UNIVERSITEIT VAN AMSTERDAM

FINAL PROJECT REPORT

Digital Twin Engineering

GROUP H

Authors:

Luca Giandomenico
Gabriel Gil Pichelbauer
Jovana Vukmirovic
Yunxuan Tang

October 15, 2025

Academic Year 2025/2026

Contents

1 Introduction

The transition toward renewable energy systems requires advanced tools that can monitor, analyze, and optimize energy production in real time. Among these, **Digital Twin** (DT) technology has emerged as a promising approach for bridging the gap between physical systems and their digital counterparts. By combining real-world data with computational models, digital twins enable continuous performance assessment, anomaly detection, and predictive maintenance.

This project focuses on developing a Digital Twin for a solar power plant environment. The system integrates real historical datasets from two distinct solar plants, each equipped with multiple weather and power generation sensors. The main purpose of this digital twin is to **replay, visualize, and predict** the behavior of the solar system under different environmental conditions. By doing so, it provides valuable insights into how factors such as ambient temperature, module temperature, and solar irradiation influence power generation.

The developed system is composed of five main components:

1. **Dataset Integration:** combining sensor readings and energy production data from two plants, using timestamps to synchronize weather and generation records.
2. **Representation Model:** establishing relationships between environmental variables and system outputs, forming the basis of the virtual model.
3. **Historical Replay System:** a FastAPI-based backend capable of replaying past states of the solar plants by querying InfluxDB for specific timestamps or time ranges.
4. **Machine Learning Module:** an LSTM-based predictive model that forecasts future solar generation based on historical data trends.
5. **Visualization output:** a 3D interactive system capable of displaying the historical data and also displaying future power generation predictions using the Machine Learning Module.

Together, these components create a functional digital twin that not only reflects the past and present behavior of the solar plants but also supports forward-looking analysis. The resulting system serves as a foundation for energy performance optimization and intelligent control, contributing to more sustainable and efficient renewable energy management.

This work was completed as part of the Digital Twin Engineering course at the University of Amsterdam, under the Faculty of Science (Informatics Institute).

2 System Purpose and Users

2.1 Purpose

The purpose of the Solar Power Digital Twin is to create a data-driven simulation environment that mirrors the operational behavior of real solar power plants. By integrating weather conditions and power generation data, the system allows users to analyze, replay, and predict the performance of solar panels under varying environmental factors.

The system also serves as a foundation for predictive maintenance and energy optimization. By coupling the replay mechanism with machine learning models, the digital twin can evolve into a proactive system that not only reflects the current state of the plant but also forecasts its future performance. This provides clear value for sustainable energy management and long-term efficiency planning.

2.2 Use cases

From a practical perspective, the digital twin provides a valuable decision-support tool for solar farm operators and energy analysts. It enables them to:

- Monitor plant performance and detect deviations in generation patterns. If specific panels don't perform as expected, it can be decided to disconnect them for maintenance.
- Understand how temperature, irradiation, and environmental conditions affect the energy yield.
- Evaluate the effectiveness of different operational strategies using historical replay.

2.3 Requirements

For the digital twin to fulfill its intended purposes, it should meet a few requirements:

- The DT simulation should be consistent with its real-world counterpart at all times.
- The user should be able to choose the time frame for the historical replay.
- The user should be able to switch between replay mode and prediction mode.
- The predictions made by the machine learning model should have a high accuracy.
- The visualization output should be clear and human-understandable, in order to further analyze it.

3 Dataset and Data Processing

The dataset that was used for the digital twin[1] consists of time-stamped sensor and power generation data from two solar power plants in India. The data was collected over a 34 day period, from 15-05-2020 00:00 until 17-06-2020 23:45.

- It contains two pairs of power generation data/weather sensor readings, each pair corresponding to a solar power plant.
- The power generation data is gathered at the inverter level (22 per plant), where each inverter has multiple solar panels attached to it, while the weather sensor data is gathered at the plant level.
- The time interval between readings is 15 minutes.

[1] https://www.kaggle.com/datasets/anikannal/solar-power-generation-data/data

The **power generation** files consist of the following columns:

- DATE_TIME
- PLANT_ID
- SOURCE_KEY – unique inverter id
- DC_POWER – DC power generated by the inverter during the interval
- AC_POWER – AC power generated by the inverter during the interval
- DAILY_YIELD – cumulative energy generated by the inverter on that day
- TOTAL_YIELD – cumulative total energy generated by the inverter

On the other hand, the **weather sensor** files contain the columns:

- DATE_TIME
- PLANT_ID
- SOURCE_KEY
- AMBIENT_TEMPERATURE – temperature at the plant
- MODULE_TEMPERATURE – temperature reading for the solar panel attached to the sensor panel
- IRRADIATION

3.1 Data preprocessing & brief analysis

3.1.1 Preprocessing

In order for the data to be used by the machine learning model and the backend infrastructure, some data preprocessing steps had to be performed.

1. In the raw CSV files, one of the power generation data files uses a different DATE_TIME string format than the other files. This was corrected to ensure proper DATE_TIME comparisons between files.
2. All DATE_TIME fields were also converted into proper datetime objects. This was especially important for the integration with InfluxDB3 (see step 6).
3. Redundant columns were removed. For example, in the weather sensor data files, the SOURCE_KEY is the same for the whole file, as the sensor readings are on the plant level. Additionally, PLANT_ID doesn't give us any extra information, because it is already clear which plant it belongs to based on the file name. Thus, both columns could be safely removed.

4. For both pair of files, the generation data file was merged with the corresponding weather sensor data on the DATE_TIME column to create unified records for each inverter at each timestamp. The two merged dataframes were additionally merged again to obtain one dataframe that contained both plants, in case this was needed for the ML model training. However, this dataframe was not stored in the final InfluxDB3 database.

5. The unique inverter IDs (the SOURCE_KEY column in the generation data files) are complex strings in the CSV files. For example, '1BY6WEcLGh8j5v7' corresponds to inverter 1 from plant 1. To make it easier to work with the data, these were all replaced with simple integer-based IDs (1-22). Similarly, PLANT_ID was replaced wth simple integers (1 and 2).

6. Lastly, we stored the finalized dataframes in InfluxDB3, which is a database that is specifically catered to event and time-series data. It detects the datetime objects and correctly stores the records with their time indices in the database. Although the machine learning model can be trained on the dataframes, the integration with a sophisticated database is especially important for the backend API, to ensure efficient querying during the DT simulation. Hence, a local private database was created in our GitHub repository using InfluxDB3 Core to store the dataframes of both solar power plants. When the local InfluxDB3 server is running, the data can be queried by time, plant number (1 or 2), inverter ID, etc. using SQL-like queries.

3.1.2 Analysis

To find out more about the underlying distribution of the data and potentially aid the ML model training process, we performed a brief analysis of the data, mainly by plotting graphs. These were the most important findings:

- There are no missing values in the data, thus no data imputation was necessary.
- AC/DC power generated, irradiation and temperature all peak during roughly the same daylight hours (6:00-18:00) and days (see Figures 1, 2, 3 for plant 1 as an example).
- A subsequent correlation test found that the aforementioned variables indeed have a strong correlation with each other, although the strength differs between plants (see Figure 4).

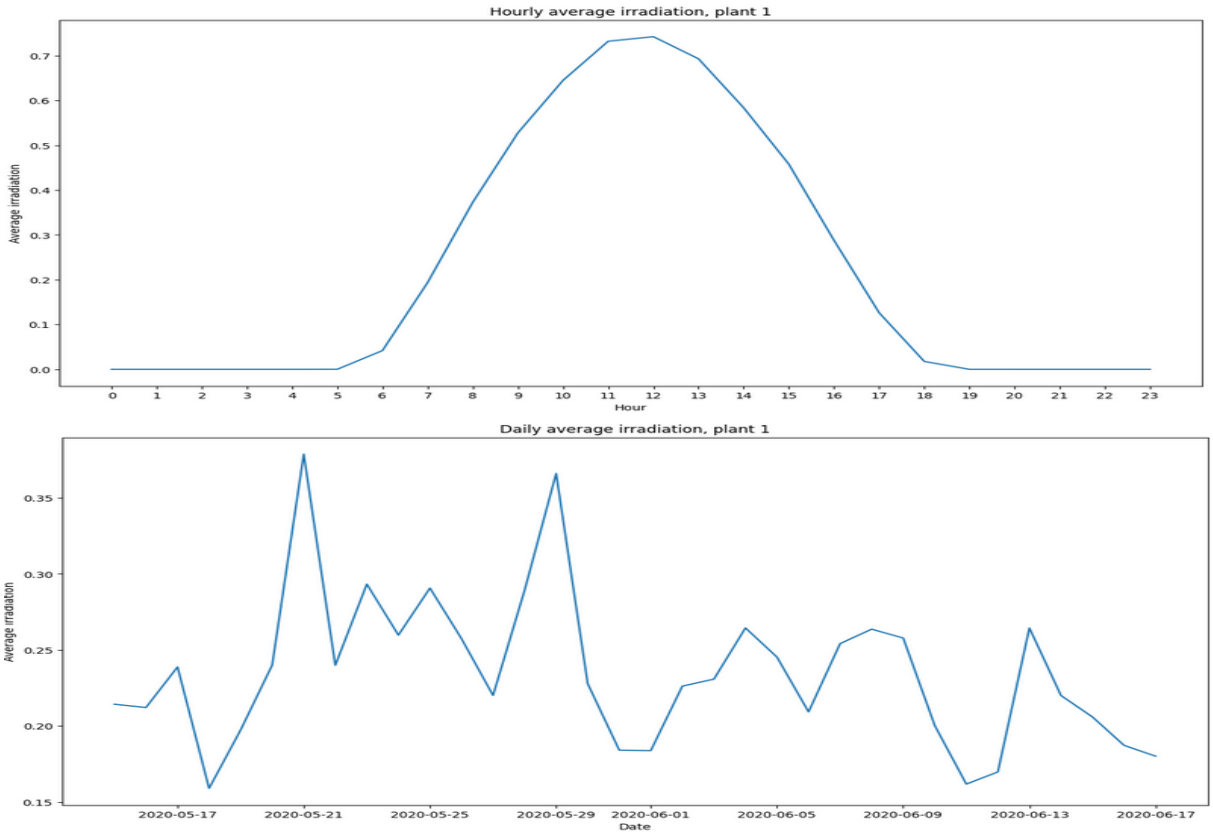


Figure 1: Daily and hourly average irradiation.

Figure 2: Daily and hourly average temperature.

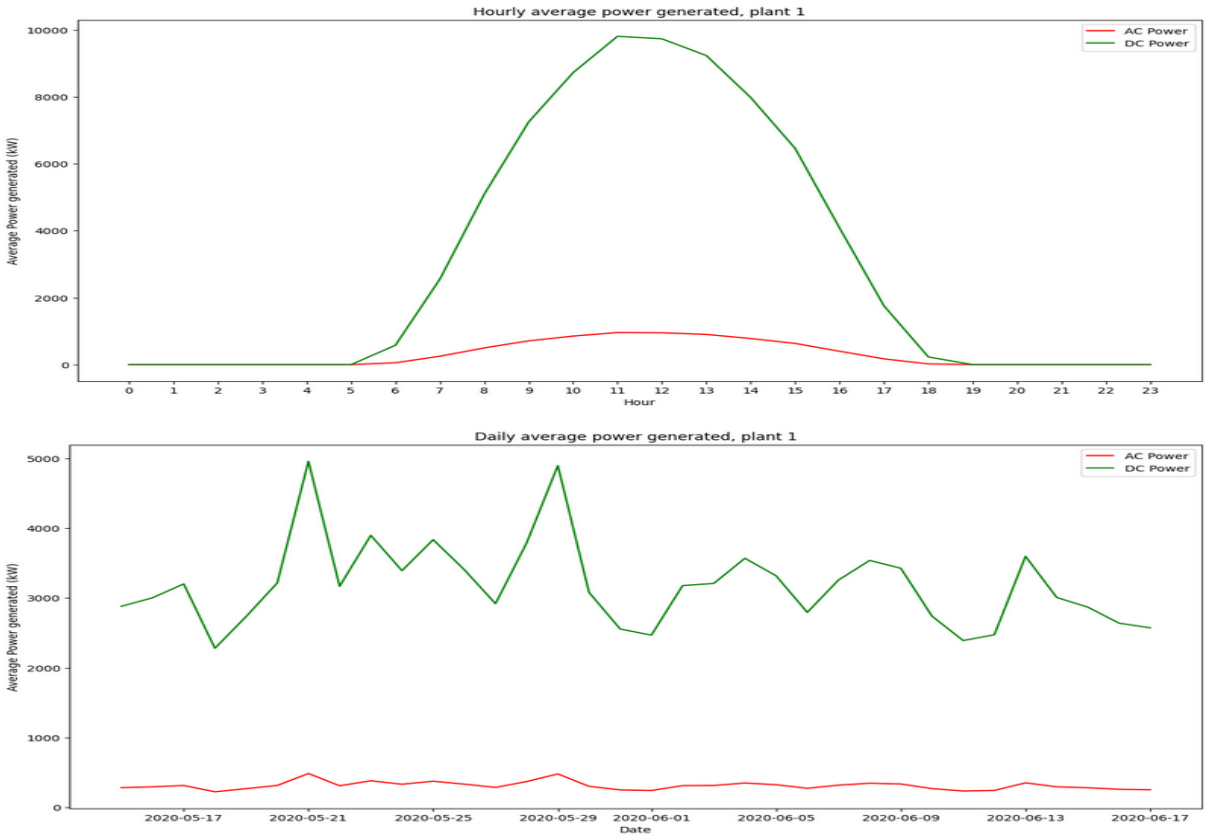


Figure 3: Daily and hourly average power generated.

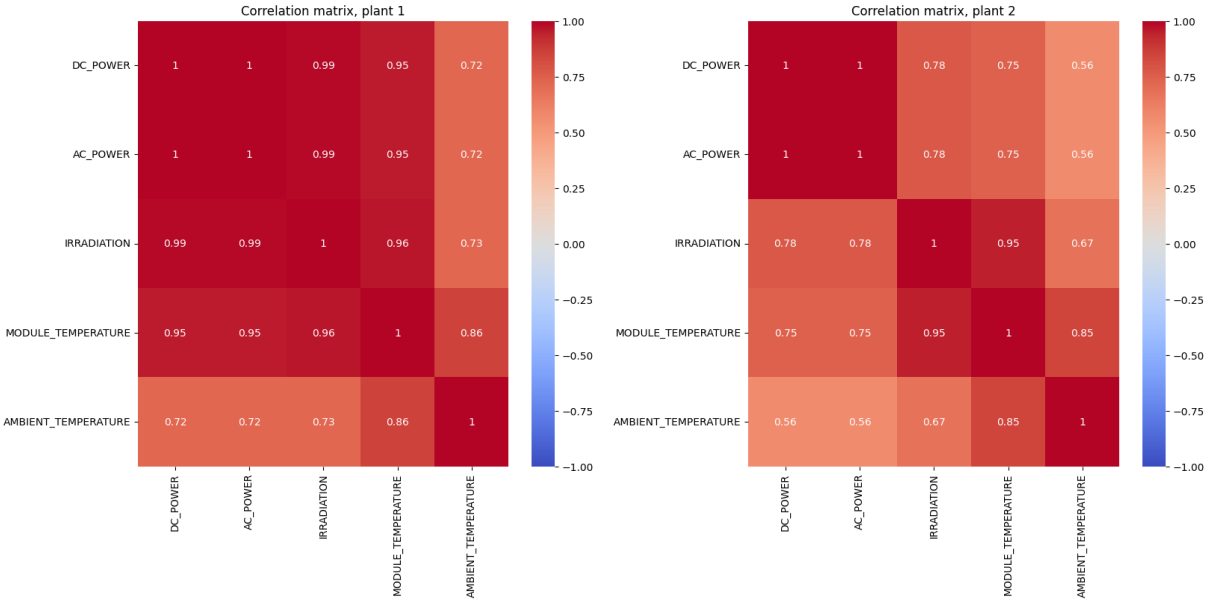


Figure 4: Correlation matrices of relevant variables. Left: plant 1, right: plant 2.

4 Representation Model

4.1 Entities, variables and relations

The DT system can be represented by the following entities, variables and relationships:

- **Solar power plant**
  - Contains all the data belonging to the specific plant. Has attribute PLANT_ID. Has 22 inverters and a weather sensor.
- **Inverter**
  - Contains all the power generation data belonging to the specific inverter. Has attributes SOURCE_KEY, DC_POWER, AC_POWER, DAILY_YIELD and TOTAL_YIELD. Belongs to one plant.
- **Weather sensor**
  - Contains the weather data belonging to a specific plant. Has attributes AMBIENT_TEMPERATURE, MODULE_TEMPERATURE and IRRADIATION. Belongs to one plant.
- **Measurement**
  - Contains all the power generation and weather data corresponding to one inverter (and thus one plant) at a specific timestamp. Has attribute DATE_TIME, next to all the previously mentioned power generation and weather attributes.

See Figure 5 for a visual diagram.

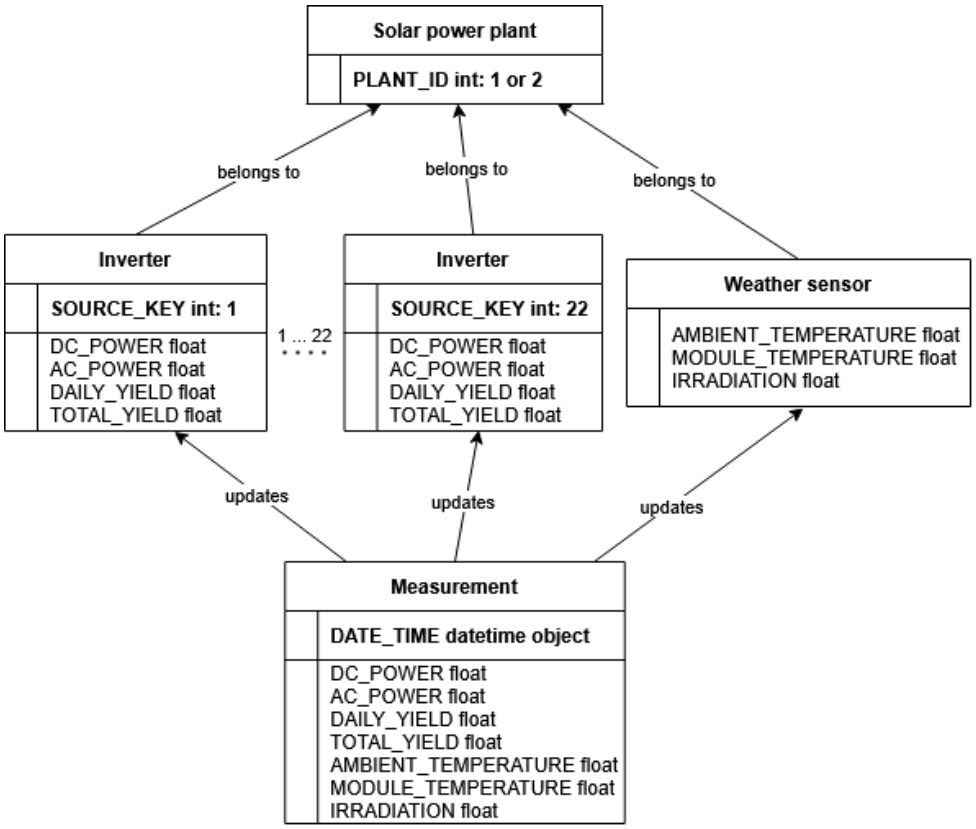


Figure 5: Visual representation of the entities, variables and relations.

4.2 Data states

Conceptually, the state of the system at any given time can be seen as a concatenation of current power generation and weather data from all inverters of both plants. This could be formally defined as $S(t) = \{s_i(t) | i = 1, 2, 3, ..., 44\}$, where each s_i stands for a specific inverter and its corresponding data:

```
{
  "PLANT_ID": 1,
  "SOURCE_KEY": 1,
  "DC_POWER": 0.0,
  "AC_POWER": 0.0,
  "DAILY_YIELD": 0.0,
  "TOTAL_YIELD": 6259559.0,
  "AMBIENT_TEMPERATURE": 25.184316133333333,
  "MODULE_TEMPERATURE": 22.8575074,
  "IRRADIATION": 0.0
}
```

However, if we're looking from the visualization perspective, only one inverter is shown at a time; thus, the state of the system could also be defined as just $S(t) = s_i(t)$, where s_i represents the current data from the specific inverter that is being shown.

When a new measurement (the next timestamp in chronological order) is retrieved from the database through the backend API, the inverter's values are updated ($s_i(t) \rightarrow s_i(t + \Delta t)$),

which is then reflected in the visualized output. There's a new measurement every 15 minutes, which translates to 3 seconds in the DT simulation during historical replay, although the speed can be adjusted. During prediction mode, the integrated machine learning model can be used to predict the next AC power generation output of an inverter. In this case, the inverter data will also include the predicted power output to be able to compare it to the actual output.

5 Event System

The system implements two distinct event handling modes:

5.1 Replay Mode (Historical Simulation):

- Events are retrieved from the historical database via REST API calls.
- The simulation progresses through timestamps sequentially, maintaining the original 15-minute data cadence.
- Time compression is configurable (1x to 20x speed), where 1x speed means 3 seconds of real time equals 15 minutes of simulated time.
- The visualization requests the next event after the configured interval, ensuring proper time order.
- Each API call fetches data for a single timestamp, preventing event mixing.

5.2 Prediction Mode:

- Events are requested on-demand for specific timestamps.
- The system fetches both historical actual data and generates ML predictions for comparison.
- Navigation controls allow stepping through available prediction timestamps in chronological order.
- Each prediction event is processed individually before moving to the next timestamp.

5.3 Timeline Consistency.

The system maintains strict temporal consistency:

- Replay Mode: Advances by exactly 15 minutes per iteration.
- Time Compression: Configurable speed multiplier (1x = 3 seconds real time per event, 20x = 0.15 seconds per event)
- Event Ordering: Timestamps are always processed in ascending chronological order
- Event Isolation: Each API call retrieves data for a single timestamp, ensuring events remain separate

6 System Transitions and Workflows

6.1 Event Handling Workflow

6.1.1 Event Request Phase

The frontend calculates the target timestamp based on current simulation state and forms a GET request to `/replay?plant=[1|2]&timestamp=[YYYY-MM-DD HH:MM:SS]`

6.1.2 Data Retrieval Phase

The backend queries the InfluxDB time-series database, filters data by plant, inverter, and exact timestamp, then returns matched inverter readings or a ”no data found” status.

6.1.3 State Transition Phase

When an event is received, the system executes the following workflow:

1. **Validate Data Existence**: Check if the API returned valid data or a ”no data found” message.
2. **Filter by Selected Inverter**: Extract the specific inverter’s reading from the returned dataset.
3. **Update System State**: Store the new timestamp, power output, temperature readings, and irradiation values.
4. **Calculate Visualization Parameters**: Convert AC power output to a normalized ratio (0-1 scale) for color mapping.
5. **Update 3D Visualization**: Apply color interpolation from red (low power) to green (high power) based on the power ratio.
6. **Update HUD Display**: Refresh all numerical displays with the new sensor readings.
7. **Schedule Next Event**: In Replay Mode, schedule the next data fetch based on the configured speed multiplier.

6.2 State Transition Diagram

6.2.1 Replay Mode States

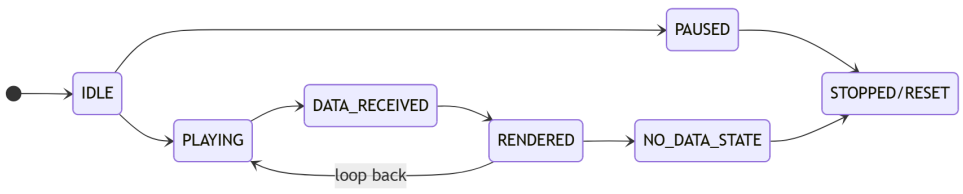


Figure 6: Replay mode states chart

6.2.2 Prediction Mode States

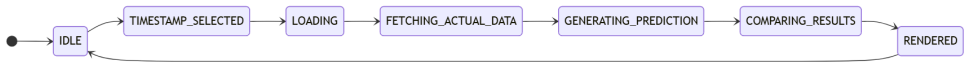


Figure 7: Prediction mode states chart

6.3 Transition Triggers and Handlers

Trigger	State Transition	Description
Play button click	IDLE → PLAYING	Starts simulation timer and begins fetching data
Timer interval expires	PLAYING → DATA_RECEIVED	Triggers API call for next timestamp
Data received successfully	DATA_RECEIVED → RENDERED	Updates visualization and HUD
Pause button click	PLAYING → PAUSED	Stops timer but maintains current timestamp
Stop button click	ANY → STOPPED	Resets all selections and returns to initial state
Generate Prediction click	IDLE → LOADING	Initiates prediction workflow with 1-second delay
Timestamp navigation	RENDERED → LOADING	Moves to different timestamp and auto-generates prediction
Jump to date/time	ANY → LOADING	Seeks to specified timestamp and loads data

Table 1: State Transition Triggers and Handlers

6.4 Replay Mode Workflow

1. User selects plant and inverter from dropdown menus
2. User clicks Play button to start simulation
3. System fetches data for current timestamp via API call
4. System validates and filters data for selected inverter
5. Visualization updates: 3D panel changes color, HUD displays metrics
6. System increments timestamp by 15 minutes
7. After configured interval (3 seconds ÷ speed multiplier), repeat from step 3
8. Process continues until user pauses/stops or end of dataset is reached

6.5 Prediction Mode Workflow

1. User selects plant and inverter from dropdown menus

2. System loads available prediction timestamps for that inverter
3. System sets first timestamp as active and displays timestamp counter
4. User clicks Generate Prediction button
5. System enters loading state (blue panels, 1-second delay)
6. System fetches historical actual data for the selected timestamp
7. System sends prediction request to ML API endpoint
8. LSTM model processes environmental features and returns predicted AC power
9. System displays both actual and predicted values side-by-side
10. Both 3D panels update with color-coded power visualization
11. User can navigate to previous/next timestamps, which auto-generates new predictions
12. User can jump to specific timestamps using date/time controls

6.6 Error Handling Transitions

6.6.1 API Connection Error

- Transition to error state
- Display error message to user
- Set panel to blue (no data) state
- Clear all HUD values with dashes
- Maintain current timestamp

6.6.2 No Data Available

- Display status message indicating no data for selected inverter/timestamp
- Set panel to blue state
- Clear HUD metrics
- Increment timestamp and continue (Replay Mode) or wait for user action (Prediction Mode)

6.6.3 Invalid Timestamp Selection

- Display warning message
- Maintain current state
- Keep previous valid data displayed

6.6.4 Missing Inverter Selection

- Display prompt message
- Remain in idle state
- Prevent simulation from starting

6.7 Time Management

6.7.1 Replay Mode Time Progression

- Base interval: 3 seconds per 15-minute increment (1x speed)
- Speed multiplier: 1x, 2x, 5x, 10x, or 20x
- Actual interval = 3 seconds ÷ speed multiplier
- Timestamp always advances by exactly 15 minutes per event
- Chronological order is strictly maintained

6.7.2 Prediction Mode Time Navigation

- On-demand event processing (no automatic progression)
- User controls timestamp selection explicitly
- Navigation buttons automatically trigger prediction generation
- Jump functionality calculates closest available timestamp
- Each prediction event is fully processed before accepting next input

6.8 State Persistence

The system maintains state across transitions:

- **Current timestamp** persists through pause/resume
- **Selected plant and inverter** remain set until explicitly changed
- **Playback speed** setting is preserved
- **Prediction index** tracks position in available timestamps
- **Panel colors** reflect most recent valid data until new event updates them

This architecture ensures proper event sequencing, prevents event mixing, maintains temporal consistency, and provides clear state transitions throughout the user's interaction with the system.

7 Machine Learning Implementation

This section describes the deep learning implementation for our digital twin model. We trained a Long Short-Term Memory (LSTM) neural network using an inverter-level prediction strategy. In this approach, each of the plant's 22 inverters is treated independently during data preprocessing to preserve its unique operational characteristics. After generating predictive sequences for each inverter, the data is concatenated into a single dataset for model training, allowing the network to learn from a diverse range of inverter behaviors while maintaining temporal consistency.

7.1 Tech Stack

For the machine learning implementation, we used `TensorFlow` and `Keras` to build and train the LSTM recurrent neural network. We utilized `Scikit-learn's` metrics classes to evaluate model performance and `Matplotlib` to visualize the results.

7.2 Why LSTM?

Traditional feedforward neural networks cannot capture temporal dependencies in sequential data. LSTMs overcome this limitation with memory cells that:

- Retain long-term patterns.
- Learn sequential dependencies.
- Handle variable-length sequences.

Because solar power generation depends on time-related factors, such as recent weather conditions, time of day, and seasonal cycles, the ability of LSTMs to remember historical context makes them well-suited for this task.

7.3 Temporal Feature Engineering

Time is cyclical by nature, but linear encodings treat values like 23 and 0 as far apart, even though they are consecutive. To capture this periodicity, we use sine–cosine transformations:

```
hour_sin = sin(2π × time_of_day / 24)
hour_cos = cos(2π × time_of_day / 24)
day_sin  = sin(2π × day_of_year / 365)
day_cos  = cos(2π × day_of_year / 365)
```

These transformations map time values onto the unit circle, where each point corresponds to a sine–cosine coordinate. As we move counterclockwise around the circle, the angle increases: $\pi/2$ (90°) aligns with 6:00 AM, π (180°) with 12:00 PM, $3\pi/2$ (270°) with 6:00 PM, and 2π (360°) returns to midnight. In this representation, hours 23 and 0 are positioned as neighbors on the circle, preserving their natural closeness. By encoding time this way, sequence models like LSTMs can more effectively learn daily and seasonal cycles without artificial breaks introduced by linear encodings.

7.4 Feature Scaling

We used 8 features (`'DC_POWER'`, `'AMBIENT_TEMPERATURE'`, `'MODULE_TEMPERATURE'`, `'IRRADIATION'`, `'hour_sin'`, `'hour_cos'`, `'day_sin'`, `'day_cos'`) to predict the target variable which is `AC_POWER`.

All features were scaled into the [0, 1] range using `MinMaxScaler`. Feature scaling is essential for LSTM training because:

- Gradient-based optimizers converge faster when all features are on a comparable scale.
- Without scaling, variables with large ranges (e.g., irradiance vs. temperature) could dominate learning.

To avoid data leakage, scalers were fit only on the training split. Separate scalers were used for features and targets to preserve independence between input normalization and output prediction.

7.5 Sequence Generation

We used fixed-length input sequences of 24 consecutive 15-minute observations (representing 6 hours) to predict the AC power value at the 25th timestep. A sliding-window approach was used to generate multiple training examples while maintaining temporal order.

7.6 Per-Inverter Processing

Inverters differ in orientation, capacity, and shading conditions. Training on mixed data without distinction risks confusing the model and introducing leakage. We therefore processed each inverter independently, generating sequences for each before concatenating the results. This ensured:

- No sequences span across inverters.
- No leakage between different hardware units.
- Each inverter's unique characteristics are preserved.

After sequence generation, data from all inverters was concatenated for training, providing diversity while preserving temporal integrity.

7.7 Data Splitting Strategy

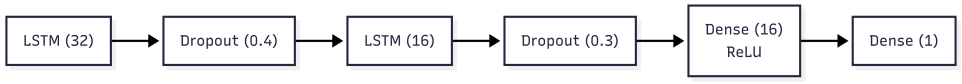
We used chronological splitting which allow the model to be tested on future data, preventing information leakage.

- 70% for training
- 10% for validation
- 20% for testing

7.8 Inverter Based Model Architecture

The model consist of:

- **LSTM Layer 1**: 32 units, `return_sequences = True`, to pass the full sequence to the next layer for hierarchical feature learning. An $L2$ kernel regularizer was applied to penalize large weights.
- **Dropout Layer 1**: 40% dropout for regularization to prevent overfitting.
- **LSTM Layer 2**: 16 units, compressing temporal representations into higher-level abstractions.
- **Dropout Layer 2**: 40% dropout for further regularization.
- **Dense Layer**: A fully connected layer with 16 neurons and a ReLU activation function to introduce non-linearity.
- **Output Layer**: A single neuron with a linear activation function for the continuous AC power regression task.



7.9 Training Configuration

- **Optimizer**: `Adam` with a learning rate of 0.001. Adam combines the benefits of momentum and adaptive learning rates, which makes it well suited for noisy, non-stationary data such as PV power signals. The chosen learning rate balances convergence speed with training stability.
- **Epochs**: Training was capped at 100 epochs, though convergence typically occurred earlier. This upper limit provides enough opportunity for the model to fit complex seasonal and daily patterns without excessive overfitting.
- **Batch size**: A batch size of 32 was selected to balance two trade-offs: smaller batches provide noisier but more generalizable gradient estimates, while larger batches improve efficiency but may converge to sharp minima. The chosen value offered stable training while remaining computationally efficient.

7.10 Regularization

- **Early stopping**: Validation loss was monitored, and training halted if no improvement was observed for 10 consecutive epochs, with the best weights restored. This prevents wasted computation, mitigates overfitting, and ensures the final model generalizes well to unseen data.
- **Model checkpoints**: We saved intermediate models during training so that the best-performing checkpoint (not necessarily the last epoch) could be retained. This guarantees that transient improvements are not lost.

7.11 Evaluation Metrics

We evaluated forecasting accuracy with multiple metrics:

- **Mean Squared Error (MSE)**: Penalizes larger errors more heavily.
- **Root Mean Squared Error (RMSE)**: Easier to interpret since it is in the same units as the target (AC power).
- **Mean Absolute Error (MAE)**: Provides a robust measure of average error magnitude.
- **R^2 (Coefficient of Determination)**: Indicates how much variance in the target is explained by the model.

7.12 Training and Validation Visualizations

Model Loss vs. Epoch

The training and validation loss curves show the model converging steadily over epochs. The validation loss tracks the training loss closely, confirming that the model is not overfitting.

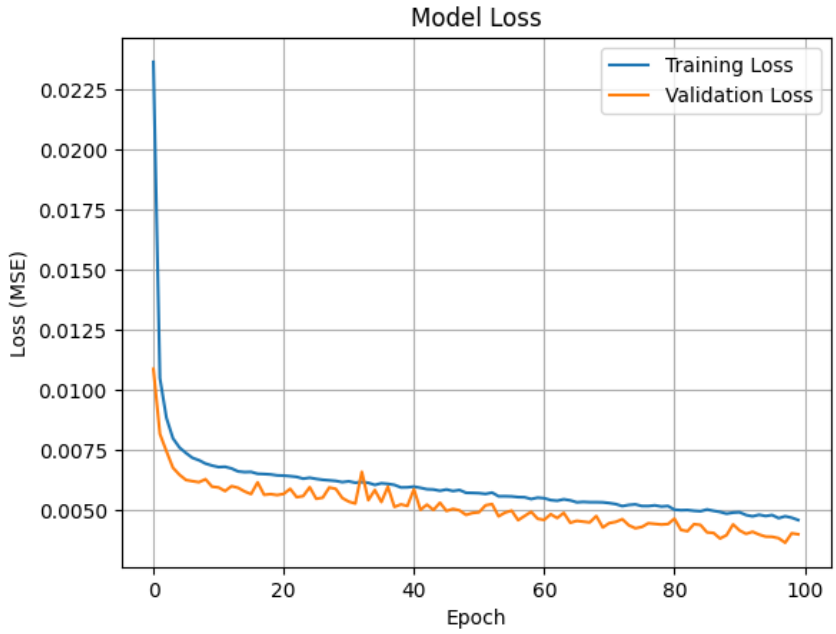


Figure 8: Model loss curve

Actual vs. Predicted AC Power (Test Set) This scatter plot of the test set shows a strong linear relationship between the actual and predicted values, with most points falling close to the ideal $y = x$ line. This visually confirms the high R^2 value and the model's accuracy.

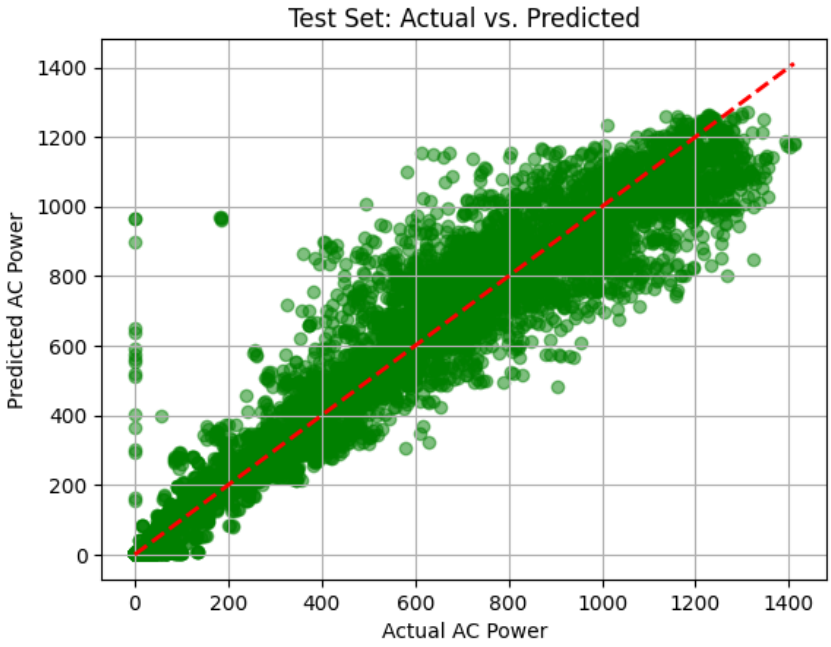


Figure 9: Actual vs. Predicted values on the test set

7.13 Improvements to the digital twin through the model

The predictive model forms the core of the Digital Twin for the solar power plant, transforming it from a static digital replica into a dynamic, forward-looking system. By forecasting PV power output, the model enables the Digital Twin to anticipate plant behavior on both short-term (hourly) and seasonal scales.

The model introduces new knowledge by capturing the cyclical patterns of solar generation, such as daily sunrise/sunset curves and seasonal variations, as well as the relationships between time, weather conditions, and power output. Time features are encoded using sine and cosine transformations, allowing the system to represent cyclical temporal information smoothly, while the LSTM-based architecture captures temporal dependencies, learning how past conditions influence future power. Together, these enrichments allow the Digital Twin to represent the system dynamically and provide actionable insights, such as predicting peak production periods, supporting operational decisions, and simulating different environmental scenarios to anticipate plant performance.

8 Visualization and Outputs

8.1 Chosen Visualization

The Solar Power Digital Twin employs a 3D interactive visualization built using Three.js (v0.160.0), presenting real-time solar panel performance in an immersive and intuitive manner. The visualization consists of physically modeled solar panels rendered in a WebGL-powered environment, complete with realistic lighting, and a circular green ground plane under a sky-blue background to simulate an outdoor solar installation.

The system operates in two distinct visualization modes, each tailored to specific analytical objectives:

- **Replay Mode:** Displays a single solar panel whose surface color dynamically changes based on historical AC power output. The color mapping uses a gradient from red (low power) to green (high power), with blue indicating no data or loading states. Users can control playback speed (1x–20x), jump to specific timestamps, and observe the panel's performance evolution over time.
- **Prediction Mode:** Presents dual side-by-side panels labeled "ACTUAL" and "PREDICTED," enabling direct visual comparison between LSTM-forecasted values and ground truth data. Each panel independently reflects its respective power output through color coding, allowing immediate identification of prediction accuracy. Navigation controls enable stepping through prediction timestamps sequentially or jumping to specific dates.

The user interface is organized into three primary components: (1) a control panel on the left for plant/inverter selection, temporal navigation, and playback controls; (2) a heads-up display (HUD) on the right showing quantitative metrics including timestamp, AC power (kW), module temperature (°C), ambient temperature (°C), and irradiation levels; and (3) interactive 3D viewport supporting mouse-based camera orbiting for multi-angle inspection. All timestamps are handled in UTC to ensure consistency across the entire system stack.

8.2 System Outputs

Beyond the primary 3D visualization, the Solar Power Digital Twin produces several critical outputs that collectively enable comprehensive analysis and validation:

1. **InfluxDB Time-Series Database:** Contains processed historical data for both Plant 1 and Plant 2, with measurements at 15-minute intervals spanning May 15 to June 17, 2020. The database stores AC power, DC power, ambient temperature, module temperature, and irradiation measurements indexed by timestamp and source key (inverter ID). This structured storage enables efficient querying for replay and prediction validation.
2. **RESTful API Endpoints:** The FastAPI backend exposes three primary endpoint categories:
   - `/replay` and `/replay_range`: Return historical data for specific timestamps or time ranges, supporting the replay visualization mode.
   - `/predict/timestamps`: Provides available prediction timestamps for a given plant-inverter combination, enabling the frontend to build navigation controls.
   - `/predict/generate`: Executes LSTM inference on demand, returning predicted AC power values alongside metadata (model version, sequence length, confidence metrics).
3. **Trained LSTM Models:** Two Keras models for plant 1 and plant 2 trained on 24-timestep sequences with cyclical time encoding (hour/day sine-cosine components) and normalized environmental features. These models serve as the predictive engine, with scalers stored for consistent preprocessing.

4. **Interactive Performance Metrics:** Real-time visual feedback through color-coded panels (gradient mapping power output to hue) and synchronized numerical displays. In prediction mode, the dual-panel layout inherently outputs a visual error signal, as color discrepancies between actual and predicted panels immediately reveal forecasting accuracy.

8.3 Contribution to System Purpose

The chosen visualization strategy directly addresses the core objectives of the Solar Power Digital Twin by transforming abstract numerical data into actionable visual intelligence:

Data-Driven Simulation Environment: The 3D representation with dynamic color mapping creates an intuitive mental model of solar plant behavior. Operators can observe diurnal patterns (color transitions throughout the day), weather impact (sudden color shifts during cloud cover), and seasonal trends (baseline color variations across the dataset), making complex time-series data immediately comprehensible.

Analytical Replay Capability: The replay mode's temporal navigation enables forensic analysis of historical events. Users can jump to specific timestamps (e.g., anomalous low-power periods) and observe the panel's state in context with environmental conditions. The adjustable playback speed (up to 20x) allows rapid scanning of extended periods while maintaining the ability to slow down for detailed investigation of transient phenomena.

Predictive Validation Framework: The dual-panel prediction mode serves as a visual hypothesis testing tool. By placing actual and predicted panels side-by-side with synchronized color scaling, forecasting errors become spatially evident rather than requiring mental arithmetic. This visual immediacy accelerates model validation, as engineers can quickly identify systematic biases (e.g., consistent over-prediction during morning hours) or random errors (scattered color mismatches).

Foundation for Proactive Systems: The modular architecture positions the digital twin for evolution toward predictive maintenance. The current visualization already highlights zero-power events (blue panels) that could indicate inverter failures. By extending the prediction horizon and integrating anomaly detection algorithms, the system could preemptively flag degrading performance (e.g., consistently lower-than-predicted output suggesting panel soiling or component wear), directly supporting sustainable energy management through early intervention.

Long-Term Efficiency Planning: The combination of historical replay and ML-based forecasting enables scenario analysis. Energy managers can replay past performance under specific weather patterns, then use predictions to estimate output under similar forecasted conditions. This dual temporal perspective (backward-looking validation and forward-looking projection) provides the quantitative foundation for capacity planning, grid integration scheduling, and economic optimization of solar assets.

9 System Architecture and Implementation

9.1 Architecture Design

The Solar Power Digital Twin follows a four-tier architecture pattern, separating concerns into distinct layers that communicate through well-defined interfaces. This modular design ensures scalability, maintainability, and independent evolution of components:

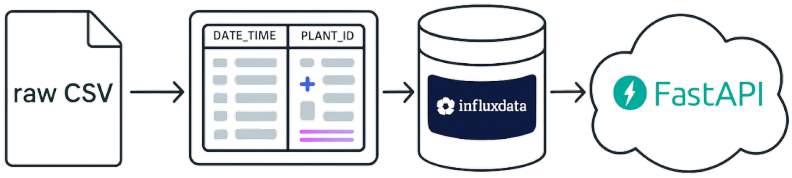


Figure 10: System architecture pipeline

1. **Data Storage Layer (InfluxDB):** A time-series database optimized for temporal queries, storing processed sensor data with microsecond-precision timestamps. The database contains two measurement tables (`plant1` and `plant2`), each indexed by timestamp and source key (inverter identifier). The schema includes fields for DC power, AC power, ambient temperature, module temperature, and irradiation, all stored as double-precision floats. InfluxDB's column-oriented storage and built-in time-based indexing enable sub-millisecond retrieval of specific timestamps and efficient range scans for historical replay.

2. **API Layer (FastAPI):** A Python-based RESTful web service serving as the system's central nervous system. The API implements three functional routers:

   - `replay.py`: Handles historical data retrieval via `/replay` (single timestamp) and `/replay_range` (interval queries) endpoints. Queries are constructed dynamically based on plant number and translated to SQL for InfluxDB execution.
   - `predict.py`: Manages ML inference lifecycle, including model loading (lazy initialization with caching), data preprocessing (MinMaxScaler normalization and cyclical time encoding), LSTM prediction, and inverse transformation. Exposes `/predict/timestamps` for prediction availability and `/predict/generate` for on-demand forecasting.
   - `utils.py`: Provides auxiliary services such as health checks and metadata queries.

   The API layer includes CORS middleware configured to allow cross-origin requests from the frontend, essential for development and deployment flexibility.

3. **Machine Learning Layer (TensorFlow/Keras):** A LSTM model trained offline on historical data and loaded into memory during API initialization. Model expects input sequences of 24 timesteps (6 hours of 15-minute intervals) with 8 features: DC power, ambient temperature, module temperature, irradiation, and four cyclical time encodings ($\text{hour}_{\sin}$, $\text{hour}_{\cos}$, $\text{day}_{\sin}$, $\text{day}_{\cos}$). The sequence-to-one architecture outputs a single AC power prediction. Models are compiled without optimization to reduce loading time (inference-only deployment), and predictions are cached during a session to avoid redundant computation.

4. **Visualization Layer (Three.js):** A JavaScript ES6 module-based frontend implementing the 3D scene, camera controls, and user interface. The visualization maintains a persistent WebSocket-like architecture using periodic HTTP polling (every 3 seconds divided by speed multiplier for replay mode). State management is event-driven, with mode switches, timestamp updates, and user interactions triggering corresponding API calls. The rendering loop runs at 60 FPS via `requestAnimationFrame`, with panel colors updated synchronously when new data arrives.

9.2 Modules and Subsystems

9.2.1 Data Preparation Subsystem (`data_prep.py`)

Preprocessing pipeline responsible for merging weather sensor data with generation data, synchronizing timestamps, storing data in InfluxDB3, and exporting clean CSVs (`plant1_final.csv`, `plant2_final.csv`).

9.2.2 Replay Subsystem (`routers/replay.py`)

Implements temporal navigation logic for historical data retrieval. Accepts plant number and timestamp(s) as query parameters, constructs parameterized SQL queries to prevent injection attacks, executes queries against InfluxDB, and returns JSON-serialized Pandas DataFrames. Error handling includes validation of plant numbers (1–2 only), timestamp format parsing (ISO 8601), and empty result detection.

9.2.3 Prediction Subsystem (`routers/predict.py`)

Coordinates the ML inference pipeline:

1. **Model Management:** Lazy loads Keras models on first request and caches them in a global dictionary (`loaded_models`). Separate models exist for Plant 1 (V2) and Plant 2, selected dynamically based on the plant parameter.

2. **Scaler Management:** Fits MinMaxScalers on the full dataset per plant (one-time operation cached in `loaded_scalers`). Scalers normalize features to [0,1] range before LSTM input and denormalize predictions back to physical units (kW).

3. **Sequence Construction:** Retrieves the 24 timesteps immediately preceding the prediction timestamp from cached plant data, applies feature engineering (cyclical encoding), normalizes features, and reshapes to LSTM input format $(1, 24, 8)$.

4. **Inference and Post-processing:** Passes the sequence through the model, denormalizes the output, and clamps negative predictions to zero (physical constraint: AC power ≥ 0).

5. **Timestamp Validation:** Cross-references requested timestamps against precomputed JSON files to ensure predictions are only generated for valid test set samples, preventing data leakage.

9.2.4 Visualization Subsystem (`simulation/main.js`)

Implements a finite state machine managing the application lifecycle:

- **IDLE:** Initial state after mode selection, awaiting user input (plant/inverter selection).
- **TIMESTAMP_SELECTED:** User has chosen a temporal target (via date picker or navigation buttons).
- **LOADING:** API request in flight; panels set to blue loading state.
- **FETCHING_ACTUAL_DATA:** (Prediction mode only) Retrieving ground truth for comparison.
- **GENERATING_PREDICTION:** (Prediction mode only) Executing LSTM inference via `/predict/generate`.
- **COMPARING_RESULTS:** (Prediction mode only) Computing color mappings for both panels.
- **RENDERED:** Final state with panels colored and HUD updated, awaiting next user action or automatic timer trigger (replay mode).

State transitions are triggered by events (button clicks, timer expirations, API responses), ensuring deterministic behavior and facilitating debugging.

9.2.5 UI Control Subsystem (`simulation/index.html`)

Provides HTML structure for the control panel, HUD, and modal dialogs. CSS-in-HTML styling defines responsive layouts, button states (enabled/disabled), and visibility toggling (replay vs. prediction mode elements). Dropdown menus are dynamically populated with valid inverter IDs (1–22) and time options (hours 00–23, minutes 00/15/30/45 to match data granularity).

9.3 Libraries and Technologies

- **Backend:**
  - `FastAPI 0.110.0`: Modern async web framework with automatic OpenAPI documentation.
  - `Uvicorn 0.29.0`: ASGI server for running FastAPI applications.
  - `influxdb3-python`: Official InfluxDB 3.0 client library for Python.
  - `pandas 2.2.2`: Data manipulation and CSV processing.
  - `TensorFlow 2.16.2`: Deep learning framework for LSTM model loading and inference.
  - `scikit-learn 1.4.2`: MinMaxScaler for feature normalization.
  - `NumPy` $\geq$ `1.24.0`: Numerical computing primitives (array operations, trigonometric functions for cyclical encoding).

- **Frontend:**
  - `Three.js 0.160.0`: WebGL-based 3D graphics library (loaded via CDN). Provides scene management, mesh geometry (BoxGeometry, CylinderGeometry, CircleGeometry), materials (MeshStandardMaterial), lighting (AmbientLight, DirectionalLight), and rendering pipeline.
  - `Vanilla JavaScript ES6`: Module-based architecture with async/await for API calls, Fetch API for HTTP requests, event-driven UI interactions.
  - `HTML5 + CSS3`: Semantic markup with flexbox layouts, absolute positioning for HUD overlays, RGBA transparency for glass-morphic panels.
- **Data Storage:**
  - `pandas 2.2.2`: Data manipulation and CSV processing.
  - `InfluxDB 3.0`: Time-series database with SQL query interface, optimized for temporal aggregations and range scans. Configured with file-based object storage for simplified deployment.
  - `influxdb3-python`: Official InfluxDB 3.0 client library for Python.
- **Development Tools:**
  - `npm + Vite`: Frontend development server with hot module reloading (`npm run dev`).
  - `Python venv`: Virtual environment management for dependency isolation.
  - `Git`: Version control with GitHub remote repository.

10 Reflection and Discussion

During the integration of the trained plant-level model into the digital twin frontend, we identified inconsistencies caused by missing inverter records in the database. In several time intervals, certain inverters had no corresponding entries, meaning that the aggregated plant-level data retrieved by the frontend represented only a subset of the total inverters. These were not missing values within existing rows but entirely absent records, which led to incomplete plant power values and, consequently, unreliable model predictions. Since the aggregated inputs no longer reflected the true state of the plant, the results produced by the model were not physically meaningful. For this reason, the plant-based model was ultimately discarded, and the focus shifted toward an alternative implementation less sensitive to missing inverter data.

Issues were also encountered when integrating different components with each other. Originally we intended on using Godot game engine for the visualization part of the project, however Godot and fastapi use different versions of the http protocol which made them incompatible with each other, causing us to shift to a different frontend framework. We also intended to containerize the whole project with docker, however we realized to late into the project that there is not readily available docker image for the database framework that we utilized therefore we had to make a script to simplify running the project on multiple devices, while

checking for the correct dependencies to be installed and guiding the user along the process in case something is missing.

11 Individual Contributions

The work among the team members was divided as follows:

- **Jovana Vukmirovic**: Dataset management, preprocessing, and preparation for model training.
- **Luca Giandomenico**: Machine Learning implementation, including model architecture design and training.
- **Gabriel Gil Pichelbauer**: Visualization implementation, system integration, and orchestration of the digital twin components.
- **Yunxuan Tang**: Backend implementation and historical replay system design.

12 Conclusion

This project successfully implemented a functional Digital Twin for solar power plants, combining time-series data processing, real-time replay, and machine learning-based prediction. The system provides both retrospective and prospective insights into solar power generation, offering a scalable foundation for future extension such as anomaly detection or predictive maintenance. Future improvements could include integration with live sensor streams and automated drift monitoring to further enhance fidelity.