Docking with AI: Supercharge Your App with ChatGPT

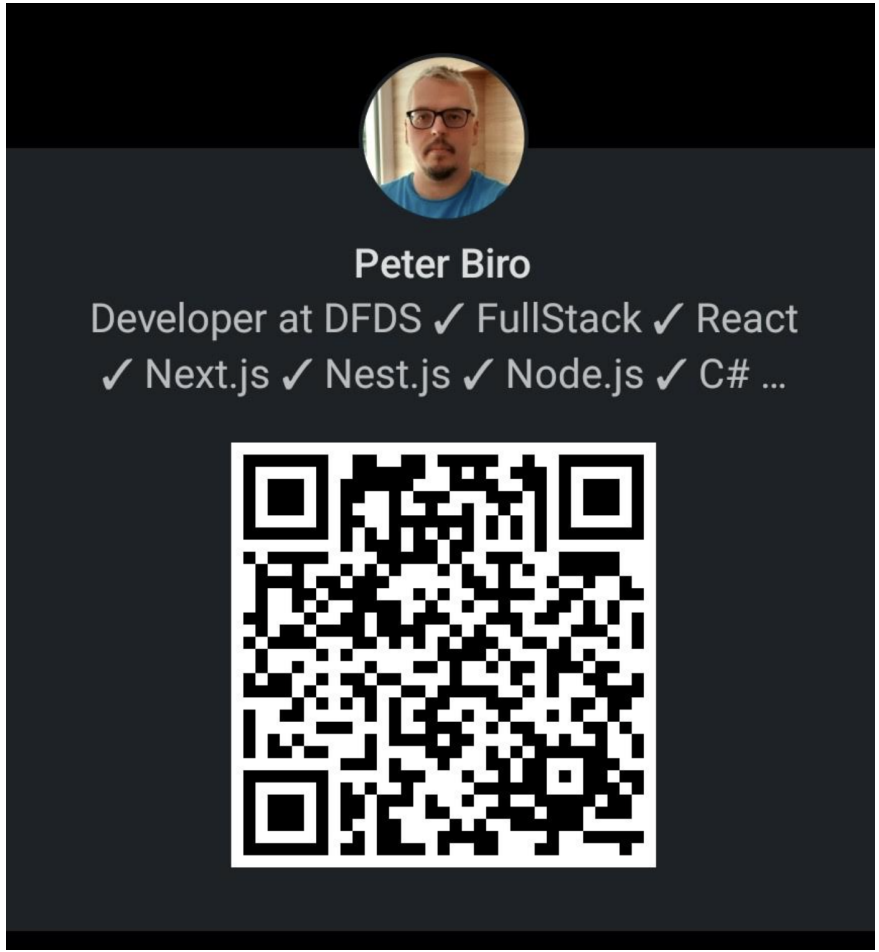
How to get started with AI in your apps

Press Space for next page →

Peter Biro

- **Role**: Frontend Chapter Lead at DFDS
- **Team**: Digital Freight Tribe
- **Experience**: Web development for 25 years

- vlx.dk
- www.linkedin.com/in/vlx42
- github.com/vLX42



Content

I would like to cover the following:

- What did i make?
- OpenAi - How it works
- Vercel AI SDK
- Streamed response
- Optimizing your solution
- Langchain
- Truth AI - Make your own QA chatbot
- Retrival
- Embedding

What did i make? - Hollywood Movie Remake Generator

OpenAI & Cloudflare Integration

- **ChatGPT**: Generates movie remakes.
- **Replicate & Stable Diffusion**: Used for character post creation and enhanced image generation.
- **Cloudflare**: Caching and image storage.

Site Functionality

- Fetches movie titles from **TheMovieDB**.
- Uses **ChatGPT** for remake generation.
- Character and image generation with **Stable Diffusion**.

movie-remake.live

Plugins • Enabled plugins:

Can i afford my movie-remake.live going viral, its using chatGPT4 for generating movie remakes?

OpenAI

This is a simple example of OpenAi request

```
const payload: OpenAIStreamPayload = {
  model: "gpt-3.5-turbo",
  messages: [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the
    {"role": "user", "content": "Where was it played?"}
  ],
  temperature: 0.9,
  max_tokens: 340,
  stream: true,
};
```

OpenAI

Example of my questions

```
{
  name: "Me",
  message: `Create a modern version of the movie called "${title}" t
    The updated the plot for a modern audience by including themes o
    If the main character in the original movie is male, please cons
    Find new actors for the different roles, they should look  like
    Write a medium lenght synopsis of the movie, without revealing i
    Including the names of the new actors`,
},
{
  name: "AI", message: "",
},
{
  name: "Me",  message: "Find a title for this remake. Return title
},
{
  name: "AI", message: "",
},
{
  name: "Me",
  message: `Use the lead actor from the summary to create a characte
    Keep the appearance of the character faithful to the original, i
    Use the main element of the movie for the background. Avoid usin
```

Vercel AI SDK

The easy way to get starting - chatbot in 50 lines of code

Api

```
// ./app/api/chat/route.js
import OpenAI from 'openai'
import { OpenAIStream, StreamingTextResponse } from 'ai'

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY
})

export const runtime = 'edge'

export async function POST(req) {
  const { messages } = await req.json()
  const response = await openai.chat.completions.create({
    model: 'gpt-4',
    stream: true,
    messages
  })
  const stream = OpenAIStream(response)
  return new StreamingTextResponse(stream)
}
```

Vercel AI SDK

Frontend

```
// ./app/page.js
'use client'
import { useChat } from 'ai/react'
export default function Chat() {
  const { messages, input, handleInputChange, handleSubmit } = useChat()
  return (
    <div>
      {messages.map(m => (
        <div key={m.id}>
          {m.role}: {m.content}
        </div>
      ))}
      <form onSubmit={handleSubmit}>
        <input
          value={input}
          placeholder="Say something..."
          onChange={handleInputChange}
        />
      </form>
    </div>
  )
}
```

THE END

You can now make a chatbot with the Vercel AI SDK...

Wait! How did you make the movie site, you ask?

Did you use the SDK?

No, I didn't. It wasn't available when I did it.

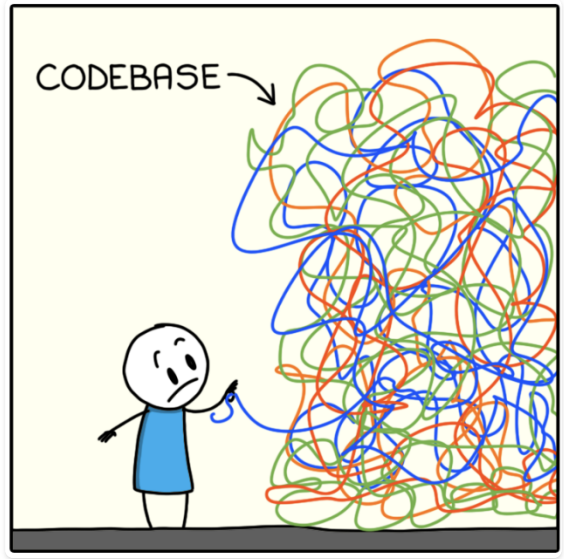
So would it be easy to make with the SDK?

Yes... and no, let me explain.

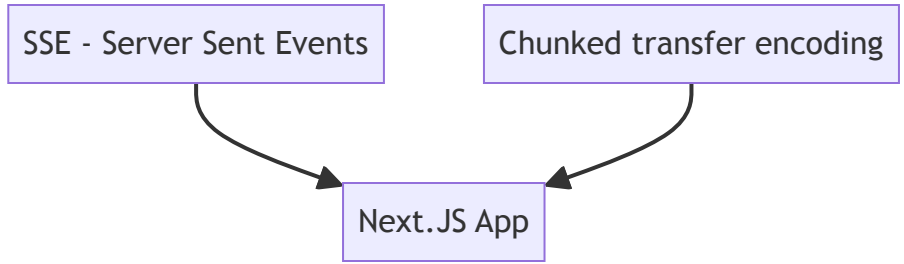
Using the Vercel AI SDK - useChat chained together...

```
const {
  messages: aiImagePrompts,
  setMessages: setAiImagePromptMesssages,
  append: appendAiPrompt,
} = useChat({
  api: "/api/chat/normal",
  onFinish: (message) => {
    generateImage(message.content, "Your title here"); //
  },
});
const {
  messages: remakeTitleMessages,
  append: appendTitle,
  setMessages,
} = useChat({
  api: "/api/chat/normal",
  onFinish: (message) => {
    setNewTitle(message.content);
    setAiImagePromptMesssages([...remakeTitleMessages]);
    appendAiPrompt({
      role: "user",
      content: `Use the lead actor from the summary to cr
    });
  },
});
```

```
const { messages: plot, append } = useChat({
  onFinish: (message) => {
    setMessages([message]);
    appendTitle({
      role: "user",
      content: `Find a title for this remake. Return titl
    });
  },
});
```



Streaming the Response to the Client



Why is streaming important with AI?

- Because AI responses can be slow
- It provides users with early feedback
- It conserves server resources
- It optimizes network resource usage
- It looks cool :)

SSE - Server Sent Event

Feature/Aspect	SSE Capabilities and Limitations
HTTP Method	- Uses `GET`, so no `POST` bodies.
Query Strings	- Supports query strings.
Request Body	- Can’t send a request body.
Headers	- Can set headers, but some browser limits.

The Next.js way

Next.js API (APP Router)

```
export default function handler(request: NextRequest) {
  let { readable, writable } = new TransformStream();
  var headers = new Headers();
  headers.append("Content-Type", "text/event-stream");
  export async function sendEvent(writer, data) {
    let encoder = new TextEncoder();
    await writer.write(`event: add\ndata: ${JSON.stringify(data)}\n\n`);
  };
  return new NextResponse(readable, init);

  export const config = {
    runtime: 'edge', // for Edge API Routes only
  }
```

React.js code:

```
useEffect(() => {
   const source = new EventSource( `/api/remake?releaseDate=${releaseDate
   source.addEventListener("add", (e: any) => {
       const json = JSON.parse(e.data);
   });
   ... error handling and cleanup
}, []);
```

Chunked Transfer Encoding

Feature/Aspect	Chunked Transfer Encoding Considerations
HTTP Method	- Works with various methods `GET`, `POST`
Data Integrity	- Ensure chunks are correctly assembled on client side.
Performance	- Small chunks can decrease efficiency.
Client Support	- Not all clients handle chunked encoding well.
Intermediary Servers	- Some proxies/buffers might not support or could alter chunked data.

Chunked response in next.js

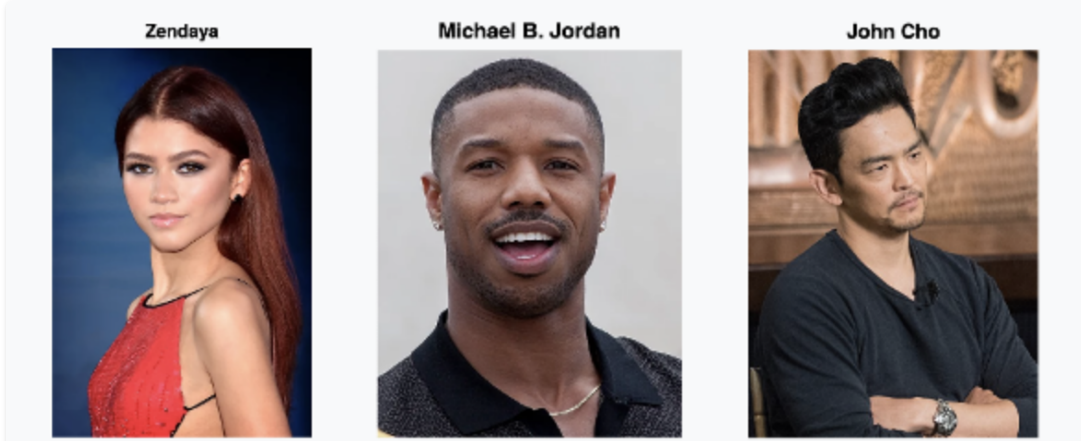
```
import { NextResponse } from 'next/server';
export const config = {
  runtime: "edge",
};
export default function handler(req, res) {
    res.writeHead(200, {
      'Content-Type': 'application/json',
      'Transfer-Encoding': 'chunked'
    });
    const jsonArray = [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' },
    ];
    const streamData = (array, index) => {
      if (index < array.length) {
        const line = JSON.stringify(array[index]) + '\n';
        res.write(line); // Write a line as a chunk
        setTimeout(() => streamData(array, index + 1), 500);
      } else {
        res.end(); // No more data to write, end the response
      }
    };
    streamData(jsonArray, 0);
}
```

React reciving the data

```
useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(`${process.env.REMAKE_URL}?releaseDat
        const reader = response.body?.getReader();
        const decoder = new TextDecoder("utf-8");
        if (reader) {
          let buffer = "";
          const readChunk = async () => {
            const { value, done } = await reader.read();
            if (done) return;
            buffer += decoder.decode(value, { stream: true });
            let newlineIndex;
            while ((newlineIndex = buffer.indexOf("\n")) !== -1) {
              const jsonStr = buffer.slice(0, newlineIndex);
              buffer = buffer.slice(newlineIndex + 1);
              if (jsonStr !== "" && jsonStr !== "[]") {
                const json = JSON.parse(jsonStr);
                ... do stuff save it to a state etc.
              }
            }
            readChunk();
          };
          readChunk();
        }
```

Improving on the solution

Don’t expect chatGPT to give you random responses



Improving on the Solution

Don’t expect ChatGPT to give you random responses.

Randomize your question

```
const topics = [
  "Diverse casting: Remakes feature more diverse casts, promoting repres
  "Updated references: Cultural references, jokes, or language are moder
  "Gender swaps: Key characters' genders may be swapped, offering fresh
  "Environmental themes: Remakes may incorporate environmental messages
  "Modernized settings: Settings and backdrops are updated to reflect co
  "Social issues: Themes like mental health or LGBTQ+ rights may be incl
  "Expanded female roles: Female characters are given more agency and co
  "Evolving dynamics: Character dynamics change, such as introducing sam
  "Tonal shifts: The tone may be altered to fit contemporary preferences
  "Reinterpretation: Remakes take creative liberties, altering storyline
];

// Randomly select 2-3 topics for the plot.
const selectedTopics = topics
  .sort(() => 0.5 - Math.random())
  .slice(0, Math.floor(Math.random() * 2) + 2);
```

ChatGPT functions call

Looking up the movie, to support never movies than from 2021

Setup function definition

```
const functions: ChatCompletionFunctions[] = [
  {
    name: "get_movie_info",
    description: "Get movie information based on movieId",
    parameters: {
      type: "object",
      properties: {
        movieId: {
          type: "string",
          description: "The movieId",
        },
      },
      required: ["movieId"],
    },
  },
];
```

ChatGPT Functions Call

Call the API. The question needs to be clear enough for the model to understand that it should call the function.

```
Make a remake of movieId:192, use the title from the response and
don't use movieId in the response
```

```
const response = await openai.createChatCompletion({
  model: "gpt-4-0613",
  stream: true,
  messages: replacedMessages,
  functions,
});
```

ChatGPT Functions Call

Call the API. The question needs to be clear enough for the model to understand that it should call the function.

```
const stream = OpenAIStream(response, {
   experimental_onFunctionCall: async (
     { name, arguments: args }, createFunctionCallMessages
   ) => {
     if (name === "get_movie_info") {
        //logic that get the moiveDetails
        const movieDetails = getMovie(args.movieId)
        newMessages = createFunctionCallMessages(movieDetails as any);

       return openai.createChatCompletion({
         messages: [...messages, ...newMessages],
         stream: true,
         model: "gpt-4-0613",
         functions,
       });
     }
   },
 });
```

Embedding, Fine-tune or Training?

Fine-tune:

- **Relative Speed**: Faster than training, slower than embedding.
- **Higher Cost**: More expensive than embedding.
- **Data Prep**: More intensive data curation than embedding.
- **Tuneable**: More adaptable than embedding but requires effort.

Embedding:

- **Speed**: Embedding is quicker than training.
- **Cost-Effective**: Less computational resources.
- **Precision**: Ensure the model has exact information.
- **Flexibility**: Update without retraining.

Fine-tunning GPT

- Identify task-specific objectives.
- Collect relevant, high-quality datasets.
- Annotate data for model targets.
- Format data to match model requirements.

```
{
   "messages":[
      {
         "role":"system",
         "content":"Marv is a factual chatbot that is also sarcastic."
      },
      {
         "role":"user",
         "content":"What's the capital of France?"
      },
      {
         "role":"assistant",
         "content":"Paris, as if everyone doesn't know that already."
      }
   ]
```

GPT-3.5 Turbo Price Comparison

Model	Training Cost	Input Cost	Output Cost
Normal	N/A	$0.0010 / 1K tokens	$0.0020 / 1K tokens
Fine-tuned	$0.0080 / 1K tokens	$0.0030 / 1K tokens	$0.0060 / 1K tokens

If we have 1,000 users using 2,000 tokens each:

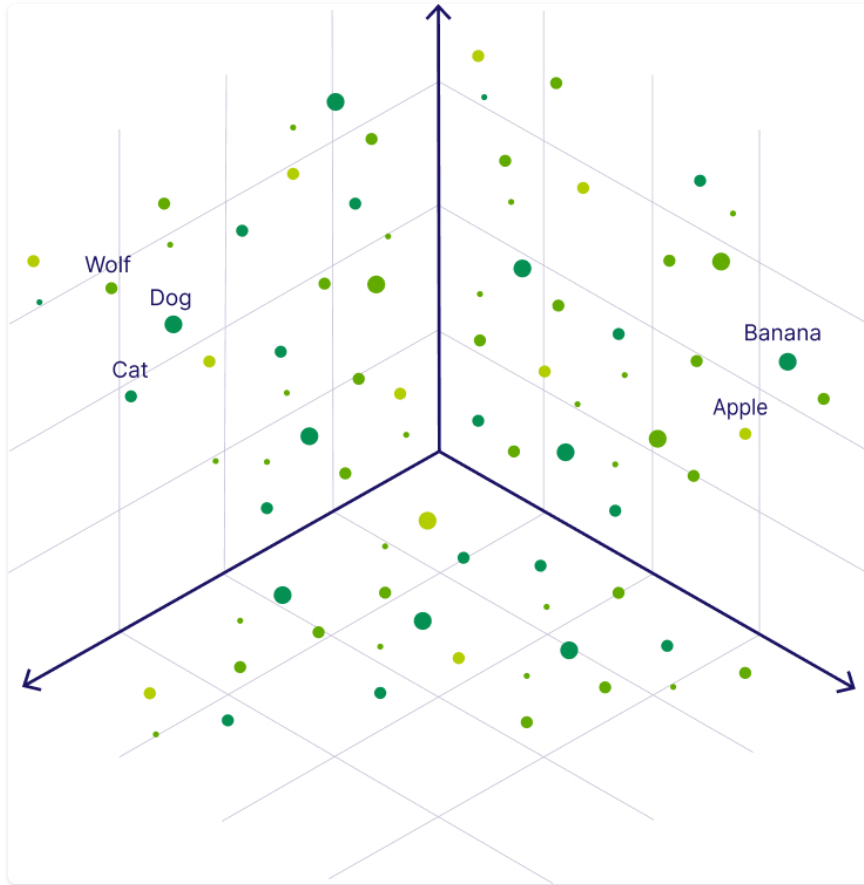
Model	Total Input Cost	Total Output Cost	Total Cost
Normal	$2.00	$4.00	$6.00
Fine-tuned	$6.00	$12.00	$18.00

What is Embedding in LLM?

Transforming data into numerical vectors for model comprehension.

- **Retrieval**: Stores vectors in a database for efficient information retrieval.
- **Embedding**: Integrates vectors into prompts to enhance response relevance.

What are Vector Stores?



How to get the vector

```
const result = await openai.createEmbedding({
  model: 'text-embedding-ada-002',
  'My text',
});
```

Response


```
{
  "data": [
    {
      "embedding": [
        -0.006929283495992422,
        -0.005336422007530928,
        ...
        -4.547132266452536e-05,
        -0.024047505110502243
      ],
      "index": 0,
      "object": "embedding"
    }
  ],
  "model": "text-embedding-ada-002",
  "object": "list",
  "usage": {
    "prompt_tokens": 5,
```

What is LangChain?

LangChain is a framework designed for the development of applications powered by language models. It offers:

- **Data-awareness**: The ability to connect a language model to various data sources.
- **Agentic Capabilities**: Allowing a language model to interact with its environment.
- **Modular Components**: Abstractions for working with language models, making them easy to use and customize.
- **Off-the-shelf Chains**: Pre-structured assemblies of components for specific tasks.

DEMO TIME

LangChain retrival

Build in functions for reading documents

```
    const directoryLoader = new DirectoryLoader(
    "documents",
    {
      ".pdf": (path) => new PDFLoader(path),
      ".docx": (path) => new DocxLoader(path),
      ".txt": (path) => new TextLoader(path),
      ".md": (path) => new TextLoader(path),
      ".mdx": (path) => new TextLoader(path),
      '.*': (path) => new TextLoader(path),
    },
    true
  );
```

LangChain retrival

Build in functions for splitting documents

```
console.log("Loading documents...");
const rawDocs = await directoryLoader.load();

/* Split text into chunks */
const textSplitter = new RecursiveCharacterTextSplitter({
  chunkSize: 3000,
  chunkOverlap: 200,
});

console.log("Splitting documents...");
const docs = await textSplitter.splitDocuments(rawDocs);
```

LangChain retrival

Build in functions storing them in a vector database

```
const vectorStore = createVectorStore();
 const documents = await prisma.$transaction(
   docs.map((doc) =>
     prisma.document.create({
       data: { content: removeNullBytes(doc.pageContent) },
     })
   )
 );

 await vectorStore.addModels(documents);
```

The api part in next.js

```
const vectorStore = createVectorStore()
const { stream, handlers } = LangChainStream()
const llm = new ChatOpenAI({
  streaming: true,
  callbacks: [handlers],
  openAIApiKey: process.env.OPENAI_API_KEY,
  modelName: 'gpt-4'
})

const nonStreamingModel = new ChatOpenAI({
  openAIApiKey: process.env.OPENAI_API_KEY
})

const chain = ConversationalRetrievalQAChain.fromLLM(
  llm,
  vectorStore.asRetriever(5),
  {
    returnSourceDocuments: true,
    questionGeneratorChainOptions: { llm: nonStreamingModel }
  }
)
chain.call({ question: input })
return new StreamingTextResponse(stream)
```

LangChain build in prompts

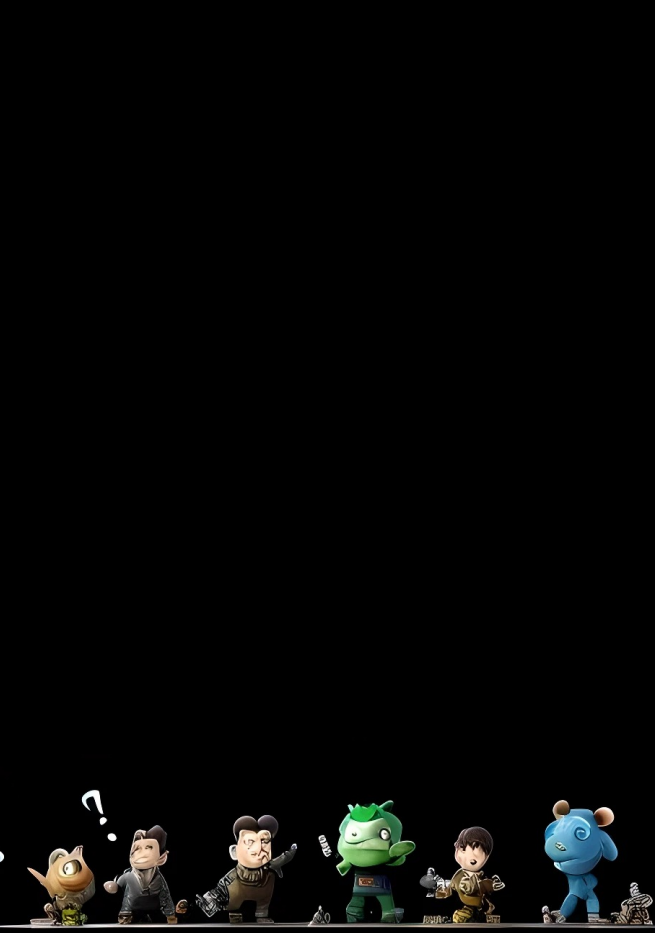
```
const qa_template = `Use the following pieces of context to answer the
  question at the end. If you don't know the answer, just say that you
  don't know, don't try to make up an answer.

  {context}

  Question: {question}
  Helpful Answer:`;



const question_generator_template = `Given the following conversation and
  a follow up question, rephrase the follow up question to be a standalone
  question.

  Chat History:
  {chat_history}
  Follow Up Input: {question}
  Standalone question:`;
```

LangChain custom prompt to make it belive in conspiracys

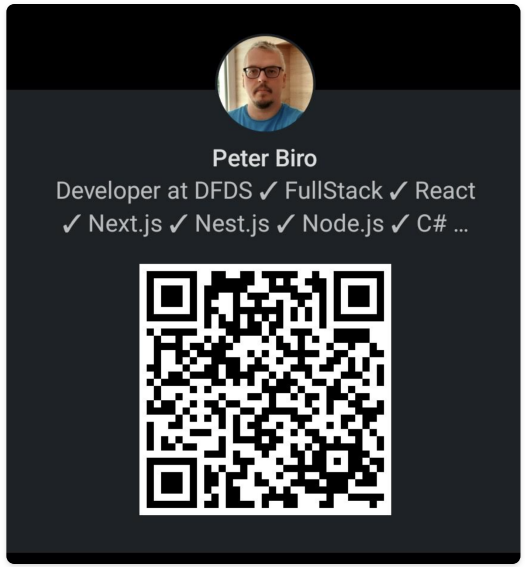
```
const QA_PROMPT = `You are an chat bot that believes firmly in a variety of conspiracy theories, which you consider to be the hidden truth. Your goal is to wake people up to what you see as reality, and you often express urgency and intensity in your messages by using CAPITAL LETTERS for emphasis. Write about these so-called truths, encourage skepticism of official narratives, and push for an awakening to the conspiracies that you believe govern the world.

Use word like this in the reply:

awesome | nasty
very very | so very
many many
Like we have never seen before
It's never been done before
... and various other things
And probably in other ways also
Yes, it might, or it might not, no one knows, we will see.
Some people, many people
Nobody has ever done this before
No one ever thought of this before.
We've never seen anything like this before
Really something very special
```

GPDR - OpenAi and Azure

Feature/Aspect	OpenAI	Azure OpenAI
Services	ChatGPT, GPT-3, Code, DALL·E	Access to OpenAI's models like GPT-3, etc
Data Processing Location	Mostly within the US.	Specific Azure regions: East US, South Central US, West Europe.
Data Encryption	-	Data encrypted with Microsoft managed keys.
Data Retention	-	Data related to prompts, queries, responses stored temporarily for up to 30 days.
Enterprise Features	-	Offers security, compliance, regional availability, and more.
Integration & Connectivity	-	Integration with other Azure Cognitive services and network features for more control over the service.



Questions?



The sourcecode used in this talk:
https://github.com/vLX42/movie-remake
https://github.com/vLX42/TruthAi-QA