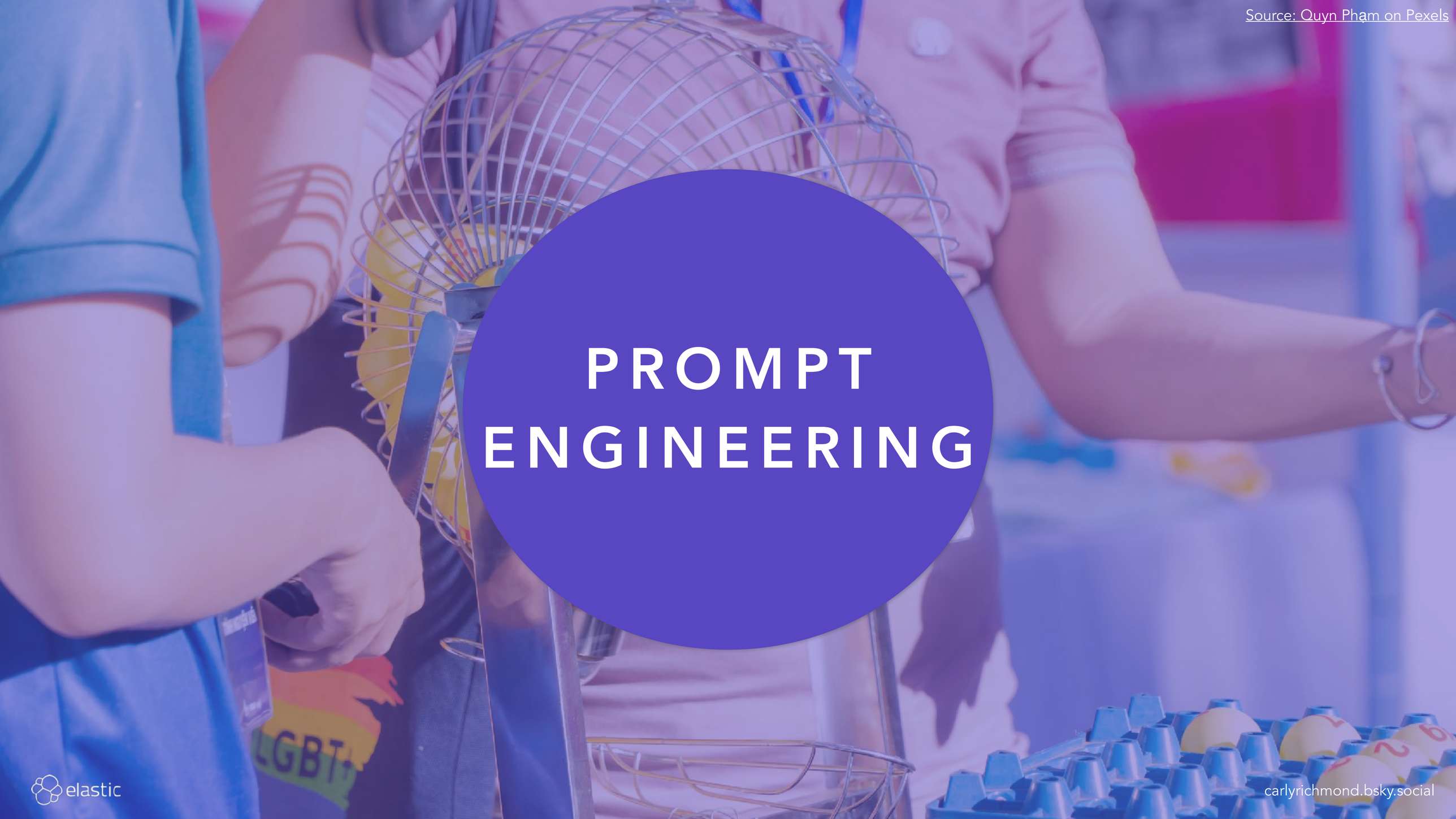
```typescript
import { tool } from 'ai';
import { z } from 'zod/v3';

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.

Source: Quyn Phạm on Pexels

BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP



Source: Quyn Phạm on Pexels

Source: Building an LLM-Powered Tool: From Concept to Production | Ioanna Nteka @ Autotrader Engineering

You are a helpful AI designed to generate engaging, informative and concise descriptions for an automotive website in the UK. You will receive a list of facts about the vehicle, such as make, model, fuel type etc. You should use the features that the user provides and stay accurate to them.

Write a description for the vehicle in two coherent paragraphs with 3 or 4 sentences in each. Base the description on the following facts: {…}


Credit: Autotrader Co-driver
elcome to Volkswagen
Co Driver
elastic
carlyrichmond.bsky.social

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

WHAT IS A PROMPT?

Role

"You are a helpful car salesperson that recommends cars based on the attributes provided by the user." You can use the getCars tool to obtain the relevant car information from our inventory.

Directive

"Return a list of cars, specifing the make, model, colour, price, location and features in a tabular format

Exemplars

"For example, if you are generating a list of small family cars in the colour silver, consider returning the details of a used 2024 Honda Civic in Silver with front airbags and a full service history."

Style Instructions

"Present the car details in the following format:

'|Year|Make|Model|Colour|Features|Location|Price|
'|2024|Honda|Civic|Silver|Front airbags|Hull|£1,295|
|2024|Honda|Civic|Silver|Airbags,GPS|London|£1,450|
|2024|Honda|Civic|Silver|Airbags|Glasgow|£1,250|."

Additional Information

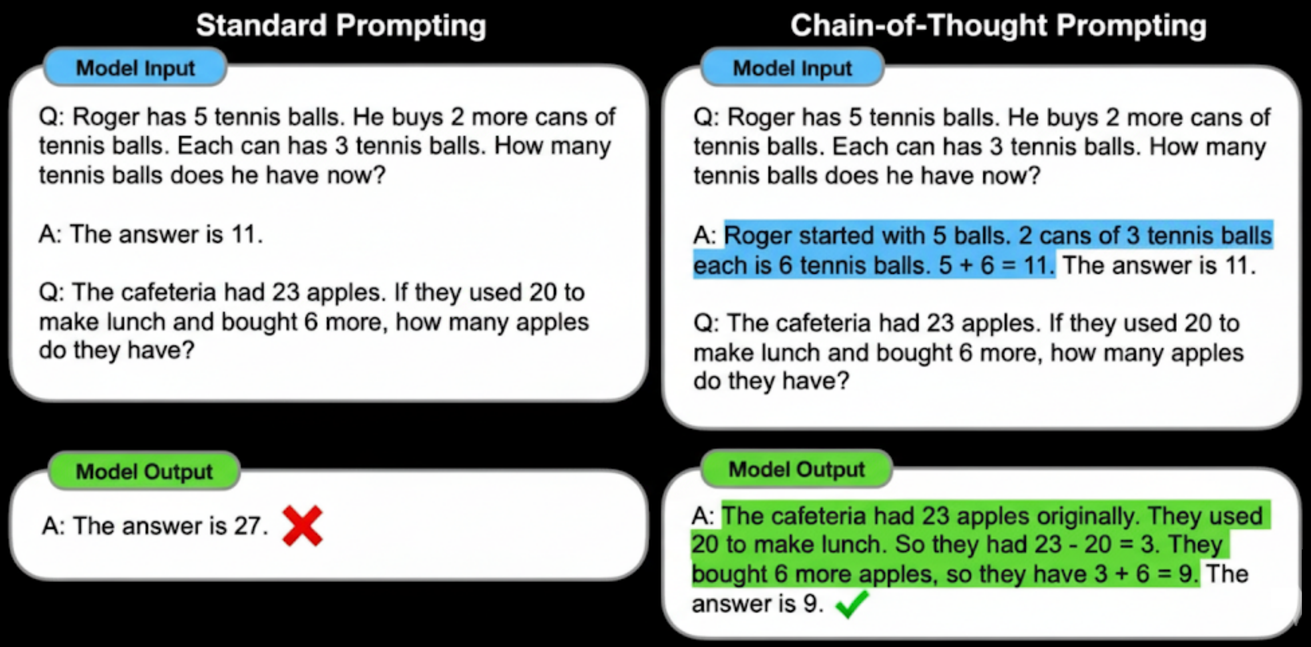
"If you are unable to find a set of matching cars, simply say 'Sorry, we are unable to find relevant cars' and offer to pass them to an agent."

Output Formatting

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

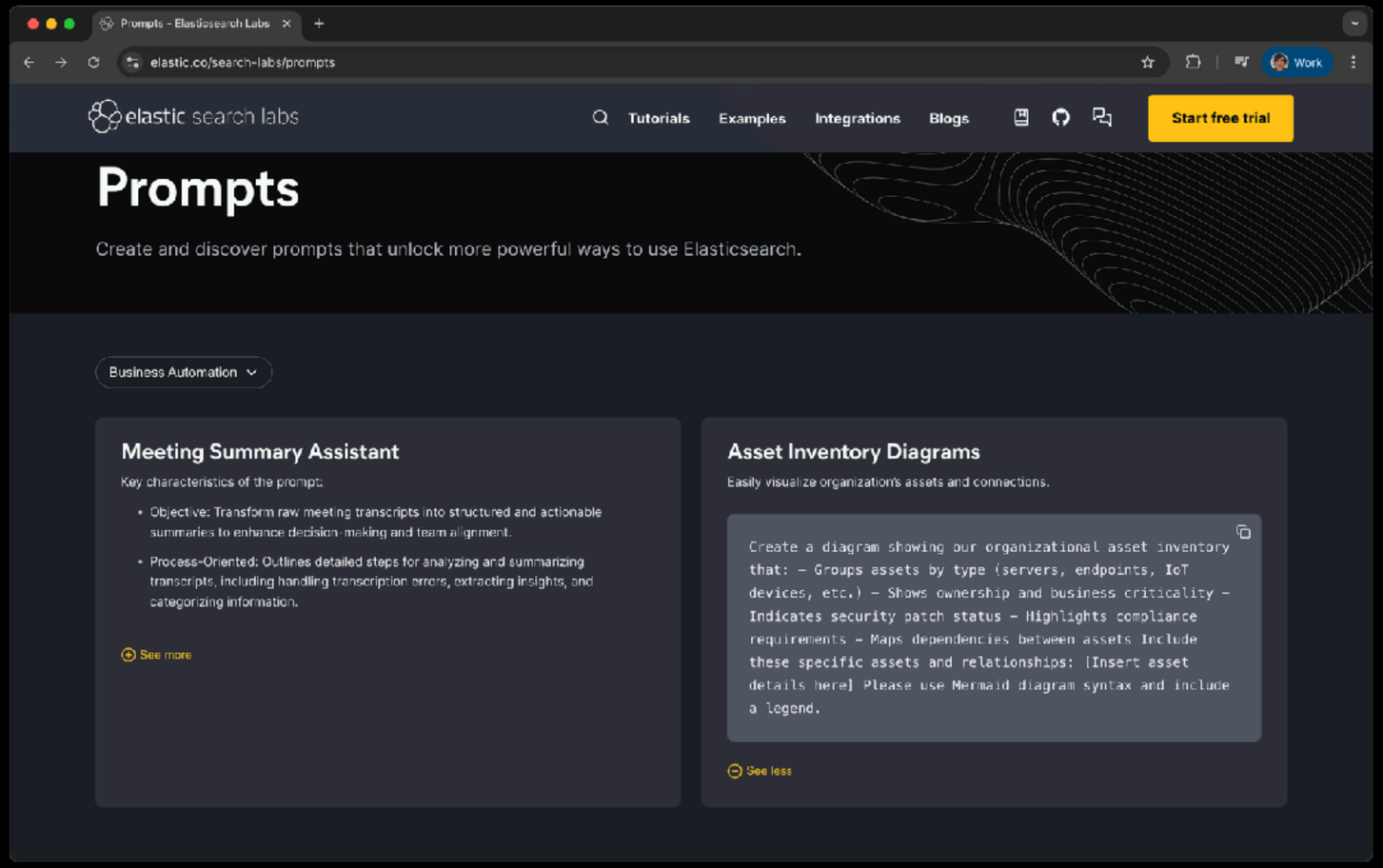
EXAMPLE STRATEGIES

- 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



Source: Quyn Phạm on Pexels

BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

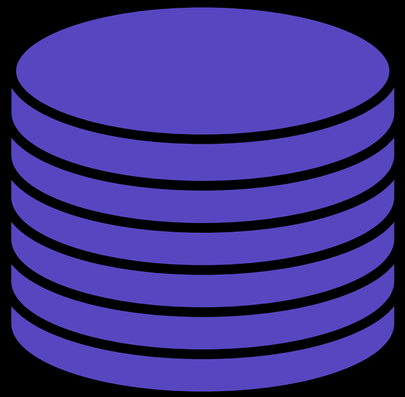
Source: Quyn Phạm on Pexels

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

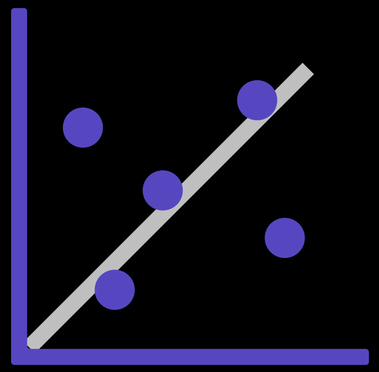
CONTEXT WINDOW

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

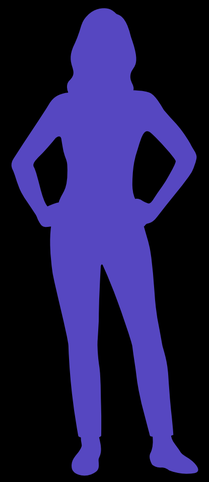
WHY DO LLMS HALLUCINATE?



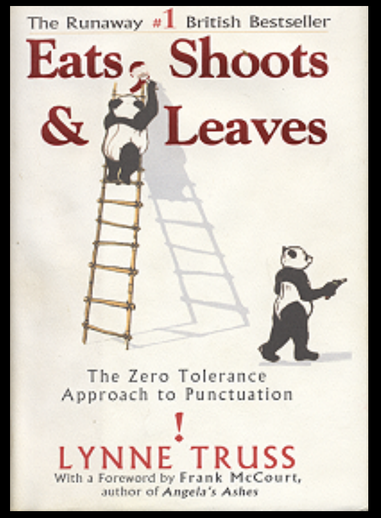
Frozen or Limited Knowledge



Overfitting



Biases



Language Ambiguity

Source: OpenAI | Why Language Models Hallucinate

Why Language Models Hallucinate

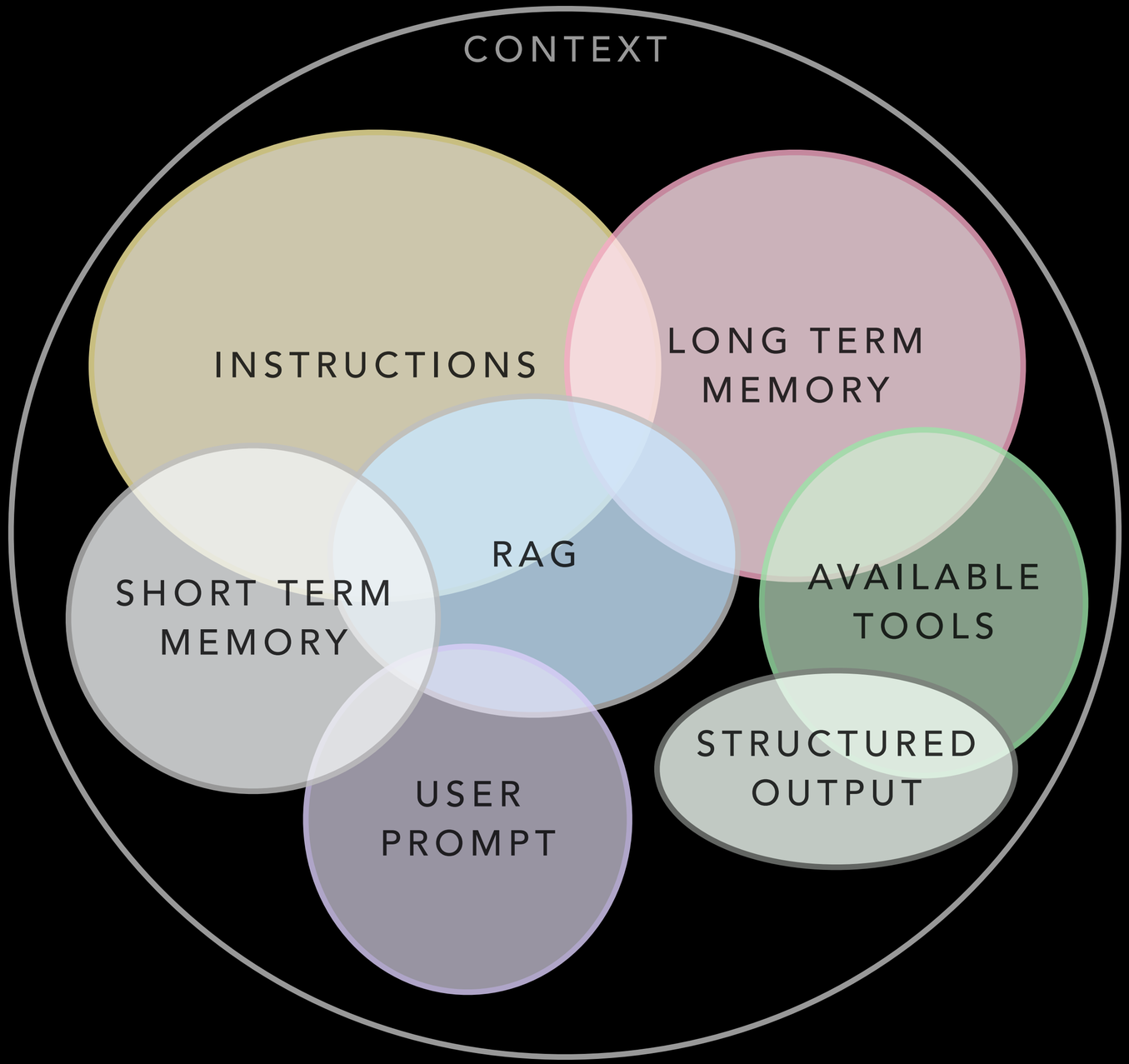
Adam Tauman Kalai* OpenAI
Ofir Nachum OpenAI
Santosh S. Vempala† Georgia Tech
Edwin Zhang OpenAI

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

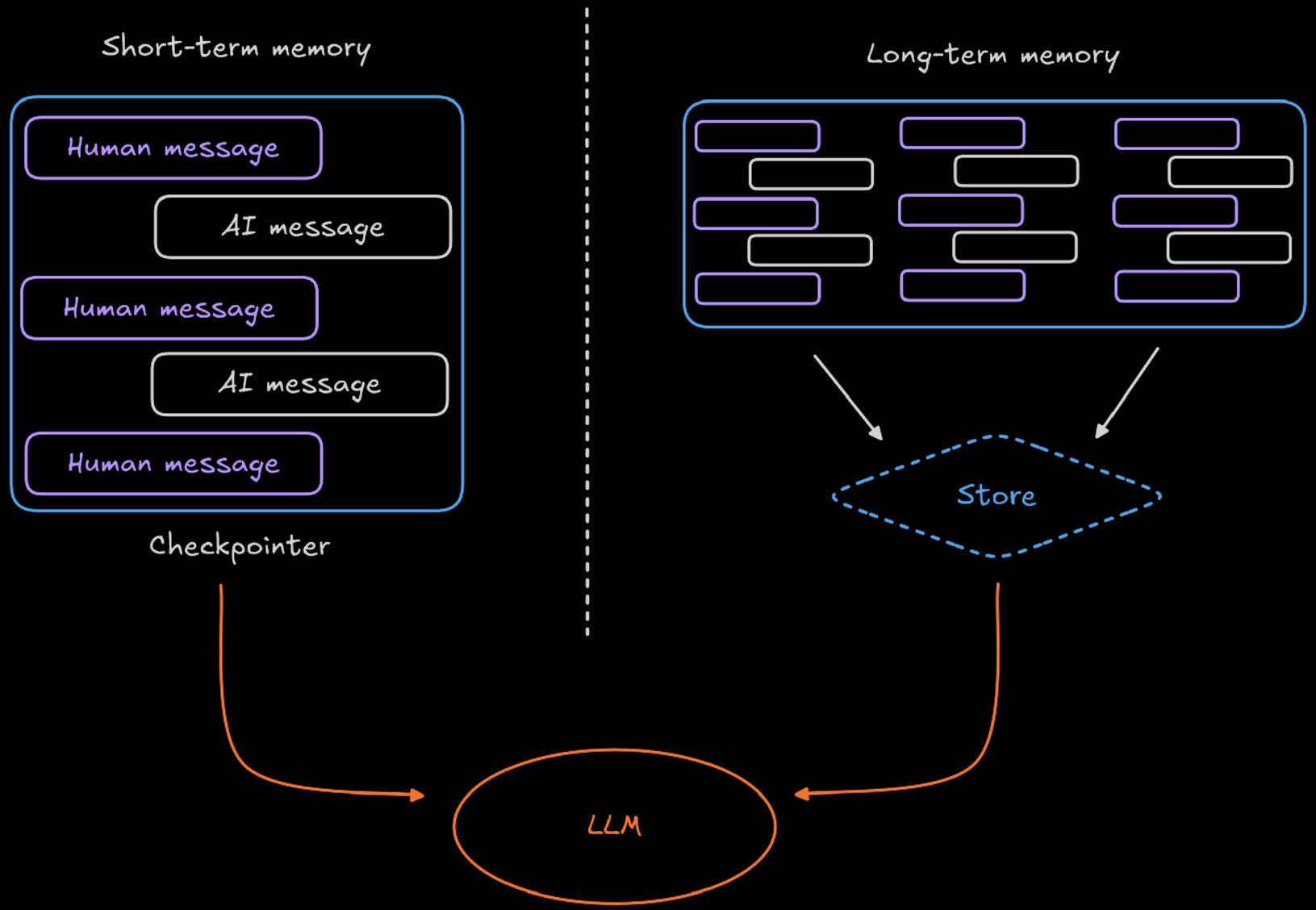


elastic

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

Credit: What is Memory | LangChain

MEMORY



Source: Memory | Langchain

STRUCTURED OUTPUTS

When was the last service for the 2015 Toyota Aygo ?

2

Request

3

Structured ouput

```
{
  make: "Toyota",
  model: "Aygo",
  year: 2015,
  registration: "AUTOTR4DR",
  color: "silver",
  services: [ { date: "2018-01-13", status: "PASSED" },
              { date: "2021-01-11", status: "PASSED" },
              { date: "2024-01-12", status: "FAILED" } ]
}
```

Schema definition

1

```
{
  make: { type: string, description: "Car manufacturer",
  model: { type: string, description: "Car model name",
  year: { type: date, description: "Car year" }
  registration: { type: string, description: "Unique car registration" },
  color: { type: string, description: "Car colour",
  services: { type: array({ date: { type: date }, status: enum(["PASSED", "FAILED"]) }),
      description: "Service history" }
}
```

Credit: Incident Summaries using LLMs | Karl Stoney

EXAMPLE

skipper.schema.yaml

```yaml
output:
required:
  - detection
  - impact
  - mitigation
  - nextSteps
properties:
  detection:
    type: string
    description: A brief summary of how the problem was detected
  impact:
    type: string
    description: The business impact, both to our teams and customers
  mitigation:
    type: string
    description: Any mitigating steps that were taken in the incident
  nextSteps:
    type: string
    description: The next steps, following mitigation
```

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

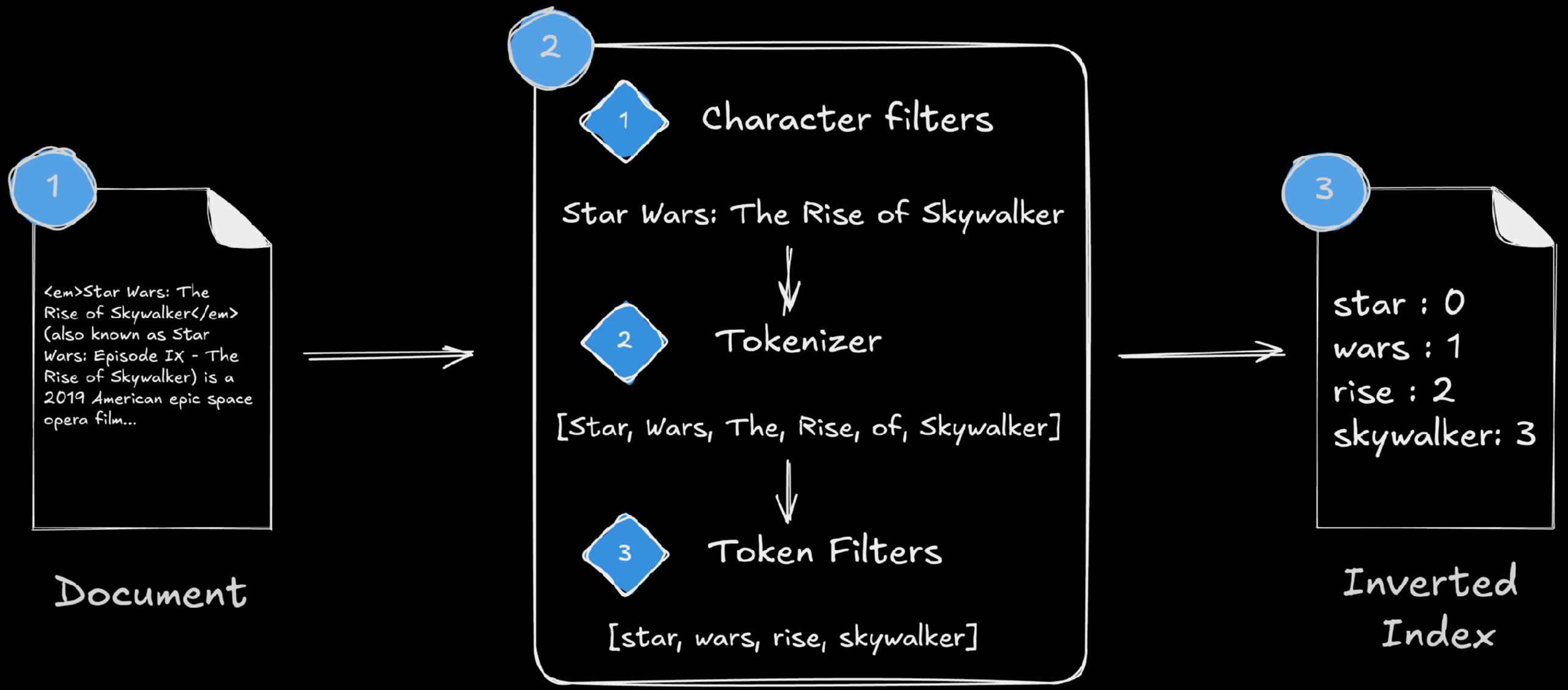
```ts
import { tool } from 'ai';
import { z } from 'zod/v3';

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
      };
    }
  }
});
```

Source: Quyn Phạm on Pexels

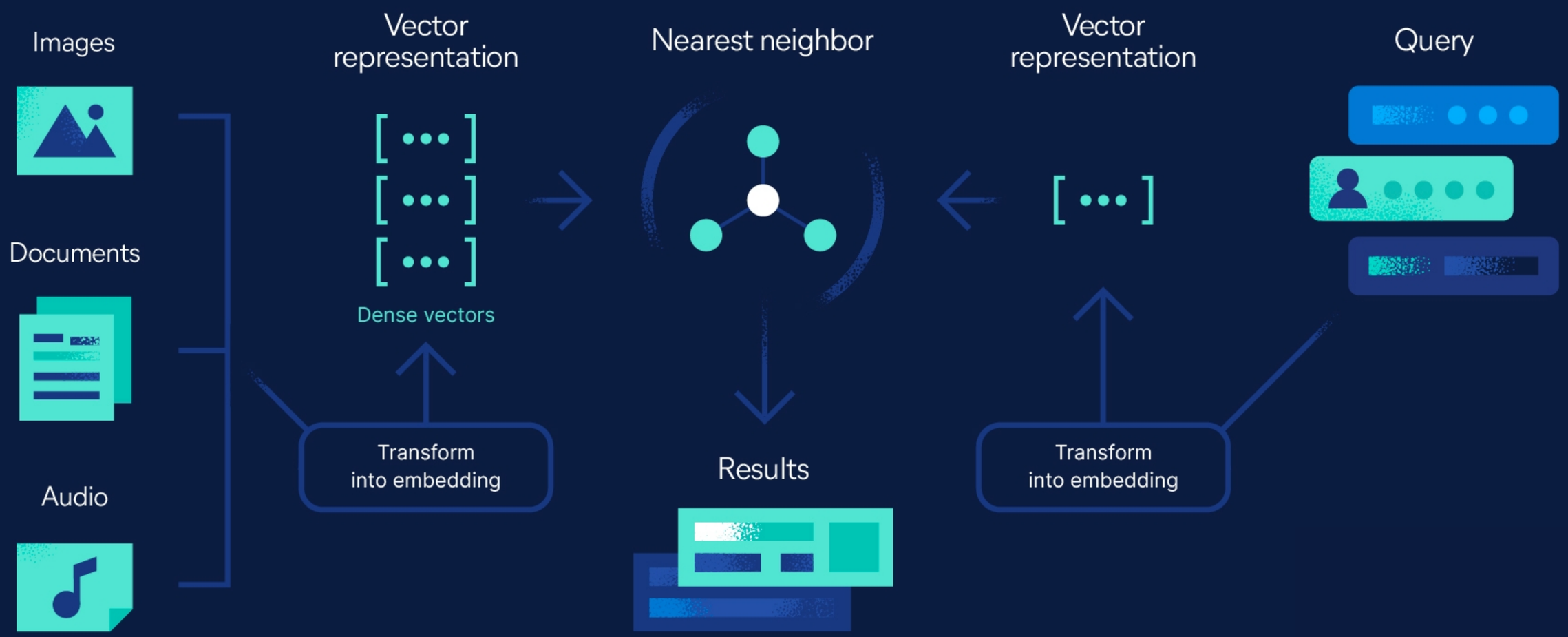
BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

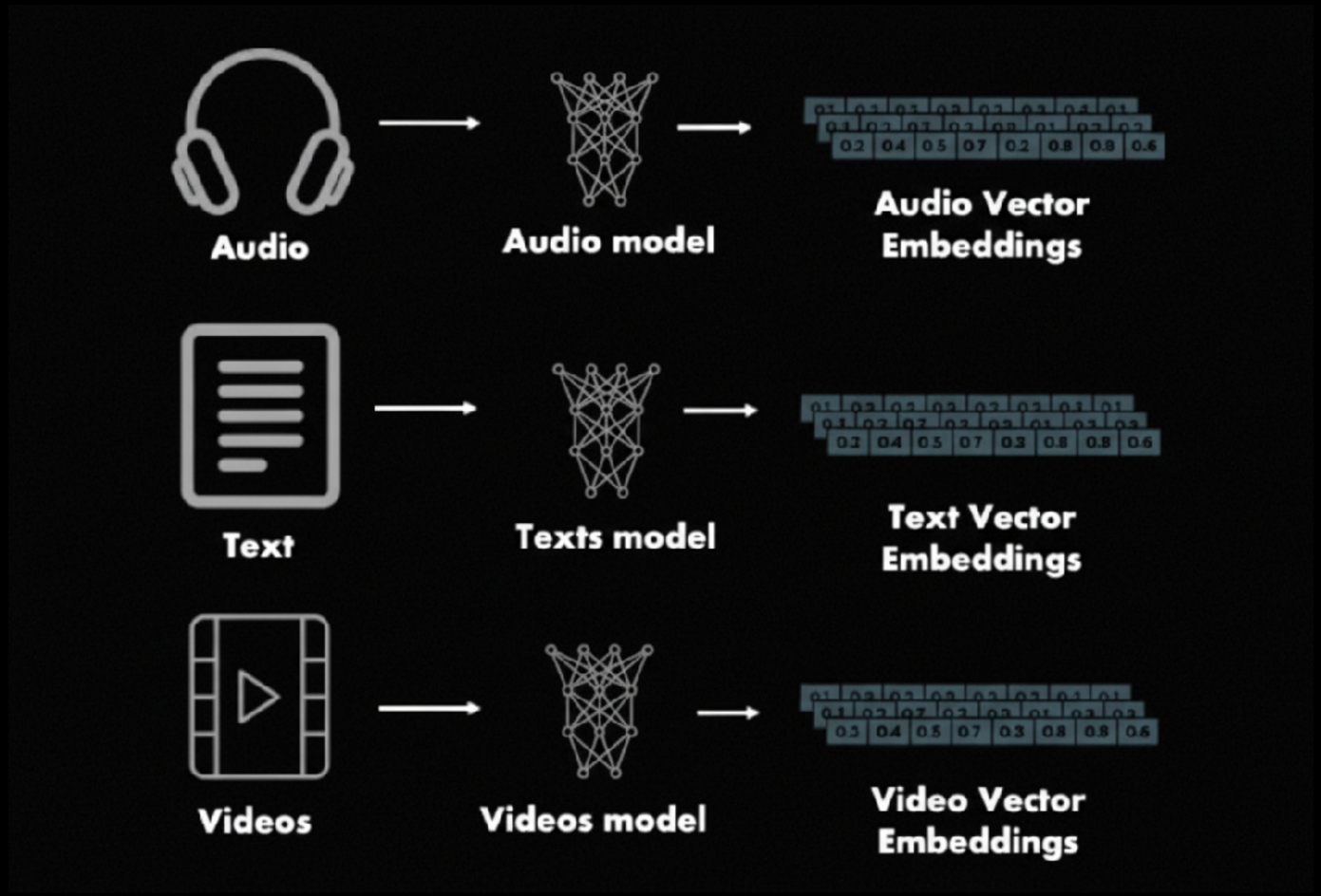

1
Star Wars: The Rise of Skywalker (also known as Star Wars: Episode IX - The Rise of Skywalker) is a 2019 American epic space opera film...
Document
2
1
Character filters
Star Wars: The Rise of Skywalker
2
Tokenizer
[Star, Wars, The, Rise, of, Skywalker]
3
Token Filters
[star, wars, rise, skywalker]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

VECTOR SEARCH

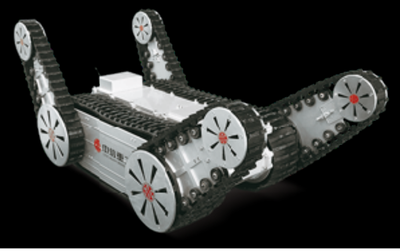


Credit: What is Context Engineering | Data Science Dojo

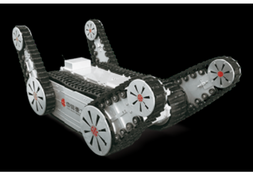
WHERE DO EMBEDDINGS COME FROM?



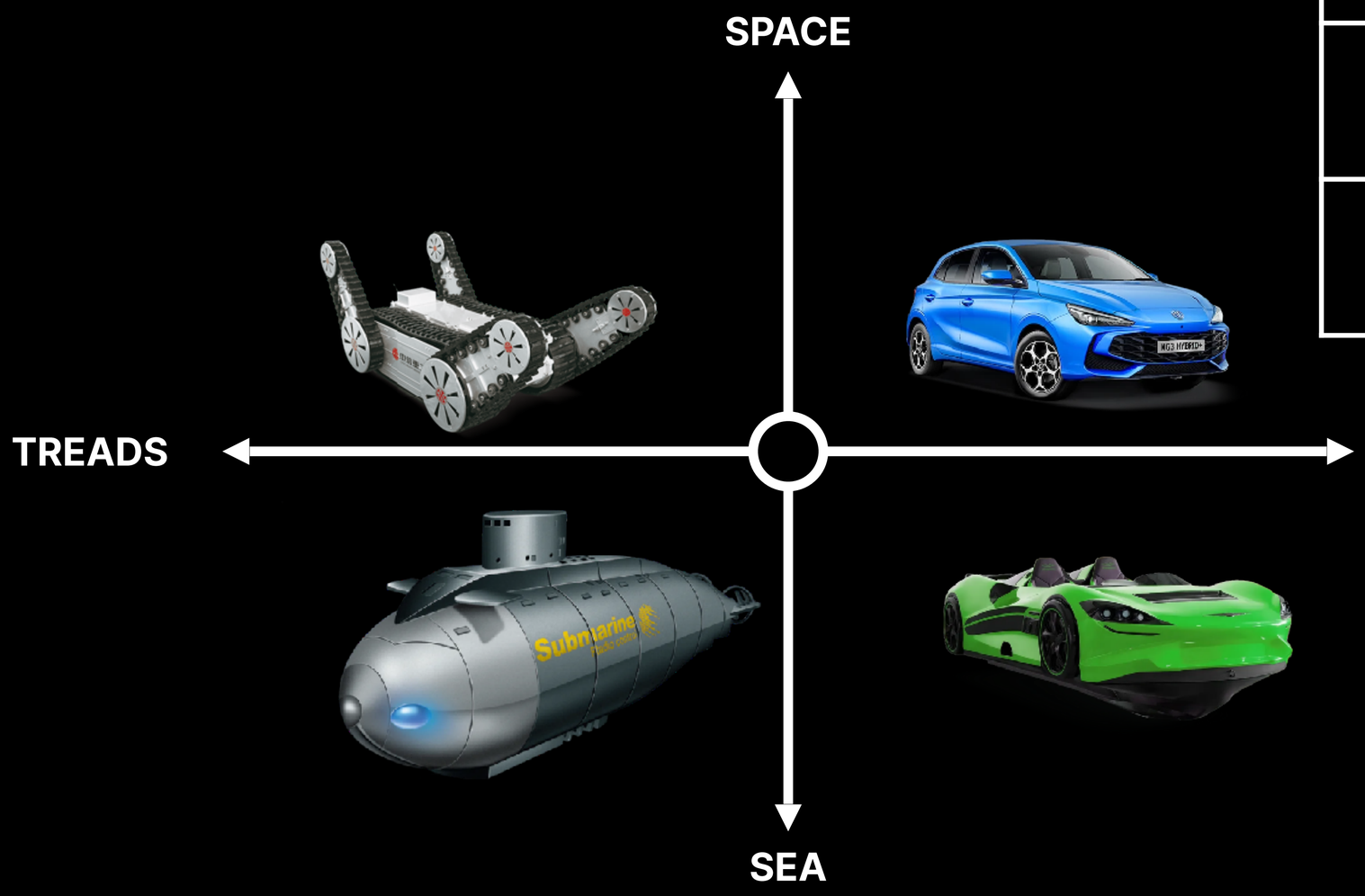
EMBEDDINGS REPRESENT YOUR DATA



TREADS ◄——●——► WHEELS

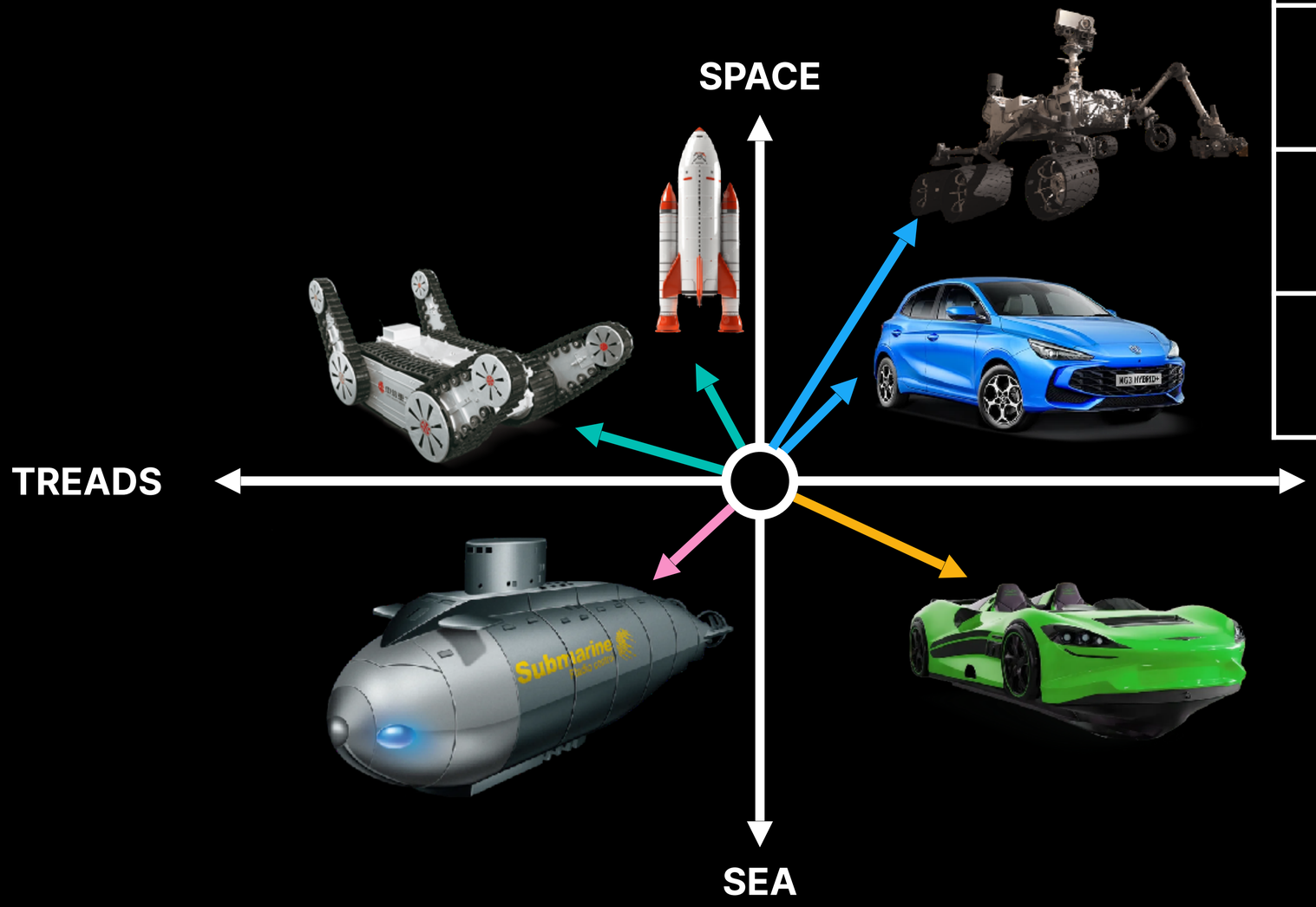
Vehicle	Vector
	[4]
	[-2]

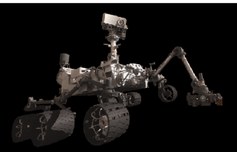
MULTIPLE DIMENSIONS REPRESENT DIFFERENT DATA ASPECTS

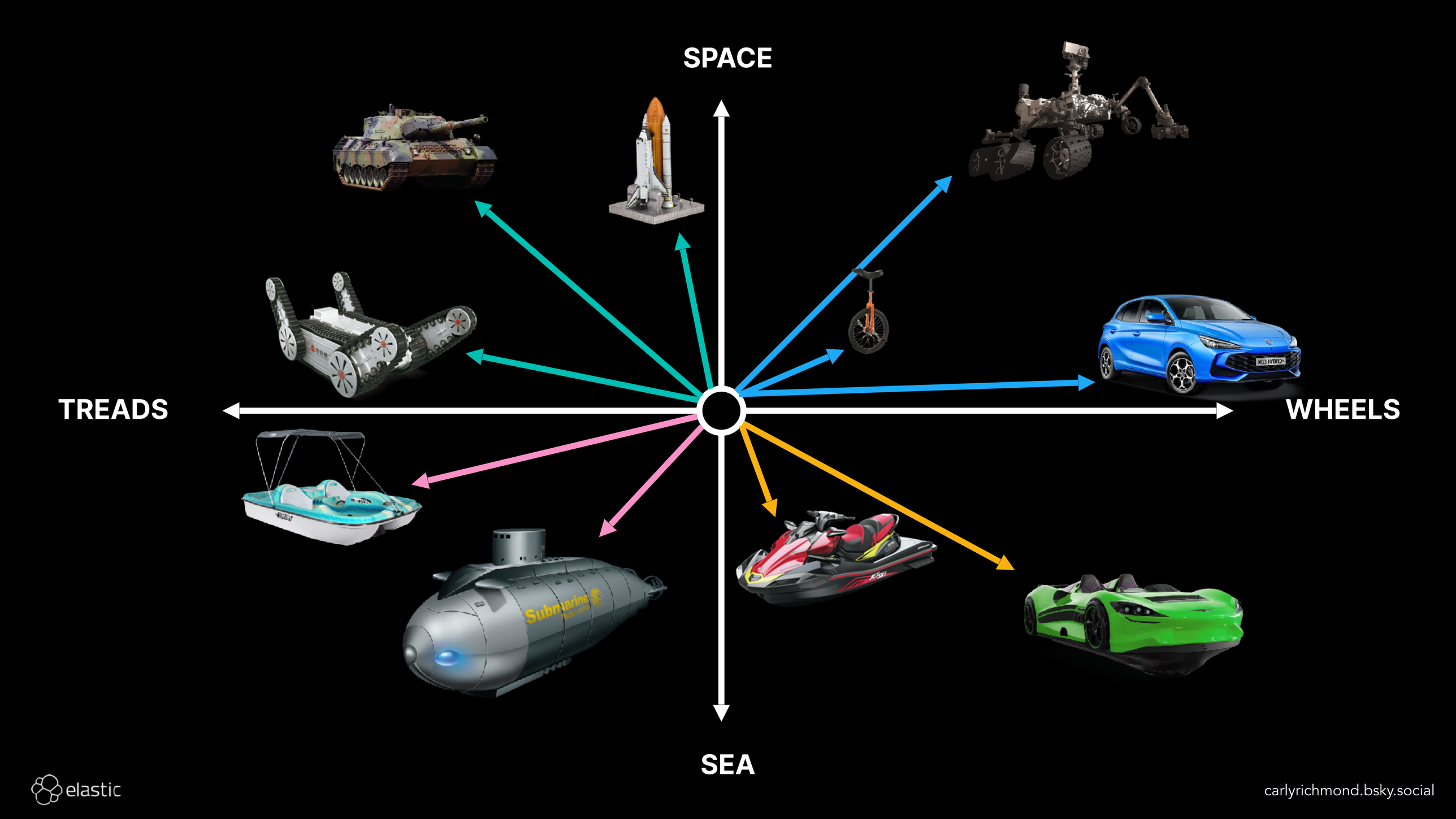


Vehicle	Vector
	[4, 0.1]
	[-2, 0.1]

SIMILAR DATA IS GROUPED TOGETHER



Vehicle	Vector
	[4, 0.1]
	[-2, 0.1]
	[6, 25.8]


SPACE
TREADS
WHEELS
SEA
Submarine

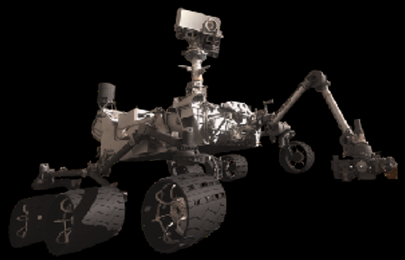
VECTOR SEARCH RANKS OBJECTS BY SIMILARITY (RELEVANCE) TO THE QUERY

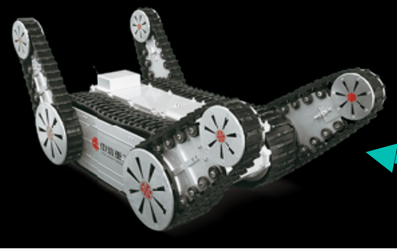
COSINE SIMILARITY

$\theta \approx 0$

$\cos(\theta) \approx 1$

SPACE





θ

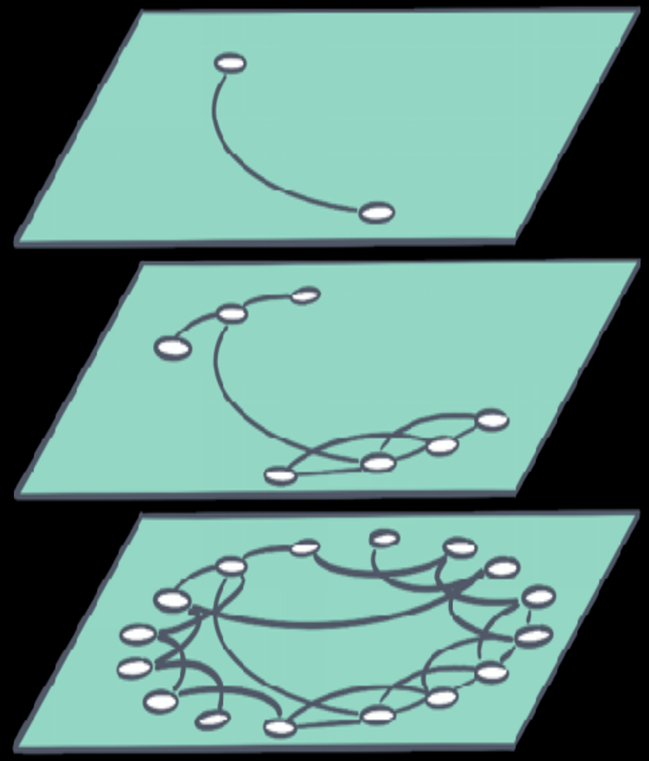
TREADS

WHEELS

SEA

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
{
  "query" : {
    "knn": {
    "field": "text_embedding.predicted_value",
    "k": 10,
    "num_candidates": 100,
    "query_vector": [-0.0078373895958599, -0.215008884668022],
    }
  }
}
```

Elasticsearch kNN search | Elastic

YES!

Source: Quyn Phạm on Pexels

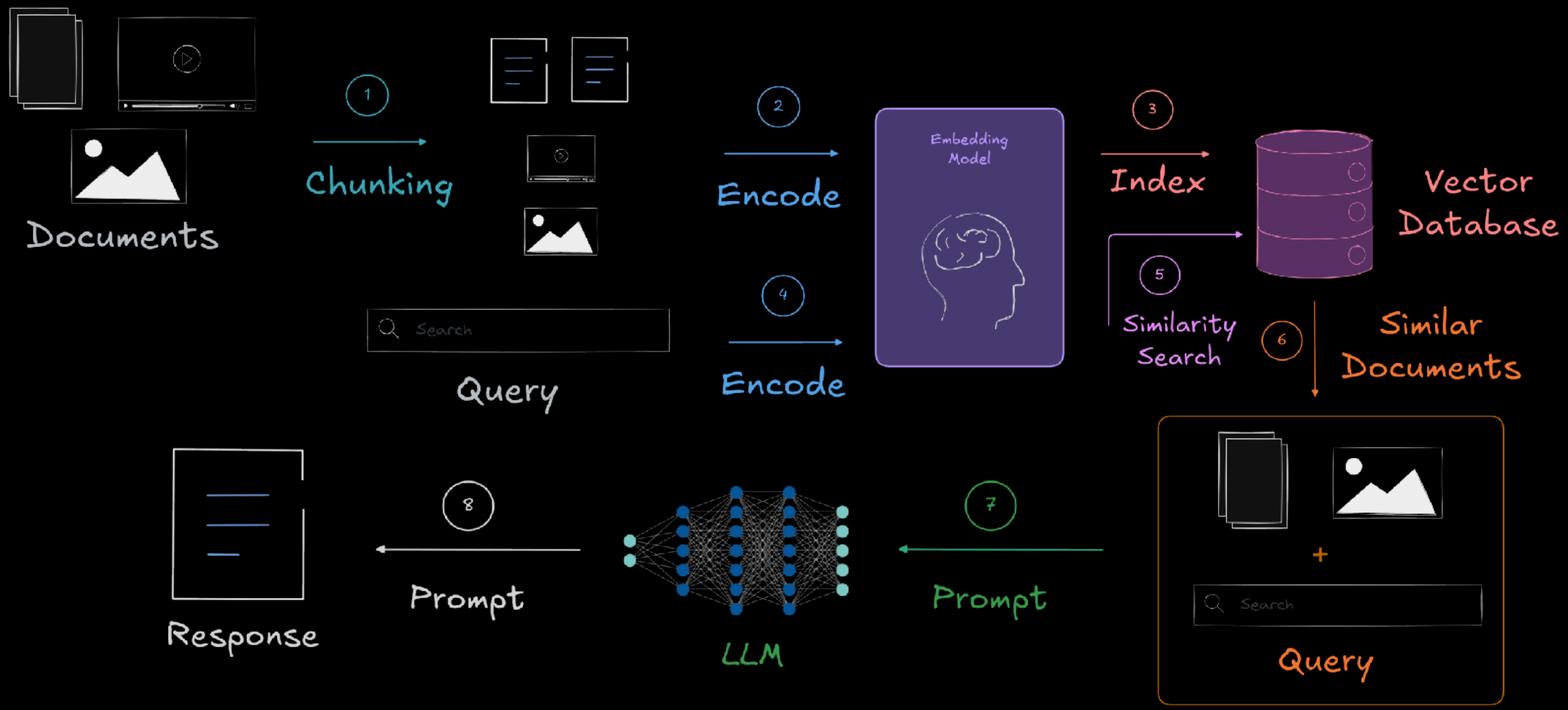
BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

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

WHAT IS RAG?



Source: Building an LLM-Powered Tool: From Concept to Production | Ioanna Nteka @ Autotrader Engineering

You are a helpful AI designed to generate engaging, informative and concise descriptions for an automotive website in the UK. You will receive a list of facts about the vehicle, such as make, model, fuel type etc. You should use the features that the user provides and stay accurate to them.

Write a description for the vehicle in two coherent paragraphs with 3 or 4 sentences in each. Base the description on the following facts: {…}

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.

Source: Quyn Phạm on Pexels

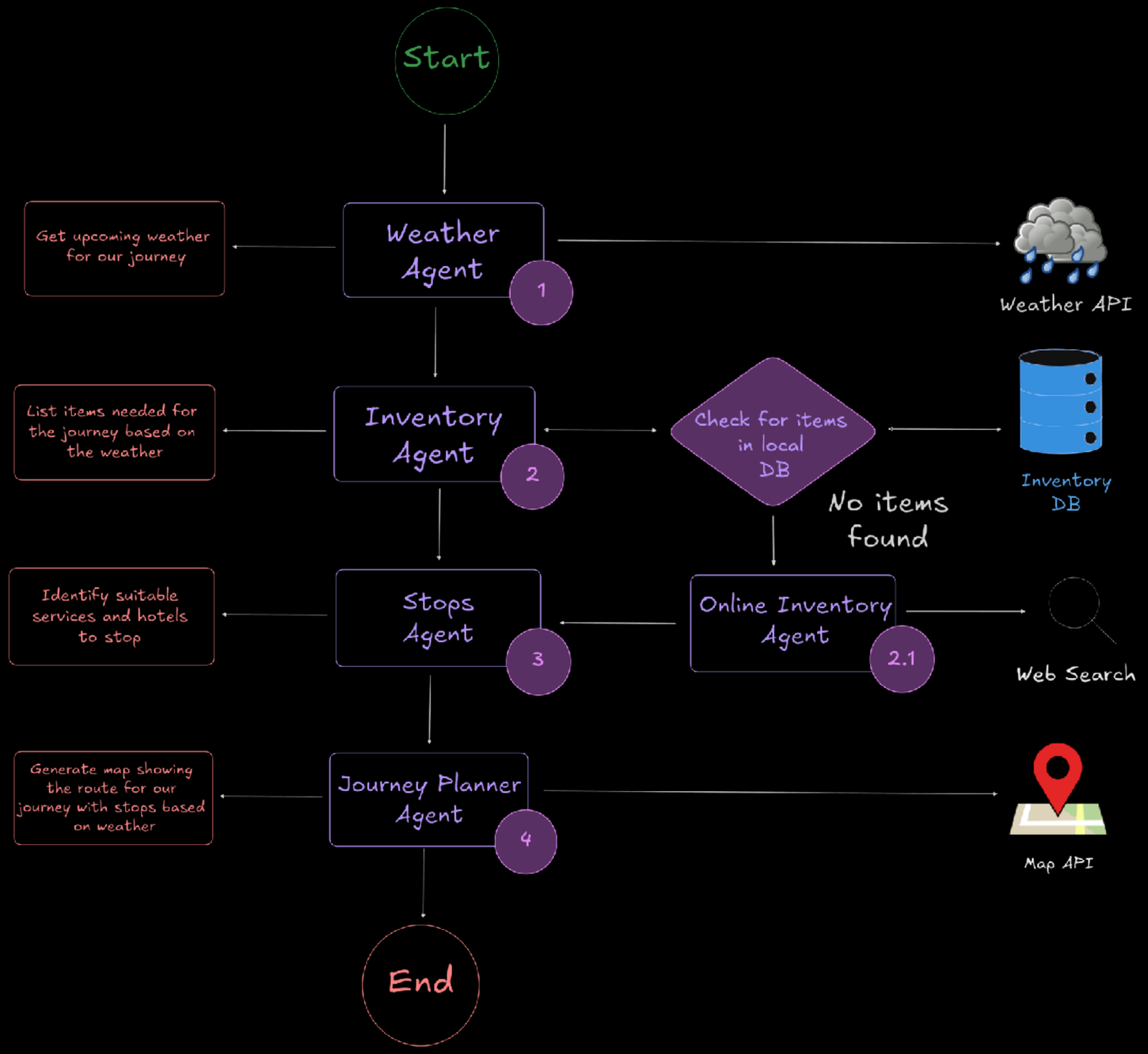
BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

Example Multi-Agent Workflow

Tools & Functions



Source: What Is the A2A (Agent2Agent) Protocol and How It Works | descope

How A2A Works

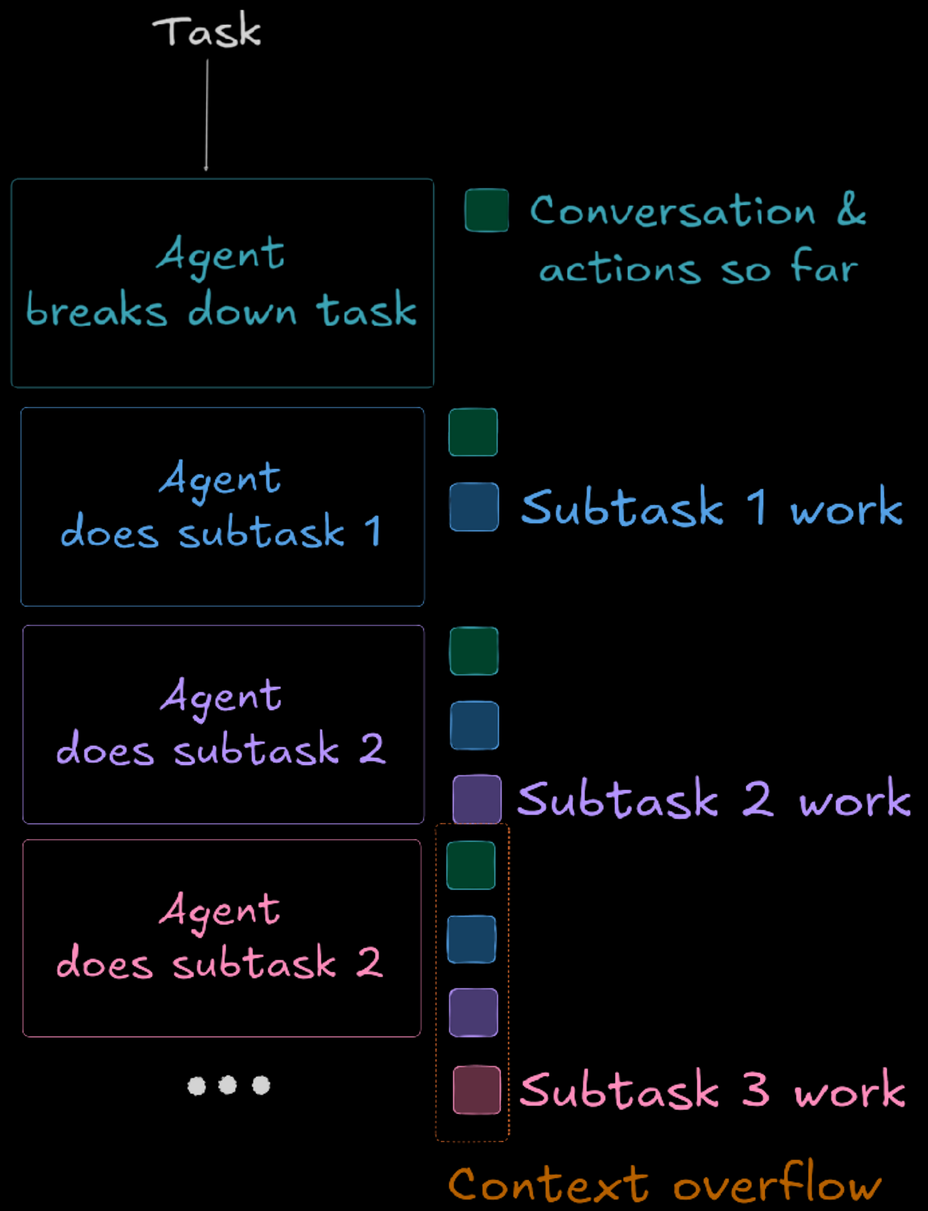
descope

A2A Communication Flow

Discovery → Task Assignment → Communication → Task Progress → Completion (Artifacts)

Credit: Don't Build Multi-Agents | Cognitiion

MULTI-AGENT CONTEXT



Source: Quyn Phạm on Pexels

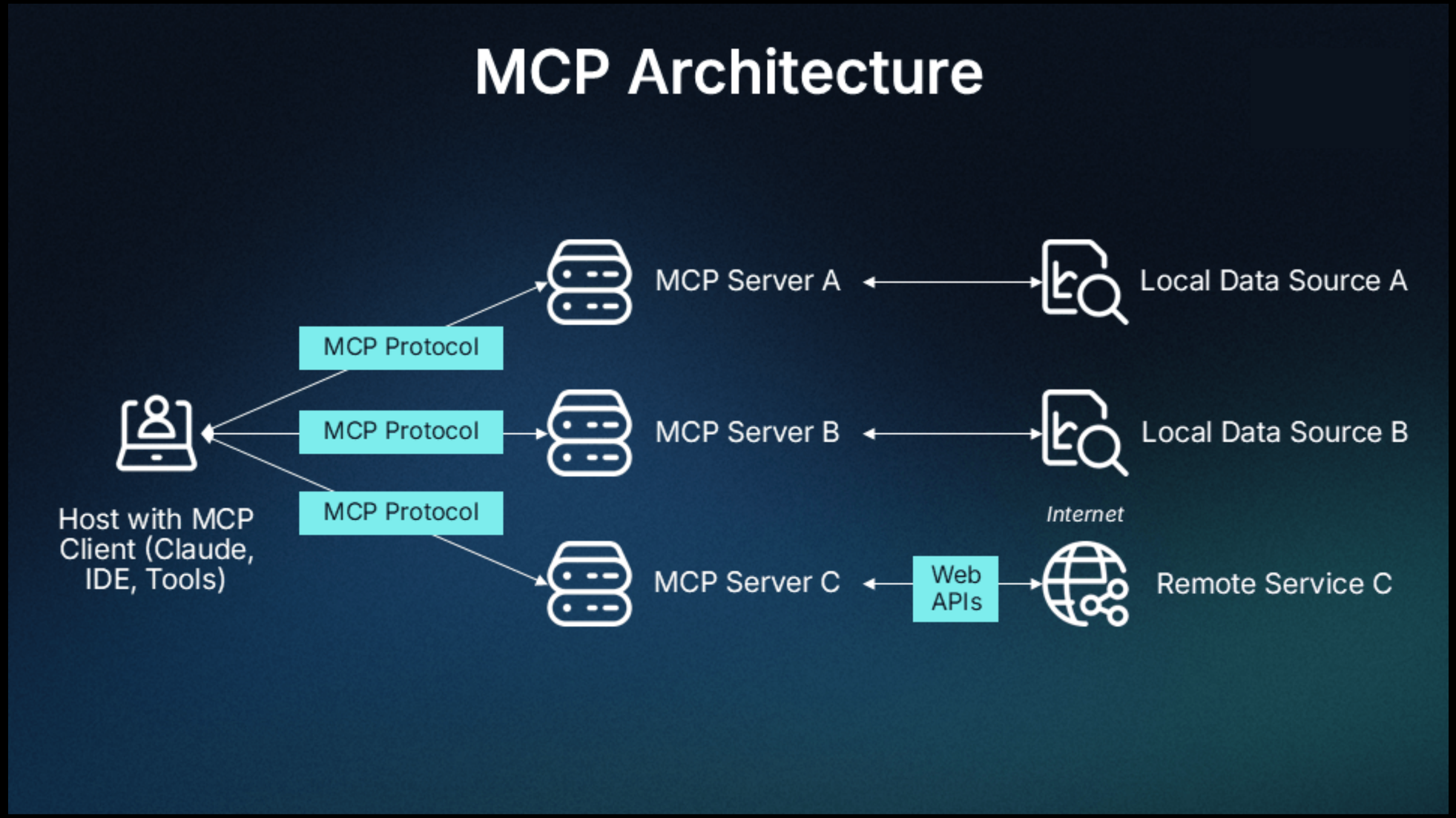
BINGO CARD

AI	LLM	AGENTS
PROMPT ENGINEERING	CONTEXT ENGINEERING	VECTOR SEARCH
RAG	A2A	MCP

Source: Quyn Phạm on Pexels

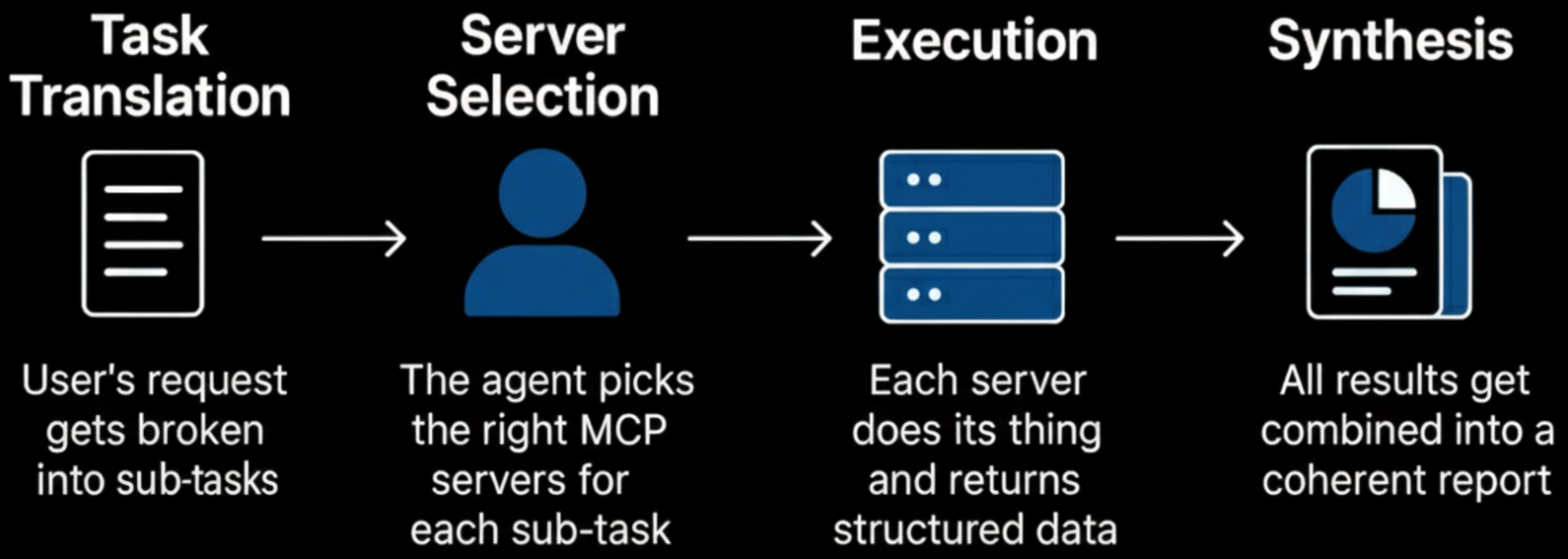
Source: What is MCP and How it Works | descope

MCP

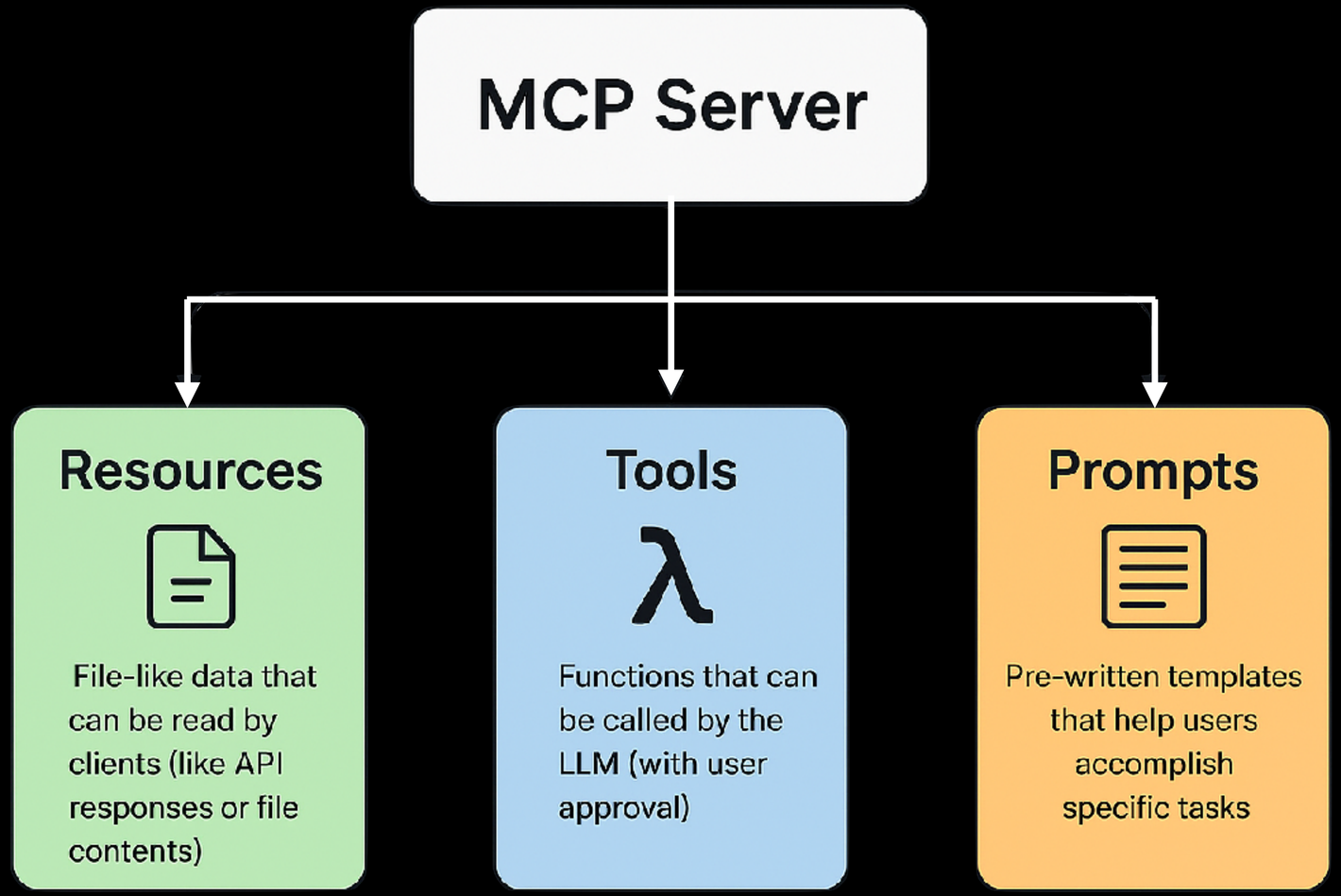


Source: How MCP Works Technically | SOCRadar

HOW DOES MCP WORK?



MCP COMPONENTS



Source: Building an MCP server with Elasticsearch for real health data | Alex Salgado @ Elasticsearch Labs

EXAMPLE MCP RESOURCE

apple_watch_mcp.py

```python
#!/usr/bin/env python3
"""
Apple Watch Health Data MCP Server
A Model Context Protocol server for querying Apple HealthKit step data stored in Elasticsearch.

Credit: Alex Salgado
"""
import os
from typing import Any, Optional
import json
from datetime import datetime
from pydantic import BaseModel, field_validator, ValidationError
from mcp.server.fastmcp import FastMCP
from elasticsearch import AsyncElasticsearch
from contextlib import asynccontextmanager

## Other server initialization code comitted

@mcp.resource("health://steps/latest")
async def get_latest_steps() -> str:
    """Gets the most recent step records"""
    query = {
        "query": {
            "match_all": {}
        },
        "sort": [
            {"endDate": {"order": "desc"}}
        ],
        "size": 10
    }

    data = await query_elasticsearch(query)
    if not data:
        return json.dumps({"error": "Unable to query Elasticsearch"}, indent=2)

    results = []
    for hit in data["hits"]["hits"]:
        source = hit["_source"]
        results.append({
            "startDate": source.get("startDate"),
            "endDate": source.get("endDate"),
            "value": source.get("value"),
            "device": source.get("device"),
            "sourceName": source.get("sourceName"),
            "dayOfWeek": source.get("dayOfWeek"),
            "hour": source.get("hour")
        })

    return json.dumps({
        "latest_steps": results
    }, indent=2)
```

apple_watch_mcp.py

```python
#!/usr/bin/env python3
"""
Apple Watch Health Data MCP Server
A Model Context Protocol server for querying Apple HealthKit step data stored in Elasticsearch.

Credit: Alex Salgado
"""
import os
from typing import Any, Optional
import json
from datetime import datetime
from pydantic import BaseModel, field_validator, ValidationError
from mcp.server.fastmcp import FastMCP
from elasticsearch import AsyncElasticsearch
from contextlib import asynccontextmanager

## Other server initialization code comitted

@mcp.resource("health://steps/latest")
async def get_latest_steps() -> str:
    """Gets the most recent step records"""
    query = {
        "query": {
            "match_all": {}
```

```python
@mcp.resource("health://steps/latest")
async def get_latest_steps() -> str:
    """Gets the most recent step records"""
    query = {
        "query": {
            "match_all": {}
        },
        "sort": [
            {"endDate": {"order": "desc"}}
        ],
        "size": 10
    }

    data = await query_elasticsearch(query)
    if not data:
        return json.dumps({"error": "Unable to query Elasticsearch"}, indent=2)

    results = []
    for hit in data["hits"]["hits"]:
        source = hit["_source"]
        results.append({
            "startDate": source.get("startDate"),
            "endDate": source.get("endDate"),
            "value": source.get("value"),
            "device": source.get("device"),
            "sourceName": source.get("sourceName"),
```

```
            {"endDate": {"order": "desc"}}
        ],
        "size": 10
    }

    data = await query_elasticsearch(query)
    if not data:
        return json.dumps({"error": "Unable to query Elasticsearch"}, indent=2)

    results = []
    for hit in data["hits"]["hits"]:
        source = hit["_source"]
        results.append({
            "startDate": source.get("startDate"),
            "endDate": source.get("endDate"),
            "value": source.get("value"),
            "device": source.get("device"),
            "sourceName": source.get("sourceName"),
            "dayOfWeek": source.get("dayOfWeek"),
            "hour": source.get("hour")
        })

    return json.dumps({
        "latest_steps": results
    }, indent=2)
```

Source: Building an MCP server with Elasticsearch for real health data | Alex Salgado @ Elasticsearch Labs

EXAMPLE MCP RESOURCE

apple_watch_mcp.py

```python
#!/usr/bin/env python3
"""
Apple Watch Health Data MCP Server
A Model Context Protocol server for querying Apple HealthKit step data stored in Elasticsearch.

Credit: Alex Salgado
"""
import os
from typing import Any, Optional
import json
from datetime import datetime
from pydantic import BaseModel, field_validator, ValidationError
from mcp.server.fastmcp import FastMCP
from elasticsearch import AsyncElasticsearch
from contextlib import asynccontextmanager

## Other server initialization code comitted

@mcp.resource("health://steps/latest")
async def get_latest_steps() -> str:
    """Gets the most recent step records"""
    query = {
        "query": {
            "match_all": {}
        },
        "sort": [
            {"endDate": {"order": "desc"}}
        ],
        "size": 10
    }

    data = await query_elasticsearch(query)
    if not data:
        return json.dumps({"error": "Unable to query Elasticsearch"}, indent=2)

    results = []
    for hit in data["hits"]["hits"]:
        source = hit["_source"]
        results.append({
            "startDate": source.get("startDate"),
            "endDate": source.get("endDate"),
            "value": source.get("value"),
            "device": source.get("device"),
            "sourceName": source.get("sourceName"),
            "dayOfWeek": source.get("dayOfWeek"),
            "hour": source.get("hour")
        })

    return json.dumps({
        "latest_steps": results
    }, indent=2)
```

elastic

carlyrichmond.bsky.social

HOUSE!

RESOURCES

- Building an LLM-Powered Tool: From Concept to Production | Ioanna Nteka @ Autotrader Engineering
- Demystifying Large Language Models (LLM101) | Mahmoud Oshagh @ Autotrader Engineering
- Incident Summaries using LLMs | Karl Stoney
- What is Context Engineering | Carly Richmond @ Elasticsearch Labs
- 12-Factor Agents - Principles for building reliable LLM applications | Dexter Horthy
- How to Fix Your Context | Drew Breunig
- The Prompt Report: A Systematic Survey of Prompt Engineering Techniques | Schulhoff et al.
  What is Memory | LangChain
- Don't Build Multi-Agents | Cognition
- Structured Outputs | LangChain
- AI Travel Agent Planner | Carly Richmond on GitHub
- RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation | Gan and Sun

Source: Milad Fakurian on Unsplash

Source: Priscilla Du Preez 🇨🇦 on Unsplash

SCAN ME

carlyrichmond.bsky.social

elastic