Specialisation Project (VT2)
FS2025

Platform for Investment Analysis - Part II

Mixed-Integer Linear Programming Optimization for Integrated Energy Systems and Gradient Boosting Decision Trees Forecasting with Python

Submitted by
Rui Vieira
Institute of Product Development and Production Technologies (IPP)

Supervisor
Dr. Giovanni Beccuti
IEFE Optimisation and Control Systems

Study Program
Business Engineering, MSc in Engineering

July 30, 2025

Imprint

Project: Specialisation Project (VT2)
Title: Platform for Investment Analysis - Part II
Author: Rui Vieira
Date: July 30, 2025
Keywords: Power system planning, mixed-integer linear programming, asset investment optimization, machine learning forecasting, gradient-boosting, photovoltaic prediction, scenario analysis, reproducible modeling, energy systems.

Study program:
Business Engineering, MSc in Engineering
ZHAW School of Engineering

Supervisor:
Dr. Giovanni Beccuti
IEFE Optimisation and Control Systems
Email: giovanni.beccuti@zhaw.ch

Abstract

This master's semester project addresses the dual challenges of long-term infrastructure investment and short-term operational forecasting in power systems facing increasing renewable integration and demand growth. The first module develops a mixed-integer linear programming (MILP) framework for multi-year asset planning, capturing investment timing, asset lifetimes, and operational scheduling in a unified, reproducible Python workflow. This approach explicitly models asset build/retirement, annualized capital costs, and the evolving generation mix—enabling transparent analysis of portfolio evolution, dispatch flexibility, and practical scenarios such as maintenance scheduling or infrastructure sizing.

The second module benchmarks machine learning (ML) techniques, with a focus on gradient boosted trees versus statistical baselines, for day-ahead photovoltaic generation forecasting. Lagged time-series features deliver strong baseline performance, while adding physically motivated, weather-based features (e.g., POA clear-sky index) yields a further 8–12% reduction in mean absolute error (MAE) on average, with particularly clear gains on non-linear or outlier days. However, the incremental value of these features remains sensitive to integration method and requires further targeted development.

By combining these modules, the project demonstrates how investment decisions, asset aging, and operational constraints interact over realistic planning horizons. While the MILP framework robustly identifies cost-effective portfolios and dispatch strategies under growth and asset turnover, its practical value depends on careful scenario definition and real-world calibration. The ML forecasting results underline both the strengths and limitations of current data-driven approaches, highlighting the importance of integrating physically meaningful indicators and operational context.

Overall, the work clarifies the opportunities and boundaries of modular, reproducible modeling for power system planning. It provides a foundation for further advances—such as robust scenario testing, improved feature design, and integrating operational uncertainty into investment analysis—to close the gap between long-term planning and short-term operation.

Keywords: Mixed-integer optimization, power systems planning, asset management, renewable integration, operational forecasting, machine learning, feature engineering, Python modeling, scenario analysis, infrastructure investment

Contents

1 Introduction

1.1 Context and Motivation

Electricity powers everything: homes, factories, and even the internet. Global energy demand is rising, and the shift to renewables like solar and wind is accelerating. But, this transition brings new challenges. Renewables are unpredictable, and increasing digitalization means more volatile and less controllable demand. This makes planning both short- and long- term demand or renewable generation more urgent than ever.

Building on last semester's investment analysis with linear programming (LP) for energy assets, we now introduce a mixed-integer linear programming (MILP) framework that optimizes both long-term investments and operational states in an integrated way. In parallel, we developed a machine learning forecasting module to predict short-term (day-ahead) availability of renewable generation.

1.2 Long-Term Planning with Mixed Integer Optimization

The initial model, developed during the HS24 semester, addressed power-flow problem optimization and investment planning with a linear programming (LP) approach: fixed demand, hand-crafted scenarios, and only continuous variables for decision-making. While this enabled proof-of-concept studies, it had significant limitations and some lack of 'mathematical elegance'. The reliance on manually defined scenarios meant the true best investment strategy might never be evaluated. As a result, the model could not efficiently or systematically explore the full solution space for integrated planning.

The new version (VT2) adopts Mixed-Integer Linear Programming (MILP), where discrete (binary) variables enable us to model on/off states, asset retirements, maintenance cycles, and investment decisions in a unified, realistic manner [1, 11]. MILP co-optimizes capital and operational expenditure (CAPEX and OPEX), preventing suboptimal investment timing or dispatch caused by treating planning and operation separately. A commercial solver (IBM CPLEX) is now used and outperforms the open-source solver (GLPK via PuLP) for large MILP problems. It lacks of parallelization too.

1.3 Short-Term Forecasting with Machine Learning

Short-term forecasting of PV and wind is critical for grid operation. Methods range from simple persistence baselines to advanced ML models. We benchmarked several, ultimately using gradient boosting decision trees for primary forecasting and a statistical model as a baseline [3, 12].

Python's mature ecosystem makes it the dominant platform for optimization and ML in energy systems. We leverage libraries such as scikit-learn, and statsmodels. Goal is to provide a baseline model to understand which factors drive the electricity availability in the short-term.

2 Methodology and Scope

This project addresses investment planning and operational scheduling for a power grid under a mix of asset types and growing demand. We justify below the core modeling choices and set boundaries for both optimization and forecasting, explaining the rationale behind the modeling of our grid setup. Technical formulations and implementation details are deferred to Sections 3 and 4.

2.1 Optimization model – Long-Term Planning

The scope is deliberately focused: all modeling and analysis are performed on a fixed transmission grid with a representative asset mix (generators, renewables, storage). The grid configuration spans from the grid size to the number of buses, lines, and different asset types/locations. Same goes for the loads and generation profiles.

Multi-year power system planning requires both a long-term investment decisions and short-term operational modeling based (which is here, a power flow problem optimzation). Linear Programming can only handle continuous decision variables. We previously used it to solve the power flow problem, modelin it with linear constraints of the form:

$$Ax \leq b, \quad x \in \mathbb{R}^n$$

A LP is unsuitable for modeling decisions involving logical conditions such as "if investment is made, then operation is allowed". Hence no ability to represent discrete (on/off) logic or in our case, unit-commitment decisions.

Mixed-Integer Linear Programming (MILP) extends LP by introducing binary variables, enabling explicit modeling of build/replace decisions and operational constraints. The integrated MILP framework used here allows capital (CAPEX) and operating (OPEX) costs to be co-optimized, simulating when and how assets are built, retired, and optimally dispatched [1, 11].

2.2 Machine Learning model – Short-Term Forecasting

Short-term forecasting is essential for effective system operation under high shares of solar and wind. Approaches range from statistical models (e.g., ARIMA/SARIMA) to machine learning methods such as Gradient Boosted Decision Trees (GBDT/XGBoost). Statistical models provide interpretable baselines and exploit seasonality and autocorrelation, while ML models capture nonlinear patterns in larger, more complex datasets. Hybrid or ensemble approaches can combine strengths of both [3, 12].

For this project, both families are benchmarked: Gradient Boost as the primary forecasting model and SARIMA as a baseline. The forecasts target day-ahead horizons, aligning with demand for markets operations or grid stability to mitigate risks.

2.3 Solver and Toolchain Selection

Energy system MILPs are computationally challenging. Open-source solvers (CBC or GLPK) are widely used in research but can struggle with large models, particularly those involving many binaries and long planning horizons. Commercial solvers (CPLEX, Gurobi) offer better performance, robustness, and advanced features such as parallelization, which can be critical for realistic studies [8]. This project uses CPLEX via its Python API (`docplex`) [2].

Python is the de facto standard language for research and development in optimization and ML due to its broad library ecosystem. Key packages in this project include:

- `docplex` for optimization and CPLEX solver integration [2]
- `scikit-learn`, `XGBoost` for ML/forecasting [3, 12, 9]
- `statsmodels` for SARIMA/statistical modeling [10]

- `Optuna` and `skopt` for hyperparameter tuning [5]
- `pvlib` for solar/weather features calculations [6]
- standard data-handling libraries (`pandas`, `numpy`, `matplotlib`) [7], [4]

2.4 Scope and Boundaries of the Study

The project's scope covers:

- **Strategic Horizon** Multi-year (typically 1–30 years but could be adapted), with three representative weeks per year (winter, summer, spring/autumn) to keep the MILP tractable but seasonally realistic.
- **System Boundaries** A fixed test grid with predefined buses, lines, and asset candidates; retail tariffs and ancillary services are excluded. The grid is fixed during the optimization but configured before.
- **Decisions** Investment timing and size (build binaries), operational scheduling (dispatch, unit commitment), and storage operation.
- **Forecasting** Independent module producing day-ahead (visually up to 2 days and metrics up to 1 week) forecasts for Photovoltaic (PV) availability with accuracy assessed by MAE, RMSE and R^2.
- **Performance indicators** Total system cost (objective function), solver performance (solve time, optimality gap), and forecast error (MAE, RMSE).

Elements outside scope include demand-side management, real-time balancing, grid expansion, and non-economic reliability constraints. All implementation and testing are performed in a reproducible Python workflow using `Poetry` for project and dependency management. Project metadata and dependencies are in the `pyproject.toml` file ; similar to VT1.

In summary, the methodology rests on (i) an integrated MILP model for joint investment and operation planning for a multi-year horizon, (ii) a day-ahead, Python-based ML/statistical forecasting pipeline, and (iii) robust solver and workflow choices. Technical details and results for each major block are presented in subsequent sections.

3 Linear to Mixed-Integer Programming Transition

Building on last semester's investment model work [**?**], we now let the model decide when to build or replace each asset. To simply put it, we add binary variables to capture theassets availability and usage and define a new objective function balancing both investment and operational costs.

The previous linear (LP) formulation fixed the asset list ex-ante; any capacity study meant running dozens of "what-if" scenarios offline. In the MILP, those scenarios collapse into one optimisation that jointly chooses dispatch *and* build schedule for the lowest net present cost. Running this logic over multiple years, generates the possibility to assess when in a horizon, should an asset be built given its lifetime and associated cost.

3.1 Key modelling ideas

i. Two binary variables per asset and year[1]

$$z^{\text{build}}_{a,y} = \begin{cases} 1 & \text{if asset } a \text{ is } \textit{commissioned} \text{ in year } y \\ 0 & \text{otherwise} \end{cases}, \quad z^{\text{on}}_{a,y} = \begin{cases} 1 & \text{if asset } a \text{ is } \textit{operational} \text{ in year } y \\ 0 & \text{otherwise.} \end{cases}$$

Two binaries per asset per year are present in the model, but:

- Only the z^{build} variables are truly "free" (decision variables).
- The z^{on} variables are "dependent" binaries: each year's installed status is determined (by constraint) as the sum of all unexpired builds.

ii. Lifetime-aware coupling
Each asset lives L_a (lifetime) years.

$$z^{\text{on}}_{a,y} = \sum_{\substack{y' \le y \\ y - y' < L_a}} z^{\text{build}}_{a,y'} \qquad \sum_{\substack{y' \le y \\ y - y' < L_a}} z^{\text{build}}_{a,y'} \le 1 \tag{2a, 2b}$$

- (a) says an asset is "on" if it was built in the last L_a years
- (b) forbids more than one build : "at-most-one-build-per-lifetime"
- y' is the year of the last build

iii. Capacity gating
Dispatch variables (Generator : P^{gen}, storage power : (P^{ch}, andP^{dis}) and state of charge : E) are multiplied by the installation flag $i_{a,y}$. If an asset is not built, its capacity is mathematically zero.

$$0 \le X^{\text{asset}}_{a,y} \le X^{\text{max}}_{a} \cdot z^{\text{on}}_{a,y}$$

iv. Annualised CAPEX via Capital Recovery Factor (CRF)
Capital expenditure (CAPEX) is annualized using the Capital Recovery Factor (CRF). It is an asset-specific financial metric used to calculate the annualized cost of an investment over its lifetime. The goal is to make investment and operational costs comparable in a single linear objective. It is defined as:

$$\text{CRF}_a \;=\; \frac{i\,(1+i)^{L_a}}{(1+i)^{L_a}-1}, \qquad A_a \;=\; \text{CRF}_a \cdot C_a. \tag{2}$$

- A_a is the *annual* cost of owning one unit of asset a.
- C_a is the *capital cost* of asset a.
- i is the *interest rate* (depreciation rate between 5–10%)

The annulized cost brings a strong advantage in the context of a multi-year analysis. Instead of charging the full investment (CAPEX) upfront or at the end of an asset's lifetime, the total cost is distributed evenly across each year that the asset is operational within the optimization horizon. This approach has several technical and economic advantages:

[1] Upper-case sets: $\mathcal{A}$ assets, $\mathcal{Y}$ years, $\mathcal{B}$ buses. Lower-case indices: a, y, b etc.

- Each asset pays a fixed annual cost while installed, simplifying the objective.
- Only pay for years the asset is in use, avoiding end-of-horizon "distortions".
- Removes cost jumps at build/retire dates.
- The annuity is linear and easy to implement, unlike full discounted cash-flow tracking.
- Slight approximation error from annualizing if the asset is not used for all its lifetime, but negligible in practice.

This is implemented in code as follows:

```
    def compute_crf(lifetime, discount_rate):
        # Capital Recovery Factor (CRF)
        if lifetime is None or lifetime <= 0:
            return 1.0
        i, n = discount_rate, lifetime
        return (i * (1 + i)**n) / ((1 + i)**n - 1)

    # Annualized cost (annuity)
    annual_asset_cost = npv * compute_crf(lifetime, discount_rate)
```

In the optimization: The annualized cost is added for every year the asset is installed, for both generators and storage units:

```
1   for y in years:
2       total_cost += annual_asset_cost * gen_installed[(g, y)]
```

Listing 1: Objective function cost term

This modeling choice is implemented in `dcopf()` and post-processed in the cost analysis script. It ensures that costs are spread proportionally and that the cost function remains robust, regardless of asset timing or lifetime relative to the analysis horizon.

v. **Load growth factor**

To mimic a time-varying load through the years, the loads are scaled within the nodal-balance by a defined factor γ_y. This induces a growing demand in the model without increasing the generation. This induces a diversification of the generation mix through the years.

$$\tilde{D}_y = \gamma_y \, D_y.$$

It was decided to model the load growth factor as a global factor for all buses, but it could be extended to a bus-specific factor.

3.2 MILP formulation

Only the constraints that changed vs. the LP are listed. For readability, the indices set such as s (season) and t (hour) were are omitted since per season or per hour concept are implied. Other non-relevant indices may be missing too.

A. Investment constraints

- Lifetime link (2a, 2b) (see above).
- Generator, storage power and energy limits as mentionned in the capacity gating bullet:
  - Generator Output Limits:

$$0 \leq P_{a,y}^{\text{gen}} \leq P_a^{\max} \cdot z_{a,y}^{\text{on}} \tag{3}$$

  - Storage Power Limits:

$$\begin{aligned} 0 \leq P_{a,y}^{\text{ch}} \leq P_a^{\max} \cdot z_{a,y}^{\text{on}} \\ 0 \leq P_{a,y}^{\text{dis}} \leq P_a^{\max} \cdot z_{a,y}^{\text{on}} \end{aligned} \tag{4}$$

  - Energy Capacity Limit:

$$-E_a^{\max} \cdot z_{a,y}^{\text{on}} \leq E_{a,y} \leq E_a^{\max} \cdot z_{a,y}^{\text{on}} \tag{5}$$

B. Operational constraints (modified)

- Nodal balance with load scaling
  For each bus b, season s, year y, hour t:

$$\sum_{a \in \mathcal{A}_b} P_{a,y}^{\text{gen}} + \sum_{a \in \mathcal{A}_b} (P_{a,y}^{\text{dis}} - P_{a,y}^{\text{ch}}) + \sum_{\ell \in \text{in}(b)} F_{\ell,y} = \gamma_y \cdot D_b + \sum_{\ell \in \text{out}(b)} F_{\ell,y} \tag{6}$$

  The only change is the growth factor γ_y on the demand term.

C. Objective function

$$\min \quad \underbrace{\sum_{s \in \Sigma} W_s \sum_y \Big(\sum_a c_a \cdot P_a^{\text{gen}} \cdot z_a^{\text{on}} \Big)}_{\substack{\text{operating costs} \\ \text{OPEX}}} + \underbrace{\sum_y \Big(\sum_{a \in \mathcal{A}} A_a \cdot z_a^{\text{on}} \Big)}_{\substack{\text{investment costs} \\ \text{CAPEX}}} \tag{7}$$

With:

- W_s: Number of calendar weeks represented by each season s (e.g., winter = 13)
- Annualised CapEx per asset a : $A_a = \text{CRF}_a \cdot C_a$

Hence the MILP simultaneously finds the least-cost *dispatch* and the cheapest *build / replace* schedule over the planning horizon. It eliminates the need for external spreadsheets computing NPV and manual scenarios comparison.

3.3 Implementation pipeline

Figures 3 and 2 show how CSV inputs and time-series flow through the four main Python modules (`pre.py`, `network.py`, `optimization.py`, `post.py`) and finally into `cost.py` for the cost analysis. The code matches the math 1:1.

3.4 Investment analysis pipeline

A. Overview and roles

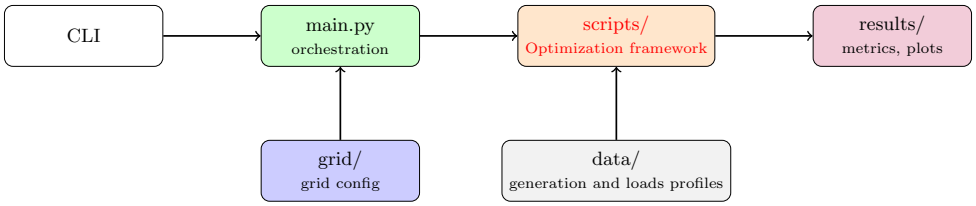

Figure 1: Compact overview of the forecasting pipeline.

- `main.py` – driver script accepting CLI flags, steering the four logical stages: preprocessing, network assembly, optimisation, and post-processing/logging.
- `pre.py` – slices three 168-h representative weeks, matches profiles to assets, and attaches analysis meta-data (years, season weights, load-growth).
- `optimization.py` – builds the "DC-OPF MILP" optimization problem with annualised CAPEX, solves it via CPLEX, then serialises all variables back into the `IntegratedNetwork`.
- `post.py` – turns the raw decision variables into human-readable implementation plans, generation-mix graphics and asset timelines.
- `analysis/costs.py` – converts dispatch into MWh, adds annuitised investment streams, and prints/plots per-asset cost breakdowns.
- `results/` – output directory for logs, results (.json) and figures.

B. Implementation

The figure below shows the deeper call graph inside the `scripts/` directory, highlighting the *three* execution phases:

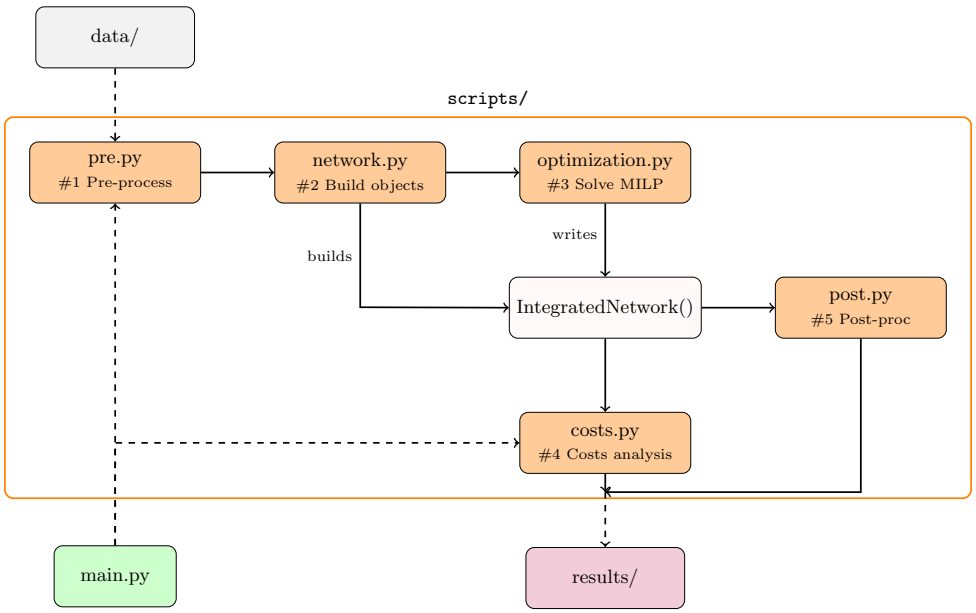

Figure 2: Detailed flow inside `scripts/`: preprocessing → object construction → MILP solution → reporting.

1. **Pre-processing**
   `pre.py` converts raw CSV/time-series into `grid_data` + `seasons_profiles`. A light sanity-check now ensures the `representative_weeks` sum to 52.

2. **Object assembly**
   `network.py` wraps every season in a `Network` (data-only) and records them inside a global `IntegratedNetwork`. Tweaks that proved essential:

   - bus-ID matching (string vs. integer)
   - automatic snapshot creation with length T
   - load-growth factors attached for later scaling

3. **Optimization formulation and solve**
   `optimization.py` hosts two core functions: `dcopf()` builds the model, while `investement_multi()` solves, extracts and stores all variables.

   Key modelling choices:

   - *at-most-one-build-per-lifetime* window → The "installed" variable at year y is the sum of active builds not yet expired:

```
      for g in generators:
            lifetime = int(first_network.generators.at[g, '
                ↪ lifetime_years'])
            for y_idx, y in enumerate(years):
                  window_builds = [gen_build[(g, yb)]
                  for yb_idx, yb in enumerate(years)
                  if (y_idx - yb_idx) < lifetime and y_idx >= yb_idx
                      ↪ ]
                        global_constraints.append(cp.sum(
                            ↪ window_builds) <= 1)

            # Installed status = sum of "active" build binaries in
            for y_idx, y in enumerate(years):
                  window_builds = [gen_build[(g, yb)]
                                   for yb_idx, yb in enumerate(
                                       ↪ years)
                                   if (y_idx - yb_idx) < lifetime
                                       ↪ and y_idx >= yb_idx]
                  global_constraints.append(
                  gen_installed[(g, y)] == cp.sum(window_builds))
```

   - Annualised CAPEX via CRF (`compute_crf`) operational costs weighted by season-weeks
   - No slack variables; nodal balances must close (same as in the LP)
   - Storage SoC forced to zero at each season edge, ⇒ breaking cross-season energy loops (same as in the LP)

4. **Post-processing**
   `post.py` renders a Gantt-like timeline, seasonal generation mixes, and an implementation plan, while `analysis/costs.py` computes MWh and $ flows using the same annuity (CRF/discount) logic as in the objective, ensuring consistency.

C. MILP model structure

Figure 3 sketches the internal control flow inside `dcopf()`.

A major bottleneck in large-scale optimization (e.g., MILP) is constraint assembly: explicit Python for-loops to create constraints individually causes excessive overhead in both `CVXPY` (matrix stuffing) and the solver's pre-processing. By leveraging vectorized expressions and a modern solver interface (CPLEX), we significantly reduced this overhead.

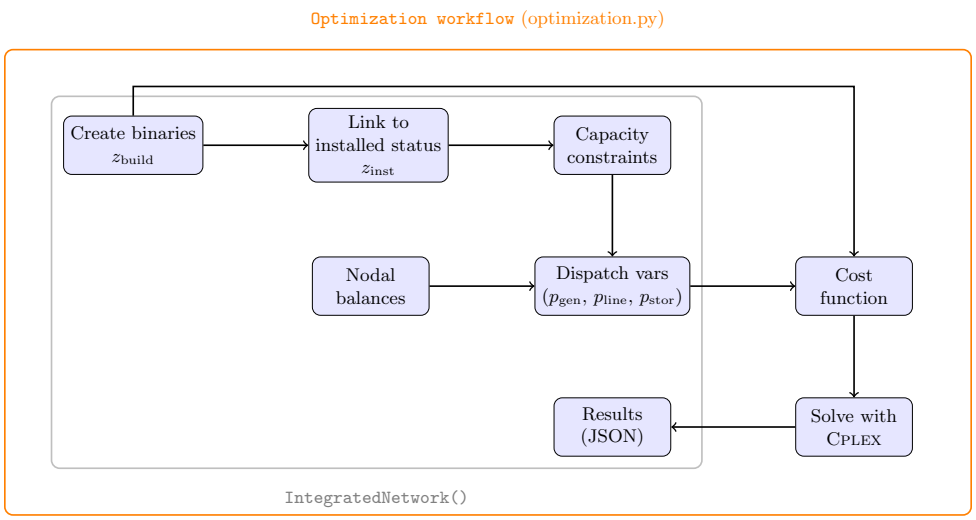

Figure 3: Internal control flow inside `dcopf()`

- **Old (CBC/PuLP):** Constraints are added one-by-one in explicit Python for-loops:

```
            for t in T:
            for i in buses:
            DCOPF += (
            gen_sum - pd_val + flow_in - flow_out == 0
            ), f"Power_Balance_Bus_{i}Time{t}"
```

  This approach is highly inefficient: each constraint is parsed and processed individually in Python, resulting in excessive pre-solve times as problem size grows.

- **New (CVXPY/CPLEX):** Constraints are constructed in bulk using array operations:

```
            for s in seasons:
            for y in years:
            for b in buses:
            flat_constraints.append(
            gen_sum + st_net + flow_in == load_vec + flow_out
            )
```

  Or, for a fully vectorized version, all buses/time steps at once:

```
            flat_constraints.append(
            cp.sum(p_gen_matrix, axis=0) + cp.sum(p_discharge_matrix,
                ↪ axis=0)
            - cp.sum(p_charge_matrix, axis=0) + cp.sum(flow_in_matrix,
                ↪ axis=0)
            == load_matrix + cp.sum(flow_out_matrix, axis=0)
            )
```

  This eliminates per-constraint Python overhead and allows CVXPY to construct and pass large sparse matrices directly to CPLEX.

In the old model, assembly times scale poorly ($\mathcal{O}(n_{\text{vars}} \cdot n_{\text{time}})$ constraints created in Python; pre-processing could take minutes for moderate n). In the new model, vectorized construction reduces

overhead by 50× to 100×; model build is near- instantaneous even for 10^5+ constraints. Main constraints (capacities, flows, balances) are fully vectorized.

For a multi-year, seasonal, multi-asset problem, model assembly dropped from ˜5 minutes (old) to <10 seconds (new), with no loss of model fidelity. The main bottleneck is now the mathematical difficulty itself (increased number of constraints and variables).

4 Forecasting

In this project, the forecasting pipeline is structured not as a monolithic Python package, but as a set of iterative improvements and notebooks organized in a `notebooks/` directory:

```
notebooks/
  |-- setA.ipynb    # Time features only
  |-- setB.ipynb    # Lag + cyclic features
  |-- setC.ipynb    # Bayesian-optimized feature set
  |-- setD.ipynb    # Recursive prediction (failed)
  |-- setE.ipynb    # POA clear-sky and weather features
  `-- setF.ipynb    # XGBoost library
```

Each notebook implements and tests its own set of features or single "forecasting strategy," incrementally building up from trivial baselines (like time-only) to advanced feature sets including weather derived ones.

We want to forecast the electricity generation for the day ahead but will forecast on 7 days. We can already see the cyclic behavior and the ramping behavior of the generation.

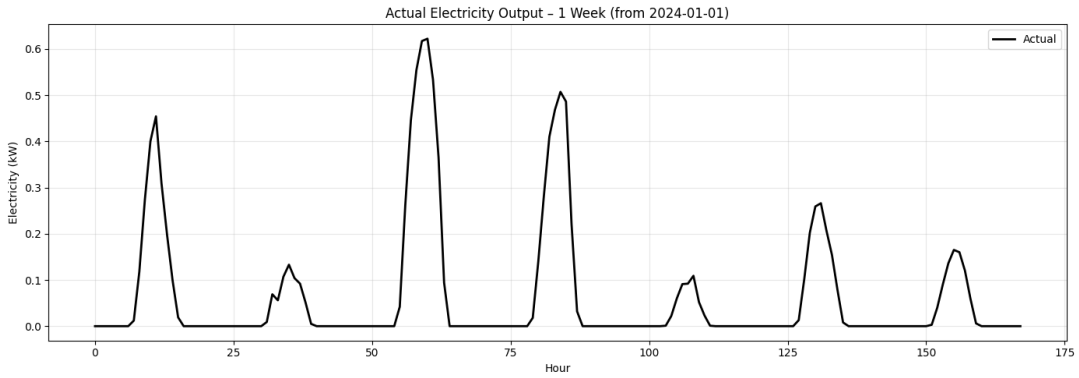

Figure 4: PV Generation normalized data (1 week)

4.1 Operational Forecasting Principle

Forecasting in the operational sense means predicting the next h hours (day-ahead or week-ahead) using only historical and current data available up to time t. Unlike "offline" statistical analyses, operational forecasts must avoid all data leakage: only lagged, cyclical, and weather-driven features that would be available before the forecast time can be used.

The predictive model is retrained on the historical period. Then, it is rolled forward to generate and evaluate true out-of-sample predictions, mimicking a real operational workflow.

4.2 Prototyping & Machine Learning Models

To benchmark our operational forecasting framework, we used MAE as main metric but also RMSE and R2. They can be interpreted as follow :

- MAE: Mean Absolute Error, interpreted in the same units as the target variable. It is the average absolute difference between the predicted and actual values.
- RMSE: Root Mean Square Error, penalizes big mistakes. It is the square root of the average of the squares of the errors.

- R2: R-squared, measures how well the model explains the variance in the target variable. It assesses the overall fit.

We required all machine learning models to outperform a SARIMA baseline. Although we began with neural networks, we ultimately switched to gradient boosted trees for their interpretability and practical strengths with engineered time features. Detailed below, both modeling approaches and the feature design supporting them.

Neural Network Models

We iterated on the following models:

1. **Simple Neural Network (NN)**
   A neural network is a computational model inspired by the human brain. Our simple model consisted of a single layer of interconnected nodes (neurons) that process data by applying weights, biases, and activation functions. Predictions are made from "patterns" learned during training.
   Bad: fails to capture strong seasonality → high bias.

2. **Multilayer Perceptron (MLP)**
   MLP is a specific type of neural network: a fully connected feedforward neural network with one or more hidden layers and nonlinear activation functions. We searched (tuned) for the optimal number of layers and units within an array of 1 to 4 layers and 1 to 128 units.

   Good: smooth forecasts; handles non-linearities.
   Bad: night-time over-fit (not 0) despite target scaling tricks.

3. **Temporal Convolutional Network (TCN)**
   TCN is a type of neural network designed for sequential data. It uses 1D causal convolutions to capture temporal dependencies and are particularly effective for handling long-term dependencies.

   Good: tracks ramps well; long memory.
   Bad: 3–5× GPU time, sensitive to feature scaling, and still overshoots zero generation at night.

We explicitly set $y_t = 0$ between dusk and dawn (computed from the solar-zenith angle in the weather feed) to remove the night signal forecasting errors. Although this removed the worst negative bias, the best TCN still landed at MAE= 4.2% and required >90 min of hyper-parameter tuning. This was not a good trade off.

Pivot to Ensemble Trees

Gradient Boosted Decision Trees (GBDT) is an ensemble machine learning method. Meaning that in the end, all the trees are combined to make one powerful model where each new tree tries to reduce the errors (called Loss) made by the previous ones. This error is minimized via a technique called gradient descent who looks at how the error changes and then adjsuts the next tree based on it.

GBDTs differ fundamentally from neural networks. They are non-parametric and rule-based, meaning they do not require extensive scaling or normalization of inputs and are typically much faster and lighter to train. Their main tunable hyperparameters include the number of trees, the maximum depth per tree, the learning rate, and regularization settings.

For model development, we split our data into three sets: training, validation, and test. The training set is used to fit the model, the validation set guides hyperparameter tuning and prevents overfitting, and the final evaluation is performed on the test set (our 7days of forecast – 1st day being the main target).

Tree ensembles have distinct strengths: they tend to be more interpretable, naturally handle tabular (structured) data, and are robust to irrelevant features or collinearity. However, unlike neural networks that can automatically model sequential dependencies, GBDTs require us to explicitly engineer time-based features (such as lags and calendar variables) to capture temporal effects.

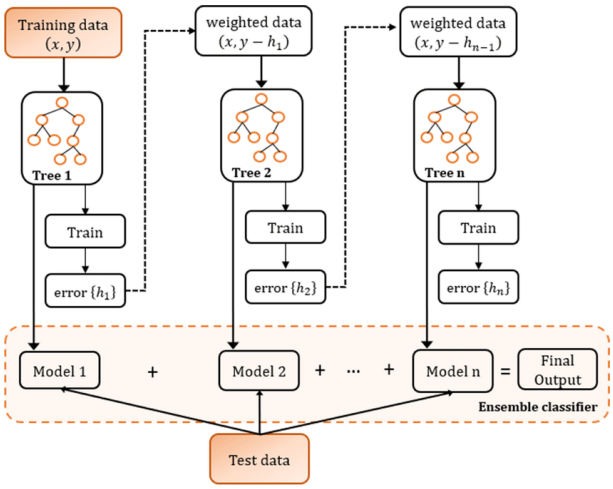

Figure 5: Gradient boosting decision tree (GBDT) illustration. Source: [**?**]

In summary, the switch to GBDT was motivated by both practical considerations and empirical results. Tree ensembles offered consistently higher accuracy, much faster training, and easier interpretability than neural networks in our forecasting task. This made GBDT the clear choice for our prototyping and final model selection.

4.3 Feature Engineering

Feature engineering is a critical step in time-series forecasting models and gradient boosted trees in particular, as it enables the extraction of temporal patterns, cyclic behavior, and relevant dependencies from the raw data. In this project, feature engineering focused on three main strategies: time features (cyclical encoding), lagged features, and "engineered" weather/irradiance features.

A. Cyclical Time Features

Hourly, daily, and seasonal periodicity in electricity generation and demand is well-known. To encode these periodic effects, we transformed the raw time variables into their cyclical (sin/cosine) representations:

$$\text{hour\_sin} = \sin\left(2\pi \cdot \frac{\text{hour}}{24}\right) \quad , \quad \text{hour\_cos} = \cos\left(2\pi \cdot \frac{\text{hour}}{24}\right)$$

This encoding allows the model to learn periodicity without artificial jumps between boundary values (e.g., hour 23 to hour 0). The 'hour' feature can be replaced by the month, day of week and day of year in our equation.

B. Lag Features

To capture autocorrelation and persistence effects in the target time series, we included lagged values of the target variable (electricity generation) as additional features. For each time t, we added the previous values at selected time intervals:

$$\text{electricity\_lag}_\ell[t] = y_{t-\ell}$$

where y_t denotes the target value at time t, and ℓ is the lag in hours.

In the final feature set, we included lags of $\ell \in \{1, 2, 3, 6, 12, 24, 48, 168\}$, corresponding to 1 hour up to 1 week (168 hours), i.e.:

$$\mathcal{L} = \{1, 2, 3, 6, 12, 24, 48, 168\}$$

This means the model, at each prediction time, has access to the target value for up to the past 7 days.

C. Weather and Irradiance Features

For the advanced models (e.g., Set E and Set F in our code), we engineered domain-specific features using both direct measurements and physical models:

- Plane of Array (POA) Irradiance: calculated with the `pvlib` library using local site, solar position parameters and the weather data from the `Renewables.ninja` API. The physical formula is :

$$G_{\text{POA}} = G_b \cdot R_b + G_d \cdot F_d + G_h \cdot \rho_g \cdot \frac{1 - \cos(\beta)}{2} \tag{8}$$

  where:

  - G_b = direct normal irradiance (DNI)
  - R_b = geometric factor for beam component (depends on solar and panel angles)
  - G_d = diffuse horizontal irradiance (DHI)
  - F_d = view factor for diffuse component (depends on tilt)
  - G_h = global horizontal irradiance (GHI)
  - ρ_g = ground reflectance (albedo)
  - β = tilt angle of the panel

- Clear-sky Index: computed as the ratio between measured POA and modeled clear-sky POA :

$$\text{POA clear-sky index}[t] = \frac{\text{poa\_total}[t]}{\text{poa\_clearsky}[t]} \tag{9}$$

For both the target and key weather features (especially POA clear-sky index), we also included recent lags (e.g., 1h, 2h, 3h, 6h, 12h, 24h) and 24h rolling means.

$$\text{poa\_clearsky\_index\_lag}_k[t] = \text{poa\_clearsky\_index}[t - k]$$

D. Maximum Lag and Buffering

A critical implementation detail is the handling of missing data at the start of each split due to lagging. For any feature with maximum lag ℓ_{max}, at least ℓ_{max} time steps are dropped at the beginning of each set (e.g. :

- $\ell_{max} = 168$ for target lags in most models : "7 days of data are needed before the first prediction"

During forecasting, a lag buffer window is reserved so that the first prediction is always made with valid historical context.

E. Feature Set Size and Selection

Adding engineered features significantly increases the dimensionality of the input data matrix. While richer features can boost model accuracy, including too many can introduce noise or lead to overfitting. To ensure our models focused on the most relevant information, we performed feature selection prior to training.

Several approaches exist for feature selection, such as forward selection (adding features one at a time) or backward elimination (removing the less important features step by step). In our workflow, we used a more efficient method: Bayesian optimization. [**?**]. It searches for the subset of features (and hyperparameters) that minimize validation error, reducing computational burden and maximizing forecast skill.

4.4 Mathematical Background

A. Statistical Model - Baseline

We used SARIMA a statistical model as a benchmark. It applies ARMA modeling on a transformed (differenced) version of the time series to capture both short-term dynamics and repeating seasonal patterns. Its power lies in modeling both the temporal structure and seasonal cycles within a single, compact framework.

ARMA (p, q) is a linear combination of two things and works only on stationary data. It means the time series must have constant mean and variance over time. It is written as:

- an *autoregressive (AR)* part of order $p \rightarrow$ how past values influence the present
- a *moving average (MA)* part of order $q \rightarrow$ how past errors influence the present

The ARMA model is written as:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

- c: constant term (mean of the process, if not differenced)
- ϕ_i: AR coefficients
- ε_t: white noise error term at time t
- θ_j: MA coefficients

SARIMA $(p, d, q) \times (P, D, Q)_s$ extends ARMA to handle trends and seasonality, in two ways:

1. **Differencing** for trend removal
   To remove non-stationary trends, SARIMA applies ordinary differencing d times:
   $$y'_t = (1 - L)^d y_t = y_t - y_{t-1} \quad (\text{if } d = 1)$$
2. **Seasonal differencing** for repeating patterns
   To remove seasonal effects (e.g., daily or yearly patterns), it applies seasonal differencing D times with period s:
   $$y''_t = (1 - L^s)^D y'_t = y'_t - y'_{t-s} \quad (\text{if } D = 1)$$

The six hyperparameters (p, d, q) and (P, D, Q) define the orders of autoregression, differencing, and moving-average smoothing at both the regular (hourly) and seasonal (daily) levels, with $s = 24$ reflecting the daily cycle. If the model residuals resemble white noise—i.e., they are uncorrelated and pattern-free—the model is considered to have successfully extracted all systematic, predictable structure.

B. Machine Learning Model - GBDT

Assume we aim to predict a target y from input features $x \in \mathbb{R}^n$. GBDT minimizes a loss function $L(y, \hat{y})$. (Minimizing mean absolute error via gradient descent between a predicted $\hat{y}$ and true y value).

Let:

$$F_0(x) = \text{initial guess} \qquad \text{for } m = 1 \text{ to } M$$

Boosting builds many shallow trees *sequentially*. Each tree tries to predict the *errors* the previous trees made:

$$r_i^{(m)} = y_i - F_{m-1}(x_i)$$

At each boosting step m, we compute the residual $r_i^{(m)}$, which is the difference between the true value y_i and the current model's prediction $F_{m-1}(x_i)$. This residual guides the new tree h_m to correct errors made so far.

The new model F_m is updated by adding a fraction $\nu \in (0, 1]$ (learning rate, controlling the step size) of the new tree $h_m(x)$ to the previous model's output. This sequential additive approach allows gradual improvement while avoiding overfitting.

$$F_m(x) = F_{m-1}(x) + \nu\, h_m(x)$$

After M trees, the final prediction is:

$$\hat{y} = F_M(x) = F_0(x) + \sum_{m=1}^{M} \nu \cdot h_m(x)$$

The final prediction $\hat{y}$ is the initial model $F_0(x)$ (often a constant like the mean of y) plus the sum of all M trees' predictions scaled by ν. This ensemble aggregates weak learners into a strong one.

Because every tree focuses on what is still unexplained, the ensemble gradually improves until additional trees no longer cut the validation loss.

4.5 Implementation Workflow

The full forecasting workflow was implemented in Python using open-source libraries (`pandas`, `scikit-learn`, `xgboost`, etc.), following a reproducible, modular structure. Each notebook (`setA`–`setF`) developed and tested a distinct feature set or model. The core implementation steps are as follows:

1. **Data Loading:** Import and preprocess the raw data, including handling timestamps and missing values.
2. **Feature Engineering:** Generate cyclical time features, target and weather lags, rolling means, and any additional domain-specific variables.
3. **Data Splitting:** Partition the dataset into training, validation, and test sets, reserving lag buffers to avoid information leakage.
4. **Feature Selection:** Use methods such as SelectKBest or Bayesian optimization to identify the most informative subset of features by minimizing the validation error (MAE).
5. **Hyperparameter Tuning:** Optimize model hyperparameters (e.g., number of trees, learning rate, max depth) using cross-validation.
6. **Validation and Evaluation:** Evaluate the model on the validation and test sets using error metrics.
7. **Analysis:** Analyze feature importance, daily error metrics, and compare model performance to baseline.

This modular workflow enables systematic experimentation, rigorous benchmarking, and easy extension with new features or algorithms.

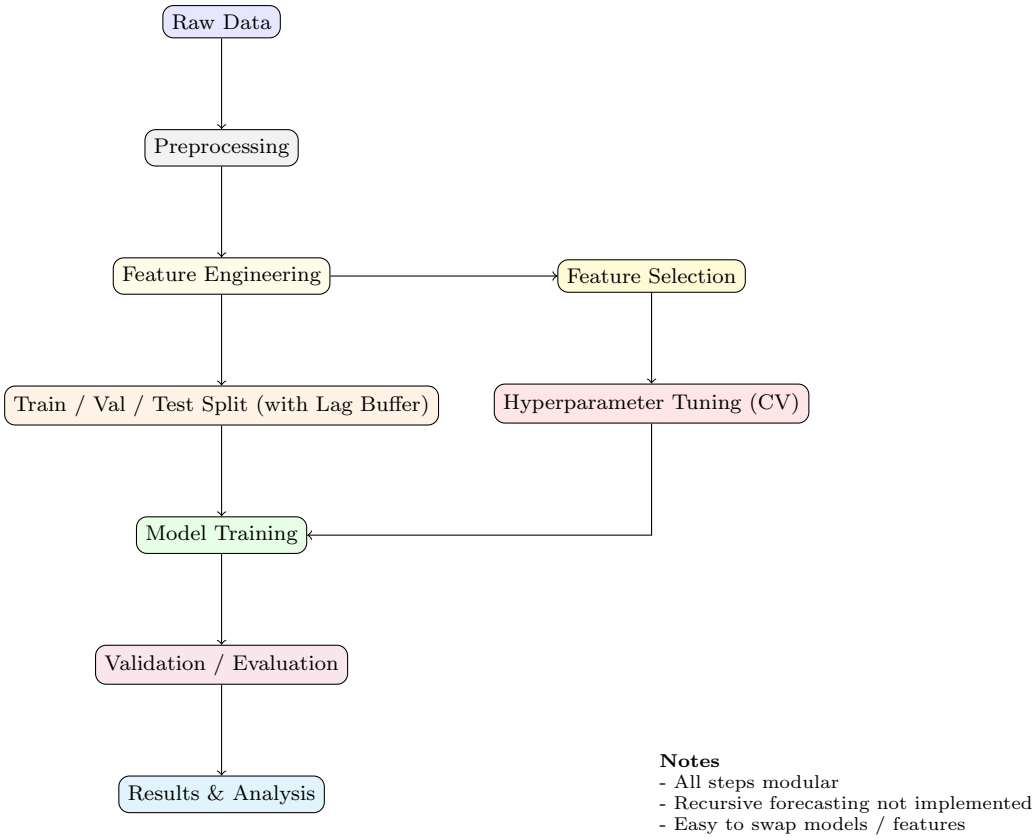

Figure 6: End-to-end forecasting pipeline: from raw data and feature engineering, through feature selection and model training, to evaluation.

5 Results

5.1 MILP Multi-Year Grid Optimization and Asset Planning

A. System Configuration

The example grid consists of five interconnected buses, each hosting a mix of generation, load, and storage assets (see Table 1). The asset portfolio includes dispatchable thermal generators (with varying capital costs and mutual exclusivity constraints), renewables (solar and wind), and an array of storage systems.

Unlike traditional static approaches, all assets are available as a *candidate pool* ; the MILP optimization selects which assets to deploy, and when, based on system needs and economics. Multiple candidate assets may be located at the same bus, reflecting both technical redundancy and the physical or geographic constraints of real networks. The line capacities and susceptances are configured to ensure a realistic representation of typical mid-scale regional grids.

To ensure the system remains responsive to evolving needs, demand is modeled to grow at a compound rate of 3% annually for the first 15 years, increasing to 5% per year thereafter. This results in a tripling of total load by year 30, imposing continuous pressure to replace 'about-to-die' assets in a timely manner.

B. Planning Horizon

The adoption of a mixed-integer linear programming (MILP) approach for multi-year planning enables the model to co-optimize both investment and operational decisions across the entire horizon. Unlike previous LP-based, scenario-driven studies, the MILP directly answers *when* and *where* to install, replace, or retire each candidate asset to meet both optimal economic and operational objectives.

ID	Type	Bus	Lifetime [yrs]	CAPEX [k$]
1001	Thermal Generator 1	1	25	10
1002	Thermal Generator 2	1	30	10
1003	Solar Generator 1	1	25	6000
1004	Solar Generator 2	2	25	8000
1005	Wind Generator 2	4	25	7000
1006	Solar Generator 3	3	25	4000
1007	Solar Generator 1	4	25	8000
1008	Wind Generator 3	4	25	8000
1009	Wind Generator 4	4	25	8500
1010	Thermal Alt. 1	1	4	70
1020	Thermal Alt. 2	1	25	950

ID	Type	Bus	Lifetime [yrs]	CAPEX [k$]
3001	Storage 1	2	10	3.5
3002	Storage 2	3	12	7.85
3003	Storage 3	1	11	4.25
3004	Storage 4	4	10	4.65

Table 1: Summary of Available Storage and Generations Assets in Grid Model

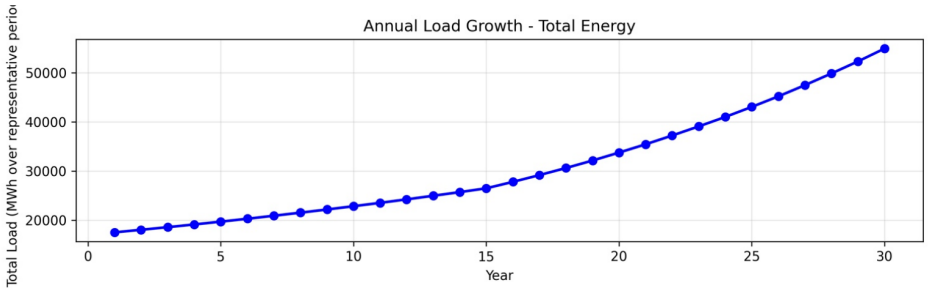

Figure 7: Annual load growth over the 30-year horizon.

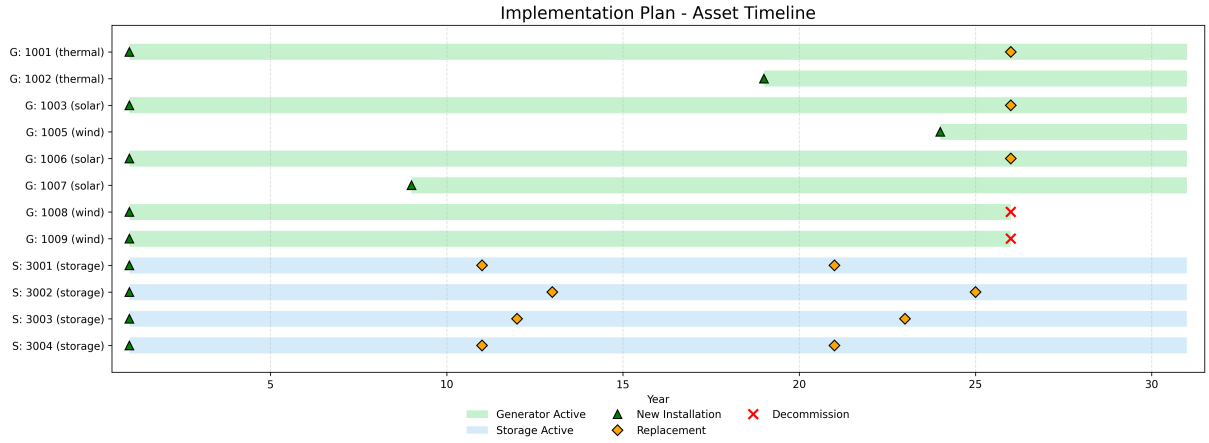

Figure 8: Asset timeline over the 30-year planning horizon

Figure 8 summarizes the resulting implementation and replacement schedule for all key assets. We notice that all available storage assets are commissioned already, while new renewables and thermal units are deployed as demand and system needs dictate.

Analysis:

- The optimization seems to prefer early deployment of 'green' assets such as solar, wind, and storage despite their relatively shorter lifetimes. It is proof of their economic relevance.

- Large-scale thermal investments are deferred until later in the planning horizon (e.g., year 19 for Thermal Generator 2). This is likely due to the fact that thermal assets are more expensive to build and operate.
- Storages also showcase their economic relevance beside their known convinience. All available storages are installed in the first period already.
- The model seems to be able to handle the load growth and the asset aging. It is able to replace the assets at the end of their technical lifetimes and increase the number of assets.

C. Energetic Mix

The new MILP model enables a detailed examination of how the system's energetic mix evolves over time. While seasonal analysis (summer vs. winter) was already feasible in the previous LP-based approach, the multi-year MILP framework now makes it possible to directly compare the system's dispatch of the planning horizon.

Figures 9 and 10 illustrate the generation mix during a representative winter week in Year 1 and Year 30, respectively. Analogous plots for a summer week are shown in Figures 11 and 12. We notice a significant difference in the asset utilization and the adaptability of the dispatch strategy across variable demands driven here by either the season or the load growth. Plot's Y-scales (Power [MW]) are not identical accross results.

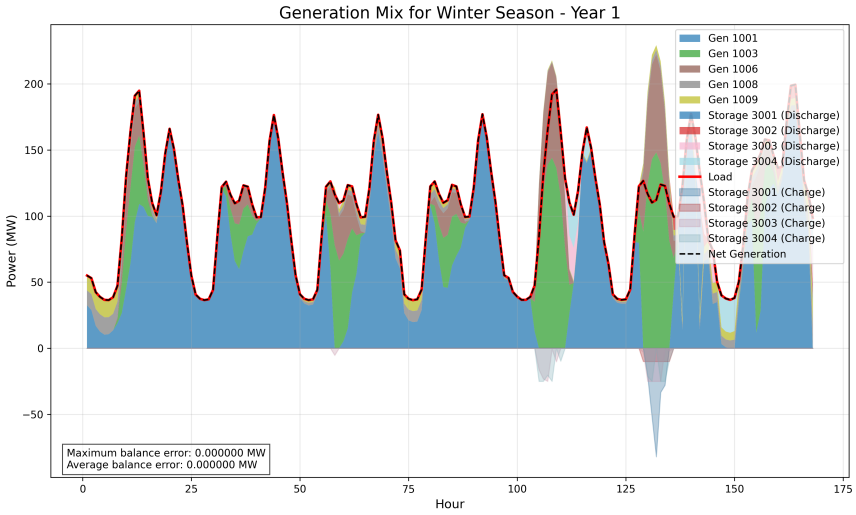

Figure 9: Winter week, Year 1

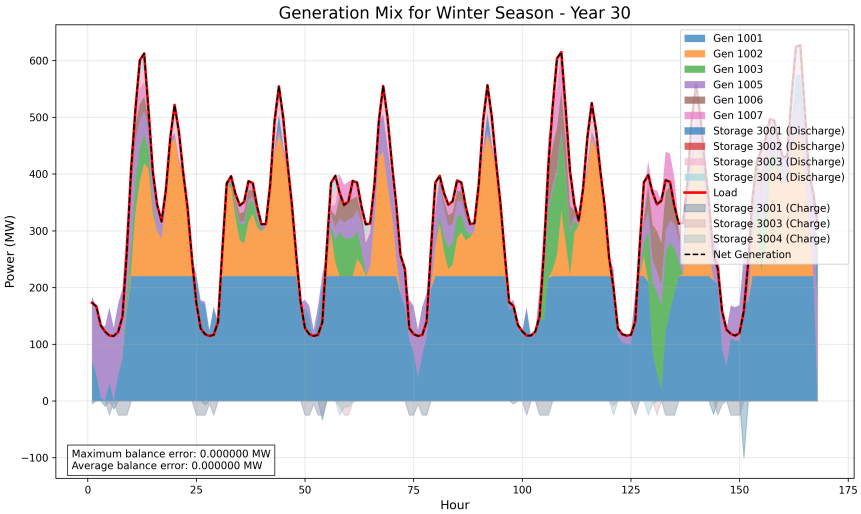

Figure 10: Winter week, Year 30

Analysis – Winter Mix

- Winter shows a heavy reliance on existing thermal assets. By Year 30, the additionnal thermal asset covers the 'plafond' of the existing thermal asset. Renewables can not provide the base load.
- Storage output increases substantially in Year 30, managing larger ramps and regular cycling. It signals a key role in maintaining a reliable grid.
- Despite the addition of renewables, thermal assets we remain vital in winter, reflecting either limited renewable resource in this season or conservative investment choices.
- If renewables are consistently underutilized or storage always reaches maximum charge/discharge, it may suggest the need for additional flexible or renewable assets, or point to over-restrictive asset pre-selection.

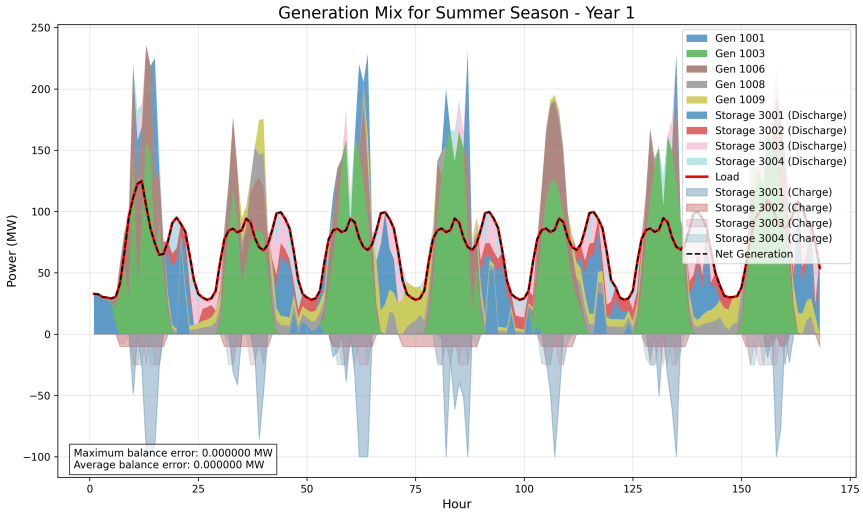

Figure 11: Summer week, Year 1

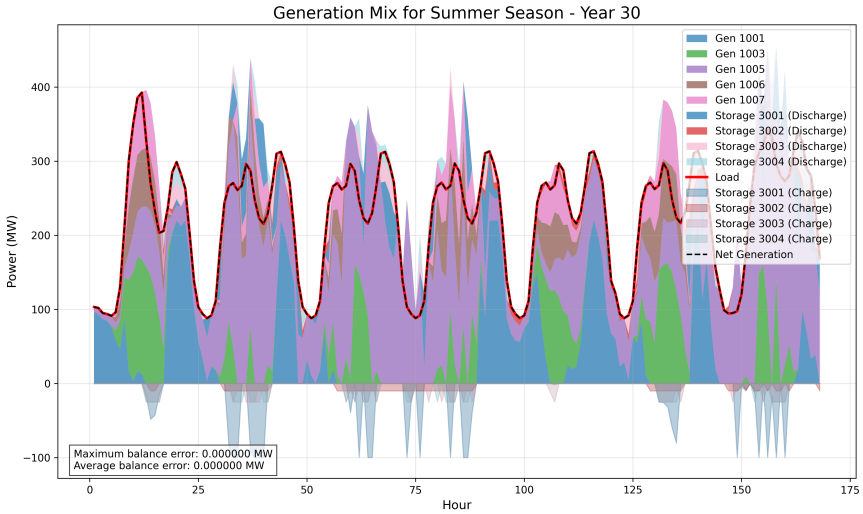

Figure 12: Summer week, Year 30

Analysis – Summer Mix

- Summer dispatch is more balanced, with renewables and storage taking a leading role, especially in Year 30. Solar output is visibly larger, and storage both absorbs surplus and releases it during evening peaks.
- Storage shows deep charge/discharge cycles, reflecting its use in smoothing the "duck curve" effect common in high-renewable summer systems.
- If storage is persistently full/empty or solar/wind are curtailed, this could signal network or asset selection limits. Should be investigated.

  We notice however the storage no longer produce as much surplus as in the first years.

Analysis – Summer vs Winter

- As seen in the 1st version of the model, summer clearly drives renewable assets penetration. High solar availability, plays a crucial role there. The detailed dispatch over the years shows that their utilization is economically meaningful with an increase in the load.
- The model is able to handle the load growth and the asset aging. It is able to replace the assets at the end of their technical lifetimes and increase the number of assets.
- Storage plays a critical, season-dependent role—absorbing midday surplus in summer. As demand grows, even winter periods begin to see sufficient surplus to enable storage cycles, highlighting how higher loads can unlock storage value year-round.
- We notice that assets are used to their full potential in winter by the last years of the planning horizon. This highlights not only the solver's ability to maximize the utilization of existing assets before commissioning new ones, but also demonstrates, on a smaller scale, how even a 3–5% annual increase in load inevitably drives the need for additional assets and infrastructure.

D. Discussion

The MILP framework offers clear advantages over static or LP-based approaches for multi-year grid planning, enabling joint optimization of investment and operations while explicitly handling asset aging and demand growth. However, several model and code limitations should be noted when interpreting the results.

Limitations

- The use of representative 'weeks' for each year speeds up simulation but may miss rare, extreme events (prolonged renewable droughts, spikes in demand, etc.).
- Asset investments are modeled as annual, binary decisions—this simplifies computation but prevents gradual or partial upgrades.
- The model is deterministic, lacking explicit treatment of uncertainty or scenario variation, and does not capture all operational details (e.g., ramping, start-up costs, maintenance downtimes).
- While tractable for this test grid, scalability to larger systems or finer time resolution could be computationally challenging.

Advantages and Disadvantages

- The main strength is the direct linkage between investment decisions and operational outcomes, allowing transparent tracking of asset utilization and portfolio evolution.
- The candidate-pool method reflects real planning flexibility, and the modular structure supports future extensions.
- The simplified operational and uncertainty modeling, as well as week-based aggregation, may limit realism—especially for system resilience to rare stress events or volatility in demand/generation.

Capabilities

- This framework is well-suited to address core planning questions, such as the marginal value and utilization of storage or renewables, tradeoffs between asset types, and how investments shift as technical/economic limits are reached.
- It can analyze resilience (e.g., under severe winter or renewable drought), track changes in system cost, CO_2 emissions, and identify when transmission or asset constraints become critical.
- While perfect foresight is assumed, the model could be extended with scenario or stochastic analysis to improve robustness.

- Further work could also include demand response, multi-objective optimization, and dynamic pricing.

Summary

The new MILP framework developed provides a technically sound and highly adaptable foundation for long-term energy system planning. It could serve as foundation model for a range of real-world applications, including asset sizing, maintenance scheduling, risk analysis and pricing strategies for executive levels.

While simplifications such as weekly time aggregation, perfect foresight, and annual binary investment decisions were necessary for tractability, the model still delivers valuable insights. Its outputs offer clear investment and operational guidance, making it a practical tool for both academic analysis and strategic planning.

Nonetheless, certain limitations remain—most notably, the absence of uncertainty modeling and the coarse temporal resolution. Future work could address these by incorporating stochastic optimization or robust planning approaches, finer time-step modeling, and coupling with demand or availability forecasting modules. It would increase decision relevance / realism. Despite these gaps, the model performs well across scenarios and offers a strong platform for scenario analysis, sensitivity testing, and deeper strategic investigation.

5.2 Forecasting module

A series of models (each building upon the previous one) were developed and compared for operational forecasting of electricity generation. Performance was evaluated over a 7-day period starting 2024-01-01. Plotted metrics are the first 4 days only. The principal metrics considered are MAE, RMSE, MAE, and R^2, computed both overall and per day. Table 2 summarizes the test results for all model variants.

Model	RMSE	MAE	R^2	Fit Time (s)
A. Time only	0.189	0.113	-0.934	0.36
B. Time features	0.123	0.059	0.178	0.41
C. Optimized features	0.217	0.112	-1.539	24.22
D. Hyperparameter tuning	0.131	0.060	0.076	140.30
E. Weather features tuned	0.103	0.051	0.428	572.58
F. Sarima (statistical)	0.116	0.056	0.275	313.02

Table 2: Forecasting model performance on test set (7 days).

A. Time Features Only

The baseline model (A), relying solely on time features, achieved poor predictive performance ($R^2 = -0.218$) on the first day, with substantial errors for the days after (see daily breakdowns).

We began with a minimal model using only time-related features (hour and day). This provided a simple benchmark and captured regular daily and weekly patterns but did not account for weather effects or recent historical trends. It confirms that pure time features are not sufficient.

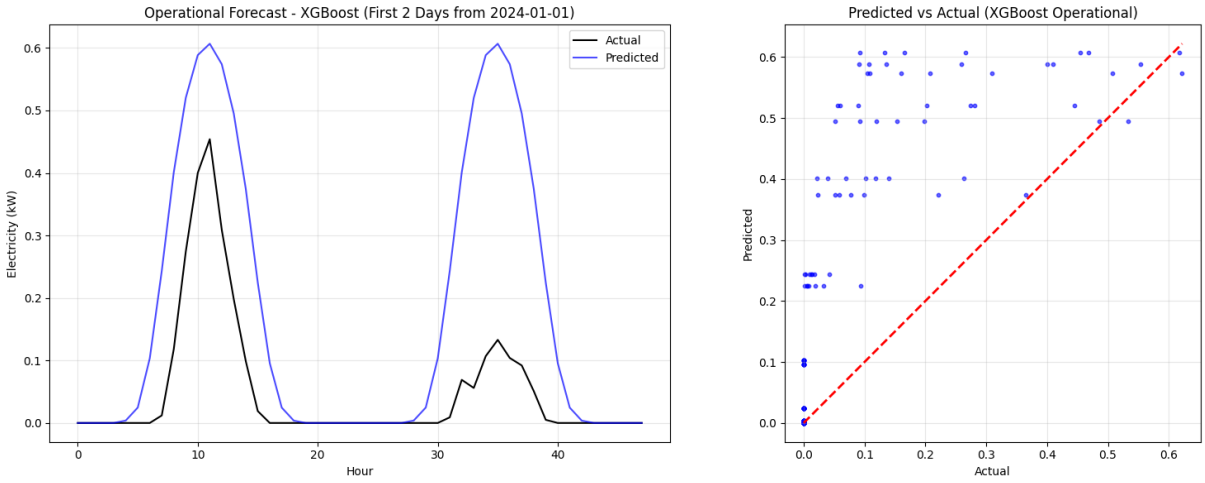

Figure 13: Set A - Time features only

Date	MAE	RMSE	R^2
2024-01-01	0.100	0.151	-0.218
2024-01-02	0.152	0.241	-31.657
2024-01-03	0.039	0.067	0.913
2024-01-04	0.072	0.114	0.585

Table 3: Set A - Daily Performance Metrics

B. Time + Lagged Features

Recognizing the autocorrelated nature of PV output, we next included lagged values of electricity generation (e.g., previous hour, previous day, previous week). However, significant errors remained on some

days, indicating the model's limited ability to capture complex outlier days – not robust in changing conditions

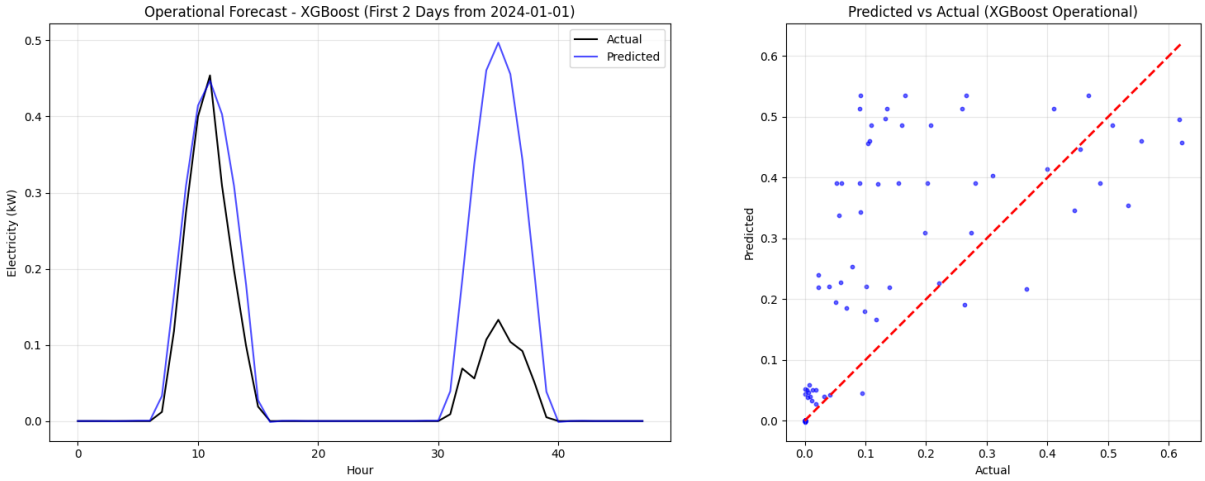

Figure 14: Set B - Time + time/cyclical features

Date	MAE	RMSE	R^2
2024-01-01	0.018	0.036	0.930
2024-01-02	0.080	0.153	-12.133
2024-01-03	0.039	0.071	0.902
2024-01-04	0.022	0.043	0.941

Table 4: Set B - Daily Performance Metrics

C. Feature Selection

We observe that lag features help the model nail the autocorrelation, which boosts performance on most days, but the improvement is not consistent. The model is still vulnerable to non-typical days, and the operational R^2 can still be negative. The feature selection was expected to filter "more" noise.

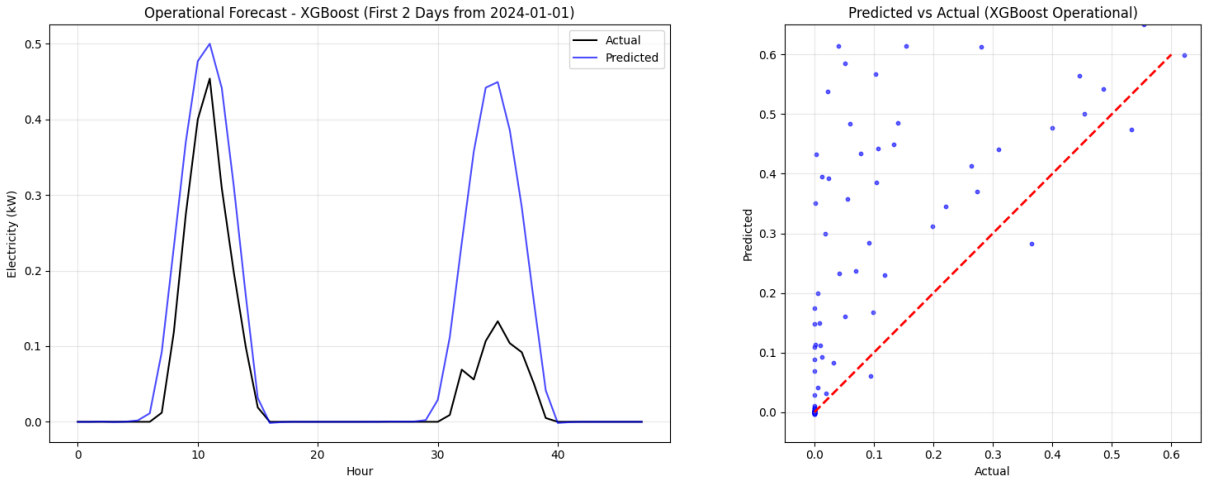

Figure 15: Set C - after feature selection

Date	MAE	RMSE	R^2
2024-01-01	0.031	0.055	0.841
2024-01-02	0.078	0.140	-10.063
2024-01-03	0.037	0.065	0.919
2024-01-04	0.082	0.145	0.333

Table 5: Set C - Daily Performance Metrics

D. Hyperparameter Tuning

We performed cross validation which splits the data into several parts (folds), trains on some, and tests on others. It should prevent overfitting. It requires a peticular attention when it comes to time series data so the splits are not randomly picked but rather sequential in time.

Also we did hyperparameter tuning. It is the search process to find the best hyperparameters for the model. Its robustness is improved The parameters are were the following:

- `colsample_bytree` fraction of columns used for training
- `learning_rate` learning rate
- `max_depth` maximum depth of a tree
- `min_child_weight` minimum weight of a leaf
- `n_estimators` number of trees
- `subsample` (fraction of samples used for training)

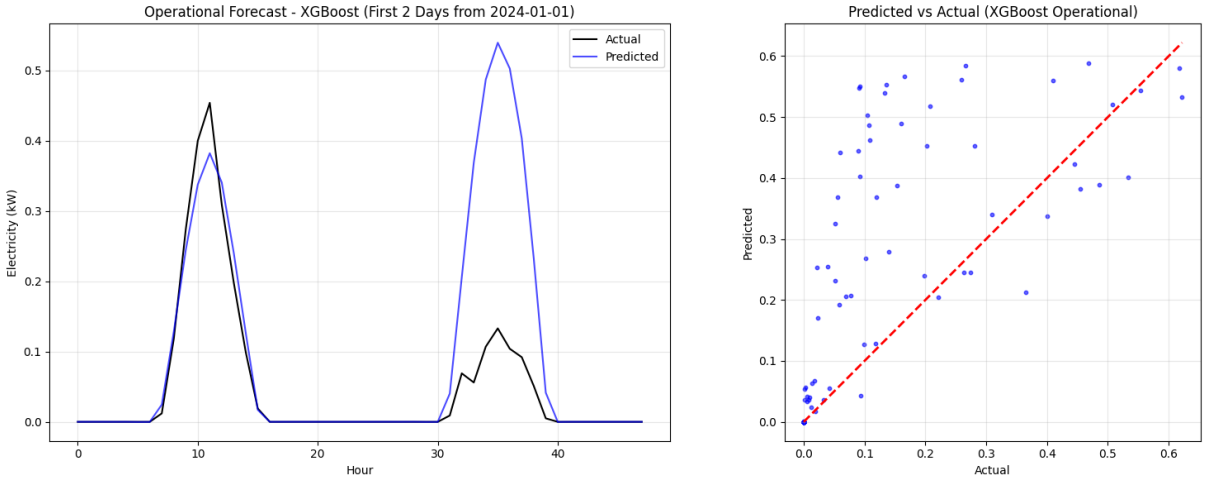

Figure 16: Set D - after hyperparameter tuning and cross-validation

Date	MAE	RMSE	R^2
2024-01-01	0.012	0.024	0.970
2024-01-02	0.092	0.173	-15.799
2024-01-03	0.022	0.047	0.957
2024-01-04	0.032	0.064	0.870

Table 6: Set D - Daily Performance Metrics

Ironically, these steps could lead to overfitting and poor generalization. With limited evaluation calls may not fully explore the searchspace, leading to suboptimal solutions. It requires a trade-off between exploration and exploitation.

E. Enhanced Features with POA Clear-Sky

Finally, we extended the feature set to include physics-based drivers—specifically, plane-of-array (POA) clear-sky irradiance and weather-driven variables.

They do provide some improvement on certain challenging days, but overall, weather features still rank low in importance. We notice that the 'engineered' features do not even rank on the top features. The problem may lie in their integration rather than their importance. It may underline a implementation issue.

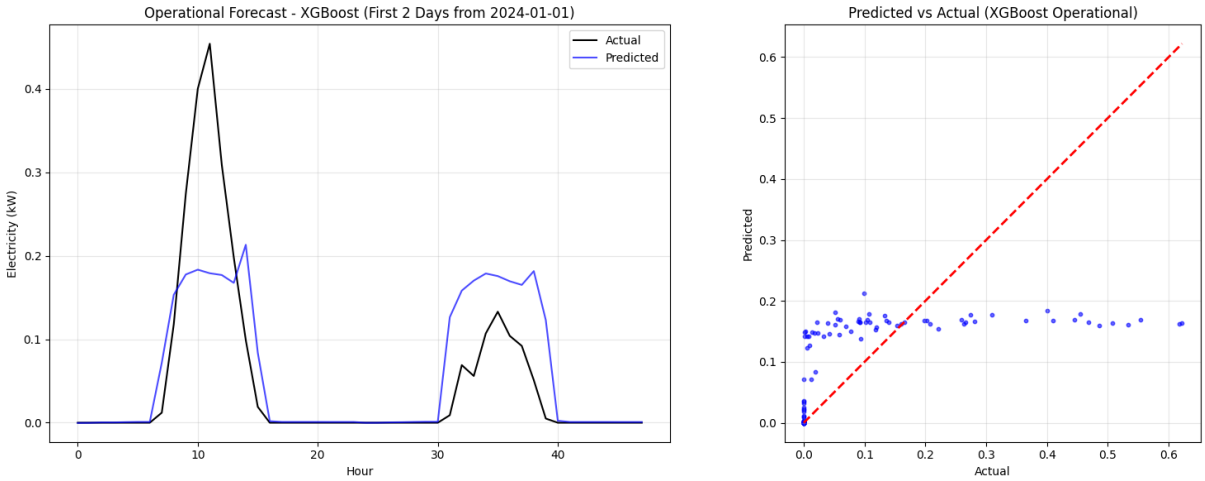

Figure 17: Set E - Weather features

Date	MAE	RMSE	R^2
2024-01-01	0.043	0.085	0.619
2024-01-02	0.035	0.059	-0.929
2024-01-03	0.101	0.187	0.326
2024-01-04	0.071	0.133	0.441

Table 7: Set E - Daily Performance Metrics

Error and accuracy metrics are improved. However, the model is still far from being perfect and computational expense is high for a sole 0.1% improvement of the MAE-error for the day ahead prediction horizon.

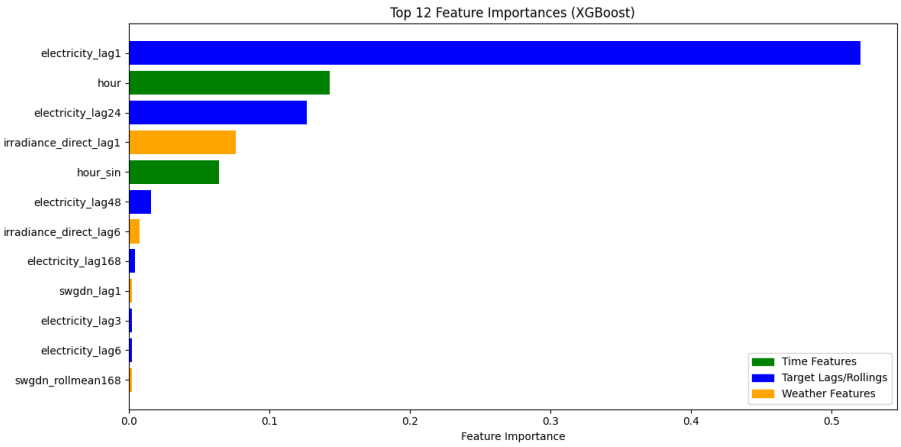

Figure 18: Feature Importance including weather-features

We performed a feature selection with the new weather-features based to understand their importance compare to the previous time-only-lags. We notice 4 additional "weather"-features in the top 12. The sum of their importance which improves the prediction performance by 8-12% of the MAE-error. Their impact remain low given the high number of features and computational cost.

F. Sarima (Statistical Model)

The autocorrelation (ACF) and partial autocorrelation (PACF) plots both showed clear seasonal structure (peaks at multiples of 24) but also a non-stationary trend in the first difference. The SARIMA output does a surprisingly good job at capturing the general daily trend (especially for normal days), but it consistently fails to capture outliers and more complex events (anticipated).

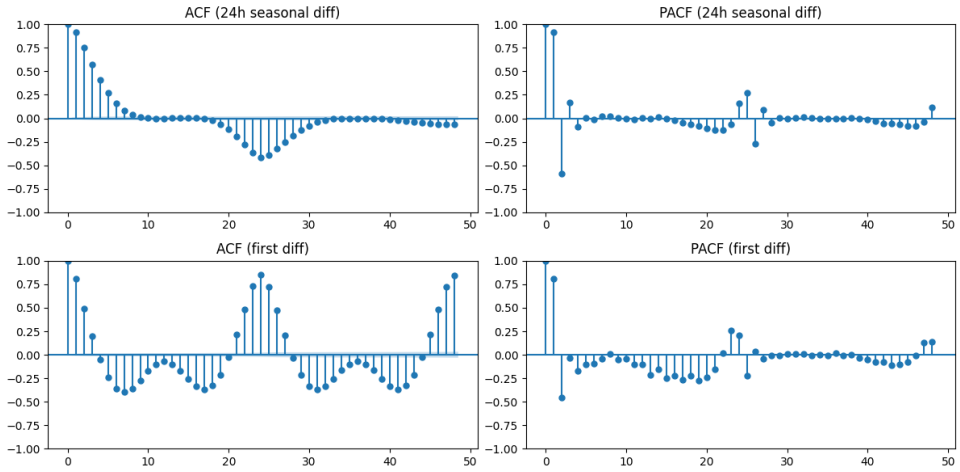

Figure 19: ACF and PACF of the SARIMA model

The daily performance is strong on "regular" days (high R^2 for Jan 3 and Jan 4), but quickly breaks down on days with unexpected events or shifts (e.g., R^2 = -12.93 on Jan 2). This reinforces the idea that statistical models are excellent baselines for trend but not for "edge" or outlier behavior.

A more exhaustive grid search over p, d, q and seasonal parameters might help, but likely won't solve the outlier issue fundamentally.

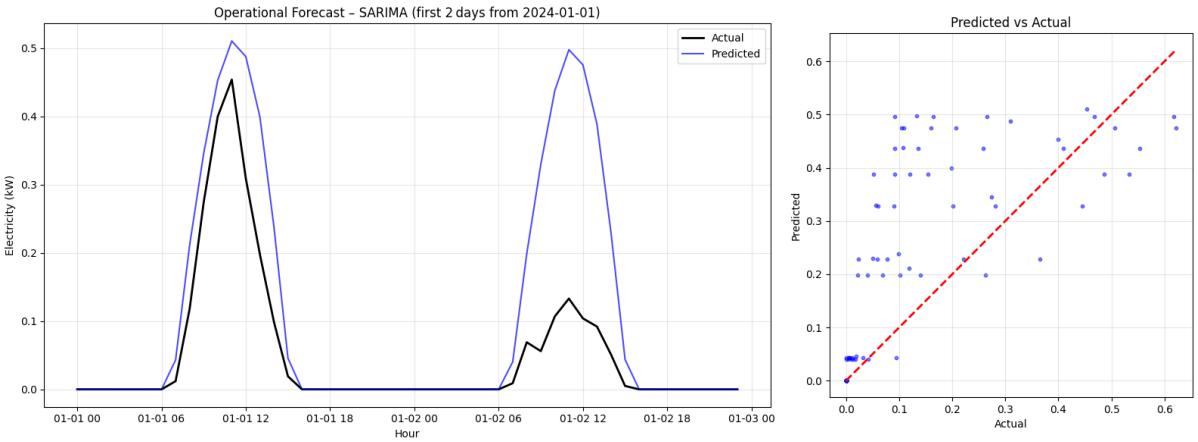

Figure 20: Set F - Sarima

Date	MAE	RMSE	R^2
2024-01-01	0.036	0.069	0.749
2024-01-02	0.084	0.157	-12.932
2024-01-03	0.038	0.068	0.912
2024-01-04	0.014	0.028	0.975

Table 8: Set F - Daily Performance Metrics

Discussion

Throughout the experiments, we observed that lagged electricity (especially `electricity_lag1`, `electricity_lag24`) features were consistently the most influential predictors in all model configurations. This makes sense, given the strong autocorrelation and daily patterns inherent in PV production. Even as more "sophisticated" time features and rolling statistics were added, these basic lags retained their dominance in the feature importance rankings. The most recent lags are the most important.

When integrating weather-derived variables such as direct irradiance, POA, and the clear-sky index, the anticipated improvement in forecast performance was not as consistent as expected. While these features should theoretically help the model adapt to dynamic meteorological changing days, their overall feature importance remained low. This may reflect either limitations in the integration or engineering of the weather data, or a need for more sophisticated approaches to extract actionable signals. That said, we did observe that weather features played a valuable role in flagging outlier days, even if they rarely supplanted lagged variables in overall importance. We can not rely on metrics only since outliers days do not come as often as normal days. A more targeted evaluation, for example focusing on rare or extreme events, could better highlight their potential value and should have been done.

Throughout the iterations, forecast accuracy showed significant variability, often failing to improve in step with added model complexity or optimization techniques. This instability may point to issues like overfitting, insufficient generalization, or perhaps simple human error—especially around the validation setup. My choice to avoid neural networks reflects both practical constraints and a lack of deep expertise. Neural models are powerful. However, they can be computationally intensive and less interpretable, which was in my sense not a fit for this study's objectives or resources.

Recursive prediction (where each new forecast feeds into the next) was tested and not documented. It proved to be a realistic but challenging operational scenario, as errors propagated rapidly and degraded performance. This suggests the appeal for hybrid approaches and statistical models like SARIMA, which, while simple, provide interpretable baselines and help reveal core data dynamics with minimal prior knowledge.

Overall, these experiments reinforced that lag-based models are robust baselines in operational PV forecasting, and that leveraging weather features is far from straightforward. The inconsistent gains across variants suggest that careful, operationally-aware validation and a critical, data-driven approach to feature selection are crucial and beneficial. There is still considerable room for progress, particularly for rare and extreme days, but these findings also caution against unnecessary complexity without clear justification.

6 Conclusions

This semester project tackled two core challenges in energy systems analysis: (i) long-term grid investment planning using Mixed-Integer Linear Programming (MILP), and (ii) operational forecasting of renewable generation using machine learning (ML). While these modules were developed separately, their eventual integration could offer a more robust foundation for future planning models.

Main contributions and findings

- The MILP framework enabled a consistent treatment of asset investment, replacement, and operational dispatch over a multi-decade horizon. Compared to the previous LP-based, the model allows explicit representation of asset lifetimes, technical constraints, and timing of investments, yielding a more realistic and flexible planning tool.
- The machine learning forecasting pipeline—primarily based on gradient boosted trees (XGBoost) demonstrated reliable day-ahead PV prediction using lagged features and cyclic encodings. More advanced feature engineering (physics/weather features) provided only incremental benefit metric-wise but significantly improved the model's ability to forecast non-standard days.
- Throughout, the codebase and workflow, the goal was to be reproducible and modular. The modules were designed to be ready-to-ship and are highly modular.

Limitations and lessons learned

- Representative-week aggregation and annual decision periods were kept from the previous LP-based model to keep the MILP computationally tractable. From an operational perspective, this is a limitation, from a investment perspective, it is a strength.
- The optimization was strictly deterministic, without scenario or stochastic elements—thus omitting an explicit treatment of operational risk, uncertainty, or market volatility.
- ML-driven forecasting, while operationally relevant and technically sound, did not yet close the loop with the investment model; forecast errors and their economic impacts were not propagated into long-term planning decisions.
- For the ML forecasting module, further complexity did not consistently yield clear improvements in this setting emphasizing the need for careful validation and a "human-critical" view of model sophistication versus marginal benefit.

Directions of improvement

This work provides a platform that is suitable for scenario analysis of investment timing and asset portfolio choices under fixed technical and economic assumptions, benchmarking forecasting techniques in an operational context, and serving as a base for more advanced modeling (e.g., uncertainty quantification, demand-side management, dynamic pricing, or larger grid cases).

Based on the experience and findings here, the most promising next steps would be:

- Integrate the ML forecasting pipeline directly with the investment optimization, quantifying the effect of forecast uncertainty on both dispatch and investment outcomes.
- Extend the MILP framework to handle explicit (real-case) scenario or stochastic optimization, allowing for risk assessment and robustness analysis under uncertain demand, supply, or price paths.
- Explore more granular time discretization and additional operational details (e.g., ramping, outages, or multi-stage investments) as computational resources and solver performance allow.
- Test scalability and transferability on larger or more realistic grids, and evaluate the model's strengths and limitations in those contexts.

Personal reflection
A key learning was how deceptively thin the barrier can be between conceptual advances—such as "if investment is made, then operation is allowed" and the real work of implementing those concepts in code. Adding a single logical condition or adjusting the cost function often required substantial restructuring and multiple iterations. The same held true for the forecasting module: progressing from one model type to another, I learned through experience what each method actually brought to the table. Throughout, the process was less about chasing complexity and more about understanding where each approach adds value.

Working on both long-term investment and short-term forecasting in the same project highlighted how the dominant factors and modeling choices shift across time scales. Ultimately, this iterative, hands-on work deepened my appreciation for reproducible research and clarified both the potential and the boundaries of increased model sophistication.

Final note The tools and results developed here provide a foundation for further research in integrated power system planning and forecasting. While substantial work remains to fully close the loop between forecast-driven operations and long-term investment, this project offers a clear step in that direction and a basis for future technical improvements.

Acknowledgements

For the redaction of this report, I would like to acknowledge the use of artificial intelligence to improve the clarity and structure of my sentences. The core observations, analyses, and personal reflections are entirely my own, drawn from my experiences during the field trip and subsequent research. The LLM usage was employed primarily for language refinement, code formatting, and orthographic corrections. Its integration helped communicate complex concepts clearly and effectively.

A Optimization formulation

A.1 MILP Formulation

Sets

- G: Set of generators
- S: Set of storage units
- B: Set of buses
- L: Set of lines
- Y: Set of years in the planning horizon
- Σ: Set of seasons
- T_s: Set of time steps in season $s \in \Sigma$

Decision Variables

- $\gamma_{g,y} \in \{0,1\}$: Binary variable, 1 if generator g is *installed* in year y
- $\gamma^{st}_{s,y} \in \{0,1\}$: Binary variable, 1 if storage s is installed in year y
- $p_{g,y,s,t} \geq 0$: Generation by generator g in year y, season s, time t
- $f_{l,y,s,t}$: Power flow on line l in year y, season s, time t
- $c_{s,y,sn,t} \geq 0$: Charging power of storage s in year y, season sn, time t
- $d_{s,y,sn,t} \geq 0$: Discharging power of storage s in year y, season sn, time t
- $SOC_{s,y,sn,t} \geq 0$: State-of-charge of storage s in year y, season sn, time t

Objective Function

$$\min \text{ TotalCost} = \underbrace{\sum_{g\in G}\sum_{y\in Y} \text{AnnCost}_g \cdot \gamma_{g,y}}_{\text{Annualized CAPEX Gen}} + \underbrace{\sum_{s\in S}\sum_{y\in Y} \text{AnnCost}^{st}_s \cdot \gamma^{st}_{s,y}}_{\text{Annualized CAPEX Storage}} + \underbrace{\sum_{y\in Y}\sum_{sn\in\Sigma} w_{sn} \sum_{t\in T_{sn}} \left[\sum_{g\in G} c^{op}_g p_{g,y,sn,t}\right]}_{\text{Operational Cost}} \tag{10}$$

where AnnCost_g and AnnCost^{st}_s are the **annualized investment costs** (including CAPEX and OPEX annuity), w_{sn} is the weight for season sn, and c^{op}_g is the marginal cost for generator g.

Constraints

1. **Build/Installed Relationship (Lifetimes):**

$$\gamma_{g,y} = \sum_{y'\leq y,\, y-y'<L_g} z_{g,y'} \quad \forall g \in G, y \in Y \tag{11}$$

$$\sum_{y'\leq y,\, y-y'<L_g} z_{g,y'} \leq 1 \quad \forall g \in G, y \in Y \tag{12}$$

$$\gamma^{st}_{s,y} = \sum_{y'\leq y,\, y-y'<L_s} z^{st}_{s,y'} \quad \forall s \in S, y \in Y \tag{13}$$

$$\sum_{y'\leq y,\, y-y'<L_s} z^{st}_{s,y'} \leq 1 \quad \forall s \in S, y \in Y \tag{14}$$

where L_g and L_s are the lifetimes of generator g and storage s.

2. **Generator Capacity (per time step):**

$$p_{g,y,sn,t} \leq p_g^{\max} \cdot \gamma_{g,y} \cdot p_{g,sn,t}^{pu} \qquad \forall g, y, sn, t \tag{15}$$

where $p_{g,sn,t}^{pu}$ is the per-unit profile (e.g., for wind/solar).

3. **Storage and SoC:**

$$c_{s,y,sn,t} \leq p_s^{\max} \cdot \gamma_{s,y}^{st} \tag{16}$$

$$d_{s,y,sn,t} \leq p_s^{\max} \cdot \gamma_{s,y}^{st} \tag{17}$$

$$SOC_{s,y,sn,t} \leq E_s^{\max} \cdot \gamma_{s,y}^{st} \tag{18}$$

$$SOC_{s,y,sn,t+1} = SOC_{s,y,sn,t} + \eta_s^{in} c_{s,y,sn,t} - \frac{1}{\eta_s^{out}} d_{s,y,sn,t} \tag{19}$$

$$SOC_{s,y,sn,0} = 0, \tag{20}$$

4. **Line Flows:**

$$|f_{l,y,sn,t}| \leq f_l^{\max} \qquad \forall l, y, sn, t \tag{21}$$

5. **Power Balance (per bus, per time step):**

$$\sum_{g \in G_b} p_{g,y,sn,t} + \sum_{s \in S_b} (d_{s,y,sn,t} - c_{s,y,sn,t}) + \sum_{l \in L_b^{in}} f_{l,y,sn,t} = \lambda_{b,y,sn,t} + \sum_{l \in L_b^{out}} f_{l,y,sn,t} \tag{22}$$

where G_b and S_b are generators and storages at bus b, L_b^{in} and L_b^{out} are lines entering and leaving bus b, and $\lambda_{b,y,sn,t}$ is the total load at bus b.

Notes

- All investments are annualized using the Capital Recovery Factor (CRF), with optional inclusion of discounted OPEX.
- The model natively supports technology lifetimes, mutually exclusive build groups, and seasonal time-series operation.
- Binary variables are only used for build and installation status; all other variables are continuous.

B Extra Plots & Test Outputs

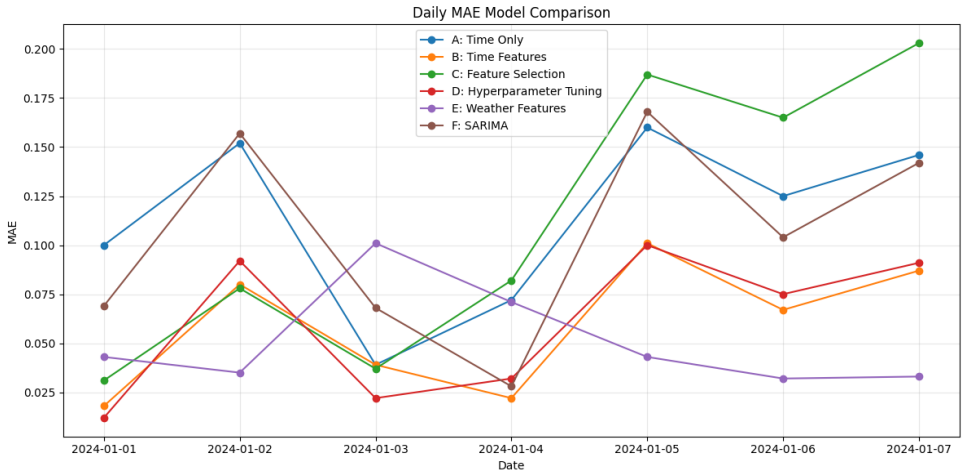

Figure 21: MAE error comparison between the XGBoost model and the statistical baseline for daily forecasts.

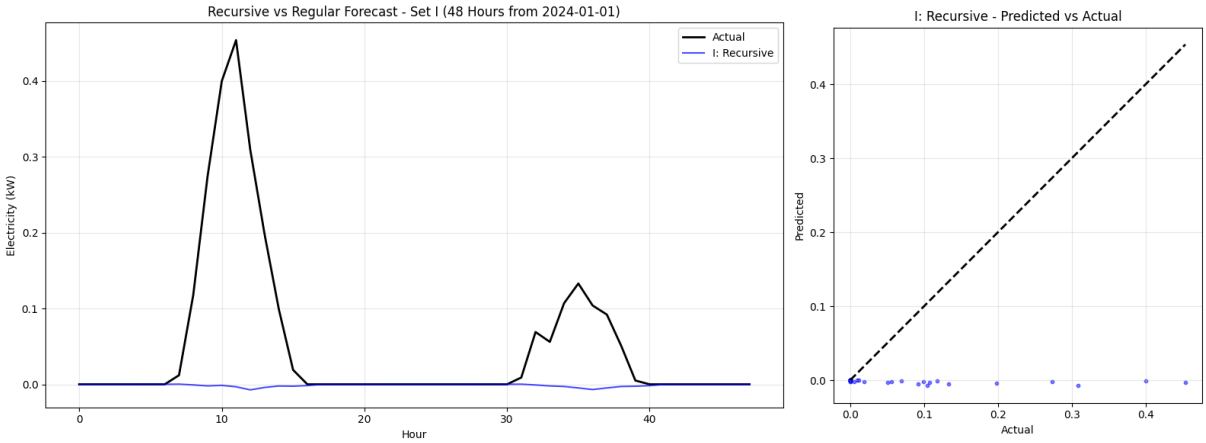

Figure 22: Recursive forecast of the XGBoost model for daily forecasts. (no weather features)

C Code attachments

C.1 Forecasting module

- A-time.py
- B-timeFeatures.py
- C-featureSelection.py
- D-hyperparameterTuning.py
- E-weatherFeatures.py
- F-sarima.py

C.2 MILP framework

- pre.py
- optimization.py
- network.py
- main.py
- post.py
- costs.py

References

[1] Göran Andersson. Modelling and analysis of electric power systems. Lecture Notes 227-0526-00, ETH Zürich, Zürich, Switzerland, March 2004. Power Systems Laboratory, ETH Zürich.

[2] IBM Documentation. Starting cplex with the python api, 2022. Accessed 2024-06-15.

[3] M Grzebyk, G Szymanski, and P Zajac. Day-ahead photovoltaic power forecasting using xgboost and weather data. *Energies*, 14(4):1012, 2021.

[4] Charles R. Harris and Millman. Array programming with numpy. *Nature*, 585:357–362, 2020.

[5] Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, et al. Scikit-optimize: Sequential model-based optimization with scipy, scikit-learn and matplotlib. *Journal of Open Source Software*, 3(30):1014, 2018.

[6] William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. pvlib python: a python package for modeling solar energy systems. *Journal of Open Source Software*, 3(29):884, 2018.

[7] Wes McKinney. Data structures for statistical computing in python. In Stefan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010.

[8] Hans D. Mittelmann. Benchmarks for optimization software, 2023. Accessed 2024-06-15.

[9] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine learning in python, 2011. Journal of Machine Learning Research, 12, pp. 2825-2830.

[10] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[11] Allen J Wood, Bruce F Wollenberg, and Gerald B Sheblé. *Power Generation, Operation, and Control*. John Wiley & Sons, 2013.

[12] Y Zhong and Q Wu. Day-ahead solar power prediction based on xgboost and lstm. *Energy Reports*, 6:767–774, 2020.