# Adding AI Capabilities to React Apps with Vercel AI SDK

Nir Tamir

# TOC

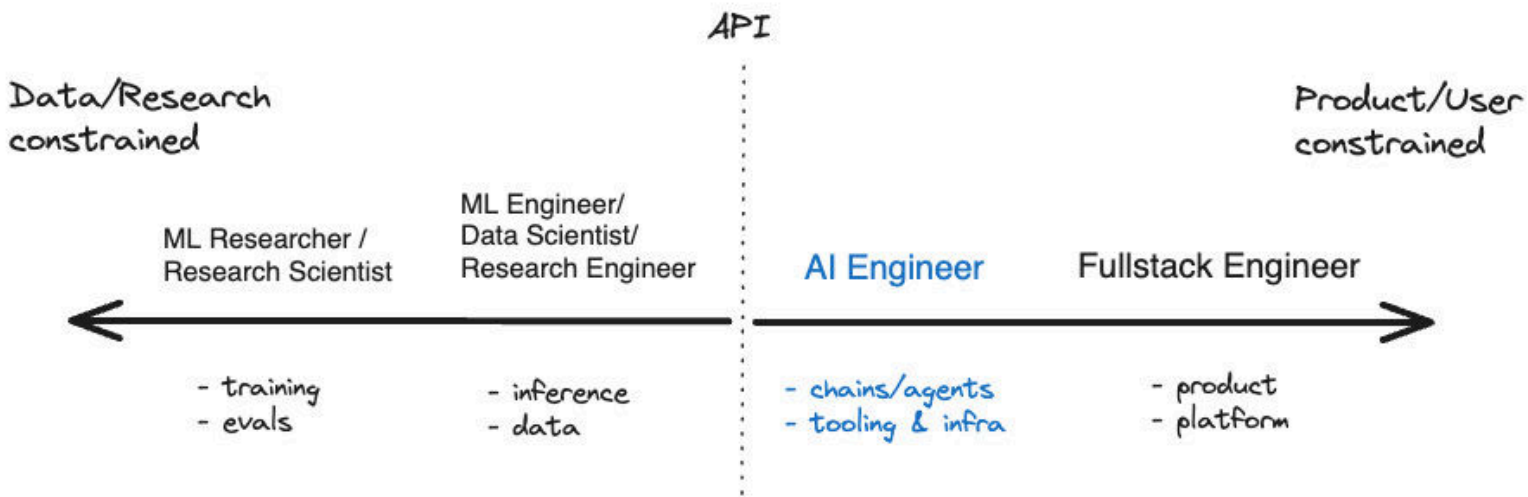
# Nir Tamir

- Senior Frontend developer
- Work at my own company helping early stage startups
- Loves open source and tooling
- nirtamir.com
- @nirtamir2
- @NirTamir
- @nirtamir2

# You don’t need to be a Data Scientist to be an AI Engineer

# AI Engineer

The Rise of the AI Engineer



# AI Engineer Skills

## Hiring AI Engineer


# My journey

# Ollama

Run open source AI models locally

# Ollama

https://ollama.com/

- Enable running open source AI models locally
- Works offline
- Open source
- Free
- CLI
- Vision models
- Integrations Copilot, Apps, Zed, Scripts…
- Local Open WebUI
- REST API
- Ollama SDK

# Calling Ollama via REST API

It has the same structure as OpenAI API hosted at http://localhost:11434

```
const response = await fetch("http://localhost:11434/v1/chat/completions", {
  method: "POST",
  body: JSON.stringify({
    model: "llama3",
    messages: [
      { role: "system", content: "You are a helpful assistant."},
      { role: "user", content: "Hello!"}
    ]}),
});
const data = await response.json();
console.log(data);
```


```
{
  "id": "chatcmpl-725",
  "object": "chat.completion",
  "created": 1719413908,
  "model": "llama3",
  "system_fingerprint": "fp_ollama",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Hello! It's nice to meet you. Is there something I can help you with, or would you like to
chat about something in particular? I'm here to listen and assist if needed."
      },
      "finish_reason": "stop"
```

# Ollama is compatible with OpenAI interface

```
import { OpenAI } from "openai"

const openai = new OpenAI({
  baseURL: "http://localhost:11434/v1",
  apiKey: "ollama", // Does not matter which key you use
  dangerouslyAllowBrowser: true, // Just because I run it in the slide
});

const response = await openai.chat.completions.create({
  model: "llama3",
  messages: [{ role: "user", content: "Tell me one line cool fact about AI" }],
});

console.log(response.choices[0].message.content)
```

```
Here's one: "AI has been used to create an algorithm that can generate realistic fake faces, so convincing that
even humans have trouble telling the difference from real-life photos - and it only takes 1-2 minutes of video data
from a person to train the system!"
```

# Calling Ollama via SDK

https://github.com/ollama/ollama-js

```
import ollama from "ollama"

const response = await ollama.chat({
  model: "llama3",
  messages: [{ role: "user", content: "Why is the sky blue?" }],
})
console.log(response.message.content)
```

# Calling Ollama via SDK

https://github.com/ollama/ollama-js

```
import ollama from "ollama/browser"

const response = await ollama.chat({
  model: "llama3",
  messages: [{ role: "user", content: "Why is the sky blue?" }],
})
console.log(response.message.content)
```

```
The sky appears blue because of a phenomenon called Rayleigh scattering, named after the British physicist Lord Rayleigh. Here's why: 1. **Sunlight**: When sunlight enters Earth's atmosphere, it contains all the colors of the visible spectrum (red, orange, yellow, green, blue, indigo, and violet). 2. **Molecules**: The atmosphere is made up of tiny molecules of gases like nitrogen (N2) and oxygen (O2). These molecules are much smaller than the wavelength of light. 3. **Scattering**: When sunlight hits these small molecules, it scatters in all directions. This scattering effect is more pronounced for shorter wavelengths (like blue and violet) than longer wavelengths (like red and orange). 4. **Blue dominance**: As a result of this scattering, the shorter wavelengths (blue and violet) are dispersed throughout the atmosphere, while the longer wavelengths (red and orange) continue to travel in a more direct path to our eyes. 5. **Our perception**: Our brain interprets the combined light from all directions as blue, which is why the sky appears blue during the daytime. Here's an interesting fact: The color of the sky can change depending on various conditions: * During sunrise and sunset, when the sun is lower in the sky, the light has to travel longer distances through the atmosphere, scattering more of the shorter wavelengths. This is why we often see hues of red and orange during these times. * At higher altitudes or in areas with less atmospheric scattering (like space), the sky can appear more intense blue due to reduced scattering. So, there you have it! The blue color of the sky is a result of the scattering of sunlight by tiny molecules in our atmosphere.
```

# Calling Ollama via SDK Streaming

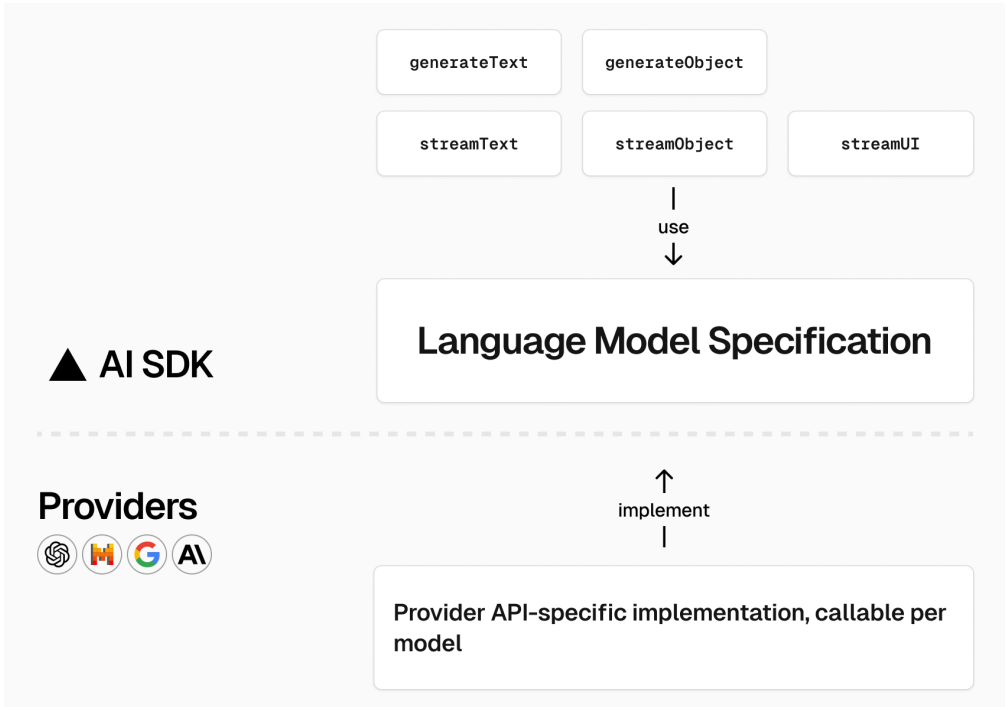
https://github.com/ollama/ollama-js

```
import ollama from "ollama/browser"

const response = await ollama.chat({
  model: "llama3",
  messages: [{ role: "user", content: "Why is the sky blue?" }],
  stream: true,
})
for await (const part of response) {
  console.log(part.message.content)
}
```

```
A
question
that
has
puzzled
humans
for
centuries
!
The
sky
appears
```

# AI SDK

https://sdk.vercel.ai



# The AI Toolkit for TypeScript

From the creators of Next.js, the Vercel AI SDK gives you the tools you need to build AI-powered products.

Get Started

```
$ npm i ai
```

# Getting started with Google Gemini AI

```
pnpm add @ai-sdk/google
```

Create `.env.local` file with your Google API key

```
GOOGLE_GENERATIVE_AI_API_KEY="YOUR_KEY"
```

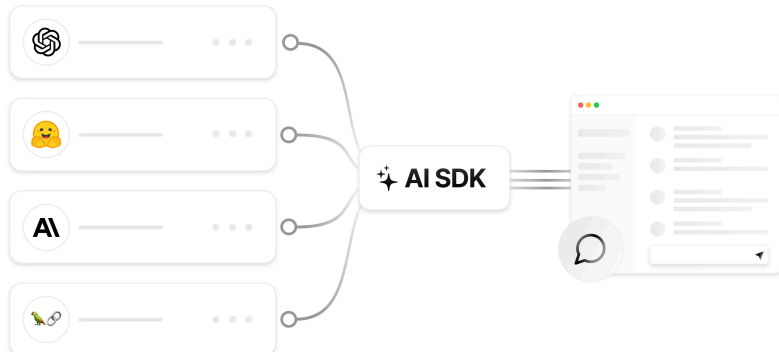
# AI SDK example

```ts
import { createGoogleGenerativeAI } from "@ai-sdk/google";
import { generateText } from "ai";
import { env } from "./env";

// Or just import { google } from "@ai-sdk/google";
const google = createGoogleGenerativeAI({
  apiKey: env.GOOGLE_GENERATIVE_AI_API_KEY,
});

const result = await generateText({
  model: google("models/gemini-1.5-flash-latest"),
  prompt: "Tell me a joke.",
});

console.log(result.text);
```

```
Why don't scientists trust atoms? Because they make up everything!
```

# AI SDK Switch models easily

```
import { openai } from "@ai-sdk/openai";
import { generateText } from "ai";

const result = await generateText({
  model: openai("gpt-4o"),
  prompt: "Tell me a joke.",
});
```



# AI SDK Image example

```
import { generateText } from "ai";
import { google } from "./google-model";

const result = await generateText({
  model: google("models/gemini-1.5-flash-latest"),
  messages: [{
      role: "user",
      content: [
        { type: "text", text: "Describe the image in detail." },
        { type: "image",  image: "https://www.nirtamir.com/_astro/portrait.9b-_4A6X_bepz7.webp" },
      ]
    }],
});

console.log(result.text);
```

```
The image shows a man standing in front of a concrete wall. He is wearing a black t-shirt and has short brown hair.
He is smiling and looking directly at the camera. The background is a textured concrete wall with a light gray
color. The man's arms are at his sides and his hands are not visible. The image is well-lit and the man's face is
clear.
```

# AI SDK Object example

```ts
import { generateObject } from "ai";
import { google } from "./google-model";
import { schema } from "./schema";

const { object } = await generateObject({
  model: google("models/gemini-1.5-flash-latest"),
  schema,
  prompt: "Generate a lasagna recipe.",
});

console.log(object);
```

```ts
// schema.ts
import { z } from "zod";

export const schema = z.object({
    recipe: z.object({
      name: z.string(),
      ingredients: z.array(
        z.object({
          name: z.string(),
          amount: z.string(),
        }),
      ),
      steps: z.array(z.string()),
    }),
  });

type Result = z.infer<typeof schema>;
```

```json
{
  "recipe": {
    "name": "Lasagna",
    "ingredients": [
      {
        "name": "Lasagna noodles",
        "amount": "12"
      },
      {
        "name": "Ground beef",
        "amount": "1 pound"
      },
      {
        "name": "Onion",
        "amount": "1"
      },
      {
```

# AI SDK Tools example

```ts
import { z } from "zod";
import { google } from "./google-model";
import { generateText, tool } from "ai";

const result = await generateText({
  model: google("models/gemini-1.5-flash-latest"),
    tools: {
    weather: tool({
      description: "Get the weather in a location",
      parameters: z.object({
        location: z.string().describe("The location to get the weather for"),
      }),
      execute: async ({ location }) => ({
        location,
        temperature: Math.floor(Math.random() * 30) + 7,
      }),
```


```
[{
  "toolCallId": "vEjzwL9",
  "toolName": "weather",
  "args": {
    "location": "Tel Aviv"
  },
  "result": {
    "location": "Tel Aviv",
    "temperature": 20
  }
}]
```

# Zod auto infer types

```
import { z } from "zod";
import { google } from "@ai-sdk/google";
import { generateText, tool } from "ai";

const result = await generateText({
  model: google("models/gemini-1.5-flash-latest"),
    tools: {
    weather: tool({
      description: "Get the weather in a location",
      parameters: z.object({
        location: z.string().describe("The location to get the weather for"),
      }),
      execute: async ({ location }) => ({
        //               we can infer the type of the location
        location,
        temperature: Math.floor(Math.random() * 30) + 7,
      }),
    }),
  },
  prompt: "What is the weather in Tel Aviv?",
});

console.log(result.toolResults);
```

# AI SDK Tools example with roundtrips

```ts
import { z } from "zod";
import { google } from "./google-model";
import { generateText, tool } from "ai";

const result = await generateText({
  model: google("models/gemini-1.5-flash-latest"),
    tools: {
    weather: tool({
      description: "Get the weather in a location",
      parameters: z.object({
        location: z.string().describe("The location to get the weather for"),
      }),
      execute: async ({ location }) => ({
        location,
        temperature: Math.floor(Math.random() * 30) + 7,
      }),
    }),
```

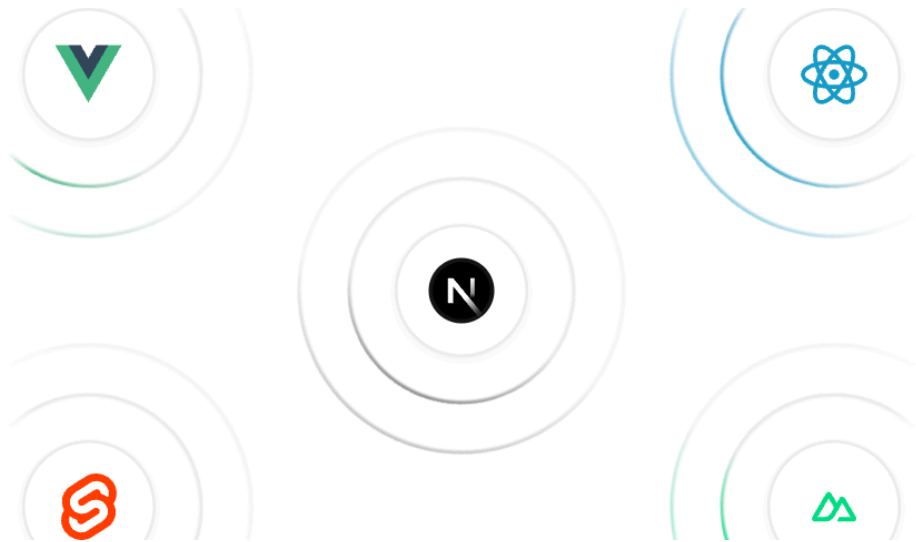
```
The weather in San Francisco is 8 degrees. The weather in Tel Aviv is 8 degrees.
[{
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": ""
    },
    {
      "type": "tool-call",
```

# Think about agents, tasks, real-time data, connect to external APIs...

- agentic - A collection of 20+ tools. Most tools connect to access external APIs such as Exa or E2B.
- browserbase - Browser tool that runs a headless browser

# AI SDK UI

It works with multiple UI frameworks, like React, Solid, Vue, and Svelte.



# Chat UI with useChat

```
"use client";
import React from "react";
import { useChat } from "ai/react";

export default function Chat() {
  const { messages, input, handleInputChange, handleSubmit } = useChat();
  return (
    <div>
      {messages.map((m) => (
        <div key={m.id}>
          {m.role === "user" ? "User: " : "AI: "}
          {m.content}
        </div>
      ))}
      <form onSubmit={handleSubmit}>
        <input
          value={input}
          placeholder="Say something..."
          onChange={handleInputChange}
        />
      </form>
    </div>
  );
```

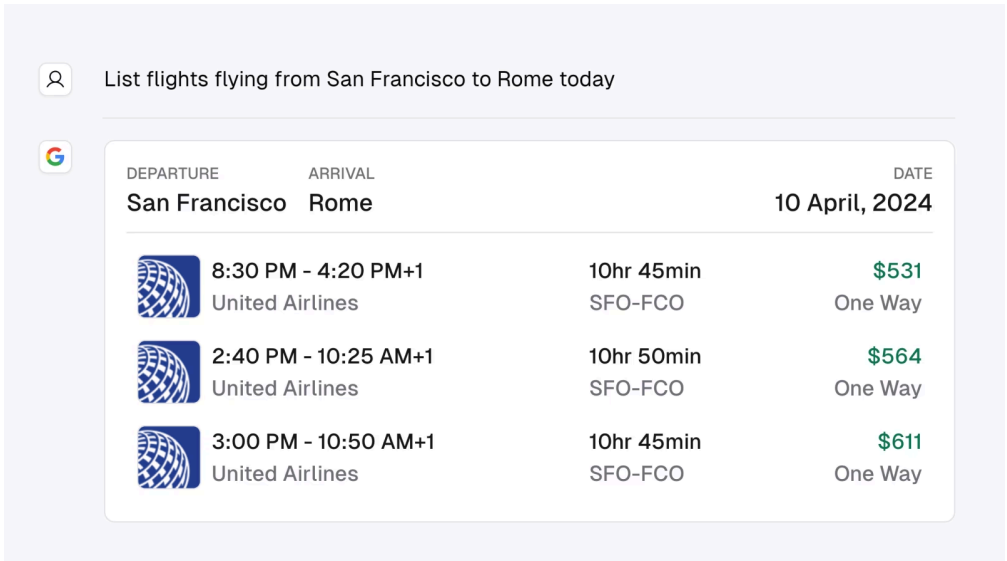
# Chat Route

```ts
// app/api/chat/route.ts
import { google } from "@ai-sdk/google";
import { streamText } from "ai";

export async function POST(request: Request) {
  const { messages } = await request.json();
  const stream = await streamText({
    model: google("models/gemini-1.5-flash-latest"),
    system: "You are a helpful assistant.",
    messages,
  });
  return stream.toAIStreamResponse();
}
```

`useChat` uses SWR internally

# AI SDK RSC

## Creating generative UI with React Server Components



# JavaScript generator functions

**Normal Function:**

Runs from start to finish each time you call it.

**Generator Function:**

Can pause execution and resume later.

**Yield Statement:**

- `yield` pauses and returns a value.
- Resumes after `yield`.

**Iterator Object**

- Returns an iterator object.
- Controls the generator"s execution

```
function* countToThree() {
  console.log("Start");
  yield 1;
  console.log("After first yield");
  yield 2;
  console.log("After second yield");
  yield 3;
  console.log("End");
}

// Create an iterator from the generator function
const iterator = countToThree();

console.log(iterator.next()); // { value: 1, done: false }
console.log(iterator.next()); // { value: 2, done: false }
console.log(iterator.next()); // { value: 3, done: false }
console.log(iterator.next()); // { value: undefined, done: true }
```

# Stream UI - server action

```tsx
// actions.tsx
"use server";
import { streamUI } from "ai/rsc";

export async function streamComponent() {
  const result = await streamUI({
    model: openai("gpt-4o"),
    prompt: "Get the weather for San Francisco",
    text: ({ content }) => <div>{content}</div>,
    tools: {
      getWeather: {
        description: "Get the weather for a location",
        parameters: z.object({
          location: z.string(),
        }),
        generate: async function* ({ location }) {
          yield <LoadingComponent />;
          const weather = await getWeather(location);
          return <WeatherComponent weather={weather} location={location} />;
        },
      },
    },
  });
```

# Stream UI - client component

```tsx
"use client";

import { useState, type ReactNode } from "react";
import { Button } from "@/components/ui/button";
import { streamComponent } from "./actions";

export default function Page() {
  const [component, setComponent] = useState<ReactNode>();

  return (
    <div>
      <form
        onSubmit={async e => {
          e.preventDefault();
          setComponent(await streamComponent());
        }}
      >
        <Button>Stream Component</Button>
      </form>
      <div>{component}</div>
    </div>
  );
}
```

# The future

# Chrome 127 Experimental Built In AI Provider

# Chrome 127 Experimental Built In AI Provider

# Chrome 127 Experimental Built In AI Provider

https://developer.chrome.com/docs/ai/built-in

chrome-ai

```
import { chromeai } from "chrome-ai";
import { generateText } from "ai";

const result = await generateText({
  model: chromeai(),
  prompt: "Tell me a joke.",
});

console.log(result.text);
```

```
AI_LoadSettingError: Browser not support
```

# Translate Text Using Chrome AI

**Your Chrome is not supported, see https://x.com/leeoxiang/status/1799334729365487869**

Enter text to translate

Type your text here...

Select target language

English

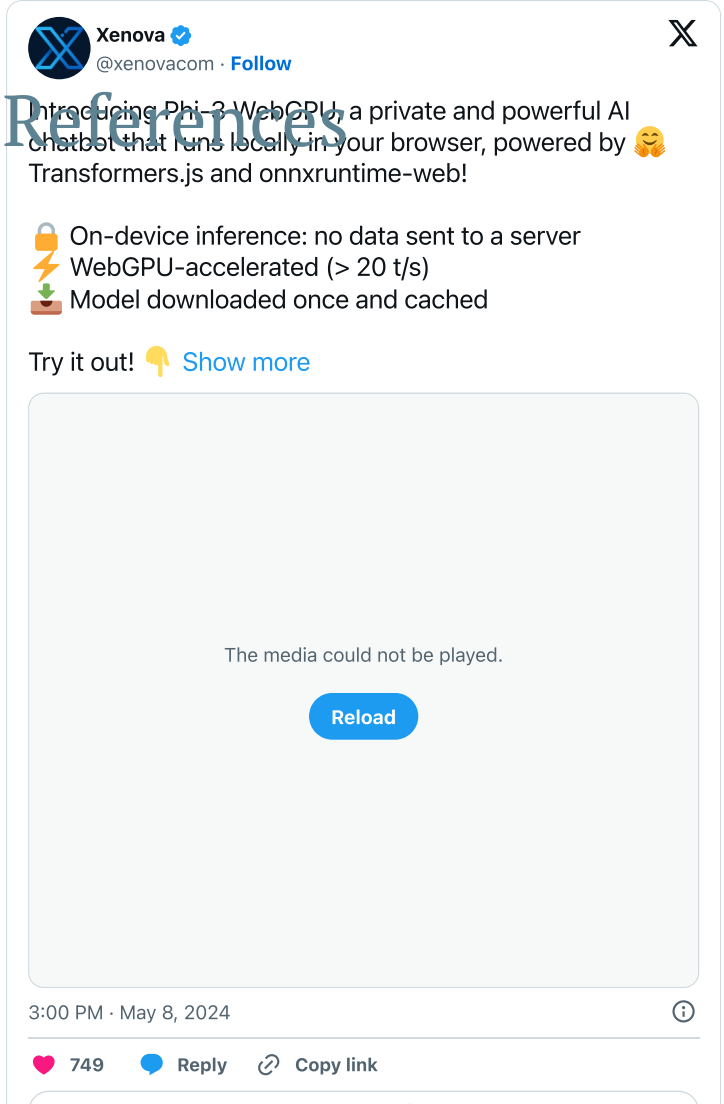
Translate

## Translated Text:

this is translated text

# WebGPU enables to run more AI models in the browser

# References



Xenova
@xenovacom · Follow

Introducing Phi-3 WebGPU, a private and powerful AI chatbot that runs locally in your browser, powered by 🤗 Transformers.js and onnxruntime-web!

🔒 On-device inference: no data sent to a server
⚡ WebGPU-accelerated (> 20 t/s)
📥 Model downloaded once and cached

Try it out! 👇 Show more

The media could not be played.

Reload

3:00 PM · May 8, 2024

749 Reply Copy link

# Further Reading

- Security
- Rate limits
- Langchain
- Prompt engineering
- Retrieval-Augmented Generation (RAG)
- Agents

# Thank you!

- nirtamir.com
- @nirtamir2
- @NirTamir
- @nirtamir2
- Slides
- Blog post

# Glossary

| | |
|---|---|
| AI | Artificial intelligence |
| LLM | Large language model (ChatGPT) |
| RAG | Retrieval-Augmented generation (Adding data) |
| genUI | UI generated by AI |