DIGITAL TWINS FOR PHOTOVOLTAIC PLANTS PERFORMANCE MONITORING

José E. C. Castilho[1], Gabriel P. Viterbo[1], Gustavo Fraidenraich[1], Tárcio A. S. Barros[1] and Denis G. Fantinato[1]

[1] Universidade Estadual de Campinas, Campinas (Brazil)

Abstract

The real-time monitoring of photovoltaic (PV) systems is essential for ensuring their operational health and supporting quick decision-making, aiming to mitigate losses caused by electrical faults or sensor misreadings. Digital Twins can be an effective solution to enhance the efficiency of PV system monitoring. This article provides a comprehensive overview of a cost-efficient implementation of a Digital Twin platform for photovoltaic systems using open-source technologies. The proposed architecture adopts a microservices approach, ensuring scalability and flexibility in managing large datasets across multiple plants. It integrates real-time data acquisition, a Python-based simulation core, anomaly detection, and customizable dashboards. A web interface allows users to build customized model chains, powered by a topological sorting algorithm that organizes the graph of models according to their dependencies. The system also provides tools for customizable data input and dashboard generation in Grafana, offering a user-friendly environment for model configuration and monitoring. The platform design emphasizes interoperability and modularity, allowing seamless integration with existing supervisory systems and data sources. The proposed architecture meets the essential performance and reliability requirements of modern solar operations, supporting its potential adoption in commercial applications and contributing to more sustainable energy practices.

Keywords: digital twins, photovoltaic systems, real-time monitoring, data-driven modeling, model chain, dashboards, container architecture

1. Introduction

The demand for electric power has been rising all over the world. According to the International Renewable Energy Agency, the total amount of electricity generation increased by approximately 2.5% per year from 2012 to 2023 (IRENA, 2025). Moreover, the demand for cleaner and renewable sources of energy has been growing even more significantly. In 2023, renewable generation grew by 5.6% compared to 2022, while non-renewable generation increased by only 1.2%. Solar energy, specifically, is the renewable energy source with the fastest growth rate in recent years, with a year-on-year increase of 25.2% in generation.

This highlights the urgency of developing tools to enhance the operation of solar power systems. Considering the large physical scale of photovoltaic plants, and the fact that most of them still rely on manual or visual inspection, improvements on real-time monitoring systems are very important. Liu et al. (2024) reported a 20% reduction in the inspection tour time and a 15% faster fault response time with the implementation of a real-time monitoring platform based on digital twins.

The first relevant use of the concept of twins goes back to the 70's with the National Aeronautics and Space Administration (NASA) Apollo mission, when a physical copy of the vehicle stayed on Earth and was used to simulate and predict the conditions of the flying one (Liu et al., 2021). The concept now evolved from the physical twin to the digital twin, and it is considered by the agency a paradigm shift that can bring unprecedented levels of safety and reliability. They defined a Digital Twin as a virtual representation of a physical system, combining multiphysics and multiscale simulations with real-time data from sensors and operational history to replicate its real-world counterpart (Glaessgen and Stargel, 2012).

A review on digital twin applications highlights their extensive use in design, manufacturing, service, and retirement phases of a product (Liu et al., 2021). Digital twins allow iterative optimization, provide data integrity, and support virtual evaluation and verification of products (Tao et al., 2019; Schleich et al., 2017). During the manufacturing phase, digital twins allow real-time monitoring, production control, performance prediction, human-robot collaboration, process evaluation, asset management, and dynamic production planning (Rosen et al., 2015; Park et al., 2019; Zhuang et al., 2018). In the service phase, they are applied to

Selection and/or peer review under responsibility of Scientific Committee
10.18086/swc.2025.05.03 Available at http://proceedings.ises.org

predictive maintenance, fault detection and diagnosis, state monitoring, performance prediction, and virtual testing of products (Tuegel et al., 2011; Xu et al., 2019; Xie et al., 2019).

The service phase is the one that best fits our purpose, as it focuses on real-time monitoring after implementation, assisting in fault diagnosis and performance evaluation. However, the proposed architecture also adapts well to the design phase. It is completely possible to create a digital twin of a plant still under design, as long as sufficient data is provided to support accurate simulations. A sensor station could easily supply the twin with live data over the course of a year before the physical installation of the plant. The user could also employ synthetically generated data to evaluate performance under specific conditions.

Mollineda and García (2024) described a Digital Twin of a photovoltaic system as a digital representation that integrates component information, system geometry, and environmental factors affecting performance. In their work, the physical model represents the system components, while mathematical models describe the behavior of elements such as solar panels and inverters, enabling simulations under different operating conditions.

However, most developments of digital twins for photovoltaic systems tend to focus either on specific models, artificial intelligence agents trained for particular plants, or the integration of new hardware (Walters et al., 2023; Mollineda and García, 2024). The architecture proposed in this work focuses on being a flexible digital-twin that can be easily implemented in any photovoltaic plant, regardless of its size or configuration. It provides multiple layers of flexibility: multiple input formats supported, and multiple combinations of models (customized model chains). It is able to handle multiple data entries while automatically generating dashboards for data visualization with an embedded alerting system, requiring no coding from the user. Both the alerting system and the model chain were designed to allow maximum flexibility when integrating new algorithms, ensuring that the only technical challenge is the algorithm itself, not the integration with the existing system.

2. Digital Twin Platform Architecture

A general overview of the proposed architecture is shown in Figure 1. The system was designed to be user-friendly, requiring no coding skills from the user, since all the actions needed to run the Digital Twin can be performed on a web interface. The components of this web interface are delimited on the image by the “Front-End” box.

A digital twin is a virtual copy of a physical system and, in order to create it, a combination of physical models, powered by real-time data, is used to represent the system, in this case, a photovoltaic plant. The number of possible model combinations can grow exponentially as more models are included. The process of hard-coding these combinations, making different models interact, can be hard, time-consuming, and may require substantial coding skills.

To address this problem, the system was designed to make it easy to integrate new models into the model-chain, as long as they are created following the correct pattern. The algorithm starts from the final metric of interest and back-propagates, iteratively expanding the available models that can be used to generate that metric. This creates a tree of models, which is treated as a graph, where all “leaf” nodes require no additional models or where the user already has measured data and does not want to model them. These leaf nodes are considered required inputs for the digital twin. Notice that, if the user has the measured data for a specific node which also has a model available, he can decide whether to use the measured data or to model it. Afterwards, depending on the user's choice, the real data can be imputed and compared with modeled one.

Finally, the user is requested to map each of these required inputs to the corresponding column names in the input data file. In case there are any other metrics in these files that the user would like to insert on the database but that are not required for the simulation, they can be included as additional inputs. In the last step, on the *Simulate* screen, the user must select the files that will provide the required data, ensuring that the column names match the mapped inputs.

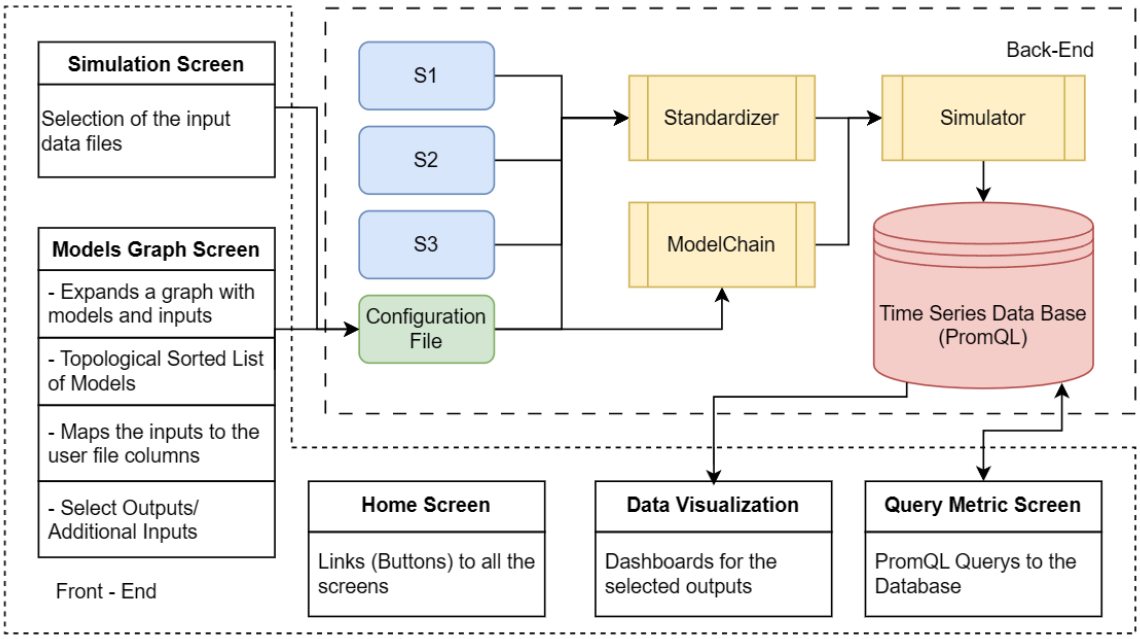

Figure 1: Simplified Digital Twin Architeture

Once the data files and the JSON configuration file are set, and the user clicks the *Simulate* button, the simulation starts. The data is processed by the *Data Loader* block. This block contains two modules: a *parser* and a *standardizer*. The parser is responsible for reading the data and merging the multiple given files, which can be provided in any of the most common formats (.json, .xlsx, .csv). The records from these merged files are aligned by the timestamp and the timezone information (specified in the configuration file) is applied. The output of the parser is a Pandas DataFrame, which is then sent to the *standardizer*, responsible for filtering the data, keeping only the required columns, already replacing the columns names with the names defined on the column mapping step.

This step is essential for a better user experience, avoiding the need for manual intervention or heavy user-side preprocessing of data, since the software is designed to automatically handle most common file structures and patterns.

After this process is complete, the customized Model Chain is internally built. This process relies on a list of parameters or fields, specified for every model. As a simplification, these fields can be seen as global variables in the scope of the simulation, representing all the inputs, intermediate parameters, and outputs of the simulation. The inputs are initially assigned to their respective fields, then, each model function is called in the correct order, assigning its outputs to the global fields in sequence. Once all models have been executed, all the desired outputs will be available.

Those outputs, combined with the inputs the user selected to send to the database during the setup step, are then sent to the *Prometheus* database, which is already connected to *Grafana*, an open-source software for data visualization. The dashboards are live-updated as new data is injected into the database. Once the records are in the database, the user can also run queries directly from the front-end. This functionality was designed so that users do not need to store their data in another database. In the subsections below, the main components, technologies, and procedures are described in more detail.

2.1 Front-End Technologies Overview

The front-end relies on Jinja2, a Python-based engine that allows the creation of dynamic HTML content. This engine makes it possible to render web pages without needing to manually generate HTML for each scenario, even if it is slightly different from the template. HTML (Hypertext Markup Language) defines the structure of the pages (forms, buttons, navigation elements) and it is visually enhanced by CSS (Cascading Style Sheets). CSS is responsible for specifying colors, fonts, spacing, and responsive layouts.

JavaScript is used to enhance interactivity and provide client-side logic. This includes validations, dynamic

content update, event handling, and communication with the back-end through HTTP requests. The requests used for this application are GET and POST, respectively responsible for fetching resources and for sending data. This communication protocol allows the front-end to request computations, retrieve results, and update visualizations dynamically.

The back-end is implemented with FastAPI, a Python framework designed to expose functionality to web clients. It acts as an interface between the front-end and Simulation Core, handling requests, serving templates, receiving uploaded files, and triggering the simulations. It provides HTTP endpoints that allow the front-end to interact with the simulation core while ensuring proper request validation and efficient execution.

2.2 Model Chain Customization

The Model Chain is dynamically built based on the final output the user selects and the available models. Each model, in its definition, lists the required inputs and produced outputs, allowing the system to evaluate the dependencies between models. Starting from the desired output metrics, the system back-propagates, expanding for each level all the available models. Only the models selected by the user are further expanded, until the process reaches metrics that the user chooses to input or that do not have any available models to generate it, and therefore are mandatory inputs.

This process, by its nature, generates a directed acyclic graph (DAG). Each node represents a model and each edge represents a dependency (from a model producing a field to a model requiring it). The acyclic nature of the graph ensures that all dependencies are represented and that there are no circular dependencies. A circular dependency happens when Model A, which produces parameter *a*, needs a parameter *b* produced by Model B, but Model B also depends on parameter *a*. These circular dependencies cannot be resolved, and the execution would fail.

Finally, to determine the correct execution order, a topological sort is performed on the graph. This algorithm sorts the models in a sequence that ensures that each model is executed only after all its required inputs have been computed and all dependencies are satisfied. The algorithm can be simplified in the following steps:

1) Identify all nodes with no incoming edges (these are models whose inputs are already available).

2) Add these nodes to a sorted list and remove them from the graph

3) For each node removed, eliminate from the graph the edges corresponding to their outputs

4) Repeats steps 1-3 until there are no nodes remaining

5) Optionally, rebuild the graph based on the dependencies and the sorted list for visualization.

A visual demonstration of the result of a topological sort is shown in Figure 2.

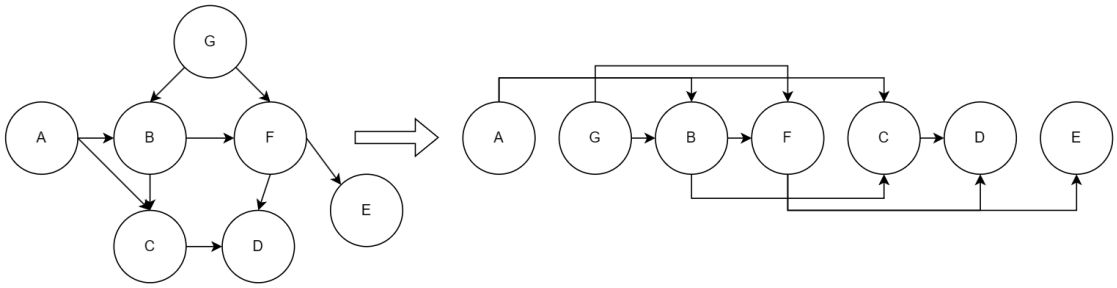

Figure 2: Visual Result of Topological Sort Algorithm

This process of selecting models and constructing the model chain is available on the “Models Graph” screen, shown in Figure 3. It is important to note that this interface was developed with a focus on the national market. Therefore, the screenshots are displayed in the authors’ native language.

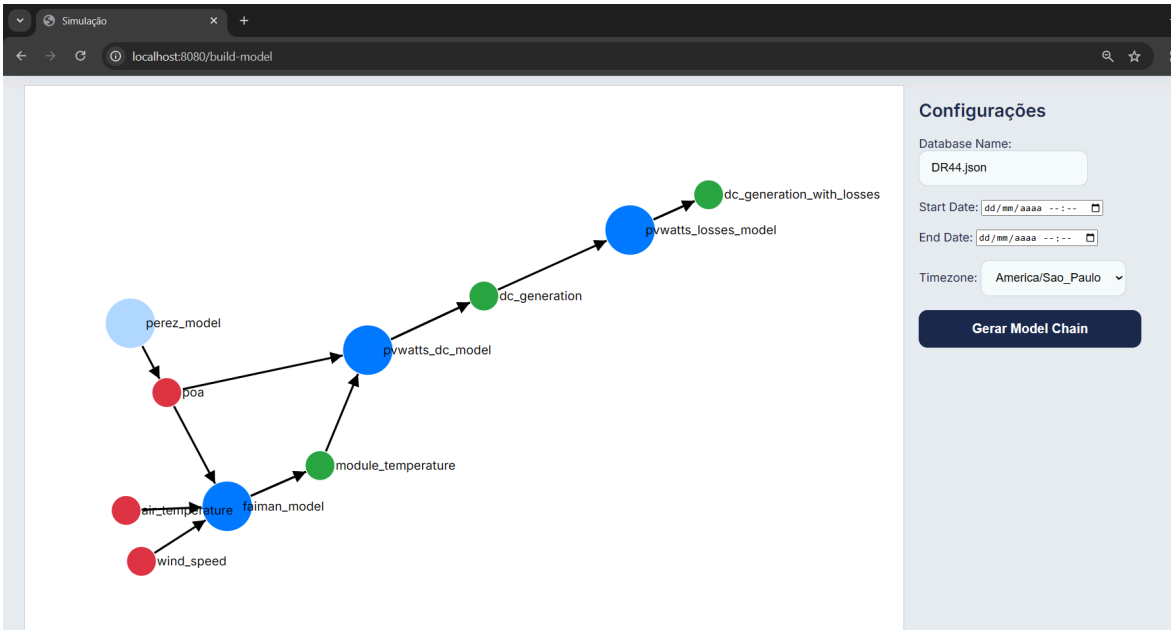

Figure 3: Custom Model Chain Graph

In the example above, the user starts from *Direct Current Generation With Losses*. The first step is to select the losses model, in this case, *PVWatts Losses* (Dobos, 2014).. Next, the user must select the *Direct Current Generation* model, for example *PVWatts* (Dobos, 2014), which requires two input metrics: *Plane of Array Irradiance* and *Module Temperature*. The *Module Temperature* can be modeled using the *Faiman* model (Faiman, 2008). The *Plane of Array Irradiance (POA)* can be obtained from several models, such as the *Perez* model (Perez et al., 1990). The *Perez* model requires multiple parameters, which will be expanded if available models exist to compute them, or will be treated as leaf nodes therefore, required user inputs.

The mapping of these inputs to the files columns can be seen in Figure 4. In this screen, the user can also include metrics that are not used in the simulation and select which outputs should be monitored on the dashboards. The output of this step is a JSON configuration file. An example of this configuration file is included in the Appendix.

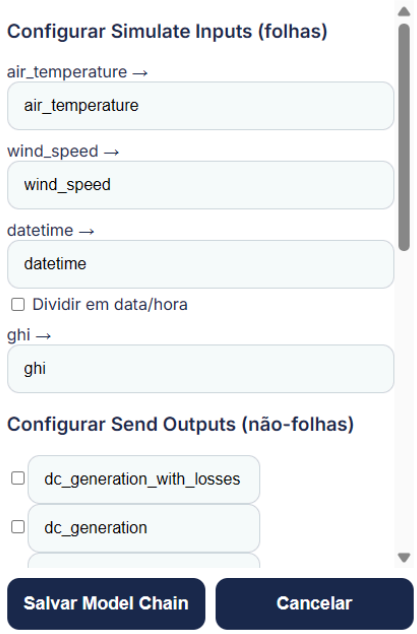

Figure 4: Mapping Inputs to Columns Names

2.3 Field Management and Model Execution Flow

The interface between models in the simulator is internally managed through the *Field Registry*. As mentioned above, each model declares its required input and output fields. These fields can be globally registered so that their values can be shared across all models in the chain. During the simulation, the Model Chain assigns the inputs (the columns mapped on the *"simulate_inputs"* section on the .json configuration file) to the corresponding fields within the Field Registry. Then, the method *compute()* of each method is called following the order determined by the topological sort. This method reads the necessary input values from the registry, performs its calculations, and updates the corresponding output fields.

As a result, after the execution of a model is completed, its outputs become available to any subsequent model that depends on them. This ensures the correct data flow and dependency resolution. Finally, after all models have been executed, the system retrieves the desired metrics from the Field Registry to generate the outputs or send them to the database.

2.4 Data Publishing Pipeline

The process described in Section 2.4 is iteratively applied to each entry in the input DataFrame. Since each entry is indexed by a unique timestamp, the output of the simulator will be a time series containing the desired metrics, which represent the temporal evolution of the modeled variables.

In addition to batch simulations (historical back-filling), the system also supports real-time data streaming, which is a core requirement of any Digital Twin. Once the model chain is configured through the user interface, external data sources can continuously send requests to the same HTTP endpoint used by the front end. Each message received is processed through the same pipeline described earlier. This unified communication layer allows the simulator to operate in both historical and real-time modes.

A logging mechanism is integrated into the system to capture errors, such as corrupted or missing data, without interrupting the simulation flow. This ensures that the simulation can proceed while keeping track of errors for later debugging and quality control.

Once the simulation is computed, both the inputs and outputs selected are formatted using the Prometheus format, where each record is represented as a labeled time series with a corresponding timestamp. These metrics are sent via HTTP POST requests to the VictoriaMetrics server, an open-source Time Series Database (TSDB) optimized for high-performance ingestion and querying of large volumes of real-time data.

Internally, VictoriaMetrics organizes data using a columnar storage engine and a time-partitioned data structure, which reduces storage cost by minimizing the additional storage required for metadata and indexing, and improves query efficiency. Prometheus Query Language (PromQL) is integrated into this database, and provides a wide set of aggregation and filtering operations, enabling users to analyze temporal behavior, compute statistical summaries, and correlate variables across different time scales. The database exposes these features through HTTP API endpoints, making it fully compatible with the monitoring and visualization tools. VictoriaMetrics also supports advanced features such as back-filling and data retention policies, which make it an optimal solution for the continuous monitoring and historical analysis required in photovoltaic systems.

Data visualization is handled by Grafana, which connects directly to the VictoriaMetrics database. Grafana provides a flexible and interactive dashboard environment, enabling plant operators and researchers to visualize all the metrics desired. Grafana supports PromQL, enabling users to build dashboards and alerts based on the same query syntax. Furthermore, it integrates with Victoria Metrics alerting tools, allowing the automatic generation of alarms in case of performance degradation or detected anomalies.

This pipeline enables real-time monitoring, comparing simulated and measured data, supports historical analyses, and forms the basis for future applications in predictive maintenance and performance optimization of photovoltaic plants.

2.5 New Models Inclusion Process

The architecture is designed to simplify the inclusion of new models. To add a model, the developer must extend the base class *Models* and define the following information:

- A unique identifier
- Required input fields (parameters)
- Output fields produced (results)
- Compute method (performs the model's calculations)

The main consideration at this stage is ensuring compatibility with the field names already used by other models. Once implemented and registered in the system, the new model is automatically recognized and made available in the user interface, allowing users to include it in a custom model chain.

2.6 Alerts Management

The Alerts System runs after the simulation, continuously evaluating the data stored in the Time Series Database according to user-defined rules. Each alert rule defines an operation type, a threshold, and the metrics to be compared (if applicable). As a proof of concept, two operation types are currently supported. The absolute difference operation compares the instantaneous values of two variables, triggering an alert when the difference exceeds the defined threshold. The moving average difference operation compares the one-hour moving averages of two metrics, computed using PromQL functions.

When an alert is triggered, a record is created containing metadata such as the dataset name, timestamp, operation type, threshold, variable values, and the computed difference. Alerts are stored as JSON files in a dedicated directory, where results are incrementally appended to preserve historical information. The platform also provides a web interface for alert configuration and visualization.

Although the current version supports only two-variable comparisons and a limited set of operations, the modular architecture of the Alert Manager enables future extensions. Possible developments include the implementation of more complex conditions and anomaly detection algorithms, as well as integration with external services for notification purposes. Moreover, Grafana and VictoriaMetrics natively support alert management, allowing alerts to be displayed on dashboard side panels. Although this feature is not yet integrated, the microservices architecture makes its addition straightforward.

3. Results and Discussion

To demonstrate the system's functionality, a simulation was performed using operational data from a photovoltaic plant located in São Paulo, Brazil. The plant consists of eight inverters, each connected to 11,340 modules, with a total installed module capacity of 3.74 MWp per inverter. The dataset includes measured plane of array (POA) irradiance, global horizontal irradiance (GHI), ambient and module temperature, wind speed, and DC generation, among other environmental variables. For this proof of concept, the data from two inverters and the environmental variables mentioned above will be used.

The dataset covers a full year of operation, sampled at 15-minute intervals, totaling 35,136 records. Each record corresponds to a single timestamp and contains multiple metrics to be computed and stored. The average simulation latency, defined as the time required to process one record, perform the simulation and make its metrics available for database insertion, was approximately 45 milliseconds per record. Each record generated seven metrics, with an average database write latency of 1 millisecond per metric. Consequently, the total processing time per record was approximately 52 milliseconds, resulting in a throughput of 19.2 records per second. In total, 245,952 metrics were inserted into the Time Series Database, requiring 1,850 seconds (approximately 30 minutes) to complete the full simulation. The simulations were executed on a personal workstation equipped with an IntelCore i7 10th generation CPU and 16 GB of RAM, running Windows 11 with Windows Subsystem for Linux (WSL). Since Windows does not fully expose all physical resources to WSL, the performance achieved could be higher on a dedicated Linux server.

It is important to note that the focus of this evaluation is on the system's operational performance, such as latency and throughput, rather than on simulation accuracy metrics. The optimal model chain depends on the specific plant and user objectives. For this proof of concept, it is sufficient to observe that the simulated metrics qualitatively follow the behavior of the real data, indicating that the models are correctly implemented, even if they do not represent the optimal combination for this plant. Figure 5 shows a 15-day

dashboard overview, and Figure 6 presents a detailed comparison for a single day, illustrating the model chain's temporal behavior.

Figure 5: 15-Day Dashboard Overview

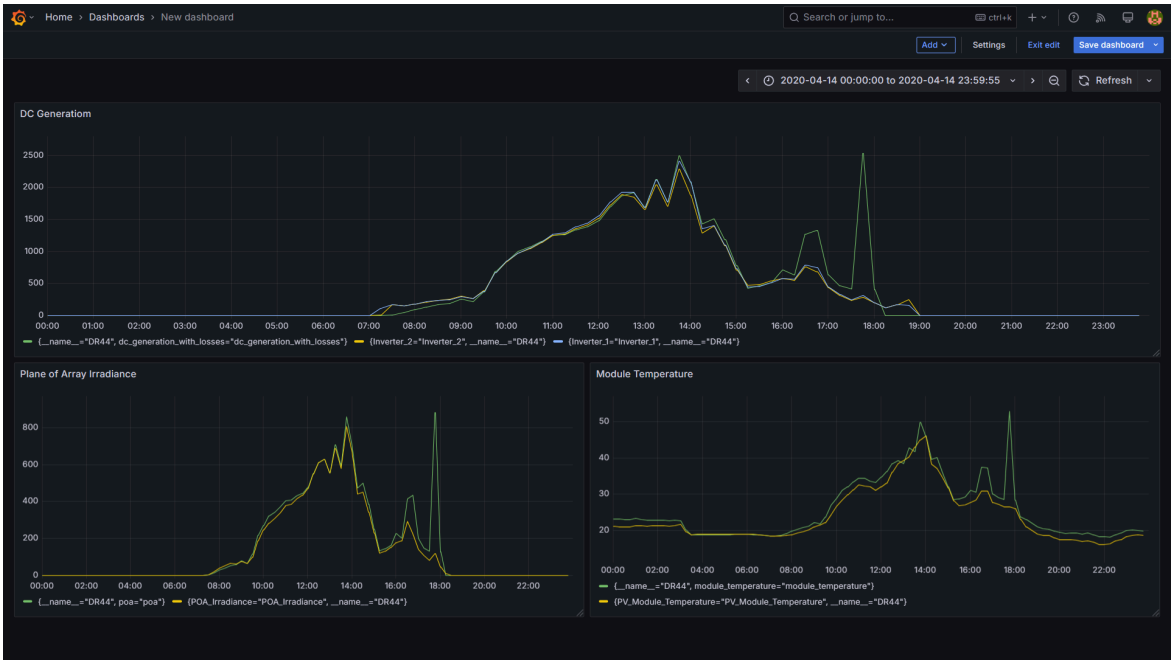

Figure 6: Detailed Dashboard For One Day of Operation

Figure 6 also illustrates a potential alert logic that could be implemented in future versions of the system. As shown in the DC generation plot, the simulated power output (green line) is higher than the measured production between 4 PM and 6 PM, which could trigger an alert indicating possible underperformance. However, when analyzing the POA and module temperature plots, which are both modeled from the solarimetric station data (GHI, wind speed, and ambient temperature), it appears that the sunset may have affected the PV array earlier than the solarimetric station. This interpretation is supported by the fact that the measured generation follows the behavior of the sensors installed in the array, rather than those at the solarimetric station. Therefore, in this scenario, the most appropriate action was not triggering the alert.

4. Conclusion and Future Work

The primary objective of this work was not to provide an optimal model chain for every photovoltaic plant in terms of simulation accuracy, since each plant may perform differently even when using the same model chain. Instead, the objective is to provide a flexible platform that allows users to test, calibrate and define their own simulation pipeline. Operators can configure model chains according to their operational objectives. A user may choose to configure a more conservative one, prioritizing robustness and anomaly detection, or a more optimistic one that focuses on predictive accuracy and system efficiency. Users can easily modify or replace components of the model chain as new models become available, or as the existing ones are refined, or even if the physical evolution of the plant (i.e., its natural degradation) is not being well captured by the current model chain.

This flexibility allows the digital twin to evolve continuously, reflecting changes in plant behavior or measurement infrastructure. The platform is designed to be scalable, supporting multiple metrics and to be able to monitor multiple plants. It can handle multiple datasets in multiple formats, and it is possible to define multiple model-chains for different periods.

Additionally, it is important to emphasize that the 30 minutes back-filling operation described in the previous section is not a time-critical operation, as it is only executed when importing historical data during the initial configuration of a plant, as an optional step. The average processing time per record is well below 1 second. Considering that most photovoltaic plants typically record measurements every 5 to 15 minutes, and that minute-to-minute monitoring occurs only in high-end plants, the observed processing times are well below typical monitoring intervals, and the computational cost can be considered negligible for practical operation.

Overall, the proposed system provides a solid foundation for real-time performance monitoring, historical analysis, and alert management, providing the flexibility to dynamically adjust the model chain without requiring additional coding. However, there is ample room for further improvements. Increasing user-friendliness and customization remains a key area. While the platform already allows dynamic configuration of the model chain, more intuitive interfaces could be incorporated, as well as guided suggestions and additional controls to facilitate operation by non-expert users. The alert system, in particular, can be significantly enhanced, exploring the integration of more advanced algorithms, anomaly detection methods, and AI-enhanced alert logic to provide more robust and informative feedback to operators.

It is worth noting that the software evolution is a continuous process: the most successful software platforms started small, improved over time and are constantly under review and refinement. This project lays the foundation for ongoing development, aiming to provide a versatile, extensible, and continuously improving digital twin platform for photovoltaic performance monitoring.

5. Acknowledgments

Castilho, J. E. C. et al. thank TotalEnergies for the financial support. We extend our gratitude to all collaborators from Universidade Estadual de Campinas (UNICAMP) and to the School of Electrical and Computer Engineering (FEEC). We also thank Henrique Pacheco Manuel for contributions to software development. The work of G. Fraidenraich was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 302077/20227.

6. References

Dobos, A.P., 2014. PVWatts version 5 manual. National Renewable Energy Laboratory (NREL), Golden, CO.

Faiman, D., 2008. Assessing the outdoor operating temperature of photovoltaic modules. Prog. Photovolt: Res. Appl. 16(4), 307–315. https://doi.org/10.1002/pip.813

Glaessgen, E. & Stargel, D., 2012. The digital twin paradigm for future NASA and U.S. Air Force vehicles. In: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Honolulu, HI, USA, 23–26 April. American Institute of Aeronautics and Astronautics.

https://doi.org/10.2514/6.2012-1818

International Renewable Energy Agency (IRENA), 2025. Renewable Energy Statistics 2025. International Renewable Energy Agency, Abu Dhabi. Available at: https://www.irena.org/-/media/Files/IRENA/Agency/Publication/2025/Jul/IRENA_DAT_RE_Statistics_2025.pdf [Accessed 28 October 2025].

Liu, M., Fang, S., Dong, H. & Xu, C., 2021. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 58(Part B), pp.346–361. https://doi.org/10.1016/j.jmsy.2020.06.017

Liu, W., Wang, P., Yang, J., Yang, Y., Pei, L., 2024. Construction of Visual Monitoring Platform for Photovoltaic Power Generation System Based on Digital Twin. Proceedings of the 5th International Conference on Clean Energy and Electric Power Engineering (ICCEPE 2024), Yangzhou, China, 398–402. https://doi.org/10.1109/ICCEPE62686.2024.10931707

Mollineda, L.G.F., García, J.R.A., 2024. Digital twin of a photovoltaic system. *Journal of Engineering and Technology for Industrial Applications (ITEGAM-JETIA)*, v.10 n.47, pp.161–172. https://doi.org/10.5935/jetia.v10i47.1166

Park, K.T., Nam, Y.W., Lee, H.S., Im, S.J., Noh, S.D., Son, J.Y., *et al.*, 2019. Design and implementation of a digital twin application for a connected micro smart factory. *Int. J. Comput. Integr. Manuf.* 32, 596–614. https://doi.org/10.1080/0951192X.2019.1599439

Perez, R., Ineichen, P., Seals, R., Michalsky, J., Stewart, R., 1990. Modeling daylight availability and irradiance components from direct and global irradiance. Sol. Energy 44(5), 271–289.

Rosen, R., von Wichert, G., Lo, G., Bettenhausen, K.D., 2015. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine* 28, 567–572. https://doi.org/10.1016/j.ifacol.2015.06.141

Schleich, B., Anwer, N., Mathieu, L., Wartzack, S., 2017. Shaping the digital twin for design and production engineering. *CIRP Ann. Manuf. Technol.* 66, 141–144. https://doi.org/10.1016/j.cirp.2017.04.040

Tao, F., Sui, F., Liu, A., Qi, Q., Zhang, M., Song, B., et al., 2019. Digital twin-driven product design framework. *Int. J. Prod. Res.* 57, 3935–3953. https://doi.org/10.1080/00207543.2018.1443229

Tuegel, E.J., Ingraffea, A.R., Eason, T.G., Spottswood, S.M., 2011. Reengineering aircraft structural life prediction using a digital twin. *Int. J. Aerosp. Eng.* 2011. https://doi.org/10.1155/2011/154798

Walters, M., Yonce, J., Venayagamoorthy, G.K., 2023. Data-Driven Digital Twins for Power Estimations of a Solar Photovoltaic Plant. 2023 IEEE Symposium Series on Computational Intelligence (SSCI), Mexico City, Mexico, pp. 246–251. https://doi.org/10.1109/SSCI52147.2023.10371834

Xie, J., Wang, X., Yang, Z. & Hao, S., 2019. Virtual monitoring method for hydraulic supports based on digital twin theory. *Mining Technology: Transactions of the Institutions of Mining and Metallurgy*, 128, pp.77–87. https://doi.org/10.1080/25726668.2019.1569367

Xu, Y., Sun, Y., Liu, X. & Zheng, Y., 2019. A digital-twin-assisted fault diagnosis using deep transfer learning. *IEEE Access*, 7, pp.19990–19999. https://doi.org/10.1109/ACCESS.2018.2890566

Zhuang, C., Liu, J., Xiong, H., 2018. Digital twin-based smart production management and control framework for the complex product assembly shop-floor. *Int. J. Adv. Manuf. Technol.* 96, 1149–1163. https://doi.org/10.1007/s00170-018-1617-6

Appendix: JSON configuration file

This appendix contains an example of a JSON configuration file produced by the front-end.


```
{
 "database_name": "test.json",
 "records": [
   {
    "date": {
     "start": "",
     "end": ""
    },
    "timezone": "America/Sao_Paulo",
    "simulate_inputs": {
     "module_temperature": "TempModule",
     "datetime": "datetime",
     "ghi": "ghi"
    },
    "send_inputs": [],
    "send_outputs": {},
    "model chain": [
     "extra_radiation_model",
     "solar_position_model",
     "erbs_model",
     "airmass_model",
     "single_axis_tracker",
     "perez_model",
     "faiman_model",
     "pvwatts_dc_model",
     "pvwatts_losses_model"
    ]
   }
 ]
}
```