Building a ChatGPT plugin to browse the internet

OpenAI released their plugins system for ChatGPT back in March. In the beginning it was only available for developers, but now all ChatGPT plus users have access. In the future, once the plugin usability and quality is high enough, I won't be surprised if OpenAI opens up their store to the broader public.

I strongly believe AI plugins are here to stay as they allow developers to enhance the capabilities of Large Language Models like ChatGPT in an easy way. Without plugins, ChatGPT is already quite useful, but it can only answer questions about data it is trained on. Not only do AI plugins give LLM's like ChatGPT the capability to access external real-time data sources, they also allow ChatGPT to take actions in the real world for its users.

> AI plugins give LLM's the potential to become a new user interface for humans to use computers

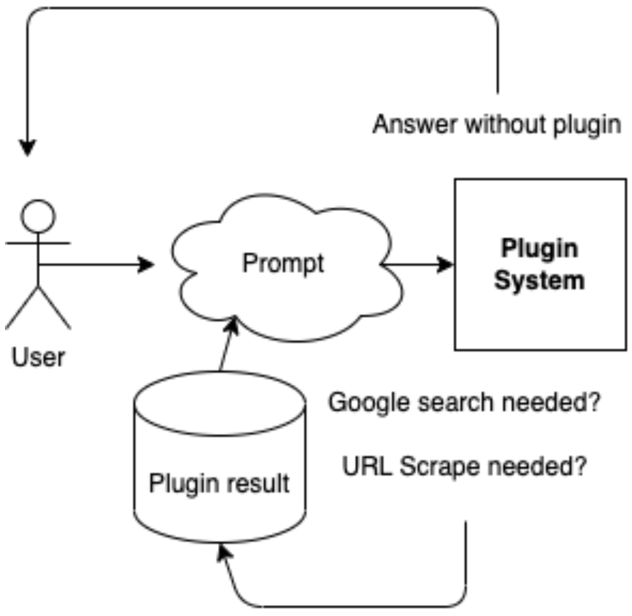
At time of writing there are over 800 plugins available in the plugin store. There are likely many more under development. Besides this, I'm sure there are many other AI companies working on similar plugin systems. If you make a plugin now, you might be able to later integrate it with dozens of other big LLM providers.

In this post, I'll show the process of making a plugin that allows ChatGPT to surf the web. The end result is the ScrapingBee GPT plugin that got released into the wild in the beginning of August.

ScrapingBee GPT

Let's first get an idea of what a plugin does. AI plugins implement a form of an MRKL system that allow a LLM to interact with functions. OpenAI later also came out with their function calling API which implements it in a more barebones way. ChatGPT plugins can be made by following a simple specification. If implemented correctly, you can enable your plugin in ChatGPT, which allows the LLM to use MRKL to opt for sometimes calling your API's. This will provide the Large Language Model with access to external data and actions.

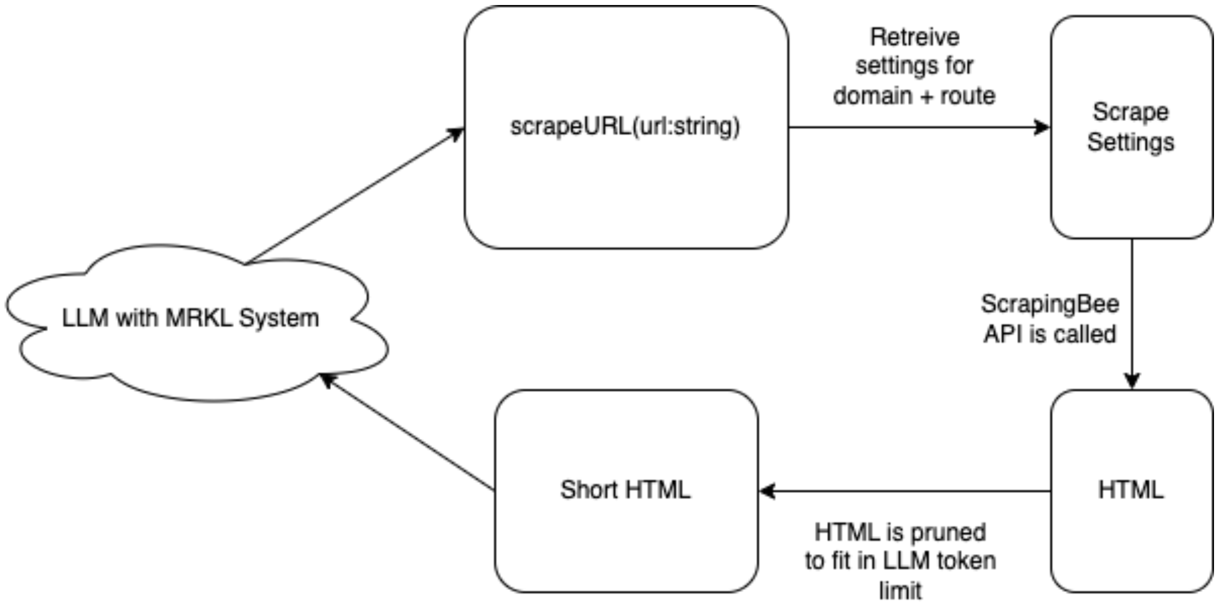
Here is a simple schematic drawing of how the ScrapingBee plugin that I made works:



The ScrapingBee plugin consists of two endpoints with clear descriptions: the url scraper allows the LLM to read website content, while the google searcher allows the LLM to search the internet when it doesn't have an URL yet.

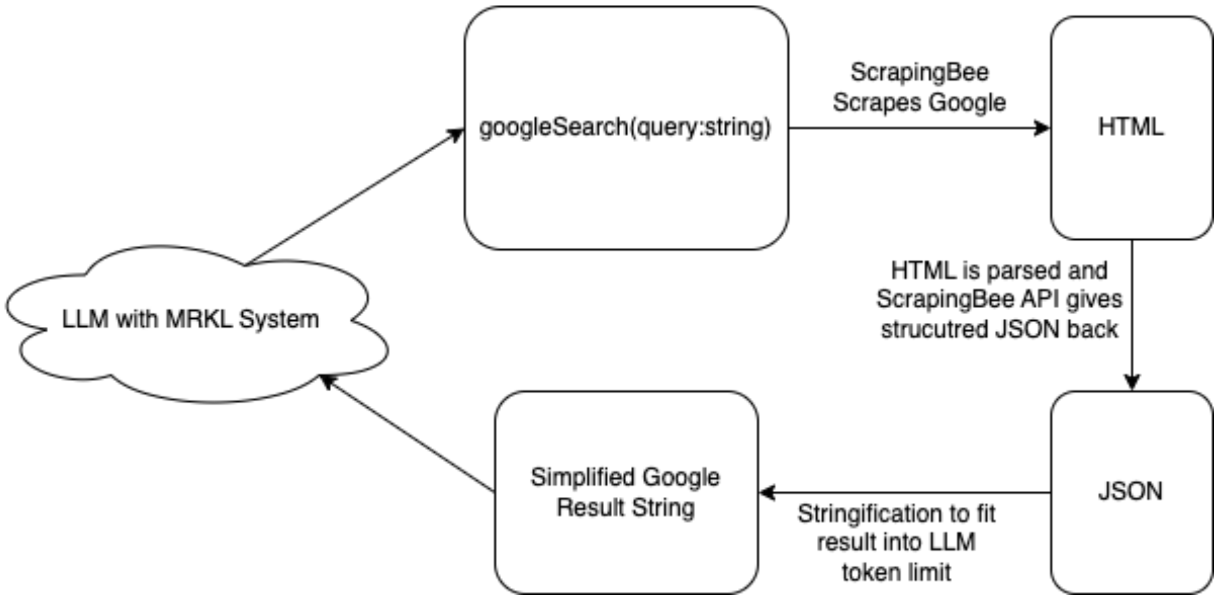
URL Scraper

Besides simply scraping the URL and providing the HTML back, we also need to ensure the scraping API call is configured properly before sending it to ScrapingBee. Afterwards, we need to ensure that the result fits into the LLM token limit, which is currently maximum 16000 tokens for ChatGPT. For this, I basically removed the HTML tags that were irrelevant, as well as removing irrelevant properties on tags, among other things.


scrapeURL(url:string)
Retreive settings for domain + route
Scrape Settings
ScrapingBee API is called
LLM with MRKL System
Short HTML
HTML
HTML is pruned to fit in LLM token limit

Google Search

The Google Search endpoint is also using the internet, but this one is much more important to be done well as it is used a lot in the plugins logic. Therefore I've used the ScrapingBee Google Endpoint to scrape Google results in a structured way. Afterwards, I only needed to simplify the JSON into a short enough string so it fits in the LLM result.


googleSearch(query:string)
ScrapingBee Scrapes Google
HTML
HTML is parsed and ScrapingBee API gives strucutred JSON back
JSON
Stringification to fit result into LLM token limit
Simplified Google Result String
LLM with MRKL System

The Specification

As you can also read in OpenAI's excellent documentation, to allow ChatGPT to use your API, you need to expose two additional endpoints: the `ai-plugin.json` file and your `openapi.json` spec.

The AI plugin JSON tells ChatGPT some metadata that is needed to show the plugin nicely in their plugin store. It also helps the AI understand what it can use your plugin for. This is how my `ai-plugin.json` for ScrapingBee GPT looks:

```
{
    "auth": { "type": "none" },
    "api": {
        "type": "openapi",
        "url": "https://plugin.scrapingbee.com/openapi.json",
        "has_user_authentication": true
    },
    "schema_version": "v1",
    "contact_email": "contact@scrapingbee.com",
    "description_for_human": "Scrape any website with any prompt. I
    "description_for_model": "Scrape any website with any prompt. I
    "legal_info_url": "https://www.scrapingbee.com/terms-and-condit
    "logo_url": "https://plugin.scrapingbee.com/logo.png",
    "name_for_human": "ScrapingBee",
    "name_for_model": "scrapingbee"
}
```

The OpenAPI spec creates a clear definition for the AI on how to use your plugin. This is how my `openapi.json` looks for ScrapingBee GPT:

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "ScrapingBee",
    "version": "v1",
    "description": "No description"
  },
  "servers": [
    {
      "url": "https://plugin.scrapingbee.com",
      "description": "Production server"
    }
  ],
  "paths": {
    "/function/flexibleScraper": {
      "post": {
        "summary": "Scrape any website with any prompt. If no url i
        "operationId": "flexibleScraper",
        "requestBody": {
          "required": true,
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "url": {
                    "type": "string",
                    "description": "Did the user specify a specific
                  },
                  "query": {
                    "type": "string",
                    "description": "What does the user want"
                  },
                  "queryType": {
                    "type": "string",
                    "enum": ["search", "images", "news", "places"],
                    "description": "Used if the user did not provid
                  }
                }
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "Standard response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/StandardResponse"
                }
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "StandardResponse": {
        "type": "object",
        "required": ["isSuccessful"],
        "properties": {
          "isSuccessful": {
            "type": "boolean"
          },
          "message": {
            "type": "string"
          },
          "priceCredit": {
            "type": "number"
```

```
                        }
                    }
                }
            }
        }
    }
```