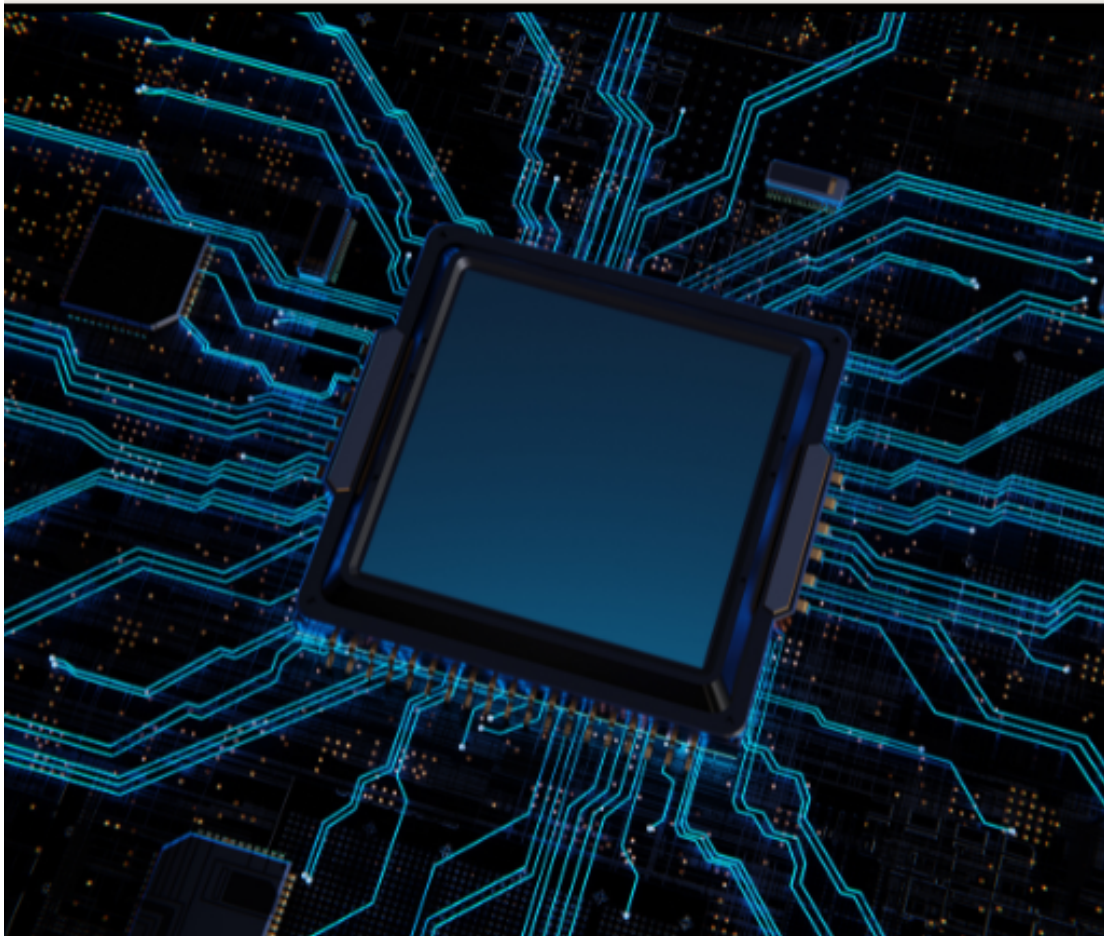
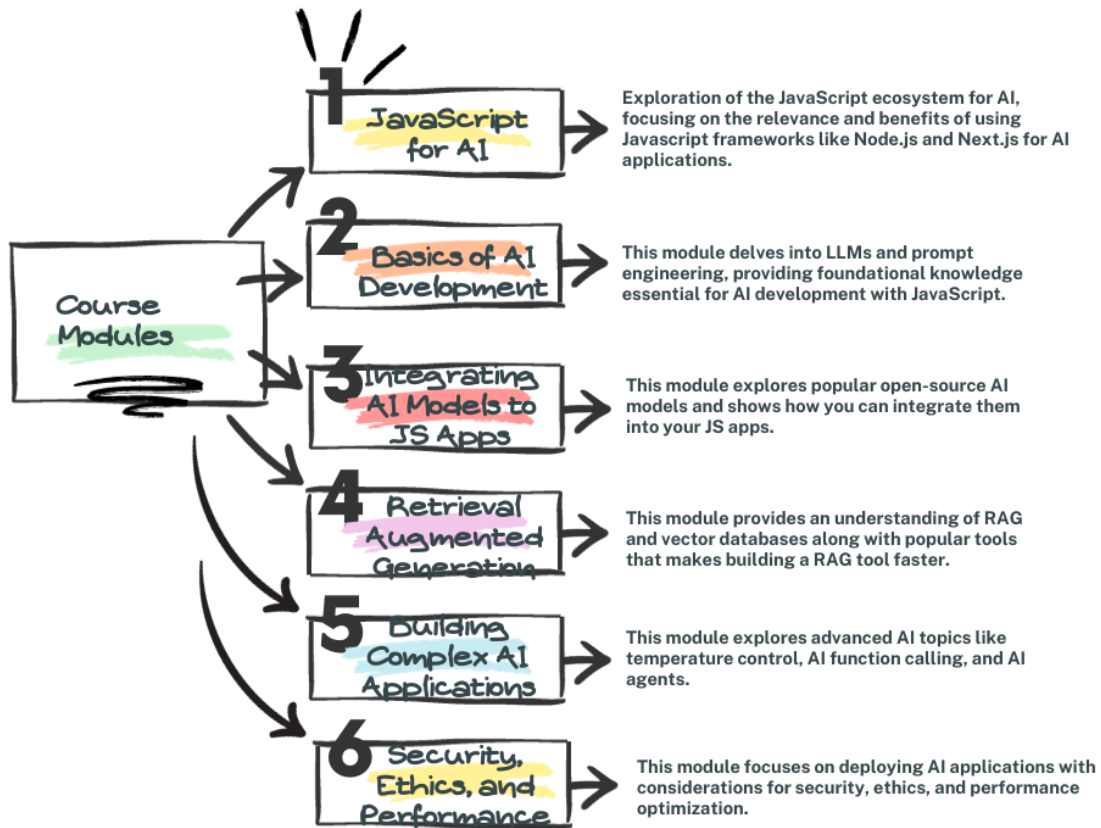


# Practical AI App Development for Javascript Developers

By Mahmud Adeleye



# Practical AI Application Development for Javascript Developers



Title:

Subtitle:

Introduction

Module 1: JavaScript for AI

## Advantages of Using JavaScript Frameworks in AI Application Development

Module Project: Setting up a development environment

1. Setting Up Node.js
2. Setting Up Next.js

Conclusion

Module 2: Basics of AI Development

Introduction to LLMs

Basics of Prompt Engineering and Model Selection

Strategies for Better Results:

Integrating LLMs with JavaScript

Module Project: Building an AI Content Generator with Node.js and OpenAI GPT 3.5 /Mistral 7B Instruct v0.2

## Prompts for Your AI Content Generator

Getting Started with OpenAI and Node.js

Writing the AI Content Generator

Getting Started with Mistral AI and Node.js

Conclusion

## Additional Resources

### Module 3: Integrating AI Models to JS Apps

#### Exploring Open-Source AI Models like Stable-Diffusion and Gemini

Exploring Closed-Source (Commercial) AI Models like Whisper and Dall-E

Working with APIs and External Libraries

#### Advantages of API Endpoints vs SDKs:

Handling AI Model Responses and Interactivity

Project: Build an Image Generation App Using Dall-E Integration

Setting Up the Project

Building the React Frontend

Conclusion

## Additional Resources

### Module 4: Introduction to Retrieval Augmented Generation

Basics of Retrieval Augmented Generation (RAG)

#### How RAG Works

Understanding Embeddings and Vector Databases

Overview of Open-Source RAG Tools of the Trade

Integrating Vercel AI SDK

Implementing RAG in Next.js

### Module Project: Building a Research Assistant Tool with Next.js and Langchain

Our Tech Stack

Prerequisites

Setting Up Next.js

Creating the `.env.local` File

Installing Necessary Packages

Step 1: One-Time Query Retrieval with MemoryVectorStore

Crafting the `/ingest-research` Route

Step 2: Conversational Chat Retrieval with Pinecone

#### Advantages of Pinecone for Conversational Retrieval

Signup for Pinecone

Building the `/chat-research` Route

Building the Frontend

Running the Application

[Conclusion](#)

[Additional Resources](#)

## **Module 5: Building Complex AI Applications**

[Advanced Prompt Engineering and Optimization](#)

[AI Function Calling](#)

[Crafting AI Agents](#)

[Module Projects:](#)

**[Project 1: Basic AI Agent with LangChain and BabyAGI](#)**

**[Project 2: AI Agents with External Data Tools \(LangChain and AutoGPT\)](#)**

**[Project 3: Building a Location-Based Suggestion Agent with OpenAI SDK and Node.js](#)**

[Conclusion](#)

[Resources](#)

## **Module 6: Security, Ethics, and Performance in AI Development**

**[1. Security Practices in AI Applications](#)**

**[Understanding AI-specific Security Risks:](#)**

**[Implementing Robust Security Protocols:](#)**

**[Regular Audits and Compliance:](#)**

**[2. Ethical Considerations and Best Practices In Building AI Apps](#)**

**[3. Privacy-focused Local LLMs](#)**

**[4. Enhancing and Scaling AI Applications - Strategies for Optimal Performance and Deployment](#)**

[Conclusion](#)

[Additional Resources](#)

[Additional Resources - Checklists, Roadmaps, Visual aids](#)

### **[Module Checklists](#)**

[Module 1: JavaScript for AI](#)

[Module 2: Basics of AI Development](#)

[Module 3: Integrating AI Models to JS Apps](#)

[Module 4: Introduction to Retrieval Augmented Generation](#)

[Module 5: Building Complex AI Applications](#)

[Module 6: Security, Ethics, and Performance in AI Development](#)

[List of more projects you can build](#)

# **Title:**

## **Practical AI Application Development for Javascript Developers**

# **Subtitle:**

An open-source course for Javascript developers to get into AI Application development with checklists, projects and demos.

---

## Introduction

I built this course hoping it would be an excellent guide for aspiring AI developers and a valuable resource for the wider JavaScript developer community.

Over the past year, the artificial intelligence industry has been at the forefront of technological advancement worldwide, with so many mind-blowing products and experiments released to the world. For example, OpenAI's ChatGPT was released in late 2022, instantly breaking all consumer records as the fastest-growing consumer product ever.

# ChatGPT Sprints to One Million Users

Time it took for selected online services to reach one million users



\* one million backers \*\* one million nights booked \*\*\* one million downloads  
Source: Company announcements via Business Insider/LinkedIn



The possibilities are endless, and if you're a curious javascript developer like me, your idea notebook is brimming full of ideas you can develop. Now you can imagine how shocked I was when the majority of early AI experiments and tutorials were released in Python. Now, don't get me wrong, I'm a huge fan of Python, and I've used Python to build many AI apps. Plus, Python has been helpful to many data science and machine learning communities in the past. But there has to be something for Javascript too. The stats support this too.

According to the [Stack Overflow Developer Survey 2023](#), JavaScript remains the most commonly used programming language for the tenth year running, a testament to its enduring popularity and versatility. Lately, there has been an increase in published JS libraries and resource materials, and this course is a contribution to that effort.

## Course Structure, Objectives, and Expected Outcomes

Structured to ensure a smooth transition from basic concepts to advanced applications, this course takes a step-by-step approach. It begins with an overview of AI fundamentals, gradually moving towards more complex topics like Retrieval Augmented Generation and ethical AI.

Each course module is designed to build upon the previous one, ensuring a gradual and comprehensive understanding of AI application development using JavaScript. The course balances theoretical knowledge with practical projects, providing a well-rounded learning experience.

As you embark on this journey, how do you envision applying AI in your JavaScript projects? Feel free to share with me your ideas on [Discord](#).

---

# Module 1: JavaScript for AI

**Summary:** Exploration of the JavaScript ecosystem for AI, focusing on the relevance and benefits of using Javascript frameworks like Node.js and Next.js for AI applications.

JavaScript is an excellent tool for creating interactive web pages and apps. As more heavily funded AI startup experiments become ready for the end consumer user, Javascript developers who are able to integrate these experiments into production-ready web and mobile apps stand to gain a lot of career advancements.

## Advantages of Using JavaScript Frameworks in AI Application Development

- **Ubiquity and Community Support:** JavaScript's dominance in web development means a vast community of developers and an abundance of community resources and support.

- **Versatility and Flexibility:** JavaScript, coupled with frameworks like Node.js and Next.js, offers unparalleled flexibility. Node.js, for instance, extends JavaScript's reach to server-side programming, enabling developers to build scalable and efficient AI-backed applications. Its non-blocking, event-driven architecture is particularly suited for handling AI's data-intensive tasks.
- **Seamless Integration with AI Tools:** Many AI tools and libraries are increasingly becoming JavaScript-friendly. AI startups like OpenAI now provide Javascript libraries that allow developers to harness the latest AI advancements in building amazing apps directly within a JavaScript environment.
- **Robust Ecosystem for Frontend and Backend Development:**  
Although I'm familiar with a lot of Javascript frameworks and encourage others to choose anyone that makes them comfortable. This course will heavily use Node.js and Next.js because they make it easier to build a fullstack AI application. With Next.js, developers can enjoy a streamlined development process, making it simpler to build AI-powered applications with universal rendering capabilities - critical for AI applications that require heavy data processing both on the client and server side.

## Module Project: Setting up a development environment



**Note:** Feel free to skip this if you already know how to do this.

Note that this setup might not always be current, for the latest setup instructions please visit

<https://nodejs.org/en> and <https://nextjs.org/docs/getting-started/installation>

Let's roll up our sleeves and get our hands dirty.

### 1. Setting Up Node.js

Node.js is a runtime environment that lets you run JavaScript on the server side. It's essential for building scalable and efficient AI applications.

#### Steps to Set Up Node.js:

1. **Download Node.js:**



- Visit the [Node.js official website](#).
- Choose the latest version appropriate for your operating system.

## 2. Install Node.js:

- Run the downloaded installer and follow the installation prompts.
- Ensure you select the option to install `npm` (Node Package Manager) as well, as it's crucial for managing your JavaScript dependencies.

## 3. Verify Installation:

- Open your terminal or command prompt.
- Run `node -v` and `npm -v` to check the installed versions of Node.js and npm, respectively.
- You should see the version numbers displayed, confirming a successful installation.

### Example Code to Test Node.js Installation:

Create a simple JavaScript file to test your Node.js setup.

- Create a file named `test.js` and add the following code:

```
console.log("Hello, Node.js!");
```

- Run this script in your terminal with:

```
node test.js
```

- If everything is set up correctly, you'll see `Hello, Node.js!` printed in your terminal.

## 2. Setting Up Next.js

Next.js is a React framework that enables server-side rendering and generating static websites for React-based web applications.

There are currently two Next.js setups. The old Pages Router and the new App router setups. I won't be making a recommendation and encourage you to try both options.

Instead of sharing the setup, I would recommend you visit <https://nextjs.org/docs/getting-started/installation> and follow the latest instructions.

## Conclusion

Congratulations! You've successfully set up your development environment with Node.js and Next.js. You're now ready to begin building AI-powered applications.

As you explore the capabilities of Node.js and Next.js, consider how you might leverage these tools in your upcoming AI projects. What innovative features can you envision bringing to life in your web applications?

---

# Module 2: Basics of AI Development

## Summary:

This module delves into LLMs and prompt engineering, providing foundational knowledge essential for AI development with JavaScript.

Imagine a world where your web applications not only respond to user inputs but also understand and interact in a way that feels almost human. This is the world of AI, where Large Language Models (LLMs) like GPT, Mixtral, and Claude have started a revolution. In this module, we're going to dive into the basics of AI development, focusing on how these powerful models can be integrated with JavaScript to create dynamic and intelligent web applications.

## Introduction to LLMs

Large Language Models represent a remarkable leap in the field of AI, offering capabilities that span various modalities, including text, vision, audio, and multimodal (a combination of different types). These models, such as OpenAI's GPT series, Google's BERT, and newer models like Mixtral and Claude, learn to understand context, generate human-like text, and even answer complex questions by being trained on vast datasets, enabling them to perform a wide array of language-related tasks.

1. **Text-Based Models:** Models like OpenAI's GPT (Generative Pre-trained Transformer) series, including the latest iteration, GPT-3, specialize in understanding and generating human-like text. They are trained on vast datasets of text and can perform tasks like translation, summarization, and content generation.
2. **Vision-Based Models:** These models are designed to understand and interpret visual data. For instance, Google's Vision AI uses machine learning to recognize thousands of objects, such as animals, landmarks, and products, in images.
3. **Audio Models:** Audio models are adept at processing and understanding sound data. Whisper, for example, is an automatic speech recognition system that can transcribe and translate speech across multiple languages.
4. **Multimodal Models:** A multimodal LLM is **a type of AI that's trained with multiple types of data**. It's a subset of AI and is similar to Generative AI (GenAI). Multimodal LLM, like Apple's Ferret, combines various types of data, such as image, text, speech, and numerical data, with multiple intelligence processing algorithms. This can help multimodal AI perform better than single-modal AI in many real-world problems.



Visit the HuggingFace dashboard for a comprehensive list of models

[https://huggingface.co/docs/transformers/model\\_doc/mixtral#:~:text=ImageProcessor-,MODELS,-INTERNAL\\_HELPERS](https://huggingface.co/docs/transformers/model_doc/mixtral#:~:text=ImageProcessor-,MODELS,-INTERNAL_HELPERS)

#### MODELS

TEXT MODELS

VISION MODELS

AUDIO MODELS

MULTIMODAL MODELS

REINFORCEMENT LEARNING MODELS

TIME SERIES MODELS

GRAPH MODELS

## Basics of Prompt Engineering and Model Selection

Prompt engineering is a vital skill in the field of AI, just as important as coding. It involves creating queries or inputs that direct language models to generate the desired output. The complexity lies in how the prompt is constructed; a well-structured prompt can utilize the model's abilities to produce highly precise and relevant responses.

### Strategies for Better Results:



OpenAI's Prompt Engineering Guide has a "**Six strategies for getting better results**" section. Check it out here:

<https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results>

When selecting a model for your application, consider factors like the model's complexity, response time, cost, and the nature of the tasks it needs to perform. Each model has its strengths and is suited to different types of applications.

<b>Feature/Model Parameters</b>	<b>GPT-4 1.5 trillion</b>	<b>Bard 1.6 trillion</b>	<b>LLaMa 1.2 trillion</b>	<b>Flan-UL2 20 billion</b>	<b>BLOOM 176 billion</b>
<b>Training Data</b>	Web Text-like corpus	Web Text-like corpus	Web Text-like corpus	Publicly available data	Multilingual web corpus
<b>Training Objectives</b>	Language modeling	Language modeling	Language modeling	Mixture-of-Denoisers (MoD)	Not specified
<b>Special Features</b>	Improved prompt design	Improved prompt design	Improved prompt design	Universally effective across NLP tasks	Open-access, multilingual
<b>How to Access</b>	Via OpenAI API	Via Google Workspace	Application required	Not specified	Open source
<b>Released By</b>	OpenAI	Google	Meta AI	Google Research	Big Science Workshop
<b>Dataset Used for Training</b>	Web Text-like corpus	Web Text-like corpus	Web Text-like corpus	Publicly available data	Multilingual web corpus
<b>Multimodal Capabilities</b>	No	No	No	No	Yes
<b>Multilingual Support</b>	Yes	Limited	Yes	Yes	Yes

Source: <https://medium.com/@aiwizard/llm-models-comparison-gpt-4-bard-llama-flan-ul2-bloom-9ad7c0c56ba5>

## Integrating LLMs with JavaScript

Integrating LLMs into a JavaScript environment involves interacting with these models via APIs/SDKs.



Note:

In the next chapter, we will further explore popular open-source AI models and show how you can integrate them into your JS apps via APIs and external libraries.

## Module Project: Building an AI Content Generator with Node.js and OpenAI GPT 3.5 /Mistral 7B Instruct v0.2

In the landscape of modern web development, the integration of AI through Large Language Models (LLMs) like GPT-3.5 is not just a trend but an evolution.



This blog post will guide you through creating a simple AI content generator using Node.js and two popular Models.

**OpenAI GPT3.5 and Mistral 7B Instruct v0.2 (mistral-tiny).**

By the end, you'll have a functional application that can generate blog posts, stories, or any text-based content.

### Prompts for Your AI Content Generator

#### ▼ Examples

To make the most of your AI content generator, it's essential to understand how prompts can shape the content it produces. Different prompts can lead to various types of content, from creative stories to informational articles. Below is a list of prompt ideas, categorized by content type, to help you explore the versatility of your content generator.

#### 1. Blog Posts

- Technology Trends: "Write a comprehensive blog post about the latest trends in blockchain technology for 2024."

## 2. Creative Writing

- Short Stories: "Tell a short story about a time-traveling adventure in ancient Egypt."
- Poetry: "Compose a poem about the beauty of the ocean at sunset."

## 3. Educational Content

- Science Explainers: "Describe the process of photosynthesis in simple terms for high school students."
- Programming Tutorials: "Write a beginner-friendly tutorial on building a basic website with HTML and CSS."

## 4. Marketing and Sales Copy

- Product Descriptions: "Generate an enticing description for a new eco-friendly yoga mat."

## 5. Personal Development

- Career Advice: "Provide insightful career advice for recent college graduates in the tech industry."

# Getting Started with OpenAI and Node.js

Before diving into the code, make sure you have Node.js installed. If you don't have it, please refer to the first module.

## Step 1: Initialize Your Node.js Project

1. Create a new directory for your project.
2. Navigate to this directory in your terminal and run `npm init -y` to initialize a new Node.js project.
3. Install the OpenAI npm package with `npm install --save openai`.

## Step 2: Setting Up OpenAI API

1. Sign up for an API key from [OpenAI](#). This key is essential to authenticate your requests.

2. Store your API key in a safe place. It's recommended to use environment variables to keep your key secure.

## Writing the AI Content Generator

Now, let's get to the exciting part – coding our AI content generator.

### Step 3: Configure OpenAI API in Your Application

Create a new JavaScript file (e.g., `app.js`) in your project directory and start by importing the OpenAI library:

```
import OpenAI from "openai";  
//const { OpenAI } = require('openai'); (use this if you're getting an error)  
  
const openai = new OpenAI({  
  apiKey: process.env.OPENAI_API_KEY,  
  dangerouslyAllowBrowser: true,  
});
```

Make sure you have set your `OPENAI_API_KEY` in your environment variables.

### Step 4: Writing the Function to Generate Content

Next, we'll write a function that uses OpenAI's SDK `chat.completions.create` method to generate content:

```
async function generateContent(prompt) {  
  try {  
    const chatCompletion = await openai.chat.completions.create({  
      messages: [{ role: "user", content: prompt }],  
      model: "gpt-3.5-turbo",  
    });  
  
    return chatCompletion.choices[0].message.content;  
  } catch (error) {  
    console.error("Error generating content:", error);  
    return null;  
  }  
}
```



```
}  
}
```

In this function, the `prompt` is the input from the user, which we send to the OpenAI generative AI model API. The API then returns the generated content based on this prompt.

## Step 5: Testing Your Application

Now it's time to test your application. You can do this by adding a simple call to `generateContent` at the end of your `app.js`:

```
const blogPostPrompt = "Write a comprehensive blog post about the  
  
generateContent(blogPostPrompt)  
  .then(content => console.log(content))  
  .catch(error => console.error(error));
```

Run your application using `node app.js` and see the AI-generated content in your console.

### ▼ Full Code

```
import OpenAI from "openai";  
//const { OpenAI } = require('openai'); (use this if you're on Node.js)  
  
const openai = new OpenAI({  
  apiKey: process.env.OPENAI_API_KEY,  
  dangerouslyAllowBrowser: true,  
});  
  
async function generateContent(prompt) {  
  try {  
    const chatCompletion = await openai.chat.completions  
      .create({  
        messages: [{ role: "user", content: prompt }],  
        model: "gpt-3.5-turbo",  
      });  
    return chatCompletion.choices[0].text;  
  } catch (error) {  
    console.error("Error generating content:", error);  
  }  
}
```

```

    });

    return chatCompletion.choices[0].message.content;
} catch (error) {
    console.error("Error generating content:", error);
    return null;
}
}

const blogPostPrompt = "Write a comprehensive blog post about

generateContent(blogPostPrompt)
    .then(content => console.log(content))
    .catch(error => console.error(error));

```

## Getting Started with Mistral AI and Node.js



Running the example below requires a Mistral AI API key.

Visit <https://docs.mistral.ai/#api-access> for instructions on how to get one.

### 1. Create a New Node.js Project

- **Initialize your project:** Create a new directory for your project and initialize it with `npm init -y`. This step creates a `package.json` file for managing your project dependencies.

### 2. Install Mistral AI Package

- **Install the package:** Run `npm install @mistralai/mistralai` in your project directory. This command installs the Mistral AI package and adds it to your `package.json`.

### 3. Set Up Your Mistral AI API Key

- **Secure your API key:** Store your Mistral AI API key in an environment variable. For local development, you can use a `.env` file and a package like `dotenv` to load environment variables. Remember, never hardcode your API keys directly into your source code, especially when pushing code to public repositories.

#### 4. Write Your Node.js Application

Create a new file, for example, `app.js`, and write your Node.js application code:

```
require('dotenv').config(); // If you're using dotenv to manage
const MistralClient = require('@mistralai/mistralai').default;

const apiKey = process.env.MISTRAL_API_KEY; // Ensure this is set

const client = new MistralClient(apiKey);

async function generateContent(prompt) {
  try {
    const chatResponse = await client.chat({
      model: 'mistral-tiny',
      messages: [{ role: 'user', content: prompt }],
    });

    return chatResponse.choices[0].message.content;
  } catch (error) {
    console.error('Error:', error);
    return null;
  }
}

// Using the blog post prompt
const blogPostPrompt = "Write a comprehensive blog post about the";

generateContent(blogPostPrompt)
  .then(content => console.log(content))
  .catch(error => console.error(error));
```

#### 5. Run Your Application

- **Execute your script:** Run your application with `node app.js`. This will execute the script, and you should see the AI-generated response in your console.

## Conclusion

You have now created a basic AI content generator using Node.js and the OpenAI API. This application can serve as a foundation for more complex AI-driven projects. You can expand it by adding a web interface, integrating it into a blog platform, or even creating a custom API around it.

As we embrace the era of AI in web development, consider the endless possibilities that such integrations can offer. What innovative applications can you envision for AI in your future projects?

## Additional Resources

1. **Prompt engineering:** <https://platform.openai.com/docs/guides/prompt-engineering>
2. **Mistral Javascript Client:** <https://docs.mistral.ai/platform/client/>
3. **Prompt examples:** <https://platform.openai.com/examples>
4. **Prompt Engineering Guide:** <https://www.promptingguide.ai/>
5. **The LLM Index** - <https://sapling.ai/llm/index>
6. **Open LLM Leaderboard** - [https://huggingface.co/spaces/HuggingFaceH4/open\\_llm\\_leaderboard](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard)
7. **AlpacaEval Leaderboard** - [https://tatsu-lab.github.io/alpaca\\_eval/](https://tatsu-lab.github.io/alpaca_eval/)

---

## Module 3: Integrating AI Models to JS Apps

This module further explores how popular AI models, both open and closed-source, can be seamlessly integrated into JavaScript applications. We will explore APIs, external libraries, and handling AI model responses, and conclude with a hands-on project that brings these concepts to life.

### Exploring Open-Source AI Models like Stable-Diffusion and Gemini

Open-source AI models offer a treasure trove of possibilities for developers. These models stand out for their accessibility and community-driven enhancements. Here are

some popular Open-source AI models:

- **Llama 2** - Developed by Meta. You can explore the demo on HuggingFace: [https://huggingface.co/spaces/ysharma/Explore\\_llamav2\\_with\\_TGI](https://huggingface.co/spaces/ysharma/Explore_llamav2_with_TGI).
- **Falcon** – Developed by the Technology Innovation Institute (TII), UAE. You can also play with their demo on HuggingFace: <https://huggingface.co/spaces/tiiuae/falcon-180b-demo>.
- **MPT-30B and MPT-7B** - Developed by MosaicML. Try it out here <https://huggingface.co/mosaicml/mpt-30b>.
- **Stable Diffusion XL**: Released by Stability AI, this model has made waves in the AI community for its ability to generate high-quality images from textual descriptions. There's an official sandbox you can play with: <https://platform.stability.ai/sandbox/text-to-image> and also a HuggingFace setup: <https://huggingface.co/spaces/stabilityai/stable-diffusion>
- **Gemini**: Released by Google, Gemini is a natural language processing multimodal model excelling in sentiment analysis, text classification, and language generation tasks, providing developers with tools to create applications that understand, interpret, and generate human-like text. **Documentation:** <https://ai.google.dev/>



It's important to note that not all these open-source models are available for commercial use.

Model	License	Commercial use?	Pretraining length [tokens]	Leaderboard score
<a href="#">Falcon-7B</a>	Apache 2.0	✓	1,500B	47.01
<a href="#">MPT-7B</a>	Apache 2.0	✓	1,000B	48.7
Llama-7B	Llama license	✗	1,000B	49.71
<a href="#">Llama-2-7B</a>	Llama 2 license	✓	2,000B	54.32
Llama-33B	Llama license	✗	1,500B	*
<a href="#">Llama-2-13B</a>	Llama 2 license	✓	2,000B	58.67
<a href="#">mpt-30B</a>	Apache 2.0	✓	1,000B	55.7
<a href="#">Falcon-40B</a>	Apache 2.0	✓	1,000B	61.5
Llama-65B	Llama license	✗	1,500B	62.1
<a href="#">Llama-2-70B</a>	Llama 2 license	✓	2,000B	*
<a href="#">Llama-2-70B-chat*</a>	Llama 2 license	✓	2,000B	66.8

<https://huggingface.co/blog/llama2#why-llama-2>

There are also other open-source models like BLOOM and Alpaca. You can check out the Sapling AI's LLM index for a list of other **Open Source LLMs**:

<https://sapling.ai/llm/index#:~:text=Link-,Open Source,-LLMs>

## Exploring Closed-Source (Commercial) AI Models like Whisper and Dall-E

Closed-source AI models, often backed by commercial entities, come with the advantage of advanced capabilities and dedicated support. Models like Whisper and Dall-E are prime examples of this category.

- **Whisper**: Developed by OpenAI, Whisper is an automatic speech recognition system that is perfect for applications requiring transcription and translation.
- **Dall-E**: Also from OpenAI, Dall-E is a cutting-edge AI model capable of generating detailed images from textual descriptions. It's an excellent tool for creative

applications that merge the worlds of art and AI.

There are also other Commercial models like [Claude by Anthropic](#) with Demo: <https://claude.ai/> and [Cohere by Cohere](#).

## Working with APIs and External Libraries

Integrating AI models into JavaScript applications often involves working with APIs and external libraries. This process includes sending requests to AI models and processing their responses. JavaScript, with its extensive library ecosystem, simplifies these integrations, allowing for seamless communication between your application and AI services.

### Advantages of API Endpoints vs SDKs:

- **API Endpoints:**
  - No need to install additional libraries or SDKs.
  - Language agnostic: you can use them with any language that can make HTTP requests.
- **SDKs:**
  - Simplify complex API interactions.
  - Often provide additional utility functions.

For example, instead of using the Mistral client SDK as we did in module 2, we can interact with the chat completions endpoint using [Node.js native fetch API](#) to create our AI content generator as shown below:

```
require('dotenv').config(); //install dotenv

const MISTRAL_API_URL = "https://api.mistral.ai/v1/chat/completions";
const apiKey = process.env.MISTRAL_API_KEY; // Ensure this is set

async function generateContent(prompt) {
  const requestBody = {
    model: 'mistral-tiny',
    messages: [{ role: 'user', content: prompt }]
  }
}
```

```

};

try {
  const response = await fetch(MISTRAL_API_URL, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json',
      'Authorization': `Bearer ${apiKey}`
    },
    body: JSON.stringify(requestBody)
  });

  const data = await response.json();
  return data.choices[0].message.content;
} catch (error) {
  console.error('Error:', error);
  return null;
}
}

// Using the blog post prompt
const blogPostPrompt = "Who is the most renowned French painter";

generateContent(blogPostPrompt)
  .then(content => console.log(content))
  .catch(error => console.error(error));

```

## Handling AI Model Responses and Interactivity

Effectively managing responses from AI models is crucial. This involves parsing the data returned by the models, error handling, and creating interactive interfaces that can showcase the AI's capabilities in a user-friendly manner. Below is a code snippet that parses the data returned, handles errors, and returns the resulting response in a simple



yet effective manner. In future examples and projects, we will show how to create interactive UIs in a way that communicates the AI responses to the user.

## Project: Build an Image Generation App Using Dall-E Integration

This blog post will guide you through setting up a simple React application using Vite, integrating OpenAI's Dall-E API to generate images, and displaying them using a form-based UI.



We will be using OpenAI Documentation on Image Generation:  
<https://platform.openai.com/docs/api-reference/images/create>

### Setting Up the Project

#### 1. Initialize a Vite + React Project:

- Ensure you have Node.js installed.
- Create a new Vite project with React by running `npm init vite@latest my-dall-e-app -- --template react`.
- Navigate to the project directory and install dependencies: `cd my-dall-e-app && npm install`.
- Start the development server: `npm run dev`.

#### 2. Install OpenAI SDK:

- In your project directory, install the OpenAI SDK: `npm install openai`.
- Remember to add your OpenAI API key to a `.env.local` file as `REACT_APP_OPENAI_API_KEY=your_api_key_here`.

### Building the React Frontend

#### 1. Modify `App.jsx`:

- Replace the existing code in `App.jsx` with the following to create a form-based UI for image generation:

```

import React, { useState } from 'react';
import OpenAI from 'openai';

function App() {
  const [prompt, setPrompt] = useState('');
  const [imageUrl, setImageUrl] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const generateImage = async (promptText) => {
    setIsLoading(true);
    try {
      const apiKey = process.env.REACT_APP_OPENAI_API_KEY; //
      const openai = new OpenAI({
        apiKey: apiKey
      });
      const response = await openai.images.generate({
        model: 'dall-e-3',
        prompt: promptText,
      });

      setImageUrl(response.data[0].url);
    } catch (error) {
      console.error('Error:', error);
    } finally {
      setIsLoading(false);
    }
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (prompt) {
      generateImage(prompt);
    }
  };
};

```

```

return (
  <>
    <h1>Dall-E Image Generator</h1>
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={prompt}
        onChange={(e) => setPrompt(e.target.value)}
        placeholder="Enter a prompt for Dall-E"
      />
      <button type="submit" disabled={isLoading}>
        {isLoading ? 'Generating...': 'Generate Image'}
      </button>
    </form>
    {imageUrl && <img src={imageUrl} alt="Generated from D
  </>
);
}

export default App;

```



Note that the button text changes based on the API response status so we can inform the user of what's going on.

```
{isLoading ? 'Generating...' : 'Generate Image'}
```

### ▼ API Endpoint Alternative

Alternatively, instead of using the SDK, you can also decide to query the API endpoint using the native `fetch` API as shown below:

```

const response = await fetch(
  'https://api.openai.com/v1/images/generations',
  {
    method: 'POST',

```

```

    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${apiKey}`,
    },
    body: JSON.stringify({
      model: 'dall-e-3',
      prompt: promptText,
      n: 1,
      size: '1024x1024',
    }),
  }
);

const data = await response.json();
setImageUrl(data.data[0].url);

```

### ▼ Bonus: AI Voice App

If you're interested in building a text-to-speech or speech-to-text-app app, use the newly-gained knowledge and apply it in integrating OpenAI's Audio models.

<https://platform.openai.com/docs/guides/text-to-speech>

## 2. Run and Test Your Application:

- Start your application: `npm run dev`. You should have something like the screenshot below:

# Dall-E Image Generator

Generate Image



- Visit the local development server URL (usually `http://localhost:3000`).
- Try entering different prompts to generate images and, of course, style it using your preferred CSS setup.

With these steps, you've successfully built an image generation application using OpenAI's Dall-E and React. This application demonstrates the power of AI in creative endeavors and how easily it can be integrated into modern web applications.

## Conclusion

As you conclude this module, reflect on the immense potential of integrating various AI models into JavaScript applications. The ability to work with both open-source and closed-source models opens up a world of creativity and innovation. How will you leverage these AI technologies in your future JavaScript projects to create applications that were once thought impossible? Feel free to share with me your ideas on [Discord](#).

## Additional Resources

1. Image Generation with Stable Diffusion and Replicate:

<https://replicate.com/docs/get-started/nodejs>

2. Google AI JavaScript SDK: <https://github.com/google/generative-ai-js>

---

## Module 4: Introduction to Retrieval Augmented Generation

Imagine a world where AI not only understands and generates text but also embarks on a quest for knowledge, seeking information from vast data oceans to enlighten its responses. This is the world of Retrieval Augmented Generation (RAG) - a realm where AI models are no longer confined to their training data but can dynamically retrieve and utilize external information.

In this module, we will explore the intricacies of RAG, understand the role of embeddings and vector databases, and learn about tools that facilitate the development of RAG applications. By the end of this module, you will be equipped to build a research assistant tool using Next.js, harnessing the power of RAG.

### Basics of Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation is a technique where a generative model, like GPT-3.5, is combined with an external knowledge retrieval step in a way that improves the accuracy and reliability of the model's responses. This hybrid approach allows the model to pull in relevant information in real-time, enhancing its responses or generation with details far beyond its training data.

### How RAG Works

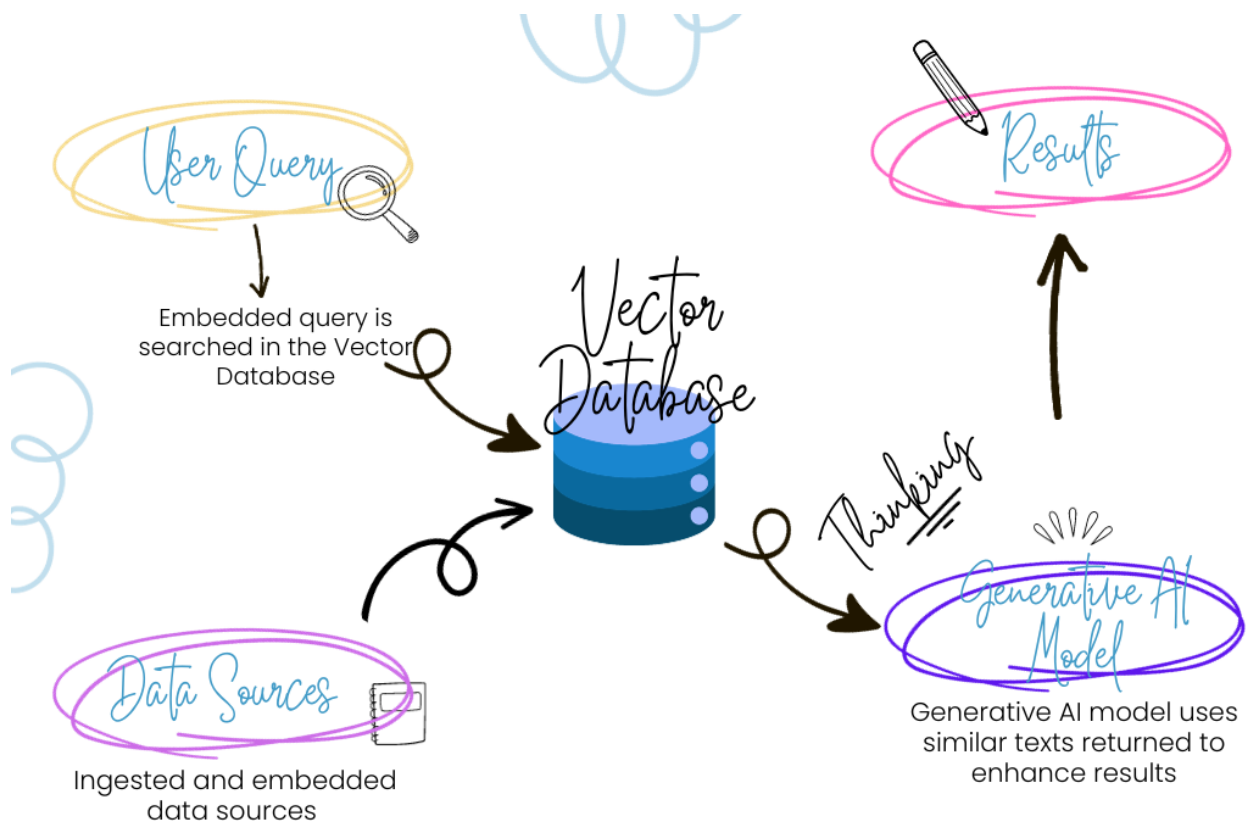
RAG operates in two steps.

#### Retrieval:

First, a query is used to retrieve relevant information from (multiple) documents or data from a large dataset.

## Content generation:

Then, this retrieved data is combined with the query to generate a response. This process ensures that the responses are not just based on learned patterns but are also informed by specific, up-to-date, and relevant external data.



## Understanding Embeddings and Vector Databases

To understand RAG, one must grasp the concept of embeddings and vector databases.

- **Embeddings:** In AI, embeddings are numerical representations of data, whether text, images, or sounds, in a high-dimensional space. For example, word embeddings represent words in a vector space where similar words are closer together. They are crucial for tasks like semantic search, where we need to find the most relevant documents based on their content.

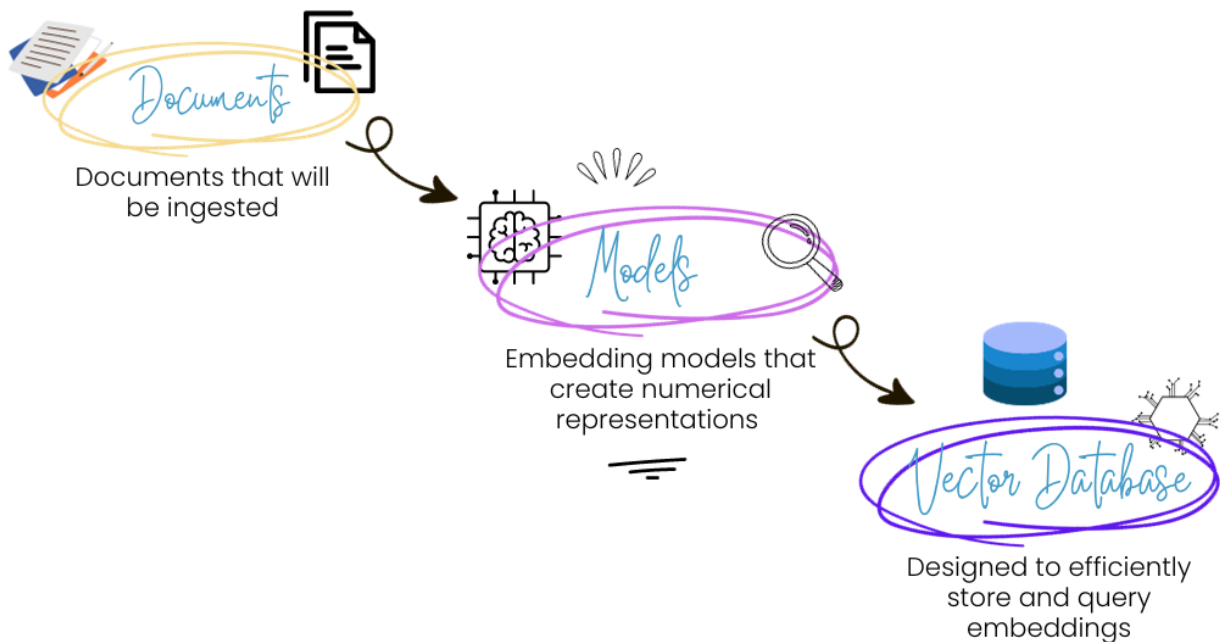
Some of the common Text Embedding models developers use in their AI

applications are developed by OpenAI, Google, Cohere, Jina and Mistral. Since we will be discussing and integrating Langchain into our module project, you can have a look at the list of embedding models available for use:

[https://js.langchain.com/docs/integrations/text\\_embedding](https://js.langchain.com/docs/integrations/text_embedding).

- **Vector Databases:** These databases are designed to efficiently store and query embeddings and enable rapid similarity searches among large collections of embeddings, essential for the retrieval step in RAG. Some of the popular vector databases are FAISS from Meta, Chroma, HNSWLib, Memory, and Pinecone. We will be using the last two for our module project. You can also take a look at the list of vector databases available for integration by Langchain:

<https://js.langchain.com/docs/integrations/vectorstores>



## Overview of Open-Source RAG Tools of the Trade



Several open-source tools have emerged to facilitate the development of RAG systems:

- **Llama Index**: A data framework that facilitates the ingestion of data, storage, and retrieval of embeddings generated from your own data, making it easier to integrate LLMs into your applications with your own data.
- **Embedchain**: This RAG framework helps extract your data into relevant chunks and embeddings, making them useful in powering contextual responses to your queries.
- **Langchain**: A framework for setting up AI applications that can interact with external knowledge bases. It provides an ecosystem of integrations that allow you to ingest data, create and store embeddings, and set up optimized retrieval interfaces.

## Integrating Vercel AI SDK

To integrate the aforementioned models and tools into a JavaScript environment like Next.js, we will be using [Vercel's open-source AI SDK](#) to streamline the AI development process.

## Implementing RAG in Next.js

Incorporating RAG into a Next.js application involves setting up the backend to handle RAG operations and creating frontend components to interact with this backend.

- **Frontend Integration**: Create UI components to take user documents and queries, communicate with the backend, and display the generated responses.
- **Backend Setup**: Implement the logic to retrieve data from the vector database and pass it to the RAG model for response generation.

## Module Project: Building a Research Assistant Tool with Next.js and Langchain

Imagine having a research assistant at your fingertips, one who can understand and provide insights from any PDF document you provide. This is not just a figment of

imagination anymore; it's a reality we're going to build together using Next.js, Langchain, and vector databases like MemoryVectorStore and Pinecone.

## Our Tech Stack

For this project, we will be using Next.js, TypeScript, Tailwind CSS, AI SDK, and Langchain. Additionally, we'll be exploring the use of vector databases by building a two-step research assistant tool, utilizing both MemoryVectorStore for one-time query retrieval and Pinecone for conversational chat retrieval. This dual approach leverages the strengths of each vector database to provide a comprehensive and versatile research tool.

## Prerequisites


Before we dive in, ensure you have Node installed on your machine. You'll also need an OpenAI API Key, which you can get from [OpenAI](#).

## Setting Up Next.js

Start by creating a new Next.js project:

```
npx create-next-app@latest research-assistant
```

Follow the CLI prompts to bootstrap your project. Opt for using the App Router, the `src/` directory, and choose to install Tailwind CSS too.

```
>_ Terminal 
```

```
1  What is your project named? my-app
2  Would you like to use TypeScript? No / Yes
3  Would you like to use ESLint? No / Yes
4  Would you like to use Tailwind CSS? No / Yes
5  Would you like to use `src/` directory? No / Yes
6  Would you like to use App Router? (recommended) No / Yes
7  Would you like to customize the default import alias (@/*)? No / Yes
8  What import alias would you like configured? @/*
```

## Creating the `.env.local` File

In your project directory, create a `.env.local` file to store environment variables:

```
OPENAI_API_KEY="openai_api_key"
```

## Installing Necessary Packages

Install the packages needed for our project:

```
npm install langchain pdf-parse ai react-dropzone @pinecone-database
```

## Step 1: One-Time Query Retrieval with MemoryVectorStore

In the first part of our application, we use MemoryVectorStore to handle one-time queries. This setup is ideal for quick document summarization and extracting key information from uploaded research papers.

## Crafting the `/ingest-research` Route

To begin building our research assistant tool, we need to create a route for ingesting research material, specifically PDF documents. This route, named `/ingest-research`, will be responsible for handling the upload of PDF files and processing them using Langchain to extract meaningful data that our AI can later utilize.

Start by creating a new file `route.ts` under the directory `app/api/ingest-research`. The code in this file will set up an API endpoint in our Next.js application, which will handle the uploading and processing of PDF files.

Here's a detailed look at the code for the `/ingest-research` route:

```
import { PDFLoader } from "langchain/document_loaders/fs/pdf";
import { NextRequest, NextResponse } from "next/server";
import { PineconeClient } from "@pinecone-database/pinecone";
import { OpenAIEmbeddings } from "langchain/embeddings/openai";
import { PineconeStore } from "langchain/vectorstores/pinecone";
import { RecursiveCharacterTextSplitter } from "langchain/text_splitters";
```

```

export async function POST(request: NextRequest) {
  // Extract FormData from the request
  const data = await request.formData();
  // Extract the uploaded PDF file
  const file = data.get("file") as File;

  // Ensure a file is provided
  if (!file) {
    return NextResponse.json({ success: false, error: "No file provided" });
  }

  // Confirm the file is a PDF
  if (file.type !== "application/pdf") {
    return NextResponse.json({ success: false, error: "File is not a PDF" });
  }

  // Load the PDF and split it into smaller documents
  const pdfLoader = new PDFLoader(file);
  const textSplitter = new RecursiveCharacterTextSplitter({
    chunkSize: 1000,
    chunkOverlap: 200,
  });
  const splitDocuments = await pdfLoader.loadAndSplit(textSplitter);

  // Create embeddings for the documents
  const embeddings = new OpenAIEmbeddings({ apiKey: process.env.OPENAI_API_KEY });
  const vectorStore = await MemoryVectorStore.fromDocuments(splitDocuments, embeddings);

  // Set up the RAG model
  const model = new ChatOpenAI({ modelName: "gpt-3.5-turbo", openAIApiKey: process.env.OPENAI_API_KEY });
  const chain = RetrievalQAChain.fromLLM(model, vectorStore.asRetrieval());

  // Generate a summary of the research document
  const response = await chain.call({
    query: "Summarize the primary arguments or findings in this document."
  });
}

```

```
});  
  
return NextResponse.json({ success: true, summary: response.summary });
```

In this setup, the Langchain `RetrievalQAChain` uses the `MemoryVectorStore` for efficient retrieval of relevant document sections, which the `ChatOpenAI` model then summarizes.



The `MemoryVectorStore` is ephemeral, which is why we need to set up Pinecone for our next step.

## Step 2: Conversational Chat Retrieval with Pinecone

The second part of our API route enhances the user experience by enabling a conversational chat interface with the documents. For this, we integrate Pinecone, a scalable vector database perfect for handling conversational context and maintaining state across queries.

### Advantages of Pinecone for Conversational Retrieval

- **Stateful Interactions:** Pinecone helps us maintain the context of a conversation, allowing the AI to provide coherent and consistent responses over a session.
- **Scalability:** It is designed to handle large-scale vector datasets efficiently, making it suitable for extensive research libraries.
- **Real-time Responses:** Pinecone's architecture ensures quick retrieval, essential for maintaining a fluid conversation.

### Signup for Pinecone

For this, sign up for a free-tier starter account at [Pinecone](#). Create an Index with the name of your choice (e.g., `research-assistant`) and set the Dimensions to 1536, aligning with OpenAI model requirements and the details to your `.env` file:

```
PINECONE_API_KEY="pinecone_api_key"  
PINECONE_INDEX_NAME="research-assistant"
```

Add the Pinecone setup as shown below after `// Create embeddings for the documents` line in the `/ingest-research` route.

```
// Initialize the Pinecone client
const pineconeClient = new PineconeClient();
await pineconeClient.init({
  apiKey: process.env.PINECONE_API_KEY ?? "",
  environment: "us-east-1-aws",
});
const index = pineconeClient.Index(process.env.PINECONE_INDEX);

// Store the processed documents in Pinecone
await PineconeStore.fromDocuments(splitDocuments, new OpenAIEmbeddings({
  pineconeIndex,
  maxConcurrency: 5,
  namespace: filename,
  textKey: 'text'
}));
```

Here's the link to the full version of the `/ingest-research` route.

## Building the `/chat-research` Route

The next step in our project is to create the `/chat-research` route. This endpoint will be the core of our research assistant, handling user queries by retrieving relevant information from our vector store and generating insightful responses using OpenAI models.

Create a new file `route.ts` under the directory `app/api/chat-research`. This file will contain the logic for processing user queries and interfacing with our AI model.

Here's the full code for the `/chat-research` route:

```
import { NextRequest, NextResponse } from "next/server";
import { PineconeClient } from "@pinecone-database/pinecone";
import { PineconeStore } from "langchain/vectorstores/pinecone";
import { OpenAIEmbeddings } from "langchain/embeddings/openai";
import { OpenAI } from "langchain/llms/openai";
```

```

import { VectorDBQChain } from "langchain/chains";
import { CallbackManager } from "langchain/callbacks";

export async function POST(request: NextRequest) {
  // Parse the JSON body from the request
  const body = await request.json();

  // Initialize the Pinecone Client
  const pineconeClient = new PineconeClient();
  await pineconeClient.init({
    apiKey: process.env.PINECONE_API_KEY,
    environment: "us-east-1-aws",
  });
  const index = pineconeClient.Index(process.env.PINECONE_INDEX);

  // Prepare the vector store with the Pinecone index
  const vectorStore = new PineconeStore(new OpenAIEmbeddings(),

  // Set up the OpenAI model
  const openAIModel = new OpenAI({
    modelName: "gpt-3.5-turbo",
    apiKey: process.env.OPENAI_API_KEY,
  });

  // Create a Langchain chain for Q&A
  const chain = new VectorDBQChain(openAIModel, vectorStore);

  // Process the query and generate a response
  const response = await chain.call({ query: body.prompt });

  return NextResponse.json({ response });
}

```

## Building the Frontend

The success of our research assistant tool also depends on its user interface. It needs to be simple yet effective, allowing users to easily upload PDF documents and interact with the AI for queries. We will develop this interface using React, leveraging the capabilities of Next.js and integrating a file dropzone for PDF uploads and an input field for AI queries.

### Step 1: Setting Up the Metadata in `layout.tsx`

Begin by updating the `layout.tsx` under the `app/` directory. This file will define the basic metadata for our application, setting the stage for a cohesive and informative UI. Here's a sample setup for your `layout.tsx`:

```
export const metadata = {
  title: "AI-Powered Research Assistant",
  description: "Interact with your research documents through AI",
};
```

### Step 2: Building the Main Page Interface in `page.tsx`

Now, let's focus on the main page of our application, `page.tsx`. This is where users will interact with the tool, upload documents, and ask questions.

Here's a detailed code structure for the `page.tsx`:

```
"use client";

import { useCallback } from "react";
import { useDropzone } from "react-dropzone";
import { useCompletion } from "ai/react";

export default function Home() {
  // When a file is dropped in the dropzone, call the `/api/ingest` endpoint
  const onDrop = useCallback(async (acceptedFiles: File[]) => {
    const file = acceptedFiles[0];

    if (file.type !== "application/pdf") {
      alert("Please upload a PDF");
      return;
    }
  }, []);

  const { openai } = useCompletion({
    endpoint: "/api/ingest",
  });
}
```



```

}

const formData = new FormData();
formData.set("file", file);

    setUploading(true);

    try {
const res = await fetch('/api/ingest-research', {
    method: 'POST',
    body: formData,
});

const result = await res.json();
if (result.success) {
    alert('File uploaded successfully!');
} else {
    alert('File upload failed!');
}
} catch (error) {
    console.error('Error uploading file:', error);
    alert('An error occurred while uploading the file.');
```

```

} finally {
    setUploading(false);
}
}, []);

// Configure react-dropzone
const { getRootProps, getInputProps } = useDropzone({
    onDrop,
});

// Vercel AI hook for generating completions through an AI model
const { completion, input, isLoading, handleInputChange, handle
```

```

    useCompletion({
        api: "/api/chat-research",
```

```

    });

    return (
      <main className="flex min-h-screen flex-col items-center p-4">
        <h1>AI-Powered Research Assistant</h1>
        <div
          {...getRootProps({
            className:
              "dropzone bg-gray-900 border border-gray-800 p-10 rounded"
          })}
        >
          <input {...getInputProps()} />
          <p>Drag 'n' drop a PDF here, or click to select a file</p>
        </div>

        <div className="mx-auto w-full items-center max-w-md py-24">
          <form onSubmit={handleSubmit} className="flex flex-col gap-4">
            <input
              className="w-full max-w-md text-black border border-gray-300 rounded"
              value={input}
              placeholder="Ask a question..."
              onChange={handleInputChange}
            />

            <button
              disabled={isLoading}
              type="submit"
              className="py-2 border rounded-lg bg-gray-900 text-white"
            >
              Submit
            </button>

            <p className="text-center">
              Completion result: {completion === "" ? "Awaiting response" : completion}
            </p>
          </form>
        </div>
      </main>
    );
  }
}

```

```
    </div>
  </main>
);
}
```

In this setup:

- We use `react-dropzone` to create a drag-and-drop area for uploading PDFs.
- The `onDrop` function handles the logic for uploading files to the `/ingest-research` endpoint.
- The `handleSubmit` function sends user queries to the `/chat-research` endpoint and displays the AI's response.
- Styling can be adjusted as per your design preference using CSS or a framework like Tailwind CSS.

## Running the Application

To test your application, run:

```
npm run dev
```

Upload the research documents of your choice (in PDF format), and once the training is complete, you can start querying the AI with prompts related to the uploaded content.

## Conclusion

You've successfully built a full-stack RAG setup capable of learning and chatting with any research document it's trained on. This tool stands as a testament to the power of combining AI with modern web development.

As you explore further, think about how this technology can be expanded. How can you utilize AI to enhance the way we interact with and learn from vast amounts of textual data?

If you have any questions or need assistance, feel free to send a message.

## Additional Resources

1. **Getting Started With Embeddings:** <https://huggingface.co/blog/getting-started-with-embeddings>
  2. **What are embeddings?**  
<https://platform.openai.com/docs/guides/embeddings/what-are-embeddings>
  3. **Embeddings:**  
[https://docs.llamaindex.ai/en/stable/module\\_guides/models/embeddings.html#](https://docs.llamaindex.ai/en/stable/module_guides/models/embeddings.html#)
  4. **Streaming:** <https://sdk.vercel.ai/docs/concepts/streaming>
  5. **AI SDK Templates & Examples:** <https://sdk.vercel.ai/docs#examples>
- 

## Module 5: Building Complex AI Applications

In the ever-evolving landscape of AI, the ability to construct complex applications stands as a hallmark of advanced development skills. In this module, we dive deep into the intricacies of advanced AI topics like temperature control, AI function calling, and the creation of AI agents. Going beyond basic interactions, this section explores how to extend AI models for specialized functions and create autonomous AI agents. This module will not only enhance your understanding but also equip you with the tools to build sophisticated AI-driven applications.

### Advanced Prompt Engineering and Optimization

The foundation of any sophisticated AI interaction lies in its prompt engineering. A well-crafted prompt can steer generative AI models in the desired direction, significantly impacting the quality and relevance of their responses.

- **The Essence of Prompt Design:** Crafting effective prompts involves a keen understanding of the AI's language model. For instance, GPTs respond differently based on the nuances of the prompt given. According to a study by OpenAI, tweaking just a few words in a prompt can lead to markedly different outputs.
- **Temperature Control for Creativity and Precision:** The concept of 'temperature' in AI controls the randomness of the generated responses. A lower temperature yields more predictable and conservative outputs, whereas a higher temperature can generate more creative and diverse responses. This feature allows developers

to tailor the AI's responses to suit the needs of their application, whether it be a creative storytelling assistant or a factual information retriever.

## AI Function Calling

Modern AI models, such as those offered by OpenAI, can perform an array of functions – from translating languages to generating code. These models can self-invoke functions provided and carry out specific tasks like summarization, translation, or even code generation.

Supported by GPT 3.5-turbo and GPT 4 models, the function calling feature is a nice way for developers like you to describe a set of defined functions that the models can intelligently use to connect to external data points, thereby enhancing their responses.

## Crafting AI Agents

An AI agent is an autonomous entity capable of performing tasks, interacting, and learning. Creating an AI agent involves not only technical prowess but also an understanding of user experience and interaction dynamics. Experts believe that AI agents are rapidly evolving and finding their place in various industries, from customer service to personal assistants.

## Module Projects:



For this module, we will build **three AI agents** that will highlight how you can fuse function calling into an AI agent setup to create amazing AI apps capable of running executable code based on user inputs.

This sub-section will guide you through building complex AI applications with increasing levels of complexity. We'll explore integrating **LangChain with BabyAGI, AutoGPT, and the OpenAI SDK** in Node.js, focusing on creating autonomous agents that can perform tasks independently and utilize external data.

Through these projects, you'll gain hands-on experience in building AI agents with varying complexities. Starting from a basic setup, you'll progress to **integrating external data tools and finally create a location-aware suggestion system.**

We will follow the structure below for both AI agents.

- **Designing the Agent:** Map out the functionalities and user interactions for your AI agent. This could range from a customer service bot to a more complex assistant capable of handling various tasks.
- **Implementation:** Utilize JavaScript and appropriate AI SDKs to bring your AI agent to life. Integrate advanced prompt engineering techniques and ensure your agent can handle function calls effectively.
- **Testing and Iteration:** Test your AI agent with various scenarios and refine its capabilities based on feedback and performance.

## Project 1: Basic AI Agent with LangChain and BabyAGI

In this tutorial, we'll create a simple AI agent that independently crafts a culinary tour itinerary in Paris without relying on external data sources. Remember to compare the results here and that of project 2.

### 1. Setting Up:

- Initialize a Node.js project and install LangChain with `npm install langchain @langchain/openai`.
- Import necessary modules from LangChain.

```
import { BabyAGI } from "langchain/experimental/babyagi";
import { MemoryVectorStore } from "langchain/vectorstores";
import { OpenAIEmbeddings, OpenAI } from "@langchain/openai";
```

### 2. Building the Agent:

- BabyAGI uses the OpenAI model to generate a culinary itinerary.
- The MemoryVectorStore manages embeddings for task processing.
- The AI is tasked with creating a unique Parisian culinary experience

```
import { BabyAGI } from 'langchain/experimental/babyagi';
import { MemoryVectorStore } from 'langchain/vectorstores/memory';
import { OpenAIEmbeddings, OpenAI } from '@langchain/openai';
```

```

const vectorStore = new MemoryVectorStore(new OpenAIEmbedding

const babyAGI = BabyAGI.fromLLM({
  llm: new OpenAI({ temperature: 0 }),
  vectorstore: vectorStore,
  maxIterations: 3,
});

async function generateParisCulinaryItinerary() {
  const response = await babyAGI.call({ objective: "Generate
console.log("Culinary Tour Itinerary in Paris:", response)
}

generateParisCulinaryItinerary();

```

## Project 2: AI Agents with External Data Tools (LangChain and AutoGPT)

This tutorial will enhance the AI agent's ability to use external data tools for an enriched culinary tour in Paris. We will also be using AutoGPT here to add a new flavor. If you want to use BabyAGI for this, [check out the documentation](#).



We will need the Serp API key for this tutorial. You can get yours here: [https://serpapi.com/users/sign\\_up?plan=free](https://serpapi.com/users/sign_up?plan=free). They have a free plan that should be okay for this tutorial.

### 1. Setting Up:

- Initialize a Node.js project and install LangChain with `npm install @langchain/openai @langchain/community`.
- Import necessary modules from LangChain.

```
import { AutoGPT } from 'langchain/experimental/autogpt';
import { ReadFileTool, WriteFileTool, SerpAPI } from 'langchain/tools';
import { NodeFileStore } from 'langchain/stores/file/node';
import { HNSWLib } from '@langchain/community/vectorstores/hnswlib';
import { OpenAIEmbeddings, ChatOpenAI } from '@langchain/openai';
```

### Integrating External Data Tools:

- Utilize tools like `ReadFileTool`, `WriteFileTool`, and `SerpAPI` for external data interaction.

### 2. Building the Enhanced Agent:

- AutoGPT combines LangChain tools with OpenAI's language model.
- External data tools enhance the agent's ability to provide detailed and context-rich itineraries.

```
import { AutoGPT } from 'langchain/experimental/autogpt';
import { ReadFileTool, WriteFileTool, SerpAPI } from 'langchain/tools';
import { NodeFileStore } from 'langchain/stores/file/node';
import { HNSWLib } from '@langchain/community/vectorstores/hnswlib';
import { OpenAIEmbeddings, ChatOpenAI } from '@langchain/openai';

const store = new NodeFileStore();
const tools = [
  new ReadFileTool({ store }),
  new WriteFileTool({ store }),
  new SerpAPI(process.env.SERPAPI_API_KEY, {
    location: "Paris, France",
    hl: "fr",
    gl: "fr",
  }),
];

const vectorStore = new HNSWLib(new OpenAIEmbeddings());
const autogpt = AutoGPT.fromLLMAndTools(
```



```

new ChatOpenAI({ temperature: 0 })),
tools,
{
  memory: vectorStore.asRetriever(),
  aiName: "Foobar",
  aiRole: "Assistant",
}
);

async function runEnhancedAgent() {
  const response = await autogpt.run(["Generate an enriched c
  console.log("Enhanced Culinary Tour Itinerary in Paris:", r
}

runEnhancedAgent();

```

## Project 3: Building a Location-Based Suggestion Agent with OpenAI SDK and Node.js

In this tutorial, we'll create a Node.js application that uses OpenAI's function-calling capability to suggest activities or foods based on the user's location. Our app will be simple: it will determine the user's location and then generate suggestions accordingly.

### Setting Up Your Project

#### 1. Initialize a Node.js Project:

- Create a new directory for your project and run `npm init -y` to initialize a Node.js project.
- Install the necessary packages: `npm install openai node-fetch dotenv`.

#### 2. Setting Up Environment Variables:

- In your project directory, create a file named `.env`.
- Add your OpenAI API key: `OPENAI_API_KEY=your_openai_api_key_here`.

### Building the Application

#### 3. Fetch Location and Set Up OpenAI

- The `fetchUserLocation` function retrieves the user's location based on their IP address.
- **Setting Up OpenAI:** We configure the OpenAI SDK with the API key and define the tools (functions) that the AI can suggest calling.

```
import OpenAI from "openai";
import dotenv from 'dotenv';
dotenv.config();

const openAI = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
  dangerouslyAllowBrowser: true,
});

async function fetchUserLocation() {
  const locationResponse = await fetch("https://ipapi.co/json");
  return await locationResponse.json();
}

const aiTools = [
  {
    type: "function",
    function: {
      name: "fetchUserLocation",
      description: "Determines user location based on IP",
      parameters: {
        type: "object",
        properties: {},
      },
    },
  },
];

const toolset = {
```

```
    fetchUserLocation,  
  };
```

#### 4. Creating the Agent Function and Generating Suggestions

- The `agent` function is where the magic happens. It communicates with OpenAI to get suggestions on which function to call and then executes the suggested function.

```
async function agent(inputText) {  
  let chatHistory = [  
    {  
      role: "system",  
      content: `You are a smart assistant. Use the provided tools to help the user.`  
    },  
    {  
      role: "user",  
      content: inputText,  
    },  
  ];  
  
  for (let attempt = 0; attempt < 5; attempt++) {  
    const aiResponse = await openai.chat.completions.create({  
      model: "gpt-4",  
      messages: chatHistory,  
      tools: aiTools,  
    });  
  
    const { finish_reason, message } = aiResponse.choices[0].message;  
  
    if (finish_reason === "tool_calls" && message.tool_calls) {  
      const functionName = message.tool_calls[0].function.name;  
      const calledFunction = toolset[functionName];  
      const functionArgs = JSON.parse(message.tool_calls[0].function.arguments);  
      const functionArgsArr = Object.values(functionArgs);  
      const functionResponse = await calledFunction.apply(  
        null,  
        functionArgsArr,  
      );  
      chatHistory.push({  
        role: "assistant",  
        content: functionResponse,  
      });  
    }  
  }  
}
```

```

        functionArgsArr,
    );

    chatHistory.push({
        role: "function",
        name: functionName,
        content: `
            The result of the last function was this:
            ${functionResponse}
        `
    });
} else if (finish_reason === "stop") {
    chatHistory.push(message);
    return message.content;
}
}
return "No conclusive suggestion could be generated. Please
}

```

## 5. Running Your Application:

To run your application:

- **Start the Application:** Use `node yourfilename.js` in your terminal.
- **Interact with the AI:** The script will execute, and you should see food suggestions based on the detected location in your console.

```

async function main() {
    const userQuery =
        "Could you suggest some local delicacies based on where I am?";
    const suggestion = await agent(userQuery);
    console.log("Suggestion:", suggestion);
}

main();

```

By following these steps, you've just created an intelligent agent capable of making function calls to provide real-world information and suggestions. This agent exemplifies how AI can be leveraged to create interactive and dynamic applications.

#### ▼ Full code

```
import OpenAI from "openai";
import dotenv from 'dotenv';
dotenv.config();

const openAI = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
  dangerouslyAllowBrowser: true,
});

async function fetchUserLocation() {
  const locationResponse = await fetch("https://ipapi.co/json");
  const locationData = await locationResponse.json();
  return locationData;
}

const aiTools = [
  {
    type: "function",
    function: {
      name: "fetchUserLocation",
      description: "Determines user location based on IP",
      parameters: {
        type: "object",
        properties: {},
      },
    },
  },
];
```

```

const toolset = {
  fetchUserLocation,
};

async function agent(inputText) {
  let chatHistory = [
    {
      role: "system",
      content: `You are a smart assistant. Use the provided t
    },
    {
      role: "user",
      content: inputText,
    },
  ];

  for (let attempt = 0; attempt < 5; attempt++) {
    const aiResponse = await openai.chat.completions.create({
      model: "gpt-4",
      messages: chatHistory,
      tools: aiTools,
    });

    const { finish_reason, message } = aiResponse.choices[0];

    if (finish_reason === "tool_calls" && message.tool_calls) {
      const functionName = message.tool_calls[0].function.name;
      const calledFunction = toolset[functionName];
      const functionArgs = JSON.parse(message.tool_calls[0].function.arguments);
      const functionArgsArr = Object.values(functionArgs);
      const functionResponse = await calledFunction.apply(
        null,
        functionArgsArr,
      );

      chatHistory.push({

```

```

        role: "function",
        name: functionName,
        content: `
            The result of the last function was this: ${
                functionResponse,
            }
        `;
    });
} else if (finish_reason === "stop") {
    chatHistory.push(message);
    return message.content;
}
}
return "No conclusive suggestion could be generated. Please
}

async function main() {
    const userQuery =
        "Could you suggest some local delicacies based on where I
    const suggestion = await agent(userQuery);
    console.log("Suggestion:", suggestion);
}

main();

```

## Conclusion

With the skills to build complex AI agents and utilize advanced AI techniques, the possibilities are endless. If you want to build more advanced AI agents, check out the links listed in the resources section.

Reflecting on your journey through this module, what kind of innovative AI agent do you envision creating that could revolutionize the way we interact with technology? Feel free to share with me your ideas on [Discord](#).

## Resources

1. [https://js.langchain.com/docs/use\\_cases/autonomous\\_agents/baby\\_agi](https://js.langchain.com/docs/use_cases/autonomous_agents/baby_agi)
  2. [https://cookbook.openai.com/examples/how\\_to\\_build\\_an\\_agent\\_with\\_the\\_node\\_sdk](https://cookbook.openai.com/examples/how_to_build_an_agent_with_the_node_sdk)
  3. [https://js.langchain.com/docs/modules/agents/how\\_to/custom\\_agent](https://js.langchain.com/docs/modules/agents/how_to/custom_agent)
  4. <https://js.langchain.com/docs/integrations/tools/>
  5. <https://vercel.com/templates/next.js/agent-gpt>
  6. <https://www.mercity.ai/blog-post/advanced-prompt-engineering-techniques>
- 

## Module 6: Security, Ethics, and Performance in AI Development

This module, an essential part of this course, delves into the critical aspects of security, ethics, and performance in AI development. It's a guide to building AI applications that are not only powerful and efficient but also secure, ethical, and respectful of user privacy.

### 1. Security Practices in AI Applications

As we integrate AI models into apps used by millions of users across the world, it's important that we recognize the unique security challenges that have emerged and how we can prevent or at least mitigate them.

The first step in doing that is:

#### Understanding AI-specific Security Risks:

Some of the risks that are emerging are:

- **Adversarial Attacks:** These are efforts to fool AI models with deceptive data. It's crucial to understand how these attacks can manipulate AI decision-making.
- **Data Poisoning:** This involves corrupting the training data of an AI model. The integrity of data is paramount, as poisoned data can lead to flawed or biased AI outputs.



- **AIJacking:** An attack where an old model or dataset name is registered by an attacker, leading to the unintentional use of malicious data. You can read this amazing expose on the matter: [Legit Discovers "AI Jacking" Vulnerability in Popular Hugging Face AI Platform.](#)

A good way to better your security posture is by going through the resources on AI security prepared by The Open Worldwide Application Security Project (OWASP). I have linked some of these in the resources section of this module.

## Implementing Robust Security Protocols:

To counter these AI-specific and other web development security risks, robust security protocols are essential:

- **Implementing Real-Time Personally Identifiable Information Safeguard:** Implementing real-time protections for personally identifiable information is critical. Tools that anonymize or redact PII in AI interactions are essential in safeguarding user privacy.
- **Encryption and Secure APIs:** Implement HTTPS in Node.js applications to secure data transmission.
- **Secure Authentication Methods:** Use modern authentication protocols like JWT (JSON Web Tokens) for secure user authentication.
- **Data Minimization and Anonymization:** Reduce access to sensitive data, create anonymized datasets for secondary uses, and document the purpose before collection.

## Regular Audits and Compliance:

It is important to conduct regular security audits, comply with data protection regulations, and employ tools to automate these processes.

- **Adherence to Data Protection Regulations:** Compliance with GDPR, CCPA, and other data protection laws is crucial. This involves regular audits and updates to security practices.
- **OWASP AI Security and Privacy Guide:** Familiarize with the OWASP guide for AI security, focusing on risks related to AI supply chain attacks and vulnerabilities.

As AI becomes more integrated into software development, understanding and implementing robust security measures is not just a technical necessity but a moral imperative. How can JavaScript developers contribute to this evolving landscape, ensuring their AI applications are as secure as they are innovative?

## 2. Ethical Considerations and Best Practices In Building AI Apps

As we build AI apps, especially for large numbers of users, understanding and implementing ethical considerations and best practices becomes increasingly important.

Here are some tips to implement while paying attention to some key Principles of AI Ethics: **Transparency, Fairness, Accountability, Reliability, Security, and Privacy.**

- **Identifying and Mitigating Biases:** Bias in AI can stem from skewed data sets or preconceived notions held by developers. Implementing diverse training datasets and conducting regular bias audits are crucial.
- **Frameworks for Ethical Decision-Making:** Introduce ethical frameworks like the Asilomar AI Principles, which emphasize safety, privacy, and fairness.
- **Ongoing Monitoring and Assessment:** Continually assess AI systems for ethical integrity, adapting and updating as societal norms and values evolve.

## 3. Privacy-focused Local LLMs

AI teams are now focusing on developing private, LLM-focused solutions that can process data locally. This is crucial for individuals and organizations who handle sensitive data and need to comply with regulations. As privacy concerns continue to grow, these solutions are becoming increasingly important.

### Advantages of Local Data Processing:

- **Reduced Data Breach Risks:** By processing data on local servers, the risk of external data breaches is significantly lowered.
- **Compliance with Privacy Regulations:** Local processing aids in compliance with stringent data protection laws like GDPR and CCPA.

Let's explore some of the available solutions in the market:

- **GPT4All:** Developed by NomicAI, this solution offers open-source large language models that run locally on CPUs and GPUs, making advanced AI accessible without compromising data privacy.
- **privateGPT:** Focuses on private interactions with documents using GPT, ensuring data privacy and preventing data leaks.
- **AnythingLLM:** An open-source, customizable enterprise-ready document chatbot providing an local alternative to the closed-source, browser-based ChatGPT experience.

## 4. Enhancing and Scaling AI Applications - Strategies for Optimal Performance and Deployment

Here are some best practices for deploying and scaling AI applications:

- **Model Optimization:** Choose the best model based on evaluation benchmarks relevant to your specific task and data. You can take a look at some of the leaderboards we've shared through this course to inform your decision.
- **User Experience and Interaction Design:** Ensure the application is user-friendly and intuitive, with a focus on interaction design for AI systems.
- **Exploring LLM Frameworks:** Encourage experimentation with various LLM frameworks like LlamaIndex, LangChain, and EmbedChain to find the best fit for the application.
- **Deployment Platforms Selection:** Choose the most suitable platform for deployment based on the application's needs. Know when to use Serverless, Server, and Background workers for, e.g., Discord AI bots.
- **CI/CD Automation for Streamlining Deployment:** Implement CI/CD pipelines for automated testing, integration, and deployment, enhancing deployment efficiency and reliability.
- **Continuous Learning and Best Practices:** Emphasize the importance of staying informed about the latest trends, techniques, and best practices in AI development.

## Conclusion

As we navigate through the complexities of AI development, we must remember that with great power comes great responsibility. Security, ethics, and performance are not just checkboxes in the development process; they are the pillars upon which trust and efficacy are built.

As you, the JavaScript developer, embark on creating AI applications, how will you balance these critical aspects to not only meet user expectations but also contribute positively to the ethical advancement of AI technology? Feel free to share with me your ideas on [Discord](#).

## Additional Resources

- 1- [OWASP Top 10 for LLM](#)
  - 2 - [OWASP TOP 10 for Machine Learning security](#)
  - 3- [OWASP AI Security and Privacy Guide](#)
  - 4- [Six Steps Toward AI Security](#)
  - 5- [Securing AI & LLM-based Applications: Best Practices](#)
- 

## Additional Resources - Checklists, Roadmaps, Visual aids

- Checklists for each module
- List of more projects you can build
- Roadmaps for learning paths and skill development
- Visual aids and infographics for complex concepts
- Community forum for collaboration and support

## Module Checklists

### Module 1: JavaScript for AI

- [ ] Understand the role of JavaScript in AI application development
- [ ] Learn about JavaScript frameworks like Node.js and Next.js

- [ ] Set up a development environment with Node.js and Next.js
- [ ] Familiarize with the JavaScript ecosystem relevant to AI.

## Module 2: Basics of AI Development

- [ ] Introduction to Large Language Models (LLMs) like GPT, BERT
- [ ] Learn the basics of prompt engineering and model selection
- [ ] Integrate LLMs with JavaScript.
- [ ] Develop a simple AI content generator using Node.js.

## Module 3: Integrating AI Models to JS Apps

- [ ] Explore open-source AI models (e.g., Stable-diffusion, GPT-4o)
- [ ] Investigate commercial AI models (e.g., Whisper, Dall-E).
- [ ] Learn to work with APIs and external libraries.
- [ ] Understand handling AI model responses and interactivity.
- [ ] Build an Image generation app using Dall-E integration.

## Module 4: Introduction to Retrieval Augmented Generation

- [ ] Understand the basics of Retrieval Augmented Generation (RAG)
- [ ] Learn about embeddings and vector databases.
- [ ] Explore open-source tools like Llama Index, Embedchain, and Langchain
- [ ] Integrate AI SDK in development.
- [ ] Implement RAG in a Next.js project.
- [ ] Develop a Research assistant tool.

## Module 5: Building Complex AI Applications

- [ ] Master advanced prompt engineering and optimization.
- [ ] Understand AI function calling and creating AI agents.
- [ ] Develop a complex AI agent as a project.

## Module 6: Security, Ethics, and Performance in AI Development

- [ ] Learn security practices in AI applications.
- [ ] Understand ethical considerations and best practices in AI.
- [ ] Explore privacy-focused Local LLMs.
- [ ] Learn performance optimization techniques for AI apps.
- [ ] Understand best practices for deploying and scaling AI apps.

### List of more projects you can build

#### 1. Sentiment Analysis Tool:

- Build a tool that analyzes and categorizes the sentiment of user input or social media posts using AI models like BERT or GPT.
- Integrate with social media APIs to fetch real-time data.

#### 2. Language Translation App:

- Create an app that uses AI models for real-time language translation.
- Incorporate speech recognition and text-to-speech for an interactive experience.

#### 3. AI-Powered Chatbot for Customer Support:

- Develop a chatbot that can handle customer queries and provide relevant information or redirect to human support.
- Integrate with websites or customer service platforms.

#### 4. Personal Finance Assistant:

- An application that uses AI to analyze personal finance patterns and provides budgeting and investment advice.

#### 5. Voice-Activated Home Automation System:

- Build a system that uses voice recognition to control home appliances and settings.

#### 6. Fitness and Diet Planner:

- An AI system that creates personalized fitness and diet plans based on user preferences and goals.

#### **7. Automated Resume Screening Tool:**

- Develop a tool that assists HR in screening resumes and predicting candidate fit based on job descriptions and resume content.
-