

Data Orchestration for Emerging Technology Analysis

Jennifer Melot



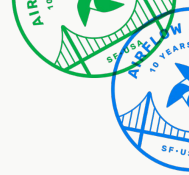
Outline

- What is CSET/ETO
- Airflow at CSET
 - Scholarly literature pipelines: handling long-running tasks
 - Sequence of SQL DAGs: abstracting away Airflow
 - Airtable DAGs: integrating human inputs
- Lessons learned



About CSET and ETO

- The [Center for Security and Emerging Technology](#) studies the security implications of emerging technologies, including AI, advanced computing, and biotech
 - Core audience is policymakers, core themes are geopolitical
 - Part of Georgetown University's School of Foreign Service
- CSET's [Emerging Technology Observatory](#) builds public data resources (tools, visualizations, datasets) to inform critical decisions on emerging tech issues

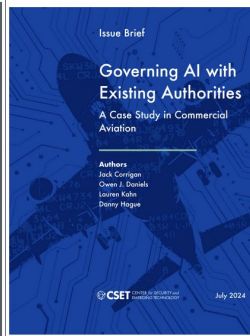


Things we build: Reports

We write reports that are read by policymakers and others needing to make decisions on emerging technology-related issues.

Some of these reports draw on our data holdings.

Airflow pipelines keep the tables used in these analyses up to date.



Analysis

[Governing AI with Existing Authorities](#)

Jack Corrigan, Owen Daniels, Lauren Kahn and Danny Hague | July 2024

A core question in policy debates around artificial intelligence is whether federal agencies can use their existing authorities to govern AI or if the government needs new legal powers to manage the technology. The authors argue that relying on existing authorities is the most effective approach to promoting the safe development and deployment of AI systems, at least in the near term. This report outlines a process for identifying existing legal authorities that could apply to AI and highlights areas where additional legislative or regulatory action may be needed.



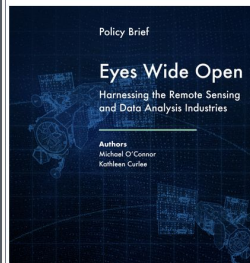
Formal Response

[Comment on Commerce Department RFI 89 FR 27411](#)

Catherine Aiken, James Dunham, Jacob Feldgoise, Rebecca Gelles, Ronnie

Kinoshita, Mina Narayanan and Christian Schoeberl | July 16, 2024

CSET submitted the following comment in response to a Request for Information (RFI) from the Department of Commerce regarding 89 FR 27411.



Analysis

[Eyes Wide Open: Harnessing the Remote Sensing and Data Analysis Industries to Enhance National Security](#)

Michael O'Connor and Kathleen Curlee | July 2024

The U.S. government has an opportunity to seize strategic advantages by working with the remote sensing and data analysis industries. Both grew rapidly over the last

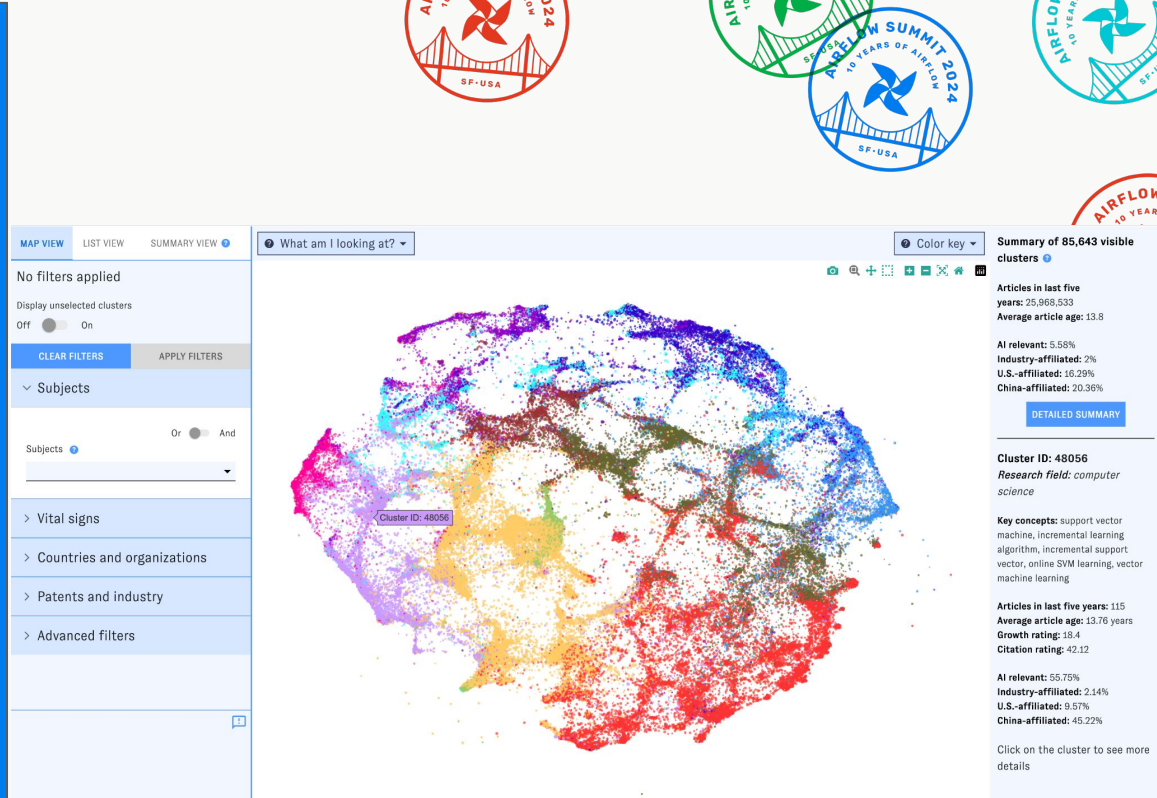


Things we build: Web Applications

We build public tools (at eto.tech) to help policymakers answer questions about emerging technologies.

At right is sciencemap.eto.tech . It's based on > 100M scholarly publications and meant to help users better understand the research landscape.

Airflow pipelines move data from BigQuery to Cloud SQL for use in these applications.



Data team structure

- ~14-person team (out of 60+ person organization)
- 3 Data Research Analysts
 - Direct support to non-coder research analysts
- 3 Data Scientists + NLP Engineer
 - Expertise in particular datasets
 - Dataset maintenance and data augmentation projects
- 1 Software Engineer + Technical Lead
 - Data pipeline maintenance and development
 - Web application development
- Other staff: translation lead, ETO analytic lead, survey specialist, director and deputy director of data science



Airflow at CSET



Airflow at CSET

- 93 dags running in Cloud Composer (Google Cloud Platform's managed Airflow)
- Used since 2020
- Generated tables often range from 100s of millions to billions of records
- Tasks range from ETL to model deployment to long-running tasks like linkage and clustering

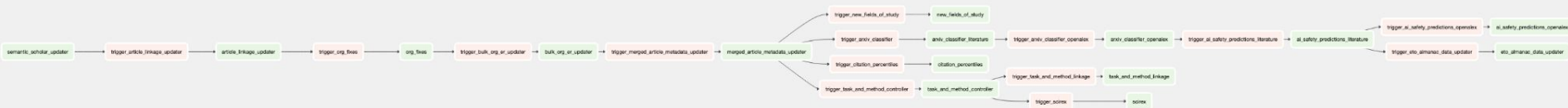


Working with scholarly literature: long-running tasks



Ingesting + Augmenting Scholarly Literature

- CSET provides analysis based on bibliometric data
 - E.g. which companies are producing the most AI-related research
- Updating bibliometric data involves running several DAGs:
 - ETL from vendors
 - Linking articles across sources + aggregating canonical metadata
 - Running classifiers
 - Clustering articles



Ingesting + Augmenting Scholarly Literature

- Most DAGs involve using BigQuery operators to run large analytic queries
- Most validation happens with BigQueryCheckOperators
- We use Google Cloud Storage to store intermediate files
- We use Dataflow operators to run inference using custom models and to do data cleaning
- We run custom Python code using Kubernetes Engine operators



Challenges with long-running tasks

- We sometimes use BashOperators to run commands over ssh on VMs with large amounts of memory
- If we do nothing to avoid it, we'll sometimes see this after several days

```
[2024-07-06, 04:03:29 UTC] {job.py:213} ERROR - Job heartbeat got an exception
```

```
Traceback (most recent call last):
```

```
File "/opt/python3.8/lib/python3.8/site-packages/sqlalchemy/engine/base.py", line 3371, in _wrap_pool_connect
    return fn()
```

```
File "/opt/python3.8/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 327, in connect
    return _ConnectionFairy._checkout(self)
```

```
File "/opt/python3.8/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 894, in _checkout
    fairy = _ConnectionRecord.checkout(pool)
```

```
File "/opt/python3.8/lib/python3.8/site-packages/sqlalchemy/pool/base.py", line 493, in checkout
    rec = pool._do_get()
```


Workaround: Sensors

- We now:
 - Run long-running tasks in the background on the remote VM
 - Upload an empty file to a particular location on Google Cloud Storage when the task has completed
 - A GCSObjectExistenceSensor waits for that file to appear and allows the DAG to proceed once it is present
- Downsides: clunky, have to ssh to the VM to view logs

Abstracting away Airflow

Airflow has a learning curve...

- Lots of enthusiasm among data team members for learning Airflow
- Not everyone focuses on building data pipelines
- Common use case - automate this sequence of sql queries, update documentation, back tables up



Generating “Sequence of SQL” DAGs

- We provide a script that configures:
 - A directory of SQL queries
 - A file that specifies query order
 - A directory of SQL data checks
 - A directory of schemas with column-level documentation
 - A file update script
- New “Sequence of SQL” DAGs are dynamically generated from these inputs
 - [Astronomer documentation on this](#)



Finding DAG configurations

We look in a “sequence directory” for CSVs containing sequences of queries.

We use the names of these files to configure a DAG that will run those queries.

We then insert that dag into the global namespace.

```
if os.path.exists(sequence_dir):
    for fi in os.listdir(sequence_dir):
        if not fi.endswith(sequence_filename_suffix):
            continue
        dagname = fi.replace(sequence_filename_suffix, "")
        globals()[dagname] = create_dag(dagname)
```

Generating DAGs

Within the DAG, we open a CSV containing sql query names and names of production datasets where those queries should eventually write their tables.

We run each of these queries in series, writing the resulting tables to a staging dataset, and then run checks, copy to production, and so on.

```
def create_dag(dag_name: str) -> DAG:
    """
    Creates a dag to write the tables specified in a sequence file named dag_name.csv
    :param dag_name: name of the dag
    :return: dag that produces the tables in dag_name.csv
    """
    dag = DAG(dag_name,
              default_args=get_default_args(),
              description=f"table updates for {dag_name}",
              schedule_interval=None)

    with dag:
        start = DummyOperator(task_id="start")
        prev = start

        lines = retrieve_uncommented_lines(f"{sequence_dir}/{dag_name}{sequence_filename_suffix}")
        production_tables = []
        for line in csv.DictReader(lines):
            next = BigQueryInsertJobOperator(
                task_id="create_"+line["table_name"],
                configuration={
                    "query": {
                        "query": "{% include '"+f"sql/{dag_name}/{line['table_name']}.sql+"' %}",
                        "useLegacySql": False,
                        "destinationTable": {
                            "projectId": PROJECT_ID,
                            "datasetId": f"staging_{dag_name}",
                            "tableId": line["table_name"]
                        },
                    },
                    "allowLargeResults": True,
                    "createDisposition": "CREATE_IF_NEEDED",
                    "writeDisposition": "WRITE_TRUNCATE"
                }
            )
        )
```



Using “Sequence of SQL” DAGs

- Benefits: no knowledge of Airflow required, boilerplate DAGs reduced
- Downsides: several components need configuration, some find this a bit overwhelming/confusing



Other dynamic DAG generation use cases

- Data ETL

- Each table can be ingested with a separate DAG on separate schedules with custom configuration

- Webscraping

- We write our webscrapers to have a standard structure - a scraper script and a parser script
- We dynamically generate our webscraper DAGs from config files pointing to these scripts

- Integrating human data annotation

Working with human inputs from Airtable



Airtable + Airflow

- Several of our workflows include analysts and work something like this:
 - A set of records are placed in Airtable (either manually or via an Airflow pipeline)
 - Humans clean and/or add additional metadata to these records
 - We ingest records that are complete back into BigQuery on a regular basis

Airtable + Airflow

- We dynamically generate dags for BigQuery to Airtable and Airtable to BigQuery workflows: <https://github.com/georgetown-cset/airtable-etl>
- Users write config files and sql queries to configure which tables are imported and the output table schemas
- The resulting DAGs are triggered from other DAGs that need to interact with Airtable, or are scheduled

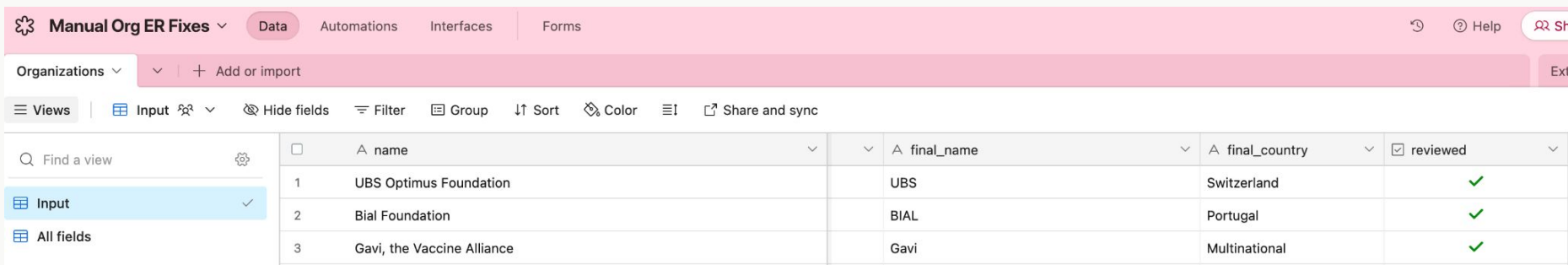
Example: updating manual ER

- Our raw data contains multiple variants of organization names
 - e.g. “International Business Machines Corporation” and “IBM (United States)”
- Various solutions in progress
 - Research Organization Registry (<https://ror.org/>)
 - Internal model-based ER effort
- In the meantime, we try to manually clean up the “worst offenders”



Example: updating manual ER

- We add “worst offender” (by number of affiliated papers or other metrics) organizations to an Airtable base
- An Airflow pipeline pulls orgs with “reviewed” checked on a weekly basis and updates a BigQuery table
- Cleaned organizations are available for e.g. our metadata merge DAG



The screenshot shows the Airtable interface for a base named "Manual Org ER Fixes". The table has four columns: "name", "final_name", "final_country", and "reviewed". The "reviewed" column has a checkmark icon in the header, indicating it is a checkbox field. The table contains three rows of data, all with the "reviewed" checkbox checked.

	A name	A final_name	A final_country	reviewed
1	UBS Optimus Foundation	UBS	Switzerland	✓
2	Bial Foundation	BIAL	Portugal	✓
3	Gavi, the Vaccine Alliance	Gavi	Multinational	✓

Lessons Learned



Lessons Learned

- Originally a data engineering team of 1, and then we evolved
 - Assigning responsibility for addressing e.g. vendor schema updates
 - Important to give the rest of the data team visibility into task failures and lineage
- Special consideration needed for long-running tasks
- Dynamically generating DAGs is a big help
 - Keeps code DRY
 - Helps team members generate DAGs without having to learn Airflow



Questions?

jennifer.melot@georgetown.edu

