# Web Scraping in LLM Era

William Brach
CODECON 2024
November 28, 2024

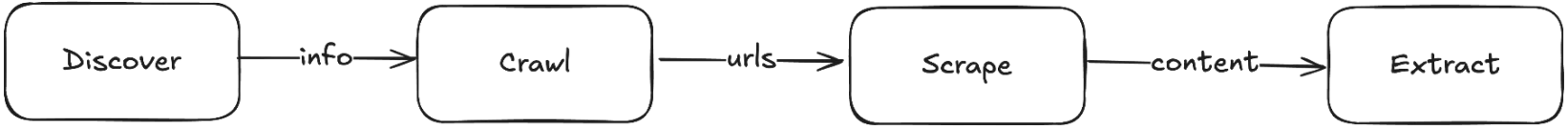
# About Me

- William Brach
- co-founder of sommify
  - AI wine digitalisation services
- phd student @FIIT STU
  - thesis - How to implement AI techniques into web scraping?
- william.brach@stuba.sk
- github - williambrach
- twitter/x - williambrach
- williambrach/webscraping-and-ai

# Your Outcomes

- Keep Your AI Always Market-Ready (**keeping up to date LLM**)
- Automate Complex Web Data Extraction (**building knowledge base**)
- Turn Raw Data into Actionable Insights (**summarization**)
- Build Custom AI Chatbots from Your Data (**chatbots**)
- Accelerate Market Discovery (**automatic discovery**)
- and more

# Stages of Web Scraping in 2024




- Target identification
- Subdomain enumeration
- API endpoint discovery
- Technology stack detection

- Web Crawling
- Robots.txt
- URL prioritization
- Rate limiting
- Proxy & user agent rotation

- HTML rendering
- HTML parsing
- Dynamic content handling
- Anti-bot bypass
- Saving

- Structured formatting
- Named entity recognition

# Example Use Case

- Scraping recipes to homemade cookbook
- 8 recipes
- Task is to extract recipe from webpage

# Recipe response object

- OpenAI Structured Outputs
- instructor-ai/instructor (~8k ⭐)
- dottxt-ai/outlines (~10k ⭐)
- pydantic/pydantic (~22k ⭐)
- different ways how to achieve structure output
- How to write your response object?

# Recipe response object

```
class Recipe(BaseModel):
    title: str = Field(
        None, description="Name of the recipe", example="Classic Chocolate Chip Cookies"
    )
    ingredients: list[str] = Field(
        ...,
        description="List of all ingredients needed for the recipe, including optional
garnishes",
    )
    instructions: list[str] = Field(
        ..., description="Step-by-step preparation instructions in chronological order"
    )
```

# Method 0 : Ground truth

- hhursev/recipe-scrapers (~1.7k ⭐)
- old way of scraping
- manually written down extraction code (python)
- should be fastest and most efficient
- *issue with dynamically rendered websites*

# Method 0 : Implementation

```python
def method_0(url: str) -> dict:
    # method 0 : recipe_scrapers
    html = requests.get(url, headers={"User-Agent": "Burger Seeker Richard"}).content
    recipe = scrape_html(html, org_url=url)
    return recipe
```

# Method 1 : Beautifulsoup + Playwright

- BeautifulSoup
- microsoft/playwright (~67k ⭐) (just better, trust me bro)
- standard implementation
- can handle dynamic content (browser rendering)
- old alternative (worse) bs+requests

# Method 1 : Implementation

```python
async def method_1(url: str, markdown: bool = False) -> BeautifulSoup:
    # Method 1 : Beautifulsoup + Playwright
    async with async_playwright() as p:
        browser = await p.chromium.launch(headless=True)
        context = await browser.new_context()

        # TODO : generate new headers

        page = await context.new_page()

        # TODO : Add random scraping timeout and wait_until (sleep)

        await page.goto(url, timeout=60000, wait_until="networkidle")
        content = await page.content()

        # TODO Save all content + update db

        soup = BeautifulSoup(content, "html.parser")
        await browser.close()

    return MarkdownConverter().convert_soup(soup) if markdown else soup.text
```

# Method 2 : Jina AI Reader

- paid api service (generous free tier)
- ~0.020eur/1M tokens
- jina-ai/reader (~7k ⭐)
- Web site (html) to **Markdown**
- javascript scraping implementation
- easy to use
- lot of options how to extract

# Method 2 : Implementation

```python
def method_2(url: str) -> str:
    return requests.get(f"https://r.jina.ai/{url}").text
```

# Method 3 : Firecrawl

- paid api service
- 3000 credits / 16$
- mendableai/firecrawl (~19k ⭐)
- scrape, crawl, map, llm extract, smart crawl

# Method 3 : Implementation

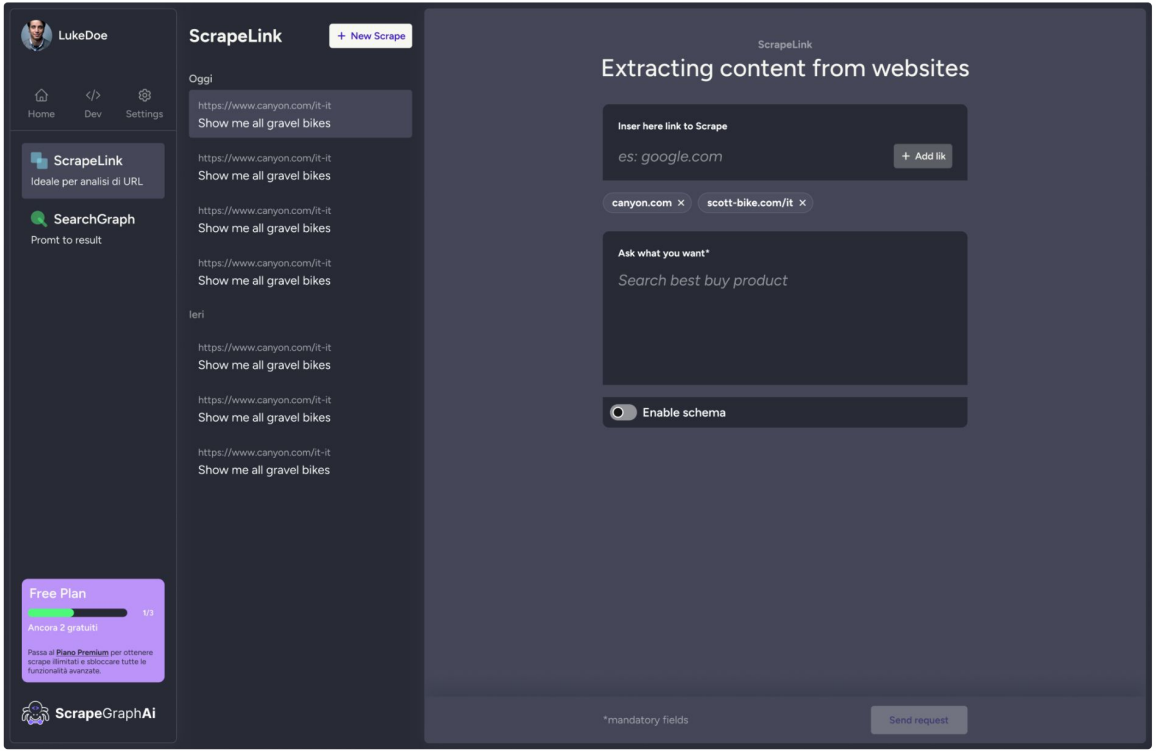
```python
import firecrawl

def method_3(url: str) -> str:
    app = firecrawl.FirecrawlApp(api_key=FIRECRAWL_API_KEY)
    scraped_data = app.scrape_url(url)["markdown"]
    return scraped_data
```

# Method 4 : Scrapegraph-ai

- Scrapegraph-ai (~16k ⭐)
- no code solution
- python api

# Method 4 : Scrapegraph-ai



# Method 4 : Implementation

```python
from scrapegraphai.graphs import SmartScraperGraph

def method_4(url: str) -> dict:

    graph_config = {
        "llm": {
            "model": "openai/gpt-4o-mini",
            "api_key": OPENAI_API_KEY,
            "base_url": OPENAI_BASE_URL,
        },
        "timeout": 60000,
        "verbose": False,
        "headless": True,
        "cache": False,
    }

    smart_scraper_graph = SmartScraperGraph(
        prompt=f"Extract recipe from the following URL in this schema {json.dumps(Recipe.schema())}",
        source=url,
        config=graph_config,
    )
    result = smart_scraper_graph.run()

    return result
```

# Large Language Model (LLM) engine

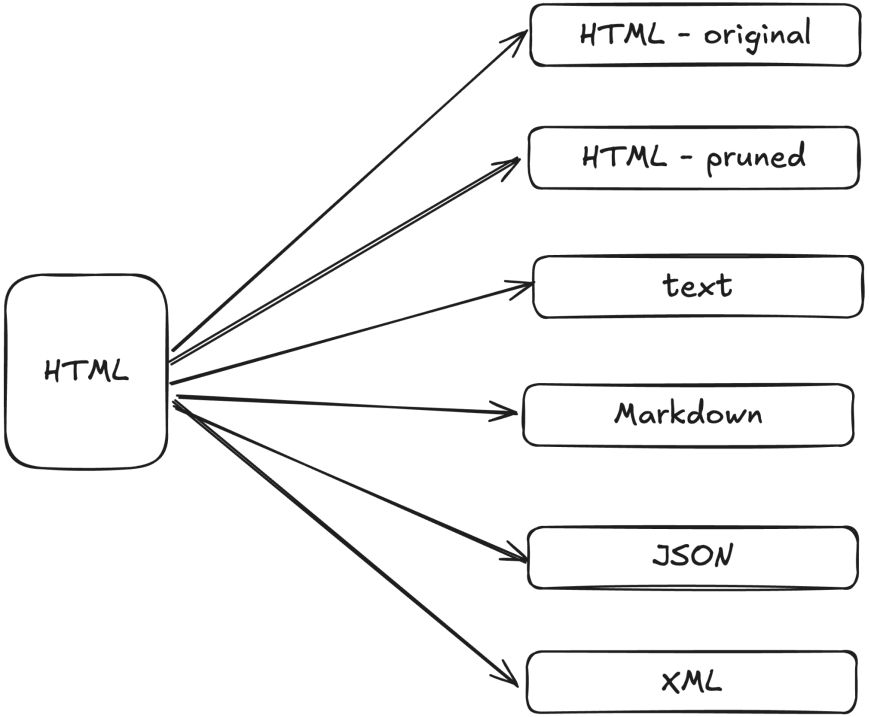
- **!! MODEL AGNOSTIC !!** approach
- gpt-4o-mini
- local llm deployment
  - ollama/ollama (~99k ⭐)
  - ggerganov/llama.cpp (~68k ⭐)
  - vllm-project/vllm (~31k ⭐)
  - others
- to use a sledgehammer to crack a nut
- prompt engineering

# "Engine" implementation

# "Engine" implementation

- LLM hallucination
  - Hallucination-evaluation-leaderboard
    - range 1.3% - 29.9%
    - gpt-4o-mini (1.7%)
    - claude-3.5-sonnet (4.6%)
    - Last updated on November 6th, 2024
- focus on issues and edge cases in output
- temperature = 0

# Content input types (string) for your LLM prompt



# "Engine" implementation

```python
from litellm import completion

def extract(text: str) -> dict:
    prompt = f"Extract the recipe from the following text: {text}"
    try:
        response = completion(
            model="gpt-4o-mini",
            base_url=OPENAI_BASE_URL,
            api_key=OPENAI_API_KEY,
            messages=[{"role": "user", "content": prompt}],
            temperature=0,
            response_format=Recipe,
        )
        return response
    except Exception as e:
        print(e)
        raise ValueError(f"Failed to extract from html: {e}")
```

# Evaluation Metrics

- volume of tokens 📊
- price per input tokens, price per output tokens 💸💸💸
- end-to-end computation time ⚡

# Recipe Methods Overview

```
📍 RECIPE_SCRAPERS
------------------
🏷️ Title: Juicy Lucy burgers stuffed with cheese & pickle
🍽️ Ingredients: ['500g 15% fat beef mince', "1 tbsp all-purpose seasoning (we used Dunn's River)", '6-8 cheese slice... 287
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1052


📍 BEAUTIFULSOUP + PLAYWRIGHT
-----------------
🏷️ Title: Juicy Lucy Burgers Stuffed with Cheese & Pickle
🍽️ Ingredients: ['500g 15% fat beef mince', "1 tbsp all-purpose seasoning (e.g., Dunn's River)", '6-8 cheese slices,... 288
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1052


📍 BEAUTIFULSOUP + PLAYWRIGHT MD
----------------
🏷️ Title: Juicy Lucy Burgers Stuffed with Cheese & Pickle
🍽️ Ingredients: ['500g 15% fat beef mince', "1 tbsp all-purpose seasoning (e.g., Dunn's River)", '6-8 cheese slices,... 283
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1051
```

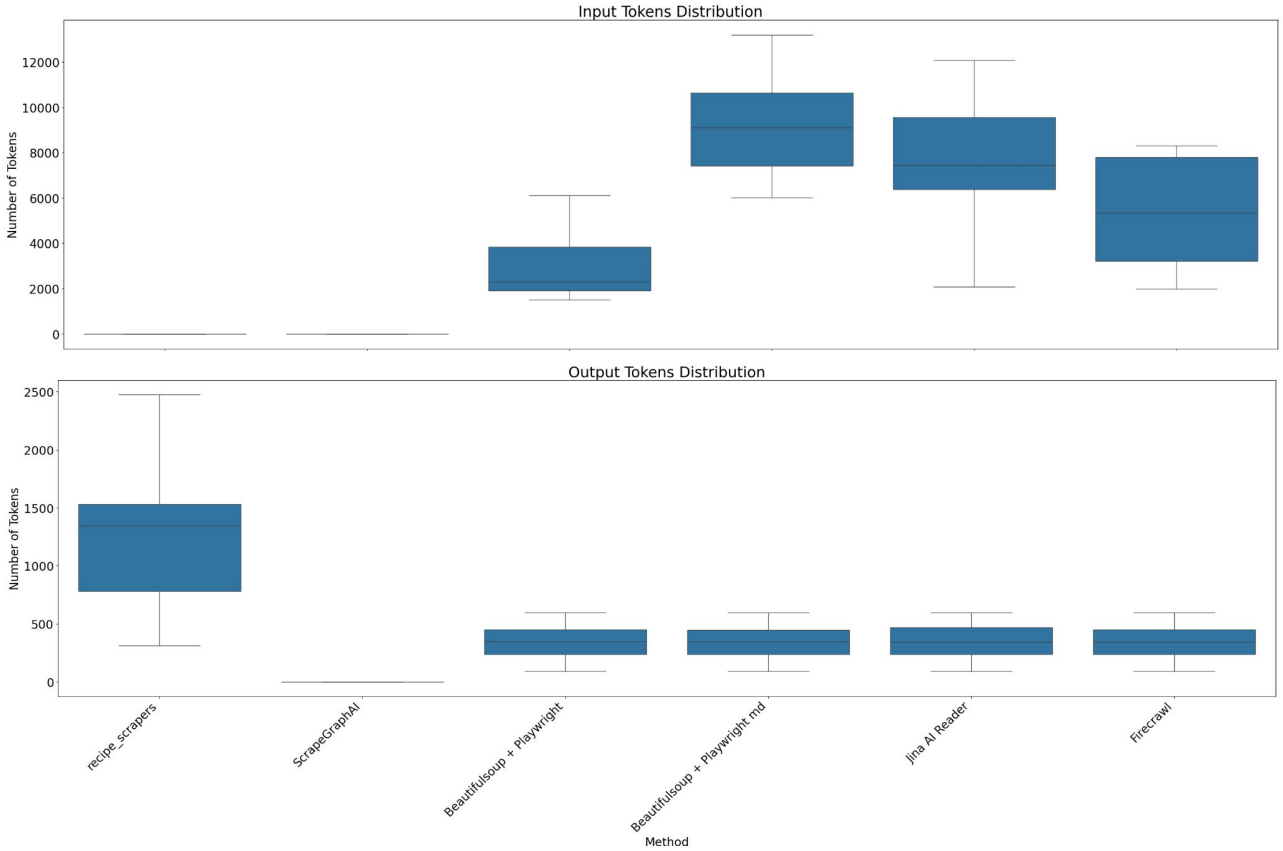
# Recipe Methods Overview

```
📍 SCRAPEGRAPHAI
------------------
🏷️ Title: Juicy Lucy burgers stuffed with cheese & pickle
🥘 Ingredients: ['500g 15% fat beef mince', "1 tbsp all-purpose seasoning (we used Dunn's River)", '6-8 cheese slice... 283
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1052

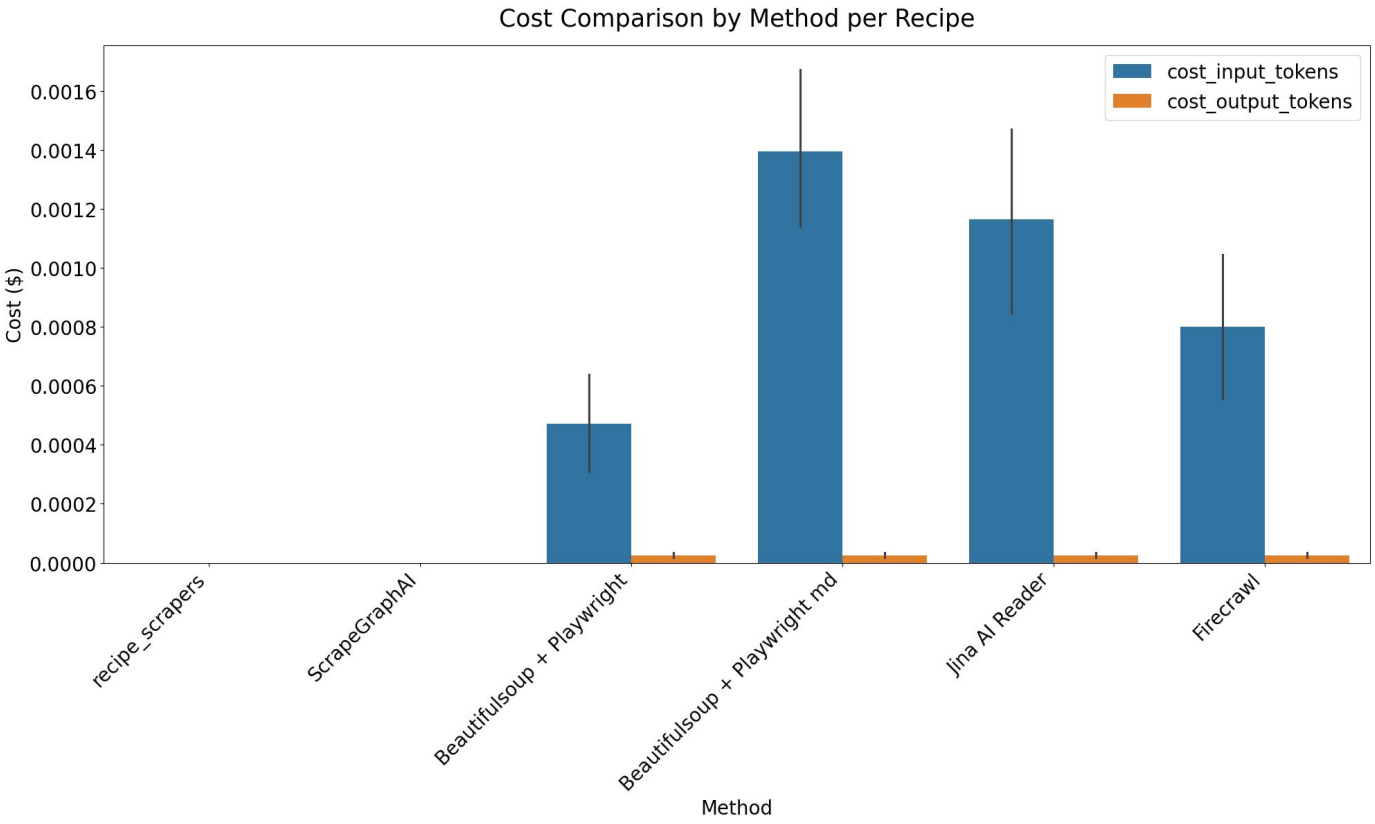

📍 JINA AI READER
------------------
🏷️ Title: Juicy Lucy Burgers Stuffed with Cheese & Pickle
🥘 Ingredients: ['500g 15% fat beef mince', '1 tbsp all-purpose seasoning', '6-8 cheese slices, chopped', '2 large p... 273
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1050


📍 FIRECRAWL
------------------
🏷️ Title: Juicy Lucy Burgers Stuffed with Cheese & Pickle
🥘 Ingredients: ['500g 15% fat beef mince', '1 tbsp all-purpose seasoning', '6-8 cheese slices, chopped', '2 large p... 261
📝 Instructions: ['Put the beef mince and all-purpose seasoning in a large bowl and season with salt and pepper. Mix ... 1052
```

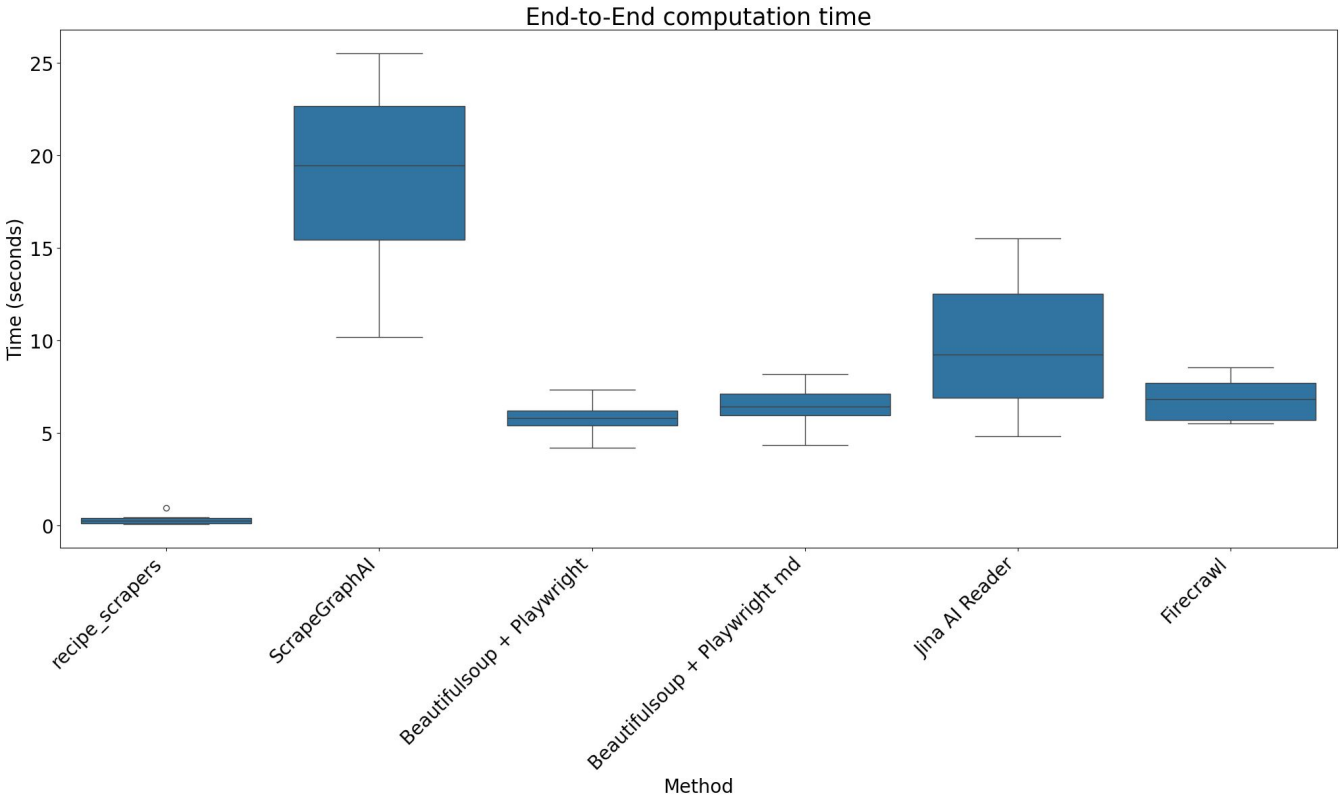
# Results : Volume of tokens



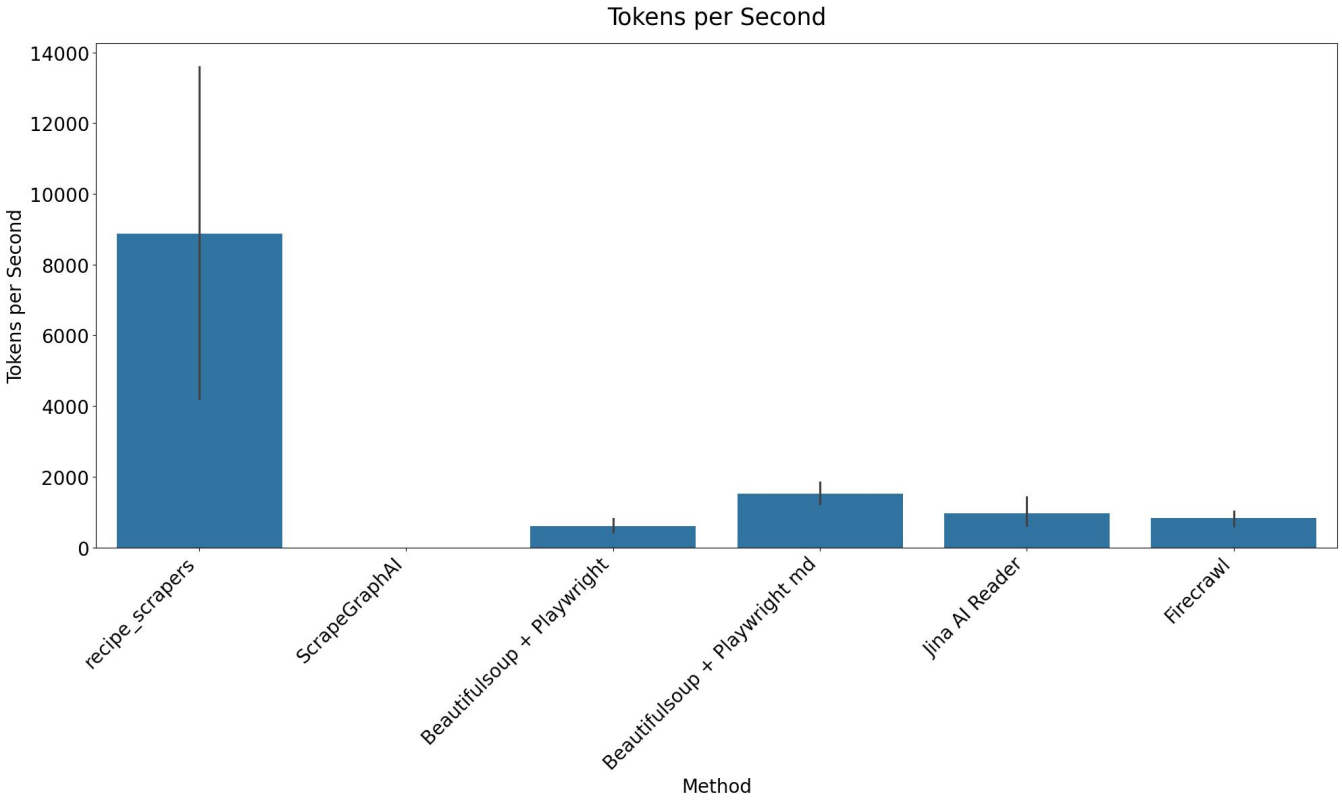
# Results : Price per input / output tokens



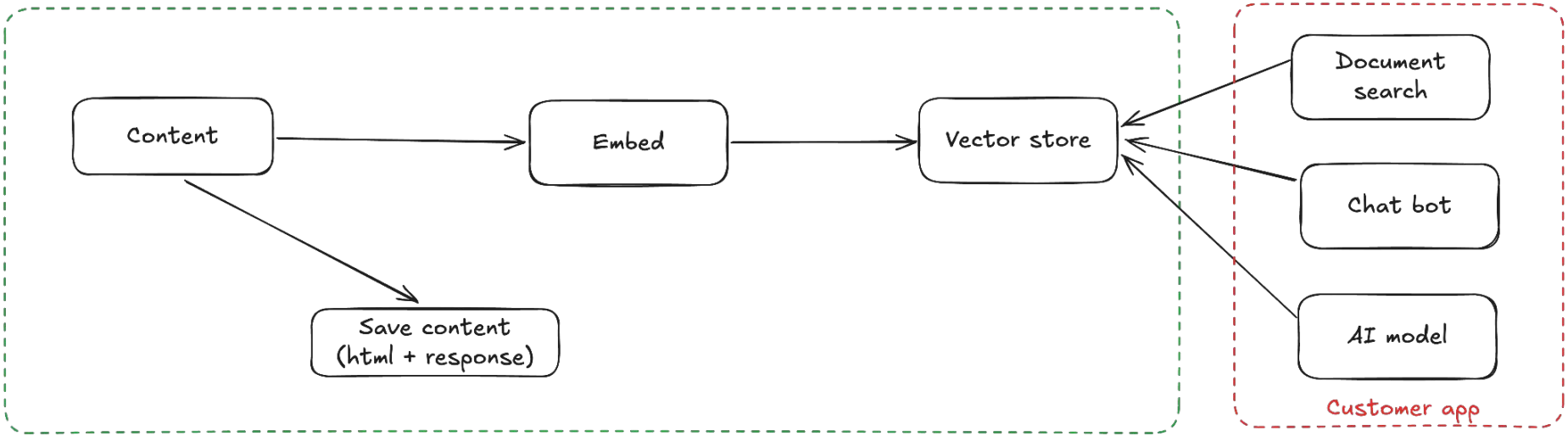
# Results : End-to-End computation time



# Results : End-to-End computation time

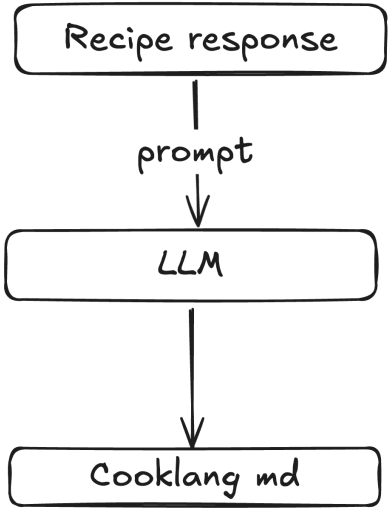


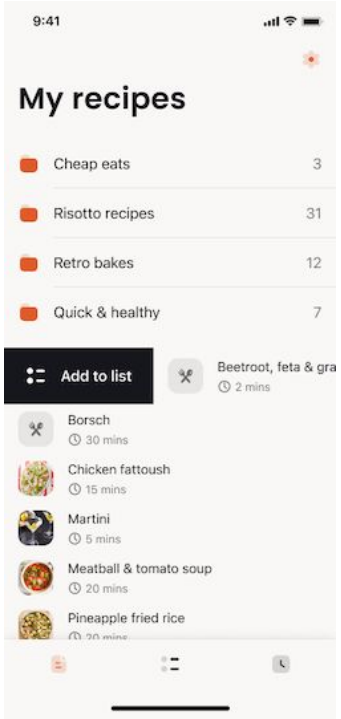
# What now?

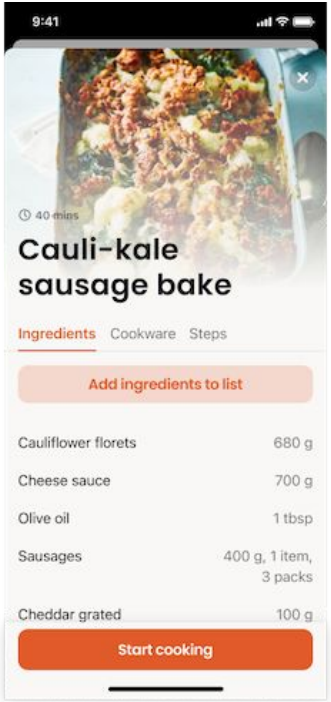

Content
Embed
Vector store
Save content
(html + response)
Document
search
Chat bot
AI model
Customer app

# Recipe to Markdown

- cooklang (~1k ⭐)




# Is web scraping agentic in 2025?

- reworkd/AgentGPT (~32k ⭐)
- Scrapegraph-ai (~16k ⭐)
- unclecode/crawl4ai (~17k ⭐)
- tinyfish-io/agentql (~300 ⭐)
- microsoft/autogen (~35k ⭐)
- crewAIInc/crewAI (~22k ⭐)
- langchain-ai/langchain (~95k ⭐) & langchain-ai/langgraph (~7k ⭐)

# Bonus : /llms.txt

- AnswerDotAI/llms-txt (~200 ⭐)
- new robots.txt?
- example
  - FastHTML docs
  - OpenPipe docs
  - Anthropic docs (claude)
  - Crawai docs

```
# Cancel a Message Batch (beta)

post /v1/messages/batches/{message_batch_id}/cancel
Batches may be canceled any time before processing ends. Once cancellation is
system may complete any in-progress, non-interruptible requests before finali

The number of canceled requests is specified in `request_counts`. To determin
batch. Note that cancellation may not result in any canceled requests if they

<Note>While in beta, this endpoint requires passing the `anthropic-beta` head


# Amazon Bedrock API

Anthropicâ€™s Claude models are now generally available through Amazon Bedroc

Calling Claude through Bedrock slightly differs from how you would call Claud
through the process of completing an API call to Claude on Bedrock in either

Note that this guide assumes you have already signed up for an [AWS account](
access.

## Install and configure the AWS CLI

1.  [Install a version of the AWS CLI](https://docs.aws.amazon.com/cli/latest
2.  Configure your AWS credentials using the AWS configure command (see [Conf
(https://alpha.www.docs.aws.a2z.com/cli/latest/userguide/cli-chap-configure.h
programmatic accessâ€  within your AWS dashboard and following the direction
3.  Verify that your credentials are working:

```bash Shell
aws sts get-caller-identity
```
```

# Summary

- **Web Scraping is Evolving**
  a. Beautifulsoup + Playwright
  b. Jina AI Reader
  c. Firecrawl,
  d. Scrapegraph-ai
- **LLM integration** offers new possibilities, providing structured outputs with built-in content understanding
  a. gpt-4o-mini, gpt-4o, open source models…
- **Content quality is crucial**
- **Cost & Performance matter** - token volume, processing costs, and computation time are key metrics for choosing the right approach
- **Code & Presentation** -> github.com/williambrach/webscraping-and-ai