Building a Parallel ERA5 Download and Cleaning Pipeline for Large-Scale Weather Analysis and AI Training

Harsh Patel, Eric Keller, Jonathan M. Snodgrass, Thomas J. Overbye
Texas A&M University, College Station, TX
Email: harshpatel@tamu.edu

***Abstract*—Large-scale weather data remains essential for power grid and renewable energy research, yet accessing decades of ERA5 reanalysis data presents significant practical challenges due to inconsistent file structures and asynchronous API constraints. This work develops a parallel, restart-safe pipeline that automates the end-to-end process of downloading, cleaning, and storing 84 years of ERA5 single-level data for 27 variables on a 0.25° × 0.25° global grid. The pipeline processes each calendar day as an independent task, retrieving one GRIB file per day from the Copernicus Climate Data Store API and consolidating five internal datasets into a unified 24-hour structure. A centralized orchestrator coordinates two worker pools—thread-based for downloads and process-based for cleaning—while enforcing CDS rate limits and enabling fully idempotent restart behavior. Benchmarked on a 64-core workstation with network-mounted storage, the parallel configuration achieves a 3.7× speedup over a minimally parallel baseline, reducing the projected 1940-2024 rebuild time from approximately 148 days to 40 days. These results demonstrate that maintaining a complete historical ERA5 archive is feasible on a single machine, enabling practical applications in large-scale analysis, database ingestion, and AI training workflows. The resulting database will support retrieval-augmented generation (RAG) systems for large language models (LLMs), providing contextual weather data for grid resilience analysis and decision support.**

***Index Terms*—ERA5, reanalysis, Copernicus Climate Data Store, data pipeline, parallel processing, weather database, power systems, renewable energy, artificial intelligence, retrieval-augmented generation**

I. Introduction

Modern power grid operations and resilience studies require access to large-scale climate and weather data to evaluate renewable energy generation and assess system performance under both normal and extreme conditions [1], [2]. Historical datasets such as ERA5, provided by the Copernicus Climate Data Store (CDS), offer detailed atmospheric information spanning over eight decades [3], [4]. However, the dataset's volume, internal structure, and the API's asynchronous behavior create substantial barriers to efficient access and management. Grid modeling frameworks require preprocessed, structured weather data, yet downloading and cleaning decades of raw ERA5 data demands significant time and technical expertise, limiting accessibility for researchers.

Copyright ©2026 IEEE. Personal use of this material is permitted. This paper was presented at the Texas Power and Energy Conference (TPEC), College Station, TX, February 2026.

To address these challenges, this work presents an automated pipeline to download, clean, and parallelize the processing of 84 years of global ERA5 hourly weather data. The pipeline is designed to handle the CDS API's evolving structure and operational constraints while producing structured, cleaned daily products that enable direct extraction of specific weather variables for modeling, simulation, and analysis—without requiring users to reimplement low-level download and cleaning procedures.

The motivation for establishing this comprehensive ERA5 database extends beyond traditional power system analysis. The cleaned, structured dataset serves as a foundation for training artificial intelligence models and implementing retrieval-augmented generation (RAG) systems for large language models (LLMs) [5], [6]. RAG enables LLMs to access domain-specific, time-series weather data dynamically, grounding model responses in authoritative historical records rather than relying solely on pre-trained knowledge [7]. This capability is particularly valuable for grid resilience applications, where accurate retrieval of historical weather patterns under extreme conditions can inform decision-making and risk assessment [8]. By creating a queryable, AI-ready database, this work facilitates the development of intelligent systems that can reason about complex climate-grid interactions with improved accuracy and interpretability.

The remainder of this paper is organized as follows: Section II describes the ERA5 data structure and variable selection. Section III outlines the pipeline architecture, including download, cleaning, and parallelization strategies. Section IV presents performance evaluation and scaling results. Section V discusses conclusions and future work, including database integration and AI training workflows.

II. Background and Data Characteristics

The ERA5 reanalysis dataset, produced by the European Centre for Medium-Range Weather Forecasts (ECMWF) under the Copernicus Climate Change Service (C3S), provides hourly estimates for numerous atmospheric, land, and ocean variables from 1940 to the present at a global grid resolution of approximately 31 km [4]. Reanalysis data are generated through data assimilation, wherein model forecasts are combined with observations from distributed sensor networks to reconstruct a physically consistent record of historical weather

conditions [9]. The dataset is publicly accessible through the Copernicus Climate Data Store, which offers both interactive and programmatic access via the CDS API [3]. Figure 1 illustrates the data assimilation concept employed by ERA5.

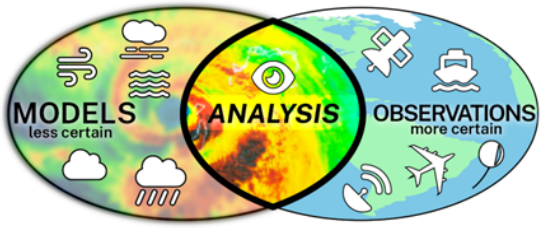

Fig. 1. Conceptual view of data assimilation: observations and numerical/physics model forecasts are combined to produce reanalysis fields (e.g., ERA5). Adapted from NOAA's "10-Year Strategy for Data Assimilation" [9].

In ERA5, not all variables are produced identically. Analysis fields represent the atmospheric state at a given hour, while others originate from short forecasts initiated at each analysis time. These forecast-derived variables employ a combination of time, step, and valid_time dimensions to describe temporal validity. Variables may be instantaneous, accumulated over intervals, or represent mean rates, resulting in a more complex structure than a simple hourly grid.

The dataset processed in this work comprises ERA5 hourly data on single levels for all latitudes and longitudes on a 0.25° × 0.25° global grid from 1940 to the present, encompassing 27 variables. These variables were selected to capture the full spectrum of atmospheric and surface processes that influence renewable generation (solar radiation, wind speed) and grid resilience (temperature extremes, precipitation, flooding, icing, drought) [10]–[12]. Each variable corresponds to one or more high-impact weather hazards, forming a comprehensive indicator set for modeling normal operations, extreme events, and grid resilience scenarios.

ERA5 data are downloaded in Gridded Binary (GRIB) format due to its space efficiency and predictable structure. Each CDS API request yields a single GRIB file containing five internally separated datasets, where each dataset groups variables sharing common characteristics. The pipeline design depends critically on understanding this structure, as each dataset exhibits distinct dimensions, metadata conventions, spatial grids, and temporal structures.

A. Problem Statement

Each daily ERA5 GRIB file contains five internally separated datasets that differ in dimensions, metadata conventions, spatial grids, and temporal structure. Accumulated surface and radiation fields include forecast steps; ocean-wave variables appear on a 0.5° grid; most near-surface analysis variables use the 0.25° grid; and certain diagnostic indices originate from the forecast stream. This mixture creates a non-uniform data layout that must be normalized before large-scale automated use. Consistently handling this structure defines the core technical challenge addressed by the pipeline architecture.

III. Pipeline Architecture

The data pipeline automates the end-to-end process of downloading, cleaning, and storing ERA5 hourly single-level weather data from 1940 to the present. The workflow comprises three main phases: Download, Cleaning, and Parallelization/Storage. Each phase addresses specific challenges in large-scale data retrieval and preprocessing.

A. Download Phase

The download phase retrieves ERA5 data from the Copernicus Climate Data Store through the CDS API. This stage manages the retrieval of raw weather data spanning 1940 to the present for 27 variables across numerous small, independent requests.

The CDS API employs asynchronous job submission that transitions through states (Queued, In Progress, Complete), necessitating sequential request handling. Data are downloaded in GRIB format rather than NetCDF due to superior predictability and stability for automation. Under the current CDS API, NetCDF requests yield 3–5 separate files per request depending on variables, increasing complexity. GRIB consistently delivers one file per request, wherein the CDS has already partitioned variables into five datasets based on their characteristics. GRIB also provides smaller file sizes and cleaner metadata.

Each download task instantiates its own CDS client, formulates its request, and writes output to a temporary .part file. Upon completion, the pipeline validates the GRIB end-of-file marker ("7777") to ensure completeness before renaming to .grib, preventing corrupted files from entering subsequent processing. The pipeline also verifies whether a given day's GRIB file already exists and is valid, enabling seamless restarts without redundant downloads.

The design emphasizes resilience through requesting exactly one day of data per API call rather than bulk multi-day ranges. This granularity simplifies error tracking, facilitates parallelization, and aligns with Copernicus recommendations. If a request fails, the affected day is immediately identifiable, and retry logic does not impact other tasks. Daily requests outperform monthly requests across metrics: smaller file sizes, reduced download latency, lower memory consumption, and improved parallelism effectiveness.

To comply with CDS API constraints—including limits on items/fields and a maximum payload size of approximately 120,000 fields or 10 GB—daily requests proved optimal. Testing revealed that exceeding approximately 10 daily requests per minute triggers HTTP 429 responses. To avoid rate limiting, the pipeline employs a token-bucket rate limiter, maintaining a steady rate of approximately 7–8 requests per minute. Each failed request is retried up to four times with exponential backoff. In practice, each daily GRIB file is approximately 1 GB and downloads in 5–7 minutes. The system can safely maintain 16–32 concurrent downloads without encountering rate-limit errors or overwhelming storage.

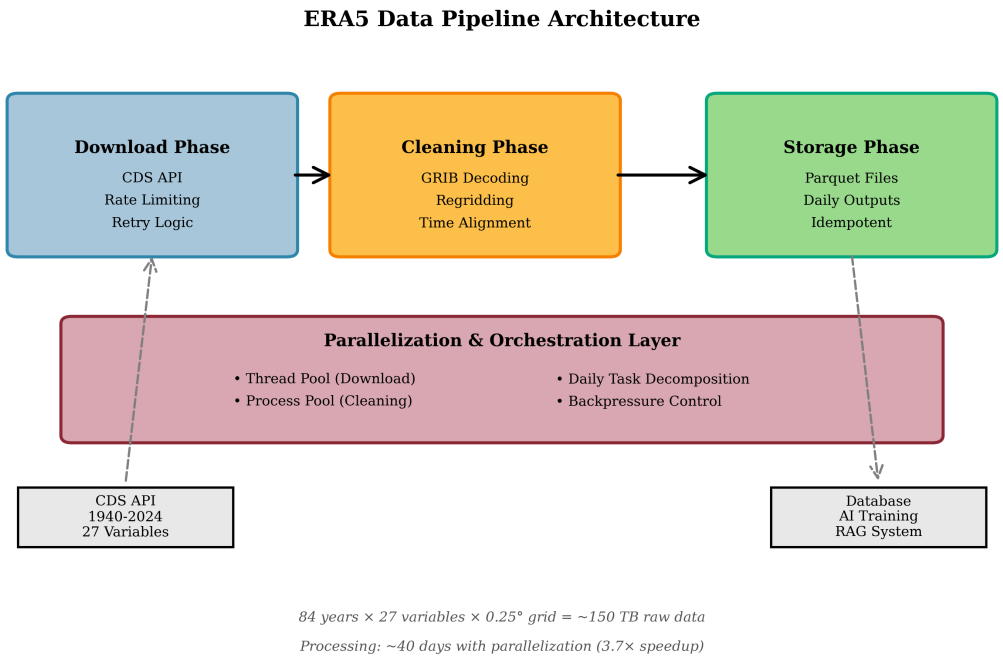

Fig. 2. ERA5 data pipeline architecture. The three main phases (Download, Cleaning, and Storage) are coordinated by a parallelization layer that treats each calendar day as an independent task and produces one cleaned 24-hour output per day for 27 variables.

B. Cleaning Phase

The cleaning phase transforms a 24-hour GRIB file (one day of ERA5 data) containing five multi-structured datasets into a single unified 24-hour dataset with common structure and grid resolution. The cleaning process comprises six main steps: coordinate removal, regridding, dimension resizing, and back-calculation of cumulative variables.

Role of the step Variable

Before describing the processing steps, it is essential to clarify the role of the step variable, which affects all datasets either structurally or substantively. For Datasets 1, 2, and 4 (soil moisture layers, ocean wave parameters, and instantaneous atmospheric/surface state variables), step exists solely as a coordinate—essentially metadata—typically remaining at 0 throughout the dataset. For these datasets, step does not influence the data values.

Datasets 3 and 5, which contain accumulated forecast variables and stability indices, include step both as a dimension and coordinate, indicating multiple forecast lead times (typically at hour 0 and hour 12). Each (time, step) combination possesses a distinct physical value. For these datasets, step must be flattened correctly into 24 hourly rows without interpolation. The variable valid_time (defined as valid_time = time + step) provides the actual wall-clock time.

Dataset 5 contains only one variable (K-index), belonging to the diagnostic forecast family. Both families derive from forecast-based fields but behave differently: diagnostic forecast variables are instantaneous values computed during the forecast, not cumulative. The flattening process for Dataset 5 mirrors that of Dataset 3.

Step 1: Drop Unused Coordinates and Trivial step

Each dataset includes unused coordinates such as number, depthBelowLandLayer, meanSea, and surface, which are irrelevant and often contain only constant values. The first cleaning step removes these unused coordinates and eliminates the trivial step coordinate when it does not vary. This is accomplished via ds.drop_vars() for each dataset, removing redundancy without losing information.

Step 2: Regrid Dataset 2 to 0.25° × 0.25°

All datasets except Dataset 2 reside on a 0.25° × 0.25° grid. To enable merging, Dataset 2 (on a 0.5° × 0.5° grid) must be converted to 0.25° × 0.25°. This is achieved using xarray's reindex method with fixed coordinates. The method matches the dataset to a new coordinate grid by exact label matching: existing grid points retain their values; non-existent points are filled with NaN; no interpolation is performed. This preserves data accuracy and avoids generating artificial data.

Step 3: Drop valid_time Where step Is Not Used

The third step removes the valid_time coordinate for datasets lacking step as a dimension. The valid_time variable is relevant only when forecast lead times are employed. In datasets without step (Datasets 1, 2, and 4), valid_time is identical to time. The pipeline drops valid_time via ds.drop_vars(), removing redundancy since time already carries the correct timestamps.

Step 4: Convert Cumulative Totals into Per-Step Amounts

Step 4 converts cumulative totals in Dataset 3 into per-step amounts by back-calculating the increment between each forecast step. Dataset 3 contains ten variables, six of which are inherently cumulative:
surface_runoff, sub_surface_runoff, snowfall, surface_solar_radiation_downwards, total_precipitation, and total_sky_direct_solar_radiation_at_surface. These extensive physical quantities accumulate over time, so differencing across steps yields the amount occurring within each interval. The pipeline computes the difference across the step dimension using `.diff()`, concatenating first-step values and differences to produce per-step amounts instead of running totals.

The remaining variables
10m_wind_gust_since_previous_post_processing,
maximum_2m_temperature_since_previous_post_processing,
instantaneous_10m_wind_gust, and
mean_total_precipitation_rate
are diagnostic or rate outputs, not cumulative totals. These fields are kept as-is and aligned using valid_time.

Step 5: Flatten (time, step) into 24-Hour Timeline

Step 5 transforms the (time, step) structure in Datasets 3 and 5 into 24 hourly rows. The pipeline uses a helper function that stacks time and step dimensions into a single index, attaches valid_time as a new coordinate (wall_time), and swaps the main dimension to wall_time. If multiple entries share the same wall_time, the pipeline groups by wall_time and retains the last record. Helper coordinates are dropped, leaving only wall_time, latitude, longitude, and variables. A clean 24-hour index from 00:00 to 23:00 is constructed, the dataset is sliced to that range and reindexed so missing hours become NaN. Finally, wall_time is renamed to time, and dimensions are reordered to (time, latitude, longitude).

Step 6: Merge All Datasets into Single 24-Hour Product

The final step merges all datasets into a single, unified 24-hour dataset. The pipeline gathers the cleaned, regridded, and transformed datasets (1 through 5) and passes them to xr.merge. The merged dataset can optionally be cast to float32 for space efficiency. Because each day is cleaned independently, this structure integrates naturally with the parallelization layer, where multiple days can be processed concurrently without cross-dependencies.

C. Parallelization Layer

The parallelization layer coordinates the download and cleaning phases to enable concurrent processing of multiple days of ERA5 data. The core principle is to treat each calendar day as an independent unit of work. This is implemented through a centralized orchestrator that manages downloading and cleaning in a non-blocking manner, assigning work to two separate worker pools—one for downloading and one for cleaning—to ensure concurrency and maintain stage independence. The overall design is summarized in Figure 3.

1) Daily Task Decomposition and Orchestrator Role: The parallelization layer decomposes the full date range into independent daily tasks managed by a centralized orchestrator. Processing data one day at a time proves safer and more efficient than requesting large multi-day blocks, which are vulnerable during failures and impose greater I/O pressure. The orchestrator constructs a list of calendar days between start and end dates, which becomes the queue for both downloading and cleaning. Since each day is self-contained, no coordination across days is required, and dependencies between them do not exist.

Daily independence enhances reliability. If one day fails during download or cleaning, the failure remains isolated. The orchestrator logs the failure and continues processing. Failed days can be retried later without affecting previously successful tasks. Another critical aspect is idempotency: before scheduling work for a day, the orchestrator verifies whether the cleaned Parquet file already exists. If so, that day is automatically skipped, enabling the pipeline to restart without losing progress. This daily structure also maintains predictable memory usage, as each worker processes exactly one day at a time.

2) Worker Pools for Downloading and Cleaning: The orchestrator distributes work to two distinct worker pools. The download workers employ a thread pool because each download spends most of its time waiting on CDS servers, network responses, or disk writes. Threads are ideal for this wait-heavy workload. Although the CDS API processes requests sequentially server-side, having multiple threads keeps more requests in flight, eliminating dead time between requests.

Cleaning employs separate worker processes. Cleaning a single GRIB file involves decoding approximately 1.3 GB of compressed binary data, regridding, flattening time-step combinations, building the 24-hour timeline, merging five datasets, and writing a Parquet file. The actual bottleneck is I/O—most cleaning time is spent opening GRIB files, loading chunks from disk, and writing Parquet files. Multiple processes provide each cleaner its own memory space and circumvent GIL limitations.

Cleaning does not scale linearly with additional workers because disk bandwidth saturates before CPU utilization peaks. Once several cleaners simultaneously load or write files, the I/O subsystem becomes the limiting factor. The pipeline uses a moderate number of cleaning workers to keep CPU cores busy without overwhelming disk bandwidth.

3) Controlled Concurrency and Pipeline Backpressure: The orchestrator enforces a limit on how many days may be in-flight simultaneously. Once active downloads reach this limit, scheduling of new days pauses until either a download completes or a cleaner becomes available. This maintains steady pipeline pacing. If downloads proceed slowly, cleaners wait. If cleaning takes longer, downloading slows correspondingly. This steady, self-balancing behavior makes the parallelization layer practical for processing multi-decade spans, as the backpressure mechanism maintains smooth progress without overwhelming the system.

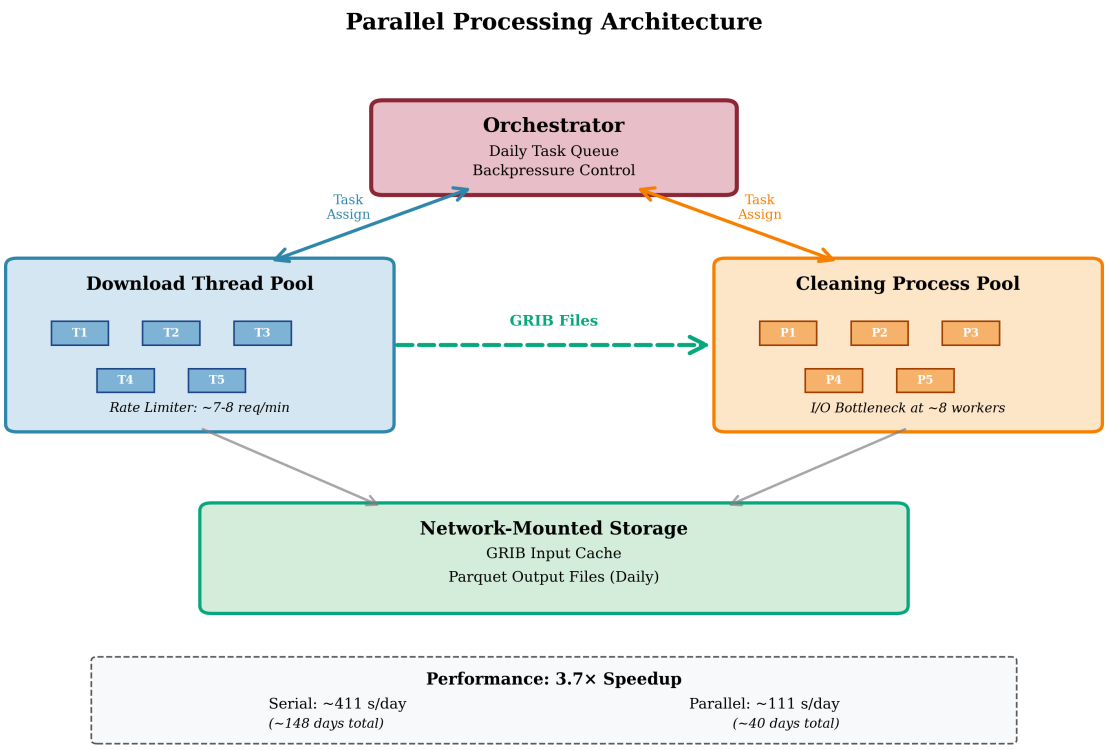

Fig. 3. Parallel processing architecture. An orchestrator manages a queue of daily tasks and coordinates a download thread pool and a cleaning process pool. GRIB files flow from the download pool to the cleaning pool and into network-mounted storage, achieving a measured 3.7× speedup over a minimally parallel baseline.

4) Failure Handling and Restartability: For downloading, every request incorporates built-in retries with exponential backoff. Failed downloads are logged and returned as `None`, but never crash the pipeline. For cleaning, failures are similarly isolated—if a worker fails on a particular day, that failure is recorded while the orchestrator continues processing remaining tasks. The most critical feature is idempotency: before performing any work on a day, the orchestrator checks whether the Parquet file already exists. This enables true restartability—if the pipeline crashes mid-run, rerunning it resumes exactly where it left off.

IV. Results and Performance Evaluation

This section evaluates parallelization layer performance. All measurements were conducted on a high-memory workstation equipped with two Intel Xeon Platinum 8562Y+ processors (64 physical cores) and 1 TB of RAM. Both GRIB inputs and Parquet outputs were stored on network-mounted remote storage. The pipeline was configured with 7 download threads, 8 cleaning processes, and a limit of 10 in-flight downloads. A 30-day slice of ERA5 was used for benchmarking to measure per-day throughput.

A. 30-Day Benchmark: Serial vs. Parallel Performance

Table I summarizes the performance comparison between serial and parallel configurations. The parallel configuration achieved approximately 3.7× speedup over the serial baseline.

TABLE I
Performance Comparison: Serial vs. Parallel Processing

Configuration	Total Time	Time/Day	Speedup
Serial (1/1)	3.43 hrs	411.6 s	1.0×
Parallel (7/8)	0.92 hrs	110.6 s	3.7×
Full Archive	**31,046 days**		
Serial projection	148 days	411.6 s	–
Parallel projection	40 days	110.6 s	3.7×

B. Projected Full-Archive Runtime (1940–2024)

The ERA5 single-level archive from 1940–2024 contains 31,046 days of data. Using measured per-day throughputs, the serial configuration would require approximately 148 days of continuous runtime (nearly five months). Under the parallel configuration, the projected runtime falls to approximately 40 days (five to six weeks). This represents a substantial improvement for a single workstation, highlighting the benefit of treating each day as an independent unit of work and overlapping downloading with cleaning.

C. Scaling Limits and Bottlenecks

Although the parallel configuration provides clear speedup, it does not scale linearly with additional workers. The cleaning stage is dominated by I/O, not computation. After several workers become active, the remote disk becomes the bot-

tleneck, with disk bandwidth saturating long before CPU utilization peaks.

Increasing cleaning workers beyond eight also introduced instability in the Windows environment. At higher parallelism levels, some workers intermittently failed to load ecCodes definition files required to decode GRIB messages. These issues arose from how Windows handles multiprocessing, library paths, and shared resources under heavy load. Reducing workers eliminated these issues.

Downloading also has inherent hard limits. The CDS API rate-limits requests, and exceeding approximately 7–10 daily requests per minute results in 429 errors. Due to CDS limits and I/O bottlenecks in the remote filesystem, adding more workers beyond the 7-download/8-clean configuration did not produce reliable behavior on Windows. Further scaling may be achievable in a Linux environment.

D. Reliability and Restartability

Since processing the full 84 years requires running the pipeline for several weeks, reliability is as important as speed. Download workers automatically retry with exponential back-off, and each GRIB file is checked for the end marker before cleaning. Cleaning failures are isolated to individual days. Idempotency makes long runs practical: before performing any work, the orchestrator checks whether the Parquet file already exists. If so, the day is skipped. This allows the pipeline to stop and restart at any time without losing progress or requiring manual cleanup.

V. Conclusion and Future Work

This work presented a parallel and restart-safe pipeline for downloading and cleaning 84 years of ERA5 single-level data in a reliable and scalable manner. By structuring the workflow around independent daily tasks and allowing download and cleaning stages to overlap naturally, the pipeline eliminates nearly all manual effort typically required to process extended spans of ERA5 data. Researchers no longer need to handle GRIB internals, forecast-step logic, or CDS rate limits. The results are both faster and more practical: a full historical rebuild that would have required approximately five months under a minimally parallel serial configuration can now be completed in approximately five to six weeks while remaining stable during extended runs and capable of restarting safely at any point.

Looking ahead, migrating the pipeline to a Linux environment should enable scaling the cleaning worker pool beyond Windows limitations. The Windows multiprocessing model degrades at higher parallel levels, particularly when multiple workers attempt to load cfgrib and ecCodes simultaneously. Another major direction involves replacing the Parquet output layer with a database-backed storage system [13], [14]. The raw 84-year ERA5 archive is on the order of 100–150 TB before indexing, making pure file-based storage challenging to manage long-term. A PostgreSQL or TimescaleDB database would enable storage in compressed chunks, maintain full queryability, and eliminate the need to reload files during analysis. TimescaleDB's hypertable structure and built-in compression make it a scalable home for this dataset, especially as expansion into additional variables or finer temporal windows occurs. Building this database layer also opens a path for training AI models directly on the historical record [2], [8], as the data would be organized, indexed, and ready for large-scale sampling for retrieval-augmented generation (RAG) systems [5], [6]. Such RAG-enabled systems would allow large language models to dynamically query historical weather patterns, grounding responses in authoritative data for improved grid resilience analysis and decision support [7].

Acknowledgment

This work is part of the "Electric Grid Resilience" Project (60NANB24D210) funded by the National Institute of Standards and Technology of the U.S. Department of Commerce. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] U.S. Department of Energy, "AI for energy: Opportunities for a modern grid and clean energy economy," U.S. Department of Energy, Tech. Rep., April 2024. [Online]. Available: https://www.energy.gov/sites/default/files/2024-04/AI_EO_Report_Section_5.2g(i)_043024.pdf

[2] I. Rojek, D. Mikołajewski, M. Macko, and P. Kotlarz, "Impact of artificial intelligence on the planning and operation of distributed energy systems in smart grids," *Energies*, vol. 17, no. 17, p. 4501, 2024.

[3] Copernicus Climate Change Service, "ERA5 single levels," https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels, 2024, accessed: 2025-01-01.

[4] H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz Sabater *et al.*, "The ERA5 global reanalysis," *Quarterly Journal of the Royal Meteorological Society*, vol. 146, no. 730, pp. 1999–2049, 2020.

[5] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023. [Online]. Available: https://arxiv.org/abs/2312.10997

[6] W. Zhang, Y. Liu, J. Wang, and M. Chen, "TimeRAG: BOOSTING LLM time series forecasting via retrieval-augmented generation," *arXiv preprint arXiv:2412.16643*, 2024. [Online]. Available: https://arxiv.org/html/2412.16643v1

[7] A. Kumar, P. Singh, and M. Zhao, "weatherarchive-bench: benchmarking retrieval-augmented generation on historical weather archives," in *arXiv preprint arXiv:2510.05336*, 2024. [Online]. Available: https://arxiv.org/pdf/2510.05336

[8] A. Kumar, P. Singh, and R. Sharma, "Machine learning applications in energy systems: current trends, challenges, and research directions," *Energy Informatics*, vol. 8, 2025.

[9] NOAA, "10-year strategy for data assimilation," National Oceanic and Atmospheric Administration, Tech. Rep., 2020. [Online]. Available: https://epic.noaa.gov/10-year-strategy-for-data-assimilation/

[10] S. Kang, "How natural hazards impact the power grid & important weather variables," Texas A&M University, Internal Report, 2024.

[11] A. Craig, J. King, D. J. Gagne II, and A. J. Rigden, "Evaluation and bias correction of the ERA5 reanalysis over the United States for wind and solar energy applications," *Energies*, vol. 17, no. 7, p. 1667, 2024.

[12] J. Olauson, "ERA5: The new champion of wind power modelling?" *Renewable Energy*, vol. 126, pp. 322–331, 2018.

[13] Timescale Inc., "TimescaleDB: A time-series database for high-performance real-time analytics packaged as a postgres extension," https://github.com/timescale/timescaledb, 2024, accessed: 2025-01-01.

[14] Y. Xie and W. Chen, "Managing time-series data: Why TimescaleDB beats PostgreSQL," *Mad Devs Blog*, 2024. [Online]. Available: https://maddevs.io/writeups/time-series-data-management-with-timescaledb/