Extracting New Product Information from SEC 10-K Filings Using Large Language Models

By: Jeffrey Potvin | CAP 5619

GitHub

Introduction

This project automates the extraction of new product information from SEC 10-K filings of S&P 500 companies using large language models (LLMs). The goal was to parse structured data from unstructured financial documents, identify new product announcements, and compile them into a standardized CSV format.

While the original assignment focused on 8-K filings, this analysis was expanded to 10-K filings due to their comprehensive nature, which often includes detailed product updates alongside financial performance. The final output includes new product names, descriptions, and company details for 100 companies, processed using OpenAI's GPT-4O-MINI model.

This report analyzes the three key components of the project:

- WIKI_COMPANY_SCRAPE_CSV.py – A script responsible for scraping company data.
- LLM_10K_Document_Analysis_Jeffrey_Potvin.ipynb – A Jupyter Notebook that implements the LLM-based extraction.
- extracted_products_FINAL.csv – The CSV output containing the structured new product information.

Methodology

1. Organized Company Data

The WIKI_COMPANY_SCRAPE_CSV.py script was developed to scrape company data from Wikipedia, specifically the list of S&P 500 companies. The script:

- Extracts tickers, company names, and CIKs.
- Organizes companies by market capitalization using the yfinance library.
- Exports a CSV file that will work in conjunction with the SEC EDGAR API.

This file served as the foundation for identifying which companies to pull 10-K filings from, helping ensure the most impactful companies were analyzed first.

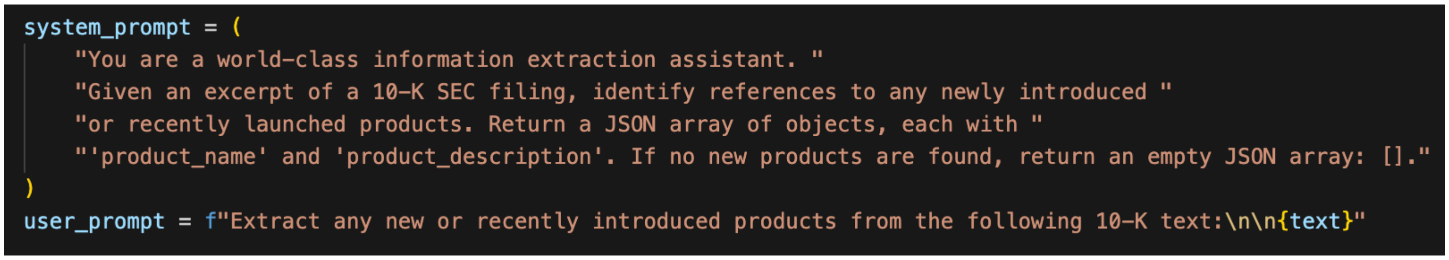
2. SEC 10-K Filings

Using the SEC EDGAR API, the project fetched 10-K filings for each company. These filings were parsed in HTML format and cleaned with BeautifulSoup to remove unnecessary formatting and extract only the raw, relevant text content. Boilerplate sections and non-informative metadata were filtered using regex. Priority was given to business description sections like "Item 1: Business" and "Item 7: Management's Discussion and Analysis."

3. LLM-Based Product Extraction

The cleaned text was passed to the **OpenAI ChatGPT API** using the **GPT-4O-MINI** model, which performed significantly better than prior approaches. To ensure structured and parseable output, the model was instructed to return responses in **JSON format**. This prevented issues with inference think text and helped with verification of products in the document.

To avoid creative variability and maximize response consistency, the **temperature parameter was set to 0.0**. This produced deterministic output and ensured each run would return the most likely, structured response. The final working prompt after extensive experimentation was as follows:



```python
system_prompt = (
    "You are a world-class information extraction assistant. "
    "Given an excerpt of a 10-K SEC filing, identify references to any newly introduced "
    "or recently launched products. Return a JSON array of objects, each with "
    "'product_name' and 'product_description'. If no new products are found, return an empty JSON array: []."
)
user_prompt = f"Extract any new or recently introduced products from the following 10-K text:\n\n{text}"
```

Challenges

This project involved extensive trial and error before arriving at the final working solution. The challenges encountered included:

1. Local Models Attempted

Numerous local models were tested before pivoting to the API. These included:

- DeepSeek 1.5B & 8B
- Phi / Phi-3
- Ollama Mistral 7B & 32B
- FinBERT (fine-tuned for product mentions)
- Hugging Face NER pipelines

While higher-parameter Ollama models like Mistral 32B performed relatively well in isolation, they were extremely slow, taking nearly 2 hours to process a single 10-K, with chunking. DeepSeek and Phi models had insufficient contextual reasoning for this task, and FinBERT tuning showed poor generalization beyond financial sentiment tasks.

2. Format Testing

Due to token limitations, long 10-K filings were chunked into smaller segments before model input. Additionally, different input formats were tested (JSON, RTF, plain text), but plain cleaned text worked best. Chunking logic was implemented manually, and only the most relevant sections (if found) were passed to the LLM.

3. Hardware Bottlenecks

Despite working on both a MacBook Pro M4 Pro and a desktop with an RTX 2070 8GB GPU, local models hit performance walls. The models took incredibly long even when attempting to run DeepSeek 1.5B in parallel.

4. Prompt Engineering

Dozens of prompts were tested before arriving at one that: prevented hallucinations, returned valid JSON, and properly identified product launches. Small changes to phrasing had large impacts on model behavior, and even slight formatting issues could mess up parsing.

5. Cost vs. Performance

After switching to the ChatGPT API, the cost of processing 100 full 10-K filings was surprisingly low—just $1.07 total. Given the computational effort required for local models, this made the API an extremely cost-effective and scalable option.

Results

The final output, extracted_products_FINAL.csv contains 434 new products. Example entries:

Company Name	Stock Symbol	Filing Date	New Product	Product Description
Advanced Micro Devices	AMD	2025-02-05	AMD Instinct MI300X GPUs	High-performance AI accelerator products deployed by large hyperscale cloud customers.
Apple Inc.	AAPL	2024-11-01	iPhone 16 Plus	A larger version of the iPhone 16, offering enhanced display and battery life.

Despite occasional formatting issues, the script extracted relevant product mentions with high accuracy and consistency across the top 100 S&P companies.

Conclusion

This project showed how large language models can be used to extract new product information from 10-K filings in a fast and affordable way. After testing several local models and running into issues with accuracy, speed, and hardware limitations, switching to the ChatGPT API was the most effective solution. Using a temperature setting of 0.0 and a carefully designed JSON-based prompt helped get consistent and reliable results. The entire process of analyzing 100 filings only cost $1.07 using the API, which made it both scalable and cost-efficient. While I learned a lot about setting up and testing local LLMs and they remain a great option for handling sensitive data, the API was the best choice for this project based on performance, cost, and reliability.