Cheat sheet

AI and Node.js

This cheat sheet outlines the basic principles of writing AI applications with Node.js, focusing on large language models. It covers essential topics such as selecting the right model, using model servers like Ollama and Podman AI Lab, and utilizing client libraries like LangChain.js. You'll also learn to implement core AI concepts like chatbots, retrieval-augmented generation (RAG), and agents with tool calling, empowering you to integrate powerful AI functionalities into your JavaScript-based web and server-side applications.

Large language models

A large language model (LLM) is a type of artificial intelligence that is trained on a massive amount of text data to understand and generate human language. LLMs are able to perform various tasks such as:

- **Question answering:** Answer questions based on the data they were trained on.
- **Text summarization:** Condense large amounts of texts into succinct summaries.
- **Language translation:** Translate between different languages.
- **Content generation:** Generate new forms of content like text, articles, and stories.
- **Conversational AI:** Create chatbots.

Choosing the right model

Choosing which model to use can be specific to the use case of the application or the organization using it. However, when choosing a LLM it is important to look at these different factors:

- **Performance metrics:** Can the model generate accurate and contextually relevant responses?
- **Scalability:** Can the model accommodate various workloads and scale in the future to ensure it is viable in the long term?
- **Resource requirements:** What type of infrastructure is required?
- **Configurability:** Can the model be customized to your organization's specific needs?
- **Training data:** Which data has the model been trained on? Did the creator of the LLM have the required rights/agreements for using that data?

- **Licenses:** Does the license allow you to use the model for the intended purpose? Will you have to pay to use the LLM for the intended purpose?

Hugging Face is a great site to discover and compare models. When in doubt, we recommend choosing a model that is open source and shares information about the data on which it was trained, such as Granite.

Model server

A model inference server acts as a bridge between the model and its users, receiving data, processing it with the model, and returning the results. These servers are often optimized for speed and efficiency, especially when dealing with complex models or large datasets.

There are two popular model inference servers that can help you get started locally: Ollama and Podman AI Lab.

Ollama

Ollama is a tool that lets you run and manage LLMs on your own computer, rather than relying on cloud-based services. It supports a wide range of open source LLMs, allowing you to experiment with different models without needing a license.

Podman AI Lab

Podman AI Lab simplifies the process of running and experimenting with AI models locally by providing containerized model servers, a user-friendly interface, and tools for easy integration into your applications.

Client

AI client libraries are packages or modules that provide pre-written code for interacting with AI APIs or services. They simplify the process of using AI technologies by abstracting away the low-level details of communication and providing easy-to-use functions.

Here are a list of popular AI client libraries that we have had success with:

- LangChain.js: A framework for developing applications powered by language models. It contains an abstraction layer that helps developers use multiple language models seamlessly.
- LangGraph.js: A low-level framework for building controllable agents.
- llama-stack-client-typescript: This library provides convenient access to the Llama Stack Client REST API from server-side TypeScript or JavaScript.

Core concepts

This section provides descriptions and examples of key AI concepts you can implement with Node.js.

Chatbots

A chatbot is a computer program that simulates human conversation with an end user. While not all chatbots use AI, modern chatbots increasingly use conversational AI techniques such as natural language processing to understand user questions and automate responses to them.

A simple chatbot with Langchain.js

```
const model = getModel();

 const prompt = ChatPromptTemplate.fromMessages([
   [ 'system',  'You are a helpful assistant'],
   [ 'human', '{input}' ]
 ]);

 const chain = prompt.pipe(model);
 const response = await chain.invoke({
   input: 'What is the weather today?'
 });

 return response;
```

Retrieval-augmented generation (RAG)

RAG is a process for optimizing the performance of an AI model by connecting it with external knowledge bases. RAG helps large language models deliver more relevant responses at a higher quality.

```
// Define a system prompt that tells the model how to use the retrieved
context
    const systemPrompt = `You are an assistant for question-answering tasks.
    Use the following pieces of retrieved context to answer the question.
    Context: {context}:`;

    // Define a question
    const question =
```

```
      "What are the main components of an LLM-powered autonomous agent
system?";

    // Retrieve relevant documents
    const docs = await retriever.invoke(question);

    // Combine the documents into a single string
    const docsText = docs.map((d) => d.pageContent).join("");

    // Populate the system prompt with the retrieved context
    const systemPromptFmt = systemPrompt.replace("{context}", docsText);

    // get a model
    const model = getModel();

    // Generate a response
    const response = await model.invoke([
      {
        role: "system",
        content: systemPromptFmt,
      },
      {
        role: "user",
        content: question,
      },
    ]);

    return response
```

Agents / tool calling

AI tools are software applications that utilize artificial intelligence algorithms to perform tasks that typically require human intelligence, such as analyzing data, recognizing patterns, and making decisions. These tools can automate processes, improve decision-making, and enhance various aspects of different industries.

```
// Tool creation
 const tools = [myTool];
 // Tool binding
 const modelWithTools = model.bindTools(tools);
 // Tool calling
 const response = await modelWithTools.invoke(userInput);
```

Model Context Protocol (MCP)

Model Context Protocol (MCP) is a protocol that allows integration between LLMs and tools, often as part of an agent. One of the interesting aspects of MCP is that tools hosted on a server can be consumed by different frameworks and even frameworks that use a different language from which the server was written.