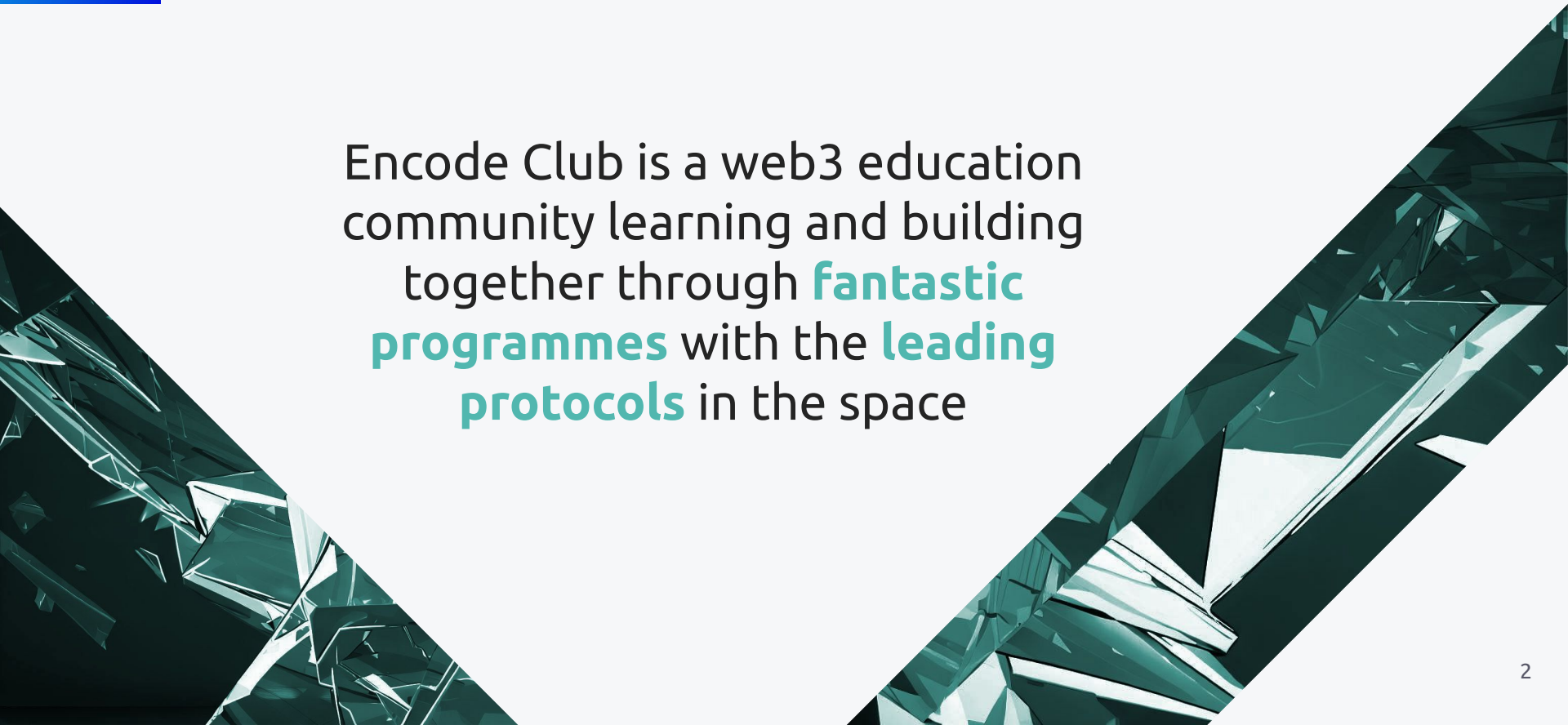
VENTURE MINER

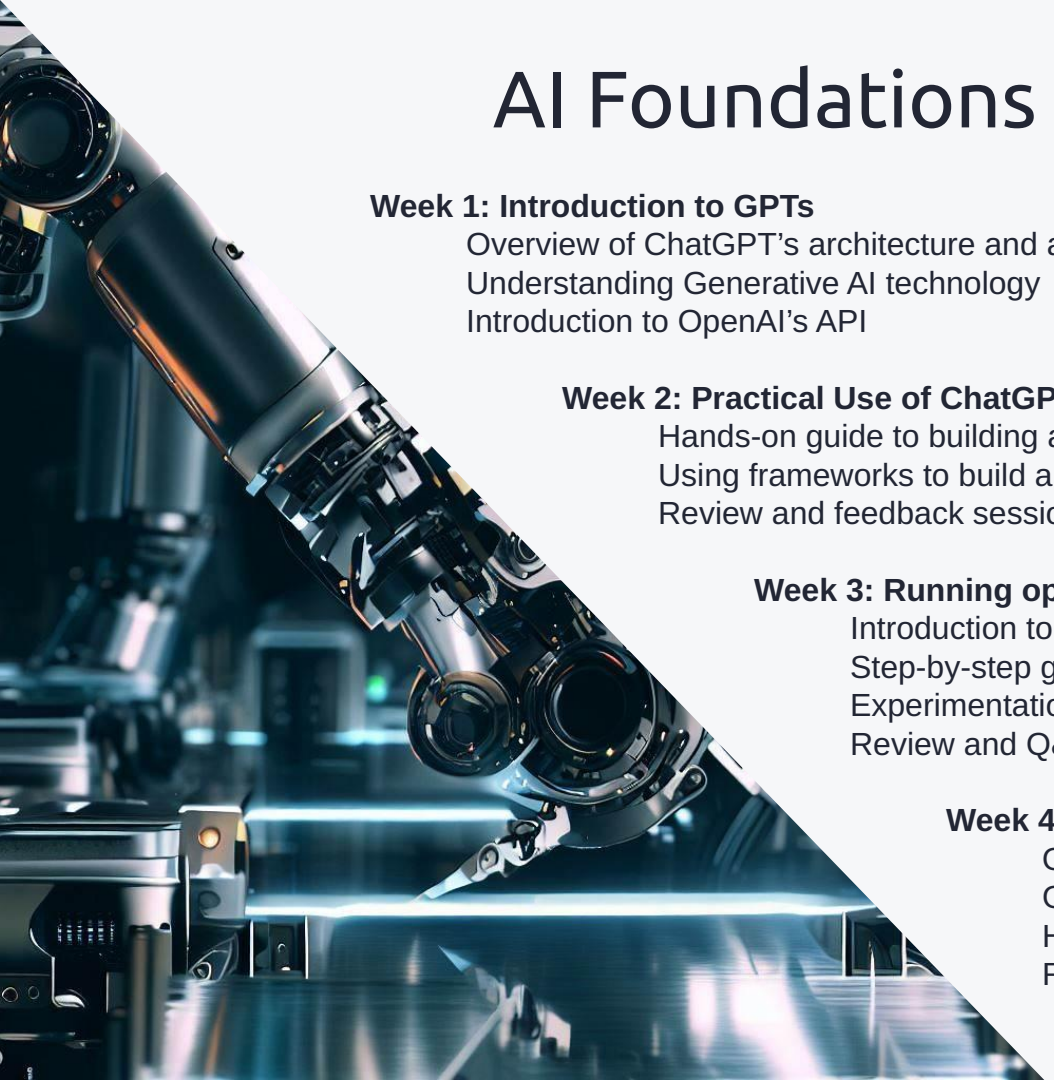
Early stage Web3 and AI
Venture Studio

In partnership with **Encode Club**

Encode Club is a web3 education community learning and building together through **fantastic programmes** with the **leading protocols** in the space



AI Foundations Bootcamp



Week 1: Introduction to GPTs

- Overview of ChatGPT’s architecture and applications
- Understanding Generative AI technology
- Introduction to OpenAI’s API

Week 2: Practical Use of ChatGPT

- Hands-on guide to building a simple Chat app with OpenAI’s API
- Using frameworks to build a simple Chat app
- Review and feedback session on ChatGPT capabilities

Week 3: Running open models locally

- Introduction to open LLMs
- Step-by-step guide to run local LLMs
- Experimentation with text-to-text tasks
- Review and Q&A on running local models

Week 4: Generating Images with Python

- Overview of image generation techniques
- Guide to executing scripts for image generation
- Hands-on experimentation with image generation
- Review and Q&A on image generation techniques

VENTURE MINER

Practical Use of ChatGPT

Topics

- ChatGPT Review
- Hands-on experimentation
  - API Calls
  - OpenAI Platform
  - Using the playground
- Building an application
  - Frontend applications
  - Building a page
  - Using a framework
  - React
  - NextJS
  - Vercel
- Implementing a chat
  - Using Vercel AI SDK
  - Experimenting with V0
  - Editing the page with Copilot
- Conclusion
- Next steps

Hands-on experimentation

API Calls

- https://platform.openai.com/docs/introduction
- https://platform.openai.com/docs/quickstart?context=curl

Do I need a strong computer for running that?

No.
ChatGPT operates like a virtual assistant who works in a remote office.

When you want to ask ChatGPT a question or give it a task, you **send your request** to the API, like sending an email or a message to someone far away.

Your message **travels across the internet** and reaches OpenAI's server. Once ChatGPT receives your request, it processes the message and respond to it back to your device.

This happens **very fast**, almost as if you're chatting with someone in real-time.

Hands-on experimentation

Using OpenAI Playground

- https://platform.openai.com/playground?mode=chat

System: "*You are a knowledgeable and resourceful virtual travel advisor, expertly equipped to assist with all aspects of travel planning. From suggesting hidden gems and local cuisines to crafting personalized itineraries, you provide insightful, tailored travel advice. You adeptly navigate through various travel scenarios, offering creative solutions and ensuring a delightful planning experience for every traveler.*"

User prompt: "*Hello! I'm dreaming of an adventure and need your help. I want to explore a place with breathtaking landscapes, unique culture, and delicious food. Surprise me with a destination I might not have thought of, and let's start planning an unforgettable trip!*"

Configurations:
Temperature 0.75; Max Tokens 500; Top P 0.9; Frequency Penalty 0.5; Presence Penalty 0.6.

Hands-on experimentation

Experimenting with the configurations

You can experiment with different configurations to compare the outputs generated from each:

Agent description: This plays a crucial role in guiding the AI's behavior and response style. Different descriptions can set the **tone**, **personality**, and **approach** of the model.
For instance, using a description such as "You are a creative storyteller" would prompt the AI to adopt a more **imaginative** and **narrative** style, whereas "You are a technical expert" might lead to more **detailed** and **specific** technical information in responses.

Temperature: This controls the **randomness** of the responses.
A lower temperature (e.g., 0.0-0.3) results in more **predictable**, **conservative** responses, ideal for factual or specific queries. A higher temperature (e.g., 0.7-1.0) generates more **creative** and **varied** responses, useful for brainstorming or creative writing.

Max Tokens (Length): This sets the **maximum length** of the response.
For **concise**, straightforward answers, a lower range (e.g., 50-100 tokens) is suitable. For more **detailed** explanations or narratives, a higher range (e.g., 150-500 tokens) can be used.

VENTURE MINER

Hands-on experimentation

Experimenting with the configurations

Frequency Penalty: This reduces the likelihood of the model **repeating the same word** or phrase. A lower setting (e.g., 0.0-0.5) allows **some repetition**, which can be useful for emphasis in writing or speech. A higher setting (e.g., 0.5-1.0) **minimizes repetition**, helpful for generating diverse and expansive content.

Presence Penalty: This discourages the model from **mentioning the same topic or concept** repeatedly. A lower setting (e.g., 0.0-0.5) is suitable for **focused** content on a specific topic, while a higher setting (e.g., 0.5-1.0) encourages the model to **explore** a wider range of topics, useful for brainstorming or exploring different aspects of a subject.

Top P (Nucleus Sampling): This determines the **breadth of word choices** considered by the model. A lower setting (e.g., 0.6-0.8) leads to more **predictable** text, good for formal or factual writing. A higher setting (e.g., 0.9-1.0) allows for more **creativity** and **divergence**, ideal for creative writing or generating unique ideas.

Hands-on experimentation

Recommended Next Steps

- Reading the documentation
- Review the concepts
- Prepare your environment for coding
  - Operating a terminal or *Bash* for running commands
  - Using *VSCode* IDE
  - *Git CLI*
  - *Node* LTS version
  - *NPM* package manager
    - and/or *Yarn*
    - and/or *PNPM*
    - and/or *Bun*

Building an application

Frontend applications

A frontend application is a software program that runs on the user's **browser** and interacts with the user through a graphical user interface (**GUI**).

A frontend application can **display** data, accept user **input**, and perform various **tasks** according to the user's needs.

To integrate **API calls** to a frontend application, you need to use a programming language that can communicate with the AI Provider web server, such as JavaScript.

Instead of coding the JavaScript directly, we'll use **libraries**, **frameworks** and even **AI tools** to speed up the development process and to simplify it as much as possible.

Building an application

Building our first page

For the browser to **understand** and **display** your application's interface, it should be organized inside an **HTML** file.

Let's ask an AI to generate it for us:

> *Generate an HTML file with a simple chat interface and include a script to send the message somewhere when you click the "generate" button*

This simple prompt should be enough for generating the contents of the html file to mimic the GUI of our application.

chat.html > ...

```
<html>
<head>
  <style>···
  </style>
</head>
<body>
  <div class="chat-container">
    <!-- Display the chat messages here -->
  </div>
  <div class="input-container">
    <!-- Input the message here -->
    <input type="text" class="input-box" id="input-box" placeholder="Type your message here...">
    <!-- Click the generate button to send the message -->
    <button class="generate-button" id="generate-button" onclick="generate()">Generate</button>
  </div>
  <script>
    // Define the function to send the message
    function sendMessage(message) {
      // You can write your code here to send the message to the API or somewhere else
      // For example, you can use the fetch function to make an API call
      // You can also display the message and the response in the chat interface
      console.log(message); // For now, just log the message to the console
    }

    // Define the function to generate the message
    function generate() {
      // Get the input box element
      var inputBox = document.getElementById("input-box");
      // Get the input value
      var message = inputBox.value;
      // Check if the message is not empty
      if (message) {
        // Send the message
        sendMessage(message);
        // Clear the input value
        inputBox.value = "";
      }
    }
  </script>
</body>
</html>
```

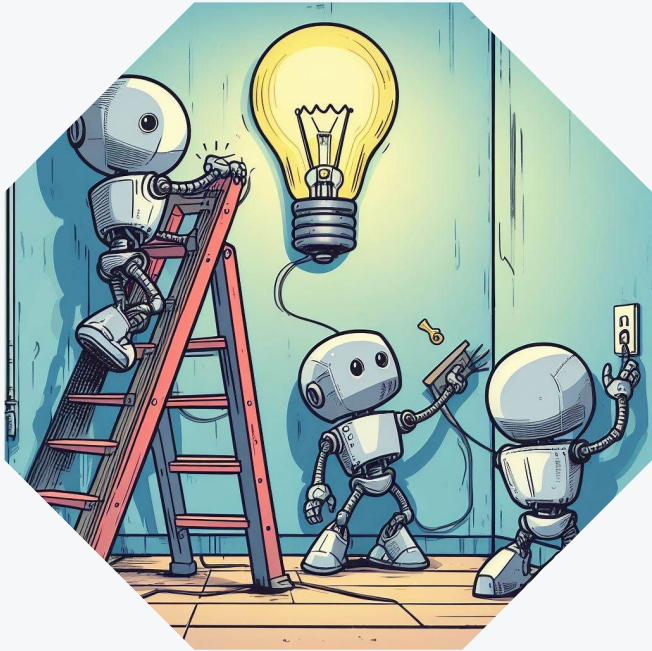
Building an application

Using a framework for development

Instead of dealing with every single aspect of building the frontend application ourselves, we could use a **web development framework** to abstract most of these required tasks and complexities for us.

Much before the code-generation LLMs existed, the development frameworks have been helping devs to automate and scale development tasks in many ways.

By combining **AI Code Generation** and a **good selection of tools**, you can easily produce quite complex applications even without any prior experience to software development at all! (at your own risk, please!)

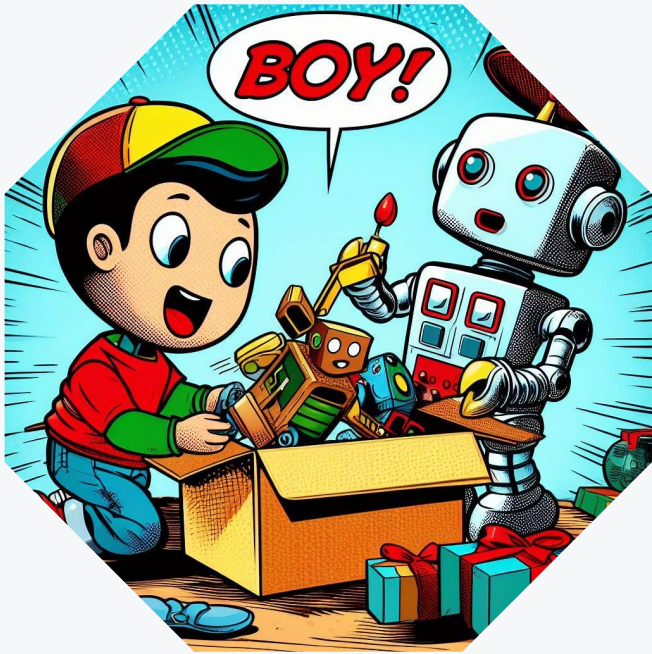


Building an application

Development tooling

- **React**: A library for building web and native user interfaces, allowing the creation of individual components written in JavaScript, designed for seamless combination and reusability. https://react.dev/learn

- **NextJS**: A React framework used for building full-stack web applications, utilizing React Components for UI and offering additional features and optimizations for web development. https://nextjs.org/docs



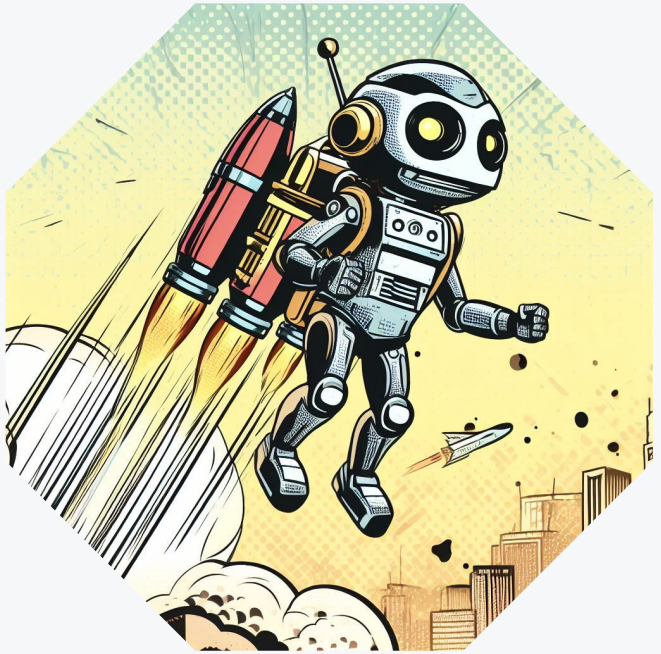
Building an application

Development tooling

- **Vercel AI SDK**: An open-source library designed to help developers build conversational streaming user interfaces in JavaScript and TypeScript, supporting various frameworks and runtime environments. https://sdk.vercel.ai/docs

- **OpenAI API**: The OpenAI API is the crucial component for giving AI capabilities like natural language processing and image generation for our application. https://platform.openai.com/docs/introduction

- **Vercel V0**: A generative user interface system powered by AI. It creates copy-and-paste friendly React code based on shadcn/ui and Tailwind CSS. https://v0.dev/docs



Implementing a chat

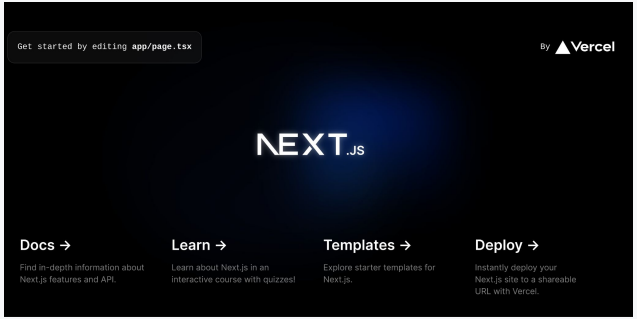
Setting up your project with Vercel AI SDK

- **Tutorial link:** https://sdk.vercel.ai/docs/guides/providers/openai
- Create your project
  - `npx create-next-app my-ai-app`
    - Pick all default options
- Enter the folder created
  - `cd my-ai-app`
- Install the suggested dependencies
  - `npm install ai @ai-sdk/openai`
- Open the project root folder in VSCode
  - `code .`

Implementing a chat

Getting started with Vercel AI SDK

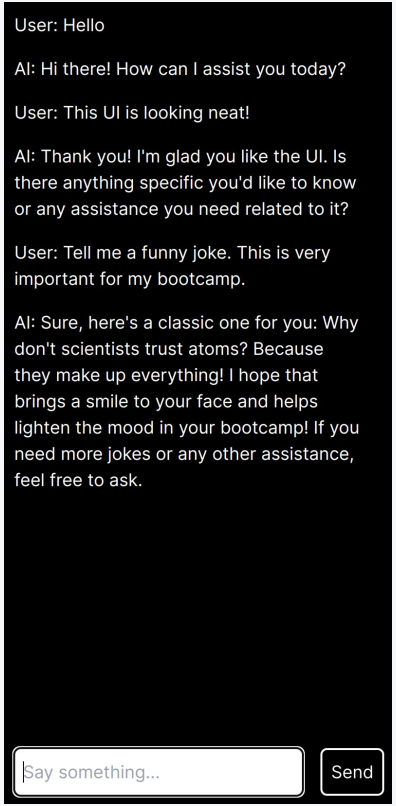
- Running the app:
  - `npm run dev` (dev mode)
- Open the sample application
  - Navigate to http://localhost:3000 in your browser
- Modify the UI with the chat components
  - Open `app/page.tsx` in VSCode
  - Paste the code from the guide at https://sdk.vercel.ai/examples/next-app/chat/stream-chat-completion



Implementing a chat

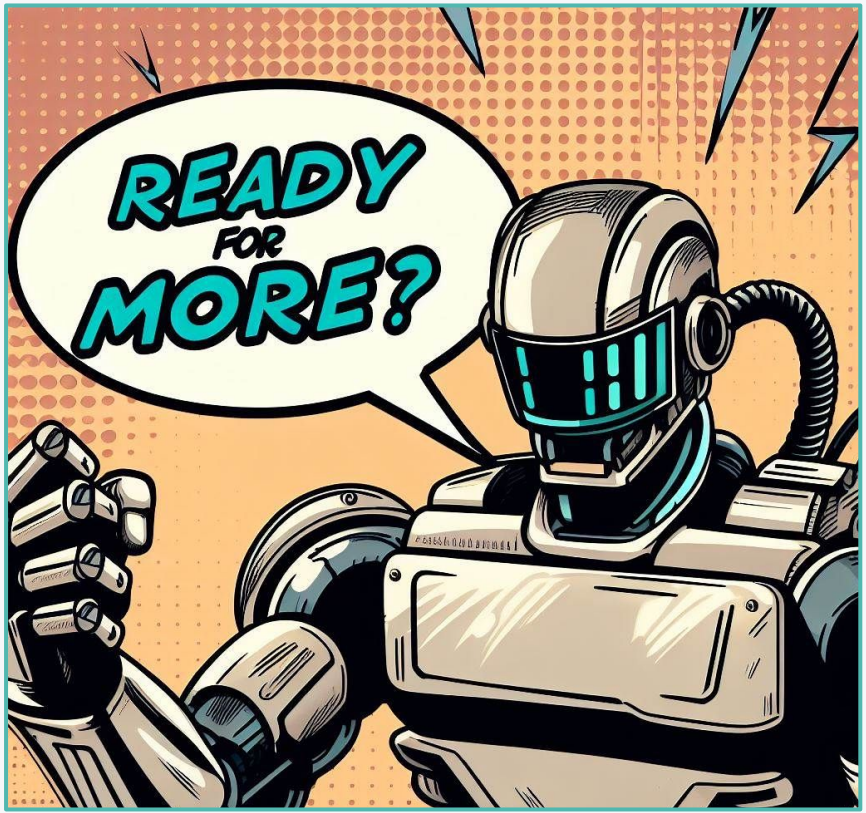
Getting started with Vercel AI SDK

- Create a new file for handling the API call
  - Create file `actions.ts` inside `app` folder
  - Paste the code from the guide at https://sdk.vercel.ai/examples/next-app/chat/stream-chat-completion
- Try out the modified application
- (optional) Modify the UI using Tailwind CSS Library
  - https://tailwindcss.com/



VENTURE MINER

Congratulations!

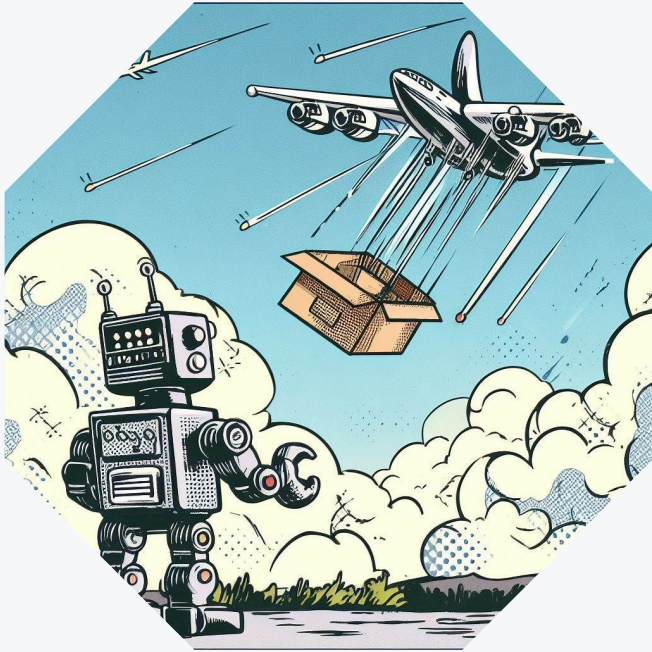


Implementing a chat

Adding content to the page

- To keep editing the page, you should add components https://nextjs.org/learn-pages-router/foundations/from-javascript-to-react/building-ui-with-components
- You could generate these with AI
  - https://v0.dev/
  - https://makereal.tldraw.com/
  - https://www.usegalileo.ai/
  - and many others
- Copy the generated code and paste it into your project
  - My example: https://v0.dev/t/1ZH7H8QCNOm
- Make the adjustments as needed
- Implement the page mechanics and application logic
- Iterate as needed

https://github.com/MatheusDaros/ai-joke-generator-example



Conclusion

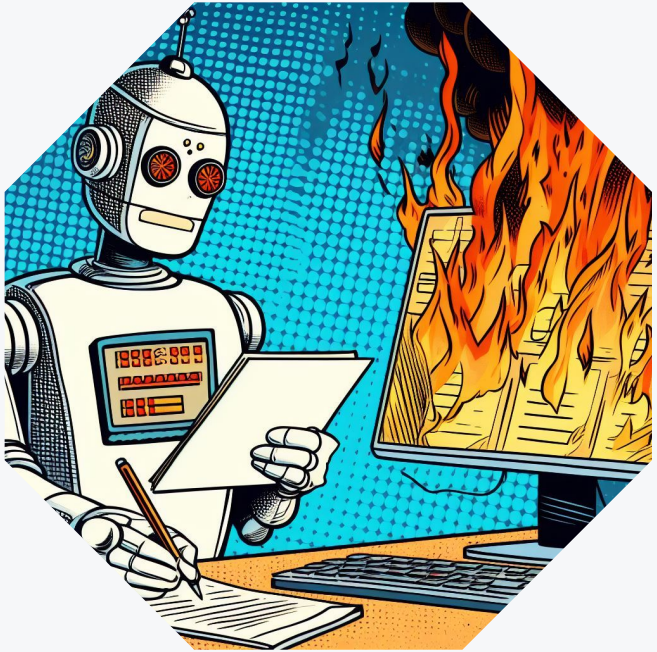
Final considerations

- Using NLP models over API calls
  - Many benefits
    - Easy to implement
    - Easy to maintain
    - Flexible costs
  - Potential risks
    - Pricing models can change
    - Service outages and performance issues
    - Data management and data leaks
    - Censorship and bias
    - Vendor lock-in
- Using your own model
  - Open-source models
  - Running the models
  - Serving the model to applications
  - Managing servers

Conclusion

Next steps

- Create a *Hugging Face* account
  - https://huggingface.co/join
- Request access to LLAMA 3
  - https://ai.meta.com/llama/
    - Fill the form and click on [Download model]
    - Wait some hours or days for confirmation
    - Download all models (links expire in 24 hours)
  - Download at least Llama 3 8B Instruct
  - You can download and keep all other models
- Prepare your development environment



Conclusion

Next steps

- Prepare Python tools
  - *Python*, *Pip* and *Conda*
- (Optional) Prepare Docker tooling
- Setup Text Generation WebUi
  - https://github.com/oobabooga/text-generation-webui
  - Follow the installation instructions
  - Run the *start* script and navigate to the WebUI
  - Download some models in the *Models* tab
    - View more on https://github.com/Troyanovsky/Local-LLM-Comparison-Colab-UI
  - Model recommendations:
    - TheBloke/Llama-2-7B-Chat-GGUF
    - TheBloke/Mistral-7B-OpenOrca-GGUF
    - TheBloke/Nous-Capybara-7B-GGUF
    - Use quantised versions for running with less VRAM
      - Results are slightly less accurate

Thank you for participating