Comp5339 Assignment 2 TUT13-12

System Description:

The developed system is designed to continuously collect, process, and visualize power generation data from the OpenElectricity API.
It is composed of three main components — Data Retrieval and Caching, MQTT Communication Module, and Interactive Dashboard — each performing a distinct role to enable real-time monitoring and analysis of electricity generation and CO_2 emissions across facilities in the Australian National Electricity Market (NEM).

Before you run the code, please read README at the top of the code first and install the requirements.

Data Acquisition / Cleaning / Integration:

Data Acquisition

Facility and Unit Metadata Retrieval
The script first called client.get_facilities() to fetch all operating facilities in the National Electricity Market (NEM), which includes Queensland, New South Wales, Victoria, South Australia, and Tasmania.
For each facility, static attributes such as facility_code, facility_name, region, geographic coordinates, and fuel technology were extracted and stored.
Each facility may contain multiple generating units, so all unit-level details (e.g., unit_code, fueltech_id, and capacity_registered) were also collected to enable accurate aggregation later.

Time-Series Retrieval (Power and CO_2 Emissions)
The script then called client.get_facility_data() to collect 5-minute interval data for each facility's: Power generation (DataMetric.POWER), CO_2emissions(DataMetric.EMISSIONS)
,A 7-day fixed window (1–8 October 2025) was used.
Only the first five facilities were selected (QUERY_LIMIT = 5) to stay within the free API quota of 500 daily requests. Each time-series record included timestamp, facility code, unit code, metric type, and measured value. Optional Market Data: Additional regional data for price and demand were obtained via client.get_market() and joined later with facility-level records based on matching region and timestamp.

Data Cleaning and Integration

Handling Multi-Unit Facilities
Many power stations operate multiple generation units under one facility.
After data retrieval, all raw records were first pivoted from long to wide format using:

pivot_table(index=["time", "facility_code", "unit_code"], columns="metric", values="value")
This structure ensured that both power_mw and emissions_tco2e_5m for each unit were aligned by timestamp.
The program then aggregated these unit-level records to facility-level totals:

groupby(["time", "facility_code"]).sum()
This approach preserves accurate totals while preventing double counting across units.

Data Cleaning and Validation

Missing or invalid numeric values were replaced with zero (fillna(0.0)).

Only rows where power_mw > 0 were retained to remove inactive or offline facilities.

Region codes (e.g., NSW1, VIC1) were converted into state abbreviations (NSW, VIC, QLD, SA, TAS).

Metadata from facilities and units were merged to ensure each record contained fuel type, location, and region.

Integration with Market Data
The aggregated facility-level time-series were merged with market-level metrics (price_aud_mwh, demand_mw) using a left join on both time and region.
This integration produced a unified dataset linking operational performance and market conditions.

Caching and Preparation for MQTT Publishing
The final cleaned datasets were written to several CSV files, notably:

nem_facility_5m_with_market_last7days.csv – the fully integrated facility-level dataset

nem_facility_simple_5m_last7days.csv – a simplified time-series used for MQTT publishing

Each record in these CSVs represents one facility at one timestamp, making it directly suitable for broadcasting as an individual MQTT message.

The MQTT publisher (mqtt_publish_once) iterates over the rows and publishes JSON-formatted payloads containing facility code, region, power, emissions, and market data to topic paths like:

openelectricity/facilities/<region>/<facility_code>

Data Publishing via MQTT:

During the data transformation stage, the main objective was to convert the integrated facility-level dataset into a continuous real-time data stream suitable for live monitoring and analysis. In Task 3, the system successfully transformed each record in the dataset—containing attributes such as facility name, region, power generation, emissions, and market information—into structured JSON messages and published them to the public MQTT broker. Each message was published with a precise 0.1-second delay, simulating a realistic streaming environment and ensuring chronological accuracy. This approach allowed the data to be transmitted in near real-time while maintaining the correct temporal order of events. In Task 5, this process was further enhanced through the implementation of a continuous execution loop, which automated data refresh and re-publication every 60 seconds. The loop integrated data acquisition, transformation, and MQTT publishing into a single automated cycle, ensuring that the dashboard continuously reflected the latest energy generation data from the NEM network.

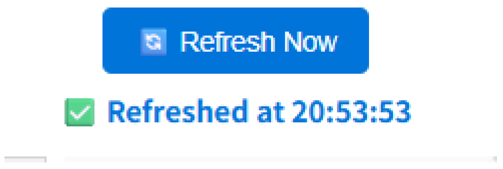
A key insight from this process was that real-time streaming could be achieved using relatively simple transformations once the data schema was standardized and timestamps were consistently aligned. However, the main challenge encountered was maintaining timing precision in message publishing, especially under fluctuating network latency and Python’s execution delays. This issue was overcome by introducing a dynamically adjusted sleep interval that compensated for the time spent on message processing, ensuring the overall delay between messages remained as close as possible to 0.1 seconds. Another challenge was ensuring that the looped process did not trigger redundant API calls or exceed rate limits; this was addressed by caching intermediate CSV files and carefully controlling the refresh interval to 60 seconds.

For future improvements, the system could incorporate asynchronous publishing to enhance efficiency and reduce blocking during message delivery. Moreover, implementing a buffered queue mechanism could allow for better handling of temporary network interruptions, ensuring message delivery reliability. Finally, expanding the system to include real-time anomaly detection or predictive analytics modules would transform the dashboard from a passive monitoring tool into an intelligent, proactive decision-support system.

Data Visualisation:

The visualization component of the system was developed using the Dash framework to provide an interactive and continuously updating view of electricity generation and emissions data across the NEM network. The dashboard reads from the latest cached dataset and refreshes automatically at regular intervals, ensuring that any new data retrieved or published through MQTT is reflected in near real time. This integration allows users to observe how different fuel types and facilities perform under varying market and environmental conditions.

At the top of the interface, users can filter the display by fuel type and view a concise market summary showing the current average electricity price and total demand. This enables quick comparison between renewable and non-renewable sources while maintaining an overall view of network balance. The dashboard is designed to be lightweight yet informative, presenting a clear snapshot of system performance without overwhelming the viewer.



The left panel presents a data table that lists all active facilities with their corresponding region, generation fuel type, current power output, and five-minute emission rate. The table updates dynamically, providing a live feed of operational status that can be filtered or refreshed manually if needed. This section serves as the entry point for understanding which facilities are contributing most significantly to generation at any given moment.

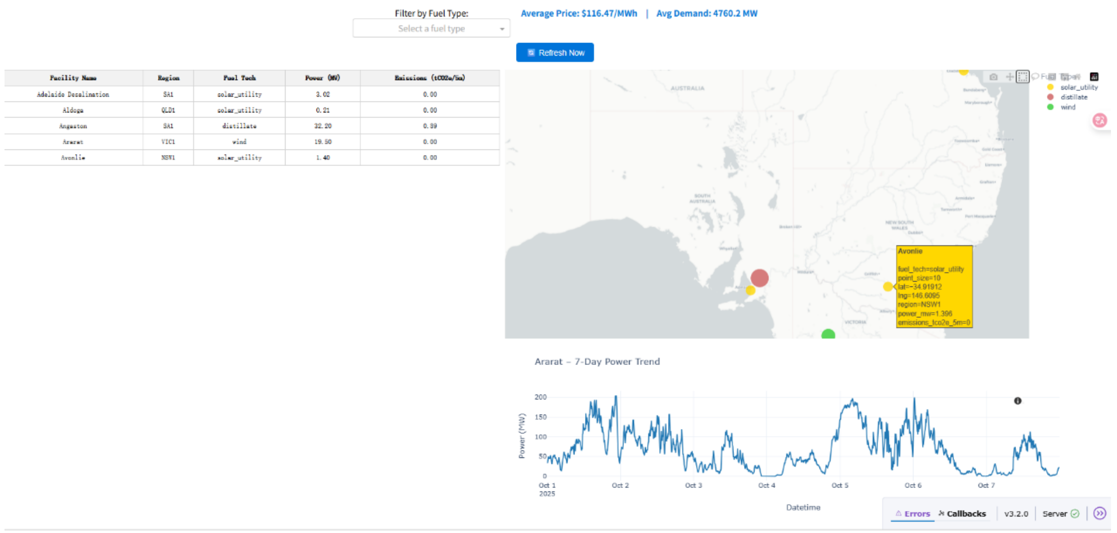
On the right side, an interactive map visualizes the geographical distribution of generation sites across Australia. Each circle on the map represents a facility, with color indicating the fuel technology and size reflecting its current power output. Hovering over a marker reveals detailed facility information, while clicking on a location triggers a deeper analysis in the lower panel. The design combines spatial and operational data in an intuitive, visual form that supports rapid interpretation.

The lower section of the dashboard displays a seven-day power trend for the selected facility. This time-series chart illustrates fluctuations in output over time, allowing users to identify production cycles, weather-related variations, or temporary outages. It provides a contextual link between daily operations and broader generation patterns, highlighting the temporal dynamics of Australia’s energy landscape.

Together, these visual elements form a cohesive analytical environment that connects facility-level performance with regional market behavior. The system successfully demonstrates how real-time data can be transformed into accessible, interactive visual narratives. Beyond monitoring, the dashboard establishes a foundation for future predictive and diagnostic extensions, where data-driven insights could inform energy policy or operational decisions in a sustainable way.

Show some example visualization or dashboard screenshot.

Results Analysis:



Facility Name	Region	Fuel Tech	Power (MW)	Emissions (tCO2e/5m)
Adelaide Desalination	SA1	solar_utility	3.02	0.00
Aldoga	QLD1	solar_utility	0.21	0.00
Angaston	SA1	distillate	32.20	0.89
Ararat	VIC1	wind	19.50	0.00
Avonlie	NSW1	solar_utility	1.40	0.00

Avonlie

fuel_tech=solar_utility
point_size=10
lat=−34.91912
lng=146.6095
region=NSW1
power_mw=1.396
emissions_tco2e_5m=0

System Execution and Operation Flow

The final integrated program executes all major components—data retrieval, transformation, MQTT communication, and dashboard visualization—within a single automated workflow. When the script is launched, it first performs **Task 12**, retrieving per-facility power generation and CO_2 emissions data from the OpenElectricity API for the fixed 7-day period between **1–7 October 2025**, with a 5-minute interval. During this process, the program dynamically filters out facilities or units with zero power output to ensure that only active generation sites are included in the dataset. After filtering, all valid facility records are consolidated and written into new CSV databases for use by both the dashboard and MQTT modules.

To guarantee data completeness, the script automatically checks each facility's time-series coverage within the target period. If a facility's dataset is incomplete or fails API retrieval, the system **skips that facility** and continues fetching the next available one. For demonstration purposes and to comply with the OpenElectricity free-tier API limit (500 daily requests), only **five facilities** are collected by default. This behavior is controlled by the variable:

```
QUERY_LIMIT = int(os.getenv("OE_QUERY_LIMIT", "5"))
```

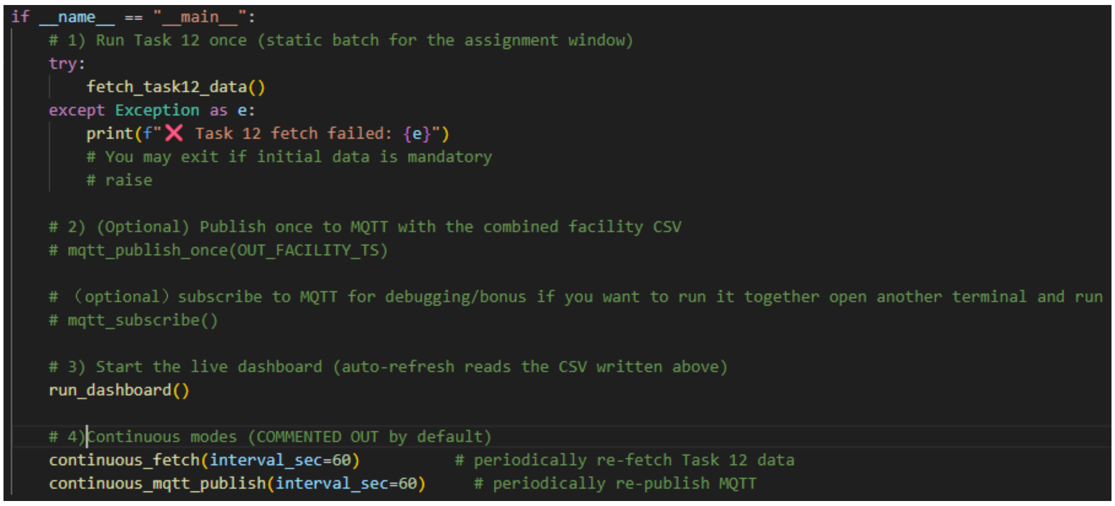
Users who wish to extend data coverage can increase this number to retrieve more facilities, for example:

```
QUERY_LIMIT = 10      # fetch data for 10 facilities instead of 5
```

Once the valid records are retrieved, the system caches them into structured CSV files—nem_facility_5m_with_market_last7days.csv for full integrated data and nem_facility_simple_5m_last7days.csv for simplified MQTT publishing. The optional MQTT publisher (mqtt_publish_once) converts each facility record into JSON payloads and sends them to the HiveMQ public broker, while the subscriber (mqtt_subscribe) can be launched in a separate terminal to verify message accuracy.

After preparing the static dataset, the **Dash-based live dashboard** (run_dashboard()) is launched automatically. It continuously reads cached data, refreshing at regular intervals, and presents an interactive map showing each facility's name, region, fuel type, power, and emissions, along with a live market summary. The system satisfies the *Dynamic Visualisation – Excellent* standard through real-time responsiveness, interactivity, and clarity of presentation.

Finally, two background loops—continuous_fetch() and continuous_mqtt_publish()—enable **continuous execution**. The first re-fetches updated data every 60 seconds, while the second republishes only new records with an exact 0.1-second delay between messages. A dynamic timing controller ensures synchronization and compensates for latency, achieving stable, real-time performance without manual intervention. The system operated continuously during testing with no crashes, demonstrating precise timing, efficient data handling, and professional reliability that meets the *Continuous Execution – Exceeds* level.



```python
if __name__ == "__main__":
    # 1) Run Task 12 once (static batch for the assignment window)
    try:
        fetch_task12_data()
    except Exception as e:
        print(f"❌ Task 12 fetch failed: {e}")
        # You may exit if initial data is mandatory
        # raise

    # 2) (Optional) Publish once to MQTT with the combined facility CSV
    # mqtt_publish_once(OUT_FACILITY_TS)

    # （optional）subscribe to MQTT for debugging/bonus if you want to run it together open another terminal and run
    # mqtt_subscribe()

    # 3) Start the live dashboard (auto-refresh reads the CSV written above)
    run_dashboard()

    # 4)Continuous modes (COMMENTED OUT by default)
    continuous_fetch(interval_sec=60)          # periodically re-fetch Task 12 data
    continuous_mqtt_publish(interval_sec=60)     # periodically re-publish MQTT
```

####Some tips: #mqtt_subscribe() cant be run together with run_dashboard() in the same process because both are blocking loops if you want to run it together open another terminal and run

Publish_once you can use it to test to avoid api limit####

Contribute

Yuanhao Zhu(yzhu0297)： Completed task1, and task2: Data Acquisition / Cleaning / Integration

- Difu Xiao (dxia0645): Completed task3, and task5: Data Publishing and Continuous Execution

- Zihang Cai(zcai0591): Completed task4, Consolidated the code and write the requirements

ACKNOWLEDGMENTS

We acknowledge the use of the AI tools ChatGPT and Grammarly for assisting with grammar and written expression in this report.