

# From taming energy market data to hyperparameter hunting at scale: leveraging Apache Beam & BigQuery

# The company / team



Simone Biondi,  
co-founder



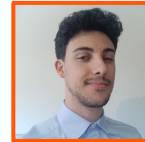
Nicholas Bonfanti,  
co-founder



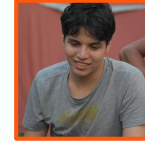
Manuele Aufiero,  
co-founder



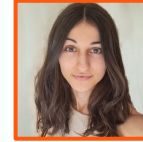
Matteo Scandolo,  
co-founder



Matteo Pacciani,  
Data Engineer



Dario Passarello,  
Software Engineer



Alessia Gualtieri,  
Software Engineer



Pietro Arsi,  
Software Engineer

Find us at [helioswitch.cloud](https://helioswitch.cloud)



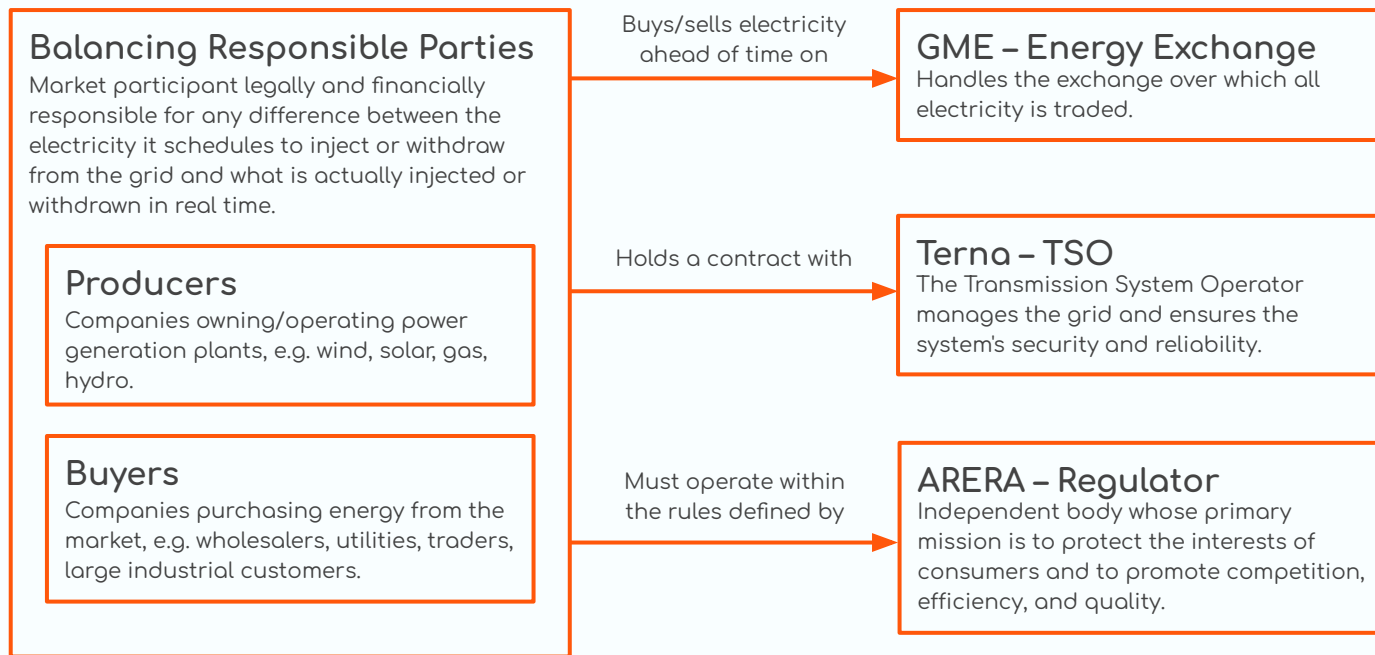
# Agenda



- The needs
  - How the (Italian) electricity market works
  - Why are day-ahead forecasts needed
  - Why is DWH scalability needed
- The solutions
  - From XML over WebDAV to BigQuery using Apache Beam
  - Apache Beam for distributed forecasting
  - The edge of distributed Bayesian optimization
- The questions
  - Next development step
  - Any questions?

# How the (Italian) electricity market works

## Italian electricity market: the very basics



# How the (Italian) electricity market works

## Italian electricity market: subdivision by trading timeframe

### MTE – Forward

**Timeframe:** Long-term, from months up to a year before the physical delivery of electricity.

**Primary Goal:** Hedging against future price volatility.

**Market Nature:** continuous trading, mostly OTC, registered over GME platform.

**Pricing Mechanism:** Pay-as-Bid in a continuous market.

### MGP – Day-Ahead

**Timeframe:** The day before the physical delivery of electricity.

**Primary Goal:** Main wholesale market for electricity. Most of the electricity volume is traded here.

**Market Nature:** GME-managed, exchange-traded and auction-based.

**Pricing Mechanism:** Marginal pricing for each hour by intersecting aggregate supply and demand for each market zone.

### XBID/MI – Intraday

**Timeframe:** After MGP, up to one hour before delivery.

**Primary Goal:** To allow participants to adjust their positions.

**Market Nature:** GME-managed continuous trading (XBID) and intraday auctions (MI).

**Pricing Mechanism:** Pay-as-Bid in the continuous XBID market and marginal price in the MI auctions.

### MSD/MB – Balancing

**Timeframe:** Day ahead to real-time.

**Primary Goal:** Allowing the TSO (Terna) to procure the resources it needs to manage the grid securely (MSD), and activate them to balance load and generation second-by-second.

**Market Nature:** Auctions with the TSO as sole buyer and qualified providers as sellers.

**Pricing Mechanism:** Pay-as-Bid, with offers accepted on economic merit and technical need.

# Why are day-ahead forecasts needed

## Scheduling non-dispatchable generation and demand

How can BRPs  
schedule how much  
to buy/sell the day  
ahead?

### Dispatchable generation and demand

Natural gas, reservoir hydro, nuclear, coal, and biomass are dispatchable: their generation can be programmed within the constraints of the underlying technology.

Programmable demand generally falls under energy storage plants or load curtailment of large consumers.

### Non-dispatchable generation and demand

Solar, wind, and hydro run-of-river are non-dispatchable: their generation is strictly related to the weather conditions, and can be only forecasted and not programmed.

Almost all demand is non-dispatchable, meaning that the end user will not schedule their actual energy consumption for the next day.

Without the option of actively planning, these must be forecasted.

While renewable generation depends almost entirely on weather forecasts, demand depend both on weather and on human activities, e.g. weekday or holidays.

Using the available historical data for generation, demand, and weather, it is possible to build ML-based forecasting models.

# Why are day-ahead forecasts needed

## Simplified case: only MGP and MSD/MB

How does the BRP  
pays/get payed for  
the unscheduled  
electricity?

### MGP – Day-Ahead

**Timeframe:** The day before the physical delivery of electricity.

**Primary Goal:** Main wholesale market for electricity. Most of the electricity volume is traded here.

**Market Nature:** GME-managed, exchange-traded and auction-based.

**Pricing Mechanism:** Marginal pricing for each hour by intersecting aggregate supply and demand for each market zone.

Energy is bought at MGP price by BRPs the day ahead.

All these offers prices are averaged out, creating an unbalance price.

All the mismatch energy between the scheduled and the real one is automatically sold/bought at this price via the contract with the TSO.

The unbalance price is more volatile than the day-ahead one, and introduces both risks and costs in erroneous scheduling.

### MSD/MB – Balancing

**Timeframe:** Day ahead to real-time.

**Primary Goal:** Allowing the TSO (Terna) to procure the resources it needs to manage the grid securely (MSD), and activate them to balance load and generation second-by-second.

**Market Nature:** Auctions with the TSO as sole buyer and qualified providers as sellers.

**Pricing Mechanism:** Pay-as-Bid, with offers accepted on economic merit and technical need.

# Why is DWH scalability needed

## The available historical data

### 1st gen (1G) meters

Used for larger consumers, they are capable of measuring data in quarter-hourly intervals, however data from these meters is typically collected and transmitted to the market operators only on a monthly basis.

### Load Profiling (PRA)

Most of the smaller consumers (< 55kW) have historically been in this category: due to the meters only registering monthly consumption, the quarter-hourly profiles are inferred on the aggregate level.

### 2nd gen (2G) meters

These meters are the latest evolution and represent a major upgrade. While they also measure quarter-hourly consumption at the POD, their consumption data is validated daily, making it available to the system operator the day after consumption (D+1).

...are being actively replaced by these by the tens of millions...

The rapid increase of POD-by-POD quarter-hourly metering has brought significant stress on legacy infrastructure design to handle mostly aggregated data.

New DWH solutions had to be designed for this paradigm shift.



# Why is DWH scalability needed

## Our company and tech stack

This increase in both the data engineering effort and possibilities has brought HelioSwitch where is it now:

1. handling demand DWH and forecasts for **12 utilities**,
2. amounting at more than **2.2 millions PODs** with years-long quarter-hourly data,
3. for a total of more than **17 TWh** of demand forecasted by our algorithms,
4. i.e. the **~5.5%** of the Italian aggregate demand.

Using Apache Beam + Google BigQuery as the main data engineering and ML heavy-lifters.



# From XML over WebDAV to BigQuery using Apache Beam

## Downloading the data

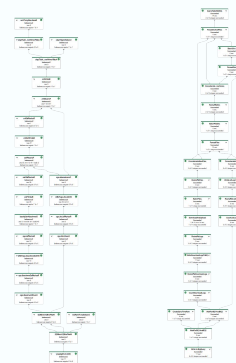
All the POD data is delivered to us with NextCloud, a software using WebDAV as underlying protocol.

It consists often in millions of XML files organized in a file-system like structure.

Dataflow pipelines download these files to GCS and continuously verifies the integrity and completeness of the data transfer.

A screenshot of a Nextcloud file browser interface. It shows a list of files in a directory. The files are XML documents with names like "ReportSummaryReport\_2024122009037.xml", "BiosocialCorrelation", and several "BiosocialCorrelation\_202412\_FPOCS\_2024122009037\_1\_FPOCS\_RoundUp" files. Each file entry shows its name, a size of 5.93, and a status of "0 read file".

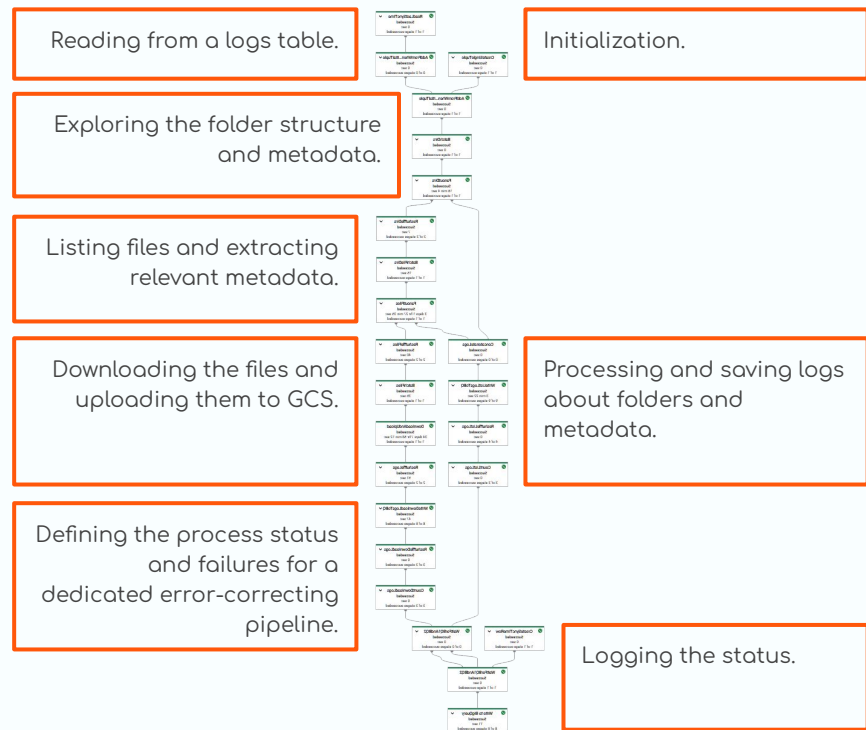
Name	Size	Status
ReportSummaryReport_2024122009037.xml	5.93	0 read file
BiosocialCorrelation	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_1_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_2_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_3_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_4_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_5_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_6_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_7_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_8_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_9_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_10_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_11_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_12_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_13_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_14_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_15_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_16_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_17_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_18_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_19_FPOCS_RoundUp	5.93	0 read file
BiosocialCorrelation_202412_FPOCS_2024122009037_20_FPOCS_RoundUp	5.93	0 read file



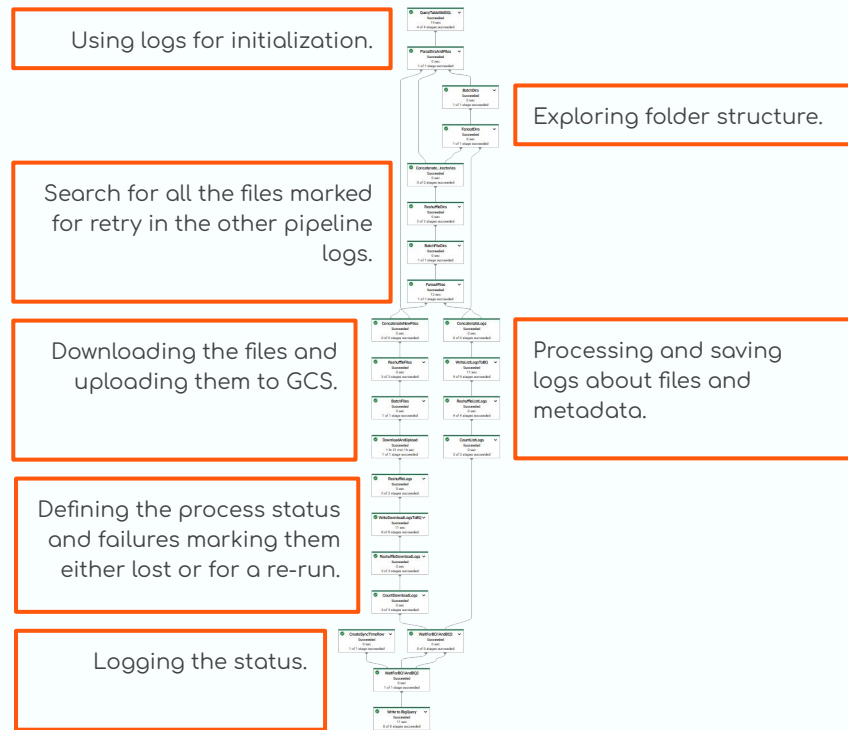
# Downloading the data

From XML over WebDAV to BigQuery using Apache Beam

## Main downloading pipeline



## Error-correction pipeline



## Downloading the data

Apart from the “usual suspect” (e.g., `beam.io.ReadFromBigQuery`), few keys components are used to ensure pipeline performance and robustness:

```
beam.BatchElements(  
    min_batch_size=...,  
    max_batch_size=...,  
    target_batch_overhead=...,  
    target_batch_duration_secs=...,  
)
```

Using WebDAV and downloading files introduces significant I/O latency.

This PTransform allows to dynamically batch elements and adapt over a latency that can easily change.

```
beam.pvalue.TaggedOutput(  
    tag=...,  
    value=...,  
)
```

This pipeline both downloads and check files, read/writes logs, and saves informations about the encountered data structure.

This feature allows to split and handle separately all these different cases.

```
beam.pvalue.AsSingleton(  
    pcoll=...,  
    default_value=...,  
)
```

This allows to treat a single-element PCollection as a DoFn side-argument.

It is essential for efficiently broadcasting a single piece of shared runtime configuration computed from the data.

From XML over WebDAV to BigQuery using Apache Beam

## Parsing the data

The data is read from the GCS bucket used by the downloading pipeline.

A Dataflow pipeline converts the XML data to JSON-like just validating the contents with the least amount of transforming possible (ELT).

All the processed files are moved in dedicated folders, indicating their status, using the **bucket folder paths** as a logging.

[illegible]

Buckets

↳ [data-database-01](#)

↳ EXCEPTIONS

Create folder

Upload

Transfer data

Other services

Filter by name prefix only

Filter: Filter objects and folders

Name	Size	Type	Created
↳ <a href="#">BETTY</a>	–	Folder	–
↳ <a href="#">GENERAL_ERROR</a>	–	Folder	–
↳ <a href="#">VERSION_FORMAT</a>	–	Folder	–
↳ <a href="#">SQL_RECURSION_ERROR</a>	–	Folder	–
↳ <a href="#">SQL_SYNTAX_ERROR</a>	–	Folder	–

Buckets

↳ [data-database-01](#)

↳ WORKING

Create folder

Upload

Transfer data

Other services

Filter by name prefix only

Filter: Filter objects and folders

Name	Size	Type	Created
↳ <a href="#">2024E8_2024-11-30T23:05:33.134Z525</a>	–	Folder	–
↳ <a href="#">2024E9_2024-12-27T11:03:21.378134</a>	–	Folder	–
↳ <a href="#">d4f84c_2024-12-01T10:03:41.738456</a>	–	Folder	–
↳ <a href="#">4a0c38_2024-12-01T11:03:15.400283</a>	–	Folder	–
↳ <a href="#">E83A_2024-04-24T19:10:23.485648</a>	–	Folder	–
↳ <a href="#">3013_2024-12-01T07:20:39.458648</a>	–	Folder	–
↳ <a href="#">40c8b_2024-04-24T20:23:13.812449</a>	–	Folder	–
↳ <a href="#">502af7_2025-05-08T01:03:55.878944</a>	–	Folder	–

# Parsing the data

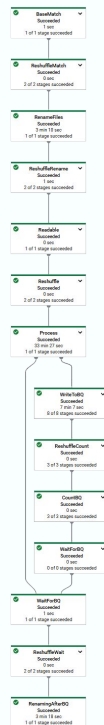
## From XML over WebDAV to BigQuery using Apache Beam

### GCS to BigQuery parsing pipeline

Initialization on matching all relevant files from the appropriate GCS folders.

Opening and parsing the files according to the expected table schema.

Moving the files according to parsing results and status.



Writing all the parsed JSON-like data to BigQuery.

## Parsing the data

The most important component of the pipeline is the **XML parser**.

It takes both the `xml.etree.ElementTree` and a `apache_beam.io.gcp(...).TableSchema`, and recursively interpret the XML data coherently with the given table schema.

Following a **ELT approach** (instead of ETL) this first step avoids any data transformation, resulting in no information loss or change after the parsing.

The **main downside** is that the **messy structure** of the original data propagates to the unprocessed table, resulting ~1800 columns, often nested, repeated, and with redundant information.

In order to optimize the amount of data read by subsequent services, a dedicated “processed” integer-typed row is added and used as **BigQuery partitioning**. The binary representation of its integer values represents which services have already parsed the row (e.g., 9 → 001001 → services n. 1 and 4).

# Apache Beam for distributed forecasting

## Pre-processing the data for the forecasting in BigQuery

In its most basic form the processed data is structured as follows:

Timestamp of the reference 15' interval

Unique POD ID

Location of the POD

Properties of the POD

Total demand in the 15' interval

Weather in the 15' at the POD location

Calendar data (e.g., holiday)

The data can be aggregated by location and/or properties of the POD before forecasting.

This is not necessary, since the models could operate on a POD-by-POD basis.

In this example we will pretend to forecast at a city and voltage level:

Timestamp of the reference 15' interval

—

City of the aggregation

Voltage of the aggregation

Total aggregate demand in the 15' interval

Weather in the 15' in the city

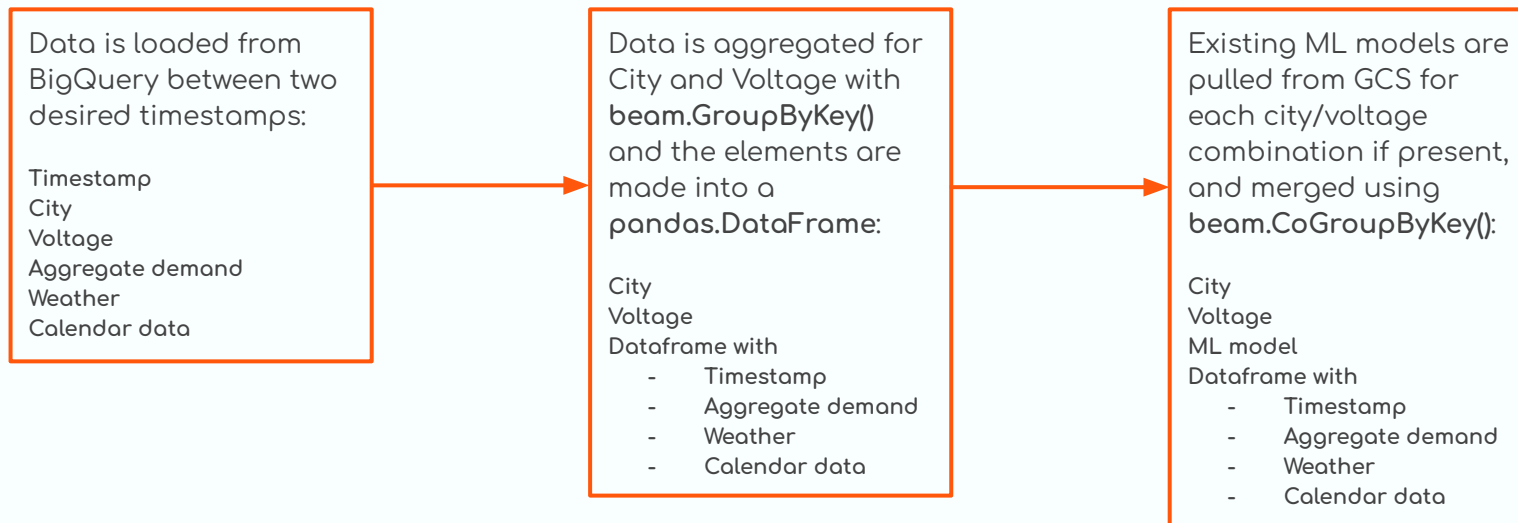
Calendar data (e.g., holiday)



# Apache Beam for distributed forecasting

## Pre-processing the data for the forecasting in Beam

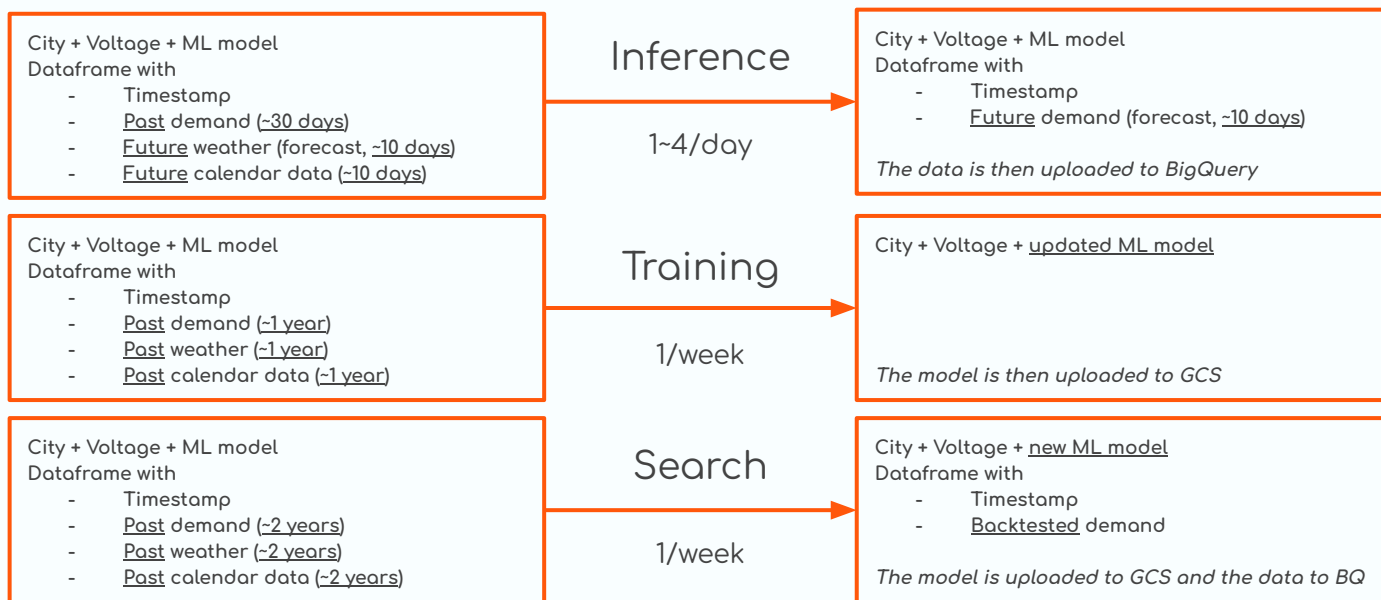
Apache Beam allows to **distribute** the tuning/training/inference of multiple “small” models, in this example one for each city-voltage combination.



# Apache Beam for distributed forecasting

## Overview of the forecasting process in Beam

Apache Beam allows to **distribute** the tuning/training/searching of multiple “small” models, in this example one for each city-voltage combination.



# Inference in Beam

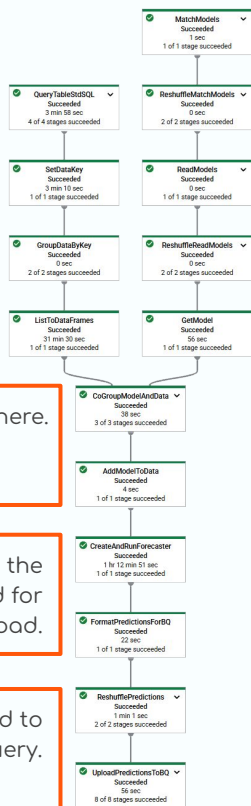
## Apache Beam for distributed forecasting

This BigQuery call extracts both the past and future entries for our inference, and groups it into a unique Dataframe for each "key" (city+voltage).

Data and model are merged here.

The models are executed and the results formatted as needed for the BigQuery upload.

The data is uploaded to BigQuery.

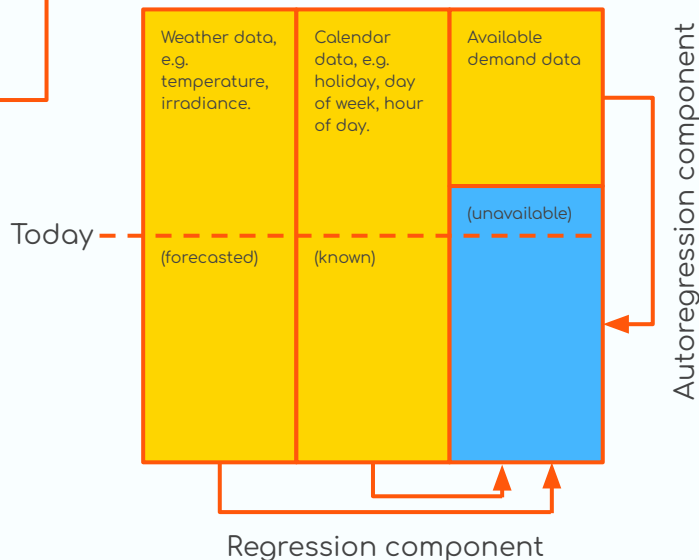


This branch loads the models from GCS, serialized with Joblib.

The models have their "keys" (in our example city+voltage) saved in the serialized object.

*The code generating this pipeline is the same of the training and search ones, but results in different DAGs based on given arguments thanks to DataFlow Flex Templates.*

This process is **distributed** over for each model used. In our example, it is repeated for each city-voltage combination.



# Training in Beam

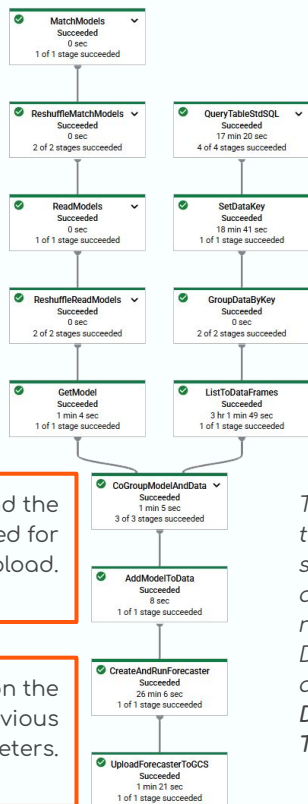
## Apache Beam for distributed forecasting

This branch loads the models from GCS, serialized with Joblib.

Whenever an existing city+voltage does not have a corresponding model, it is instantiated with default hyperparameters and configs.

The models are executed and the results formatted as needed for the BigQuery upload.

The models are trained on the new data, using the previous model hyperparameters.



This BigQuery call extracts historic entries for the training, and groups it into a unique Dataframe for each "key" (city+voltage).

*The code generating this pipeline is the same of the forecast and search ones, but results in different DAGs based on given arguments thanks to DataFlow Flex Templates.*

Once a week the models are training over at least one year of data, in order to capture the demand dynamics over all seasons.

Depending on the necessity, the training can be initialized with the existing weights, or from scratch, keeping only the hyperparameters and configuration from the downloaded one.

The models are autoregressive and recurrent, and always need a window of past demand data.

The coupling of recurrency and weather dependency allows us to model complex dynamics such as thermal inertia of buildings.

# Searching in Beam

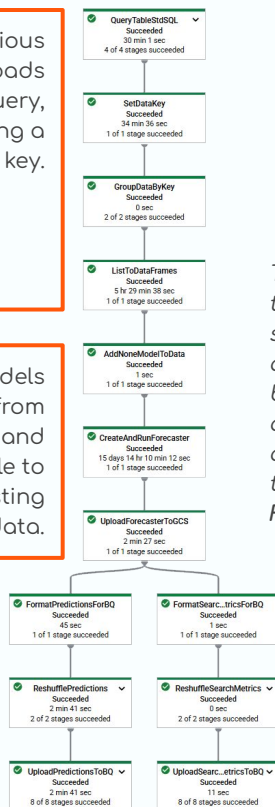
## Apache Beam for distributed forecasting

As for the previous pipelines, this one loads the data from BigQuery, and groups it, creating a Dataframe for each key.

In this example the models are instantiated from scratch, executed, and uploaded. It is possible to start from preexisting model data.

Backtesting prediction are formatted and uploaded to BigQuery.

*The code generating this pipeline is the same of the forecast and training ones, but results in different DAGs based on given arguments thanks to DataFlow Flex Templates.*



Once a month the models undergo a hyperparameter optimization process.

This Dataflow pipeline groups ~2 years of data, creates multiple models with **random hyperparameters**, run a training with **cross-validation**, and evaluates the models on the validation set.

The best performing models are identified and used to **update the distribution** from which the hyperparameters are sampled, and re-samples them, iterating the process.

More than “*hyperparameter hunting*”, this presentation title should say “*hyperparameter trawling*”.

Model performance metrics are formatted and uploaded to BigQuery.

# The edge of distributed Bayesian optimization

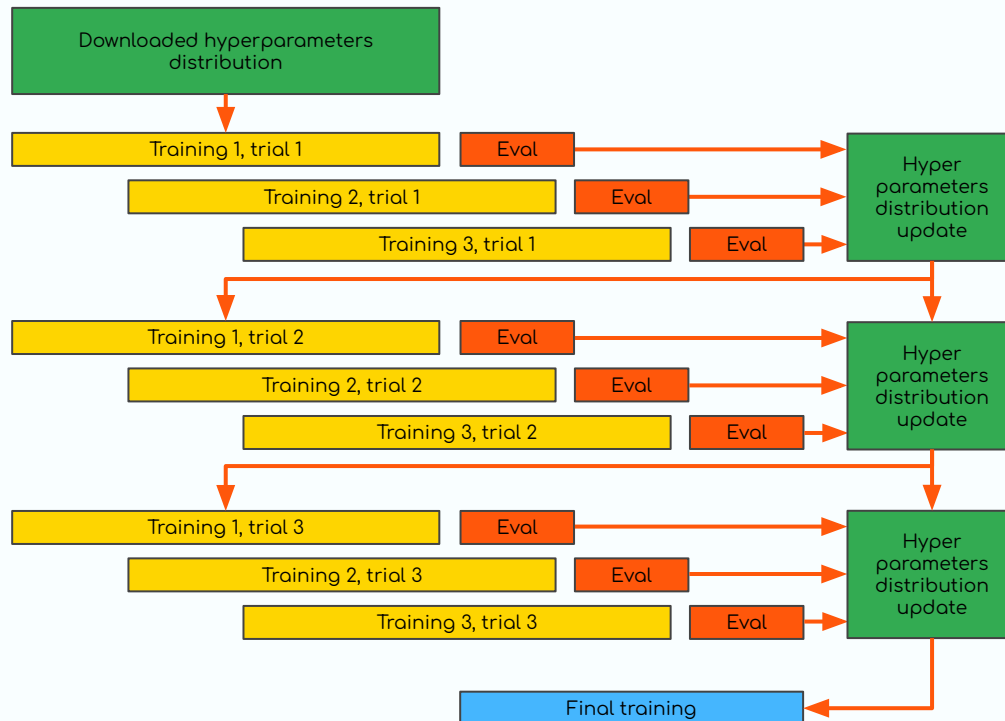
## How model Bayesian optimization works

When there are from **thousands to millions of models**, manually experimenting with hyperparameters quickly becomes unfeasible.

Every city-voltage (in this example) combination has not only a trained model, but a **hyperparameter distribution** stored on GCS.

When simply re-training a model we can either keep the existing hyperparameters, or sample from the distribution.











When searching for the optimal hyperparameters, we **update our distribution** base on the average error on a **validation set**, unseen in the training process.



# The edge of distributed Bayesian optimization

## How to run the bayesian optimization

At this point our process has multiple “scales”:

- iterations within the training   Parallelized
- trainings within the trial   Parallelized
- trials within the optimization   Now serial, potentially distributed with Beam
- optimizations over all the models   Distributed with Beam
- updated models month by month   Serial

We chose to distribute over the second to last: the first two are more suited for standard parallelization, and the complexities of distributing between trials over the same optimization (mainly updating and being updated by the same distribution concurrently) make it a sub-optimal choice with respect to the second to last one, which is intrinsically easier to distribute. The last one is intrinsically serial due to data availability.

### What now?

While this approach ensures HelioSwitch supplies SOTA forecasts for the energy markets, there are multiple ways to improve.

These are some of the ideas:

- Distributing with Beam not only over the models, but **over the trials** of a single model optimization process.
- Combine the information of the hyperparameter distribution among the different models to create a **zero-shot hyperparameter learner**.
- Switch to a **single hyperparameters distribution** among all the models that can be conditioned on metadata, allowing for information-exchanging among the models.



Any questions?

# THANKS! QUESTIONS?

Nicholas Bonfanti  
<https://www.linkedin.com/in/nicholas-bonfanti>

Matteo Pacciani  
<https://www.linkedin.com/in/matteo-pacciani>

HelioSwitch  
<https://www.linkedin.com/company/helioswitch>  
<https://helioswitch.cloud>