1. **Quick Start Guide**

**Step 1: Create a virtual environment (Highly recommended)**

It is best that you create a virtual environment to run this project to avoid it interfering with your current Python environment. You can skip this step if that is not a concern

On macOS / Linux:

# Create the virtual environment

python3 -m venv my\_env

# Activate the virtual environment

source my\_env/bin/activate

On Windows:

# Create the virtual environment

python -m venv my\_env

# Activate the virtual environment

.\my\_env\Scripts\activate

**Step 2: Install Dependencies**

With your virtual environment activated, install the required Python packages using the requirements.txt file:

pip install -r requirements.txt

**Step 3: API keys and Model selection**

Navigate to the .env file and paste in your API keys for Google Maps API, Gemini API **(Not to be confused with Vertex AI API)**, and your model of choice.

**Step 4: Run the project:**

You can either run the direct\_run.py file or use the command-line cli.py.

**For direct\_run.py:**

You can use any of the provided company names and urls, or change the COMPANY\_NAME and COMPANY\_URL variables to those of your choosing.

**For cli.py:**

You can run the solution via console with the following syntax:

python cli.py “COMPANY\_NAME” “COMPANY\_URL”

For example:

python cli.py “OpenStream AI” “https://www.openstream.ai/”

After a minute or two, you will find in the parent directory the nop\_slug.md file, along with a satellite\_images folder containing the satellite image obtained from the extracted company location address. The markdown report should also contain the same image.

1. **Codebase Explanation:**
2. **Code**

The main codebase contains 8 .py files, with 6 being discrete classes used in the pipeline, and 2 being the two mentioned above used to run the pipeline.

* **CompanyResearchAgent.py**: This class contains the main logic for scraping data from the company website. Its main tasks include identifying key pages, extracting text, and finding one physical company address.
* **SummaryAgent.py**: This class contains the main logic for background summarisation and product/service listing using the data gathered by the CompanyResearchAgent using Gemini API.
* **SatelliteAnalysisAgent.py**: This class contains the main logic for analysing the company’s satellite image fetched by Google Maps API using Gemini API.
* **ReportGeneratorAgent.py**: This class contains the main logic for compiling the data from previous steps into a markdown-formatted string using Gemini API.
* **GeminiAPI.py**: This class contains the logic for initialising a Gemini API client as well as a method for content generation.
* **GoogleMapsAPI.py**: This class contains the logic for initialising a Google Maps API client as well as methods for extracting the city, state, and country and for fetching an satellite image of a given address

For more detailed explanation of the code, please refer to the documentations inside each file.

1. **Additional files:**

There are 3 additional files used to facilitate the working of the project.

* **.env**: This file is where the API keys and the selected model name is stored
* **requirements.txt**: This file contains the necessary dependencies that need to be installed (as mentioned above in the **Quick Start** section)
* **README.md**: This file contains important information about the project (that you are reading :))

**III: Key Assumption:**

A key assumption was made for this project, which is that I was required to create a fully automated pipeline. In other words, the system is designed to require only two inputs, a company name and its website URL, and execute the entire workflow without manual intervention. This automated pipeline includes data collection, background summarisation, product listing, satellite image analysis, and final report generation.

**IV: Important Observation:**

1. **Address Extraction: Robustness vs. Cost**

A critical challenge in this assignment was reliably extracting a company's physical address from scraped web content.

Traditional methods like regular expressions (regex) are fast and cost-effective. However, they are inherently brittle. For example, a regex pattern designed for US addresses will fail when encountering different international formats (e.g., those in Singapore or Australia), making it unsuitable for a robust, general-purpose tool. To address this, I included an implementation leveraging Gemini to identify and extract the address from the scraped text. This approach proved highly effective during testing, successfully parsing addresses from websites for companies based in the United States, Singapore, and Australia.

This LLM-based implementation, however, carries additional overhead. It requires an additional API call, which increases both operating cost and processing time. While more robust, the simpler programmatic method remains a viable option where cost and latency is the primary constraint. Furthermore, while powerful and much more robust, using LLMs carries a risk of hallucination, which could result in an altered or entirely fabricated address. Programmatic methods, when they work, are 100% deterministic. I have implemented both approaches in the CompanyResearchAgent class, with the regex-based implementation (*extract\_specific\_address\_block()* method) working well for US-based companies (OpenStream AI and Texwin) but failing in others, while the LLM-based implementation (*extract\_specific\_address\_block\_llm()* method) successfully extracts addresses from all tested websites.

Please refer to the *Find a company location address* section of the run\_full\_research() method of the CompanyResearchAgent class to change between the two. Please note that the *extract\_specific\_address\_block()* method as currently implemented only works for a specific US address format.

1. **URL Selection for Content Scraping**

Another challenge is having to identify the correct source URL (e.g., the definitive page for company background) when multiple links contain similar keywords. Two primary strategies were implemented and considered for this task.

1. **"First Match" Strategy**

This approach assumes the first URL found that matches a keyword (e.g., "about") is the correct one. Its main advantage is efficiency, as it minimises the amount of data passed to the LLM, thereby reducing token consumption and cost. However, it carries the risk of selecting an irrelevant page, leading to incomplete or missing information in the final report. This strategy is implemented as the *find\_key\_page\_urls ()* method in the CompanyResearchAgent class and works well for straightforward websites like Texwin's.

1. **"Brute-Force" Strategy**

This approach scrapes content from all candidate URLs found for a category and concatenates the text. By leveraging the large context window of models like Gemini, it ensures all potentially relevant information is available for analysis. The significant drawback is the increase in input tokens, which raises costs, and the additional "noise" may reduce the LLM's effectiveness in summarising the content. This strategy is implemented as the *find\_key\_page\_urls\_many()* method in the CompanyResearchAgent class.

**IMPORTANT NOTE:** The “Brute-Force” strategy is the main approach used for this solution. You can refer to the provided comments in the *run\_full\_research()* method in the CompanyResearchAgent class if you want to test the “First Match” strategy, but it was found to only work well for the Texwin website for reasons detailed above.

1. **Suggestion for future implementation: Intelligent Disambiguation**

A more advanced solution would be to build a disambiguation mechanism. This could be a set of hard-coded rules or a lightweight machine learning model designed to analyse URLs and some contextual text to select the best candidate(s). This hybrid approach would balance token efficiency with content relevancy but requires additional investment in development time. This final approach was not implemented in the current project but could be taken into consideration for future development.