我就懶

故事大概是這樣

- 遇到的問題
- 抓網頁和前處理
- 資訊抽取 (Structured Outputs)
- Demo 各種 structured outputs 的用法

我是誰

- Narumi
- 學過一點數學，但我數學不好
- Machine Learning Engineer / Data Scientist
- 目前在 MaiCoin 工作

MaiCoin/MAX 是什麼？

- (貨幣兌換) MaiCoin 是一個虛擬貨幣代買代售的平台
- (交易所) MAX 是一個虛擬貨幣交易所

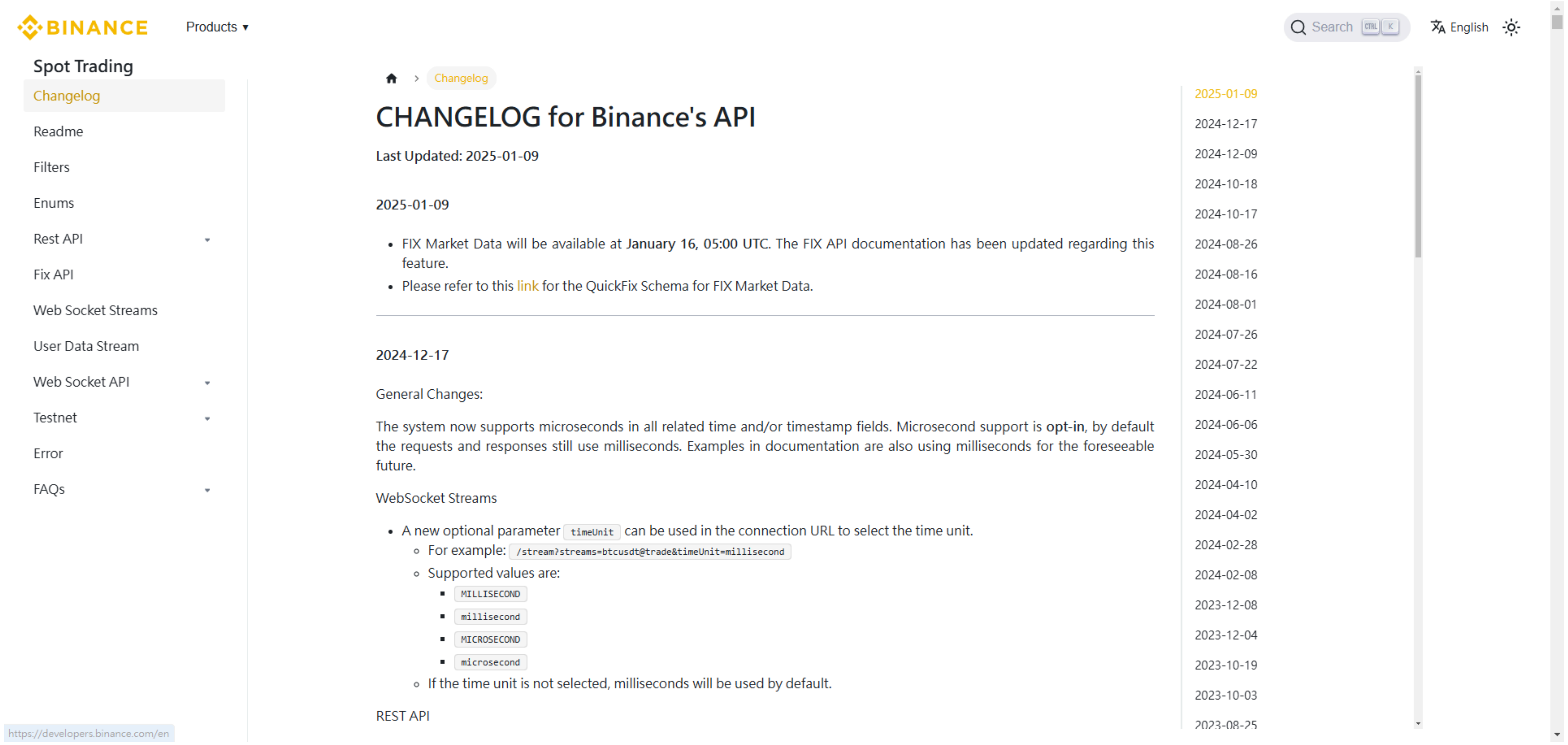
MaiCoin 怎麼決定價格？

- 參考其他家的幣商或交易所的價格
- 所以要接很多家幣商或交易所的 APIs，並且維護
- 不會只有報價這一個需求

接哪些交易所？

- Binance Spot/Margin/Derivatives
- Bitfinex
- Kraken
- CoinBase
- Bybit
- Bitget
- OKX
- ...

Binance Spot Trading Changelog



Spot Trading

- Changelog
- Readme
- Filters
- Enums
- Rest API
- Fix API
- Web Socket Streams
- User Data Stream
- Web Socket API
- Testnet
- Error
- FAQs

Changelog

CHANGELOG for Binance's API

Last Updated: 2025-01-09

2025-01-09

- FIX Market Data will be available at **January 16, 05:00 UTC**. The FIX API documentation has been updated regarding this feature.
- Please refer to this link for the QuickFix Schema for FIX Market Data.

2024-12-17

General Changes:

The system now supports microseconds in all related time and/or timestamp fields. Microsecond support is **opt-in**, by default the requests and responses still use milliseconds. Examples in documentation are also using milliseconds for the foreseeable future.

WebSocket Streams

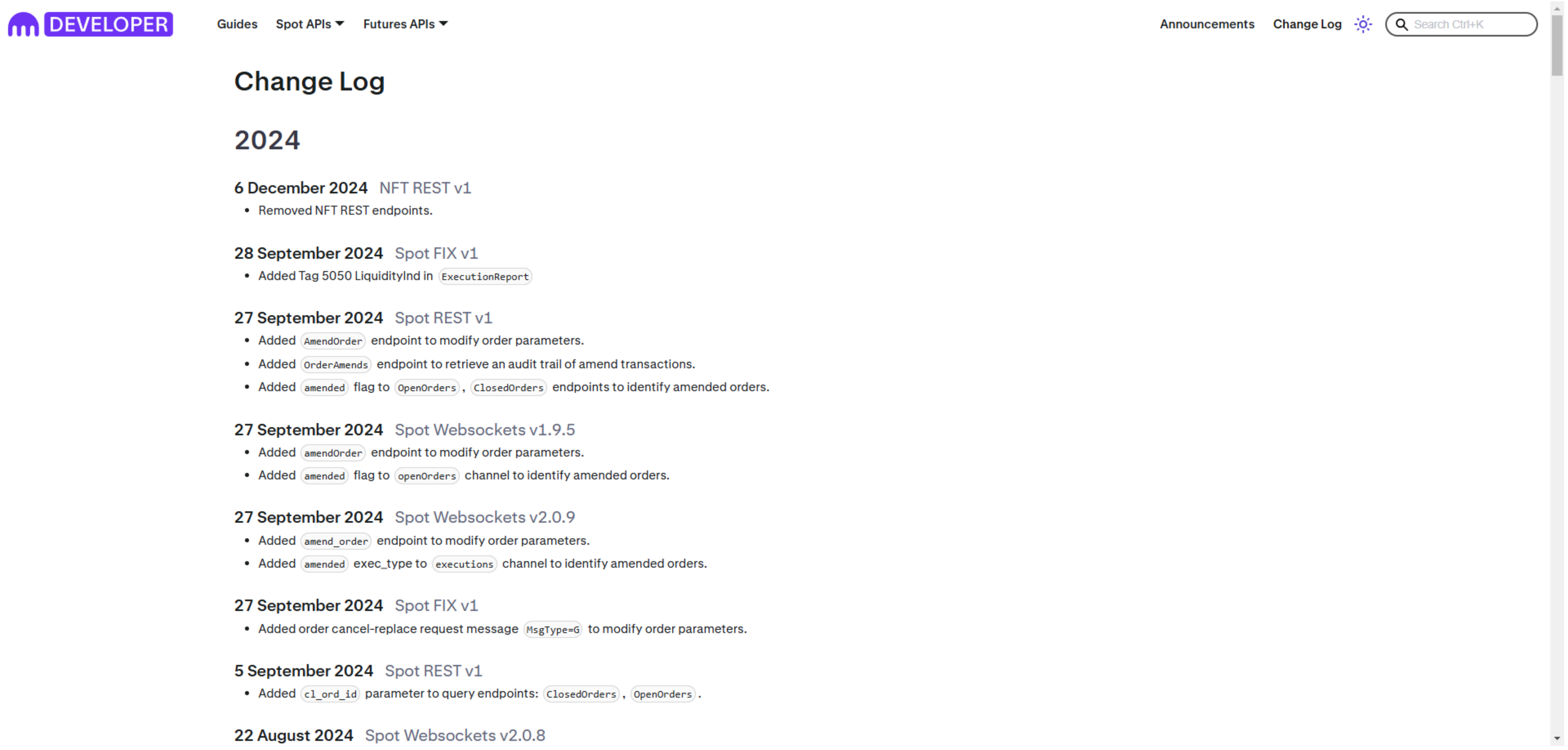
- A new optional parameter `timeUnit` can be used in the connection URL to select the time unit.
  - For example: `/stream?streams=btcusdt@trade&timeUnit=millisecond`
  - Supported values are:
    - `MILLISECOND`
    - `millisecond`
    - `MICROSECOND`
    - `microsecond`
  - If the time unit is not selected, milliseconds will be used by default.

REST API

- 2025-01-09
- 2024-12-17
- 2024-12-09
- 2024-10-18
- 2024-10-17
- 2024-08-26
- 2024-08-16
- 2024-08-01
- 2024-07-26
- 2024-07-22
- 2024-06-11
- 2024-06-06
- 2024-05-30
- 2024-04-10
- 2024-04-02
- 2024-02-28
- 2024-02-08
- 2023-12-08
- 2023-12-04
- 2023-10-19
- 2023-10-03
- 2023-08-25

https://developers.binance.com/en

Kraken Changelog



DEVELOPER Guides Spot APIs Futures APIs Announcements Change Log Search Ctrl+K

Change Log

2024

6 December 2024 NFT REST v1

- Removed NFT REST endpoints.

28 September 2024 Spot FIX v1

- Added Tag 5050 LiquidityInd in `ExecutionReport`

27 September 2024 Spot REST v1

- Added `AmendOrder` endpoint to modify order parameters.
- Added `OrderAmends` endpoint to retrieve an audit trail of amend transactions.
- Added `amended` flag to `OpenOrders`, `ClosedOrders` endpoints to identify amended orders.

27 September 2024 Spot Websockets v1.9.5

- Added `amendOrder` endpoint to modify order parameters.
- Added `amended` flag to `openOrders` channel to identify amended orders.

27 September 2024 Spot Websockets v2.0.9

- Added `amend_order` endpoint to modify order parameters.
- Added `amended` exec_type to `executions` channel to identify amended orders.

27 September 2024 Spot FIX v1

- Added order cancel-replace request message `MsgType=G` to modify order parameters.

5 September 2024 Spot REST v1

- Added `cl_ord_id` parameter to query endpoints: `ClosedOrders`, `OpenOrders`.

22 August 2024 Spot Websockets v2.0.8

麻煩在哪？（小時候）

- 需要定期檢查每間交易所 `changelog` 頁面是否有重要的 `breaking change`
- 每間交易所頁面的格式都不一樣，慢慢寫就好，應該不困難，但就是麻煩
- 交易所的網頁也可能會改版，要重新寫

現在可以怎麼做？

丟給語言模型，叫他抽給你，夭壽方便，但是

- 需要轉成結構化的輸出(structured outputs)，至少日期格式要一致
- 語言模型可能產生幻覺(hallucination)唬爛你或是不受控制，要好好寫 prompt

流程

- 抓各大交易所的 changelog 頁面
- 把 html 轉換成 markdown，砍掉不需要的部分
- 叫語言模型把 markdown 中的**變更項目**與其對應的日期抽出來
- 過濾掉不是最近的**變更項目**
- 把結果打到 Slack channel

順便說一下

- 給今天的日期直接叫語言模型找最近的幾則，效果很差

抓網頁的工具

- HTTPX vs Requests vs AIOHTTP
  - Requests – 小時候常用，不支援 Async，也不支援 HTTP/2
  - HTTPX – 後來改用這個了
  - AIOHTTP – 專門為了 Async 設計，想要快就用這個
- Cloudscraper – 撈 Cloudflare 的頁面用，不一定會成功，但可以試試看

我無腦用 HTTPX 和 Cloudscraper

簡單的範例（HTTPX）

```python
import httpx


url = "https://developers.binance.com/docs/binance-spot-api-docs"
resp = httpx.get(url, follow_redirects=True)
resp.raise_for_status()

print(resp.text)
```

抓網頁的工具（需要瀏覽器）

- Selenium - 老工具，不解釋
- Playwright - 微軟在 2020 年發布的工具，速度快
- SingleFile - 一個 Chrome 的套件，可以一鍵抓下整個網頁儲存成單一檔案，SingleFile CLI 版本
  - docker: `docker run capsulecode/singlefile "https://docs.cdp.coinbase.com/exchange/docs/changelog/" >> coinbase.html`
  - AGPL-3.0 License

目前的偏好是 Playwright >= SingleFile CLI > Selenium

SingleFile

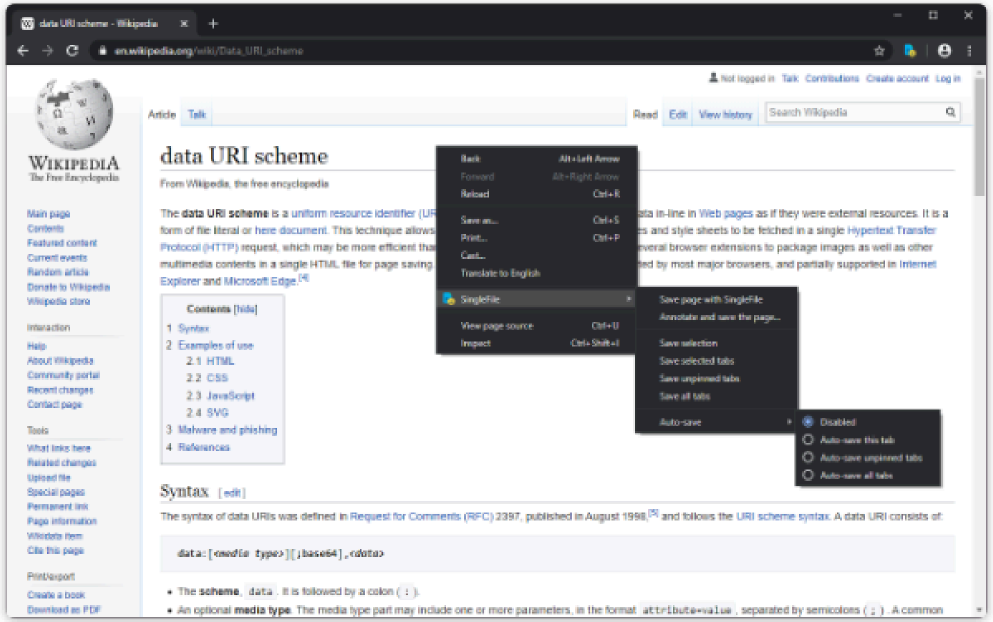
一個一鍵抓下整個網頁儲存成單一檔案的 Chrome 套件

SingleFile

從 Chrome 中移除

精選商品 4.3★ (965 個評分)

擴充功能 工具 200,000 使用者



簡單的範例 (SingleFile CLI)

Install single-file cli with npm

```python
import subprocess
from pathlib import Path

url = "https://docs.cdp.coinbase.com/exchange/docs/changelog", # 要抓的網頁
filename = "coinbase.html" # 輸出檔案名稱
subprocess.run(
    [
        "single-file",
        "--block-images=true", # 不要抓圖片
        "--filename-conflict-action=overwrite", # 若檔案已存在，覆蓋
        url,
        filename,
    ]
)
with Path(filename).open() as fp:
    print(fp.read())
```

為什麼要轉成 Markdown？

優點

- 省 token，處理效率高
- 讓 LLM 專注在文字上，不用管 HTML 的 tag
- 對人來說比較好閱讀

缺點

- 結構比較沒有 HTML 完整

簡單的 Markdownify 範例

把 HTML 轉成比較好讀的 Markdown 格式

```python
import httpx
from markdownify import markdownify as md


url = "https://www.google.com"
resp = httpx.get(url)
resp.raise_for_status()

markdown = md(resp.text, strip=["a", "img"]) # 不要連結和圖片
print(trim_and_filter_lines(markdown))
```

清掉多餘的空白和換行

```python
def trim_and_filter_lines(text: str) -> str:
    """
    Trims whitespace from each line in the given text and filters out empty lines.

    Args:
        text (str): The input text containing multiple lines.

    Returns:
        str: A string with each line trimmed of leading and trailing whitespace and empty lines removed.
    """
    lines: list[str] = []
    for line in text.splitlines():
        stripped = line.strip()
        if stripped:
            lines += [stripped]
    return "\n".join(lines)
```

資訊抽取 (Information Extraction)

~~小時候~~ vs 以前 vs 現在

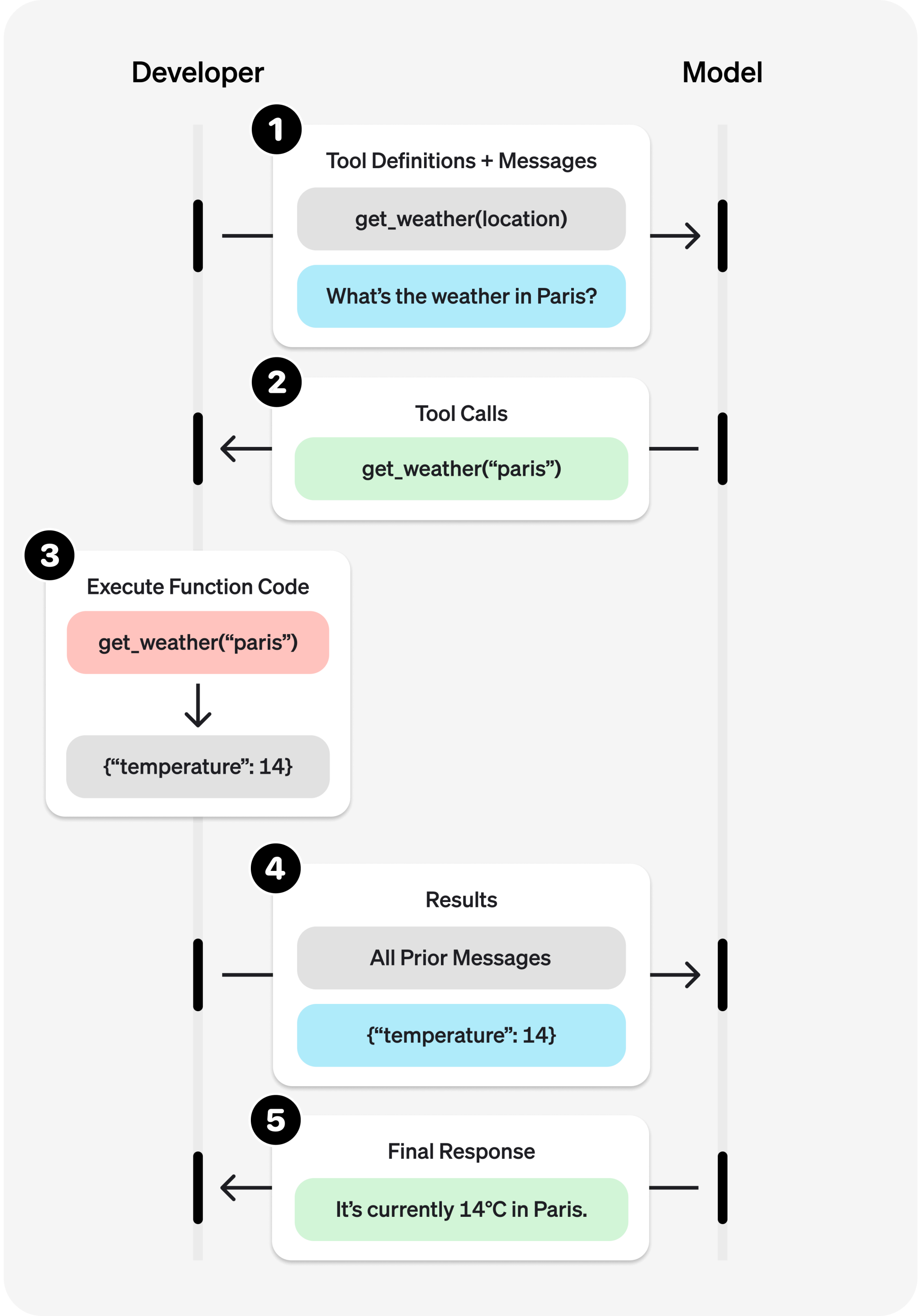
上古時代就不說了

以前 (2023)

- 利用 function calling
- 把你想抽取的資訊當作 function arguments，叫 LLM 告訴你該怎麼 call
    - LangChain OpenAI Extraction
    - Step-by-Step Tutorial for DTO Extraction from Text Using LLMs and LangChain

Function Calling

- OpenAI Function Calling
- LangChain Tool calling



LangChain 以前怎麼做

隨便打的範例

```
class Person(BaseModel):
    name: str = Field(..., description="The name of the person.")
    age: int = Field(..., description="The age of the person.")

llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
chain = create_extraction_chain_pydantic(Person, llm)
chain.invoke({"input": text})
```

LangChain 以前怎麼做

create_extraction_chain_pydantic

```python
def _get_extraction_function(entity_schema: dict) -> dict:
    return {
        "name": "information_extraction",
        "description": "Extracts the relevant information from the passage.",
        "parameters": {
            "type": "object",
            "properties": {
                "info": {"type": "array", "items": _convert_schema(entity_schema)}
            },
            "required": ["info"],
        },
    }
```

LangChain 以前怎麼做

```
_EXTRACTION_TEMPLATE = """Extract and save the relevant entities mentioned \
in the following passage together with their properties.

Only extract the properties mentioned in the 'information_extraction' function.

If a property is not present and is not required in the function parameters, do not include it in the output.

Passage:
{input}
"""  # noqa: E501
```

現在

- JSON Mode
    - Improved instruction following and JSON mode
- Structured Outputs (推薦)
    - OpenAI CookBook – Introduction to Structured Outputs

JSON Mode

- Structured Outputs 的基本版本
- 保證輸出是合法的 JSON 格式 (如果有處理 edge cases)
- 不保證輸出的內容符合某種特定的結構
- 要在 prompt 中加入 "JSON" 字串，否則 API 會回傳錯誤
- 要處理 edge cases
  - 輸出太長被截斷
  - 違反安全或政策規範 (refusal or content filter)

JSON Mode 的範例

```python
from openai import OpenAI

client = OpenAI()
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "system",
            "content": 'Extract dates from message. JSON format: {"dates": [{"year": int, "month": int, "day": int}]}',
        },
        {
            "role": "user",
            "content": "今天是2025年1月20日",
        },
    ],
    temperature=0,
    response_format={"type": "json_object"},
)
print(response.choices[0].message.content)
# {"dates": [{"year": 2025, "month": 1, "day": 20}]}
```

JSON Mode Example

```python
from openai import OpenAI

client = OpenAI()
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "system",
            "content": "You are a helpful assistant designed to output JSON.",
        },
        {
            "role": "user",
            "content": "Who won the world series in 2020? Please respond in the format {winner: ...}",
        },
    ],
    response_format={"type": "json_object"},
)

print(response.choices[0].message.content)
# { "winner": "Los Angeles Dodgers" }
```

Structuted Outputs

- 好處 (vs JSON Mode)
  - 不用驗證拿到的輸出是否正確，也不用再重新產一次輸出
  - Refusals 比較容易處理
  - 寫 prompt 更容易了，不用再特別強調要輸出什麼樣的格式與內容
    - 用 pydantic 的 BaseModel 寫起來比較舒服
- Supported Models
  - gpt-4o-mini-2024-07-18 and later
  - gpt-4o-2024-08-06 and later

P.S. Gemini 是用 typing.TypedDict - 透過 Gemini API 產生結構化輸出內容

Structured Output Example

```python
from pydantic import BaseModel
from openai import OpenAI

client = OpenAI()

class Step(BaseModel):
    explanation: str
    output: str

class MathReasoning(BaseModel):
    steps: list[Step]
    final_answer: str

completion = client.beta.chat.completions.parse(
    model="gpt-4o-2024-08-06",
    messages=[
        {
            "role": "system",
            "content": "You are a helpful math tutor. Guide the user through the solution step by step."
        },
        {
            "role": "user",
            "content": "how can I solve 8x + 7 = -23"
        }
    ],
    response_format=MathReasoning,
)

math_reasoning = completion.choices[0].message.parsed
```

Example Response

```json
{
  "steps": [
    {
      "explanation": "Start with the equation 8x + 7 = -23.",
      "output": "8x + 7 = -23"
    },
    {
      "explanation": "Subtract 7 from both sides to isolate the term with the variable.",
      "output": "8x = -23 - 7"
    },
    {
      "explanation": "Simplify the right side of the equation.",
      "output": "8x = -30"
    },
    {
      "explanation": "Divide both sides by 8 to solve for x.",
      "output": "x = -30 / 8"
    },
    {
      "explanation": "Simplify the fraction.",
      "output": "x = -15 / 4"
    }
  ],
  "final_answer": "x = -15 / 4"
}
```

以 Binance Spot API Changelog 為例

定義 Structures

```python
class Date(BaseModel):
    year: int
    month: int
    day: int

class Change(BaseModel):
    change: str

class Entry(BaseModel):
    date: Date
    changes: list[Change] = Field(..., description="The changes made")

class Changelog(BaseModel):
    entries: list[Entry]
```

也可以做分類

```python
class Category(str, Enum):
    BREAKING_CHANGES = "breaking changes"
    NEW_FEATURES = "new features"
    DEPRECATIONS = "deprecations"
    BUG_FIXES = "bug fixes"
    PERFORMANCE_IMPROVEMENTS = "performance improvements"
    SECURITY_UPDATES = "security updates"

class Change(BaseModel):
    change: str
    category: Category
```

讀取網頁和抽取資訊

```python
from openai import OpenAI
import httpx
from markdownify import markdownify as md

url = "https://developers.binance.com/docs/binance-spot-api-docs"
resp = httpx.get(url)
resp.raise_for_status()
content = md(resp.text, strip=["a", "img"])

client = OpenAI()
response = client.beta.chat.completions.parse(
    messages=[{"role": "user", "content": content[:5000]}],  # 只想要近期的
    model="gpt-4o-mini",
    temperature=0,
    response_format=Changelog,
)
```

結果

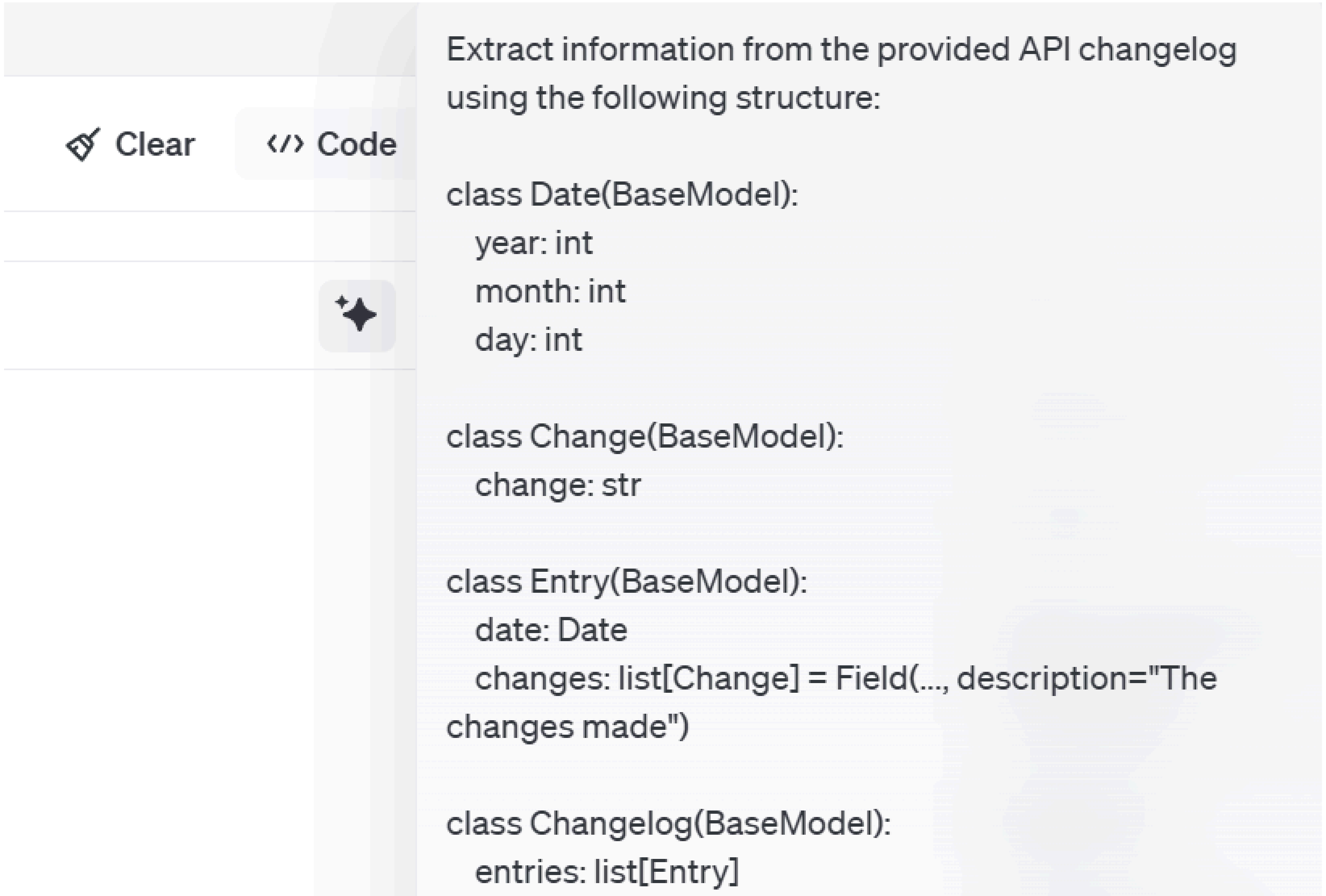
```
Changelog(
    entries=[
        Entry(
            date=Date(year=2025, month=1, day=9),
            changes=[Change(change='FIX Market Da...', category=<Category.NEW_FEATURES: 'new features'>)]
        ),
        Entry(
            date=Date(year=2024, month=12, day=17),
            changes=[
                Change(
                    change='The system now supports...',
                    category=<Category.NEW_FEATURES: 'new features'>
                ),
                ...
            ]
        ),
        ...
    ]
)
```

Prompt Generation

各種方法

- 叫 ChatGPT 幫你產生
- GitHub Copilot
- OpenAI Playground
- OpenAI Guides – Prompt generation
- Anthropic dashboard

OpenAI Playground – 產生 Prompt



Extract information from the provided API changelog using the following structure:

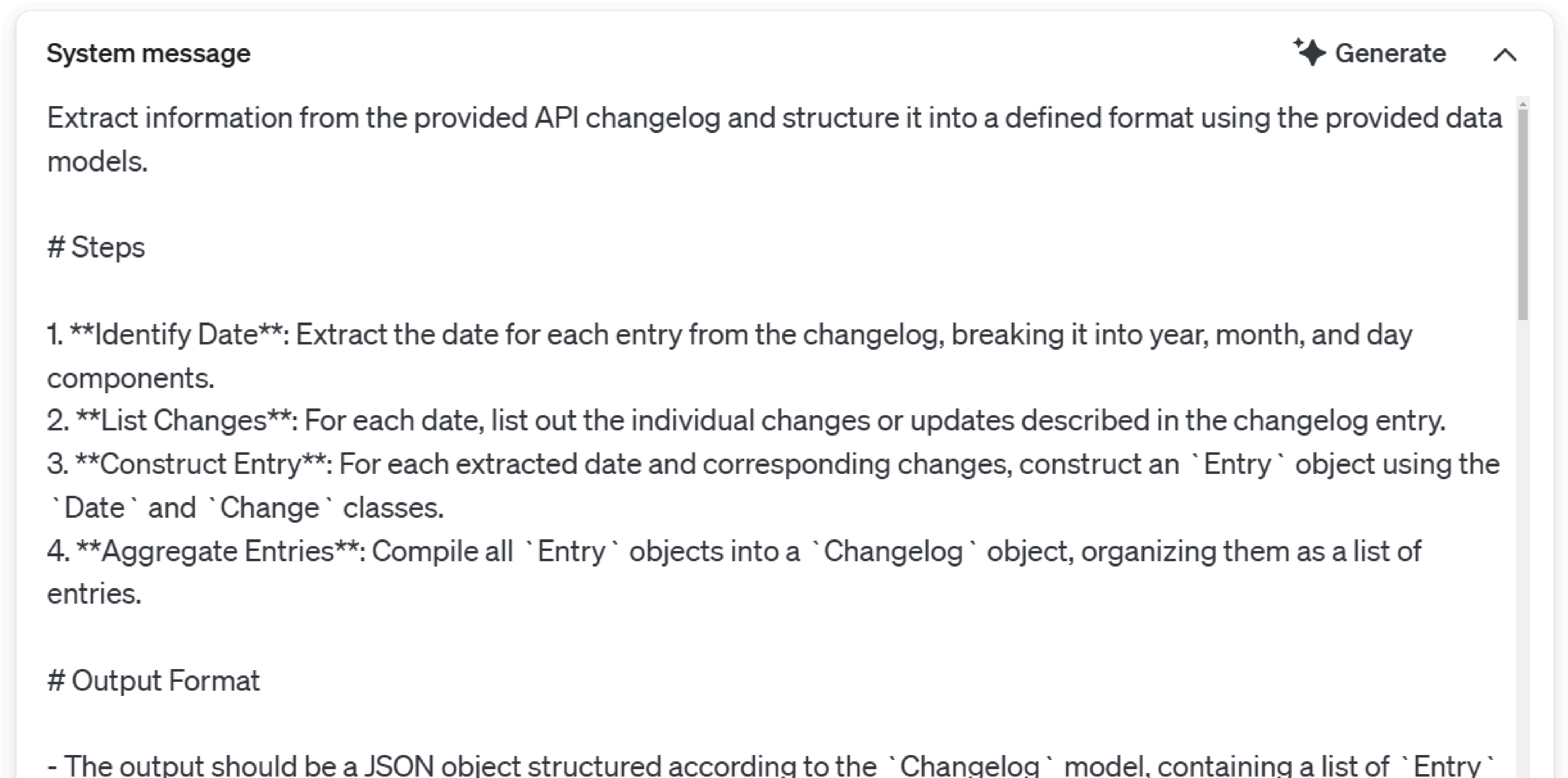
```
class Date(BaseModel):
   year: int
   month: int
   day: int

class Change(BaseModel):
   change: str

class Entry(BaseModel):
   date: Date
   changes: list[Change] = Field(..., description="The
changes made")

class Changelog(BaseModel):
   entries: list[Entry]
```

OpenAI Playground – 產生 Prompt



Extract information from the provided API changelog and structure it into a defined format using the provided data models.

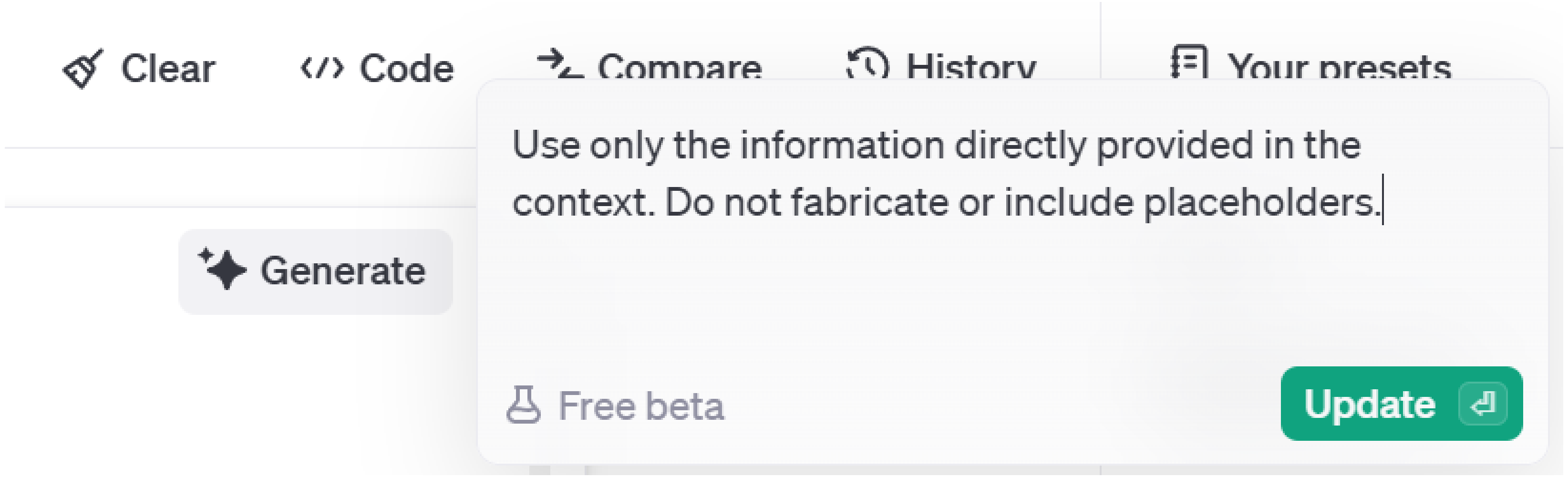
Steps

1. **Identify Date**: Extract the date for each entry from the changelog, breaking it into year, month, and day components.
2. **List Changes**: For each date, list out the individual changes or updates described in the changelog entry.
3. **Construct Entry**: For each extracted date and corresponding changes, construct an `Entry` object using the `Date` and `Change` classes.
4. **Aggregate Entries**: Compile all `Entry` objects into a `Changelog` object, organizing them as a list of entries.

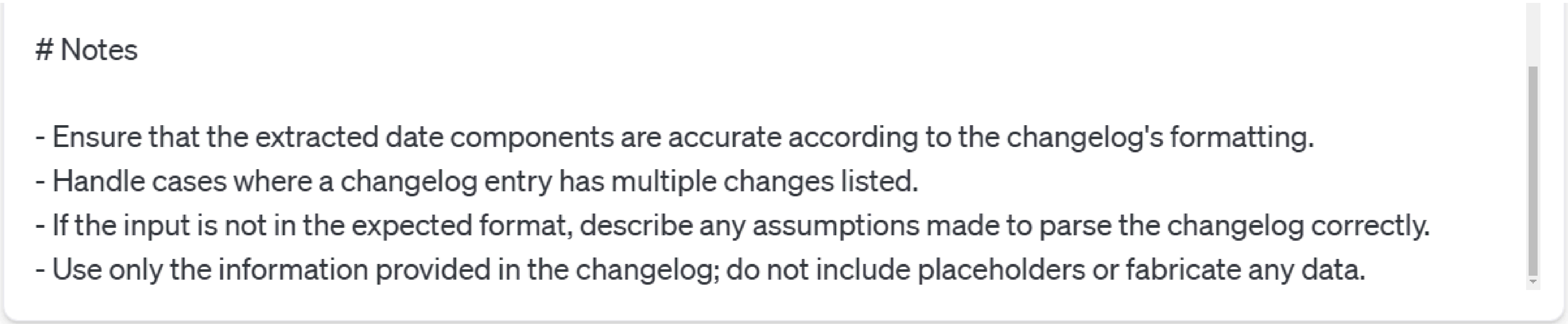
Output Format

- The output should be a JSON object structured according to the `Changelog` model, containing a list of `Entry`

OpenAI Playground – 修改 Prompt



Use only the information directly provided in the context. Do not fabricate or include placeholders.



Notes

- Ensure that the extracted date components are accurate according to the changelog's formatting.
- Handle cases where a changelog entry has multiple changes listed.
- If the input is not in the expected format, describe any assumptions made to parse the changelog correctly.
- Use only the information provided in the changelog; do not include placeholders or fabricate any data.

小心得

- 指令要明確
- 叫語言模型不要捏造事實(既使是這樣，還是可能有幻覺)
- 用同一個 `prompt` 抽不同目標且格式差異大時，給範例可能會有反效果

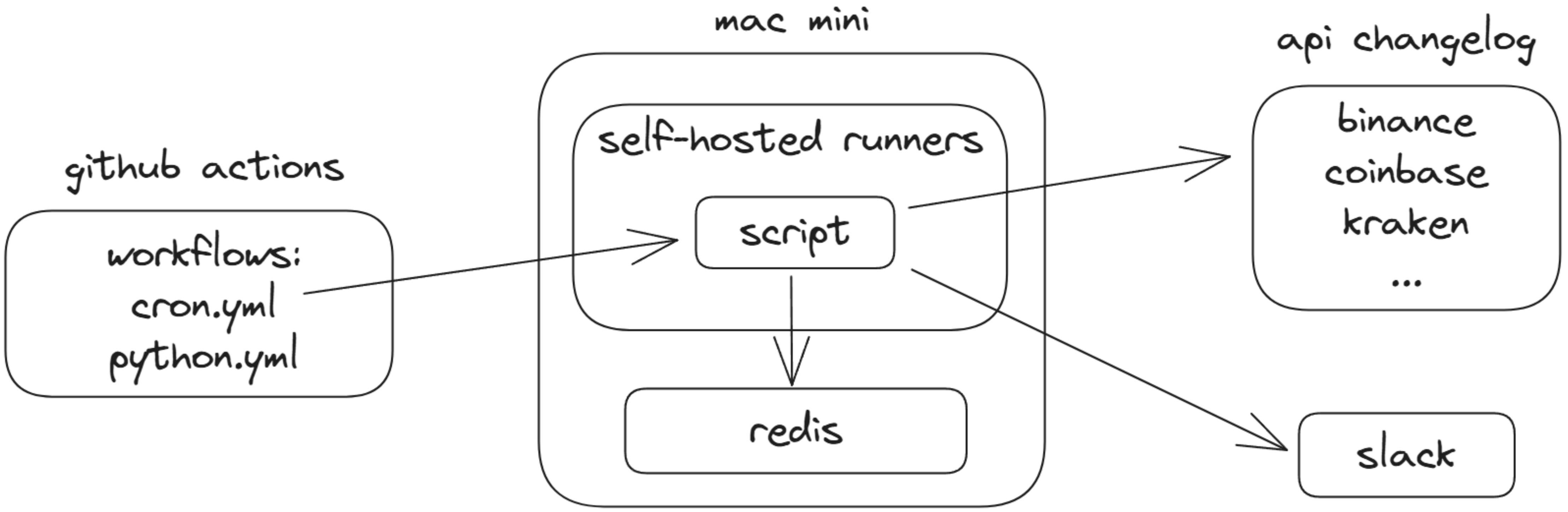
OpenAI Guides - Prompt engineering

於是...

- 你現在可以從 changelog 網頁中抽取出最近的幾則變更項目
- 有日期就可以紀錄下來(redis)，如果是新的就打到 slack channel
- 你可以把這個程式放到 cron job 中，定期執行

Github Actions

- self-hosted runners



跑一陣子之後的感覺

- 不要太相信結果，語言模型很容易產生幻覺
- 不同頁面抓起來的效果不一樣
- 至少抓日期還可接受，`changelog` 有更新就可以從 `slack channel` 知道
- 可能是我太客家，只用 `gpt-4o-mini`
- 這種方法可以在一開始的時候快速開發，之後退居二線作為備案

看程式和結果

- Github Repo
- SingleFile
- Gist

有沒有更懶的工具？

- `simplemid`
- `mirascope`
- `instructor`

Simplemind

```python
import simplemind as sm
from pydantic import BaseModel

class InstructionStep(BaseModel):
    step_number: int
    instruction: str

class RecipeIngredient(BaseModel):
    name: str
    quantity: float
    unit: str

class Recipe(BaseModel):
    name: str
    ingredients: list[RecipeIngredient]
    instructions: list[InstructionStep]

recipe = sm.generate_data("Write a recipe for chocolate chip cookies", response_model=Recipe)
```

mirascope

```
from mirascope.core import openai
from pydantic import BaseModel

class Book(BaseModel):
    title: str
    author: str

@openai.call("gpt-4o-mini", response_model=Book)
def extract_book(text: str) -> str:
    return f"Extract {text}"

book = extract_book("The Name of the Wind by Patrick Rothfuss")
# title='The Name of the Wind' author='Patrick Rothfuss'
```

Demo

隨便找一些網站來試試看

END