COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Programming an Online Dashboard to Generate Scenarios in the OpenMASTER Energy Model

Autor: Vicente Martín García-Marcos

Director: Pedro Linares

Co-Director: Manuel Pérez Bravo

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Programming an Online Dashboard to Generate Scenarios in the OpenMASTER Energy Model

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Vicente Martín Fecha: 28/06/2025

[illegible]

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por LINARES LLAMAS PEDRO - ***3450** el día 29/06/2025 con un certificado emitido por AC FNMT Usuarios

Fdo.: Pedro Linares Fecha:

Fdo.: Manuel Pérez Bravo Fecha:

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Programming an Online Dashboard to Generate Scenarios in the OpenMASTER Energy Model

Autor: Vicente Martín García-Marcos

Director: Pedro Linares

Co-Director: Manuel Pérez Bravo

Madrid

Agradecimientos

Quiero agradecer a mis padres por brindarme la oportunidad de estudiar lo que siempre quise. Sin ellos nunca habría llegado hasta donde estoy hoy.

A mis amigos de carrera Carlos, “Charlie”, Javi, “Del Eagle” y “Mike” por empujarme a ser mejor ingeniero de lo que jamás habría podido yo solo.

Y por último agradecer a Carlos, David y mi hermano Manuel por acompañarme a desconectar de vez en cuando y observar las cosas desde otra perspectiva.

Gracias a todos.

PROGRAMACIÓN DE UN PANEL DE VISUALIZACIÓN EN LÍNEA PARA GENERAR ESCENARIOS EN EL MODELO ENERGÉTICO OPENMASTER

Autor: Martín García-Marcos, Vicente
Supervisores: Linares, Pedro y Pérez Bravo, Manuel
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El panel de control renovado de openMASTER convierte la exploración de escenarios energéticos de largo plazo en un proceso interactivo y totalmente reproducible. En lugar de ejecutar y graficar cada caso de forma aislada, la herramienta precarga *n* escenarios, generados por la combinación sistemática de los parámetros con mayor influencia en las salidas del modelo energético. Mediante deslizadores discretos, rangos y filtros multiselección, cualquier conjunto de valores se puede invocar al instante; los callbacks reactivos de Dash localizan el directorio correspondiente y actualizan las figuras Plotly sin refrescar la página. La interfaz es modular y editable: widgets, métricas y tipos de gráfico pueden añadirse o reordenarse sin alterar la arquitectura, y el código Python sigue un diseño limpio que facilita extensiones. Un comparador colapsable permite fijar hasta tres configuraciones de referencia y confrontarlas lado a lado (diagramas de Sankey, curvas de coste o trayectorias de emisiones) con rotulado automático y leyendas consistentes. Todo ello se sirve desde una caché en memoria y puede escalarse horizontalmente con contenedores para albergar miles de escenarios si el espacio de diseño experimental crece.

En conjunto, el tablero pasa de ser un visor estático a una plataforma de análisis paramétrico, apta para investigadores, reguladores e industria que necesiten evaluar cómo las decisiones sobre capacidad, combustibles o tasas de descuento alteran la senda de descarbonización proyectada por openMASTER.

Palabras clave: openMASTER, modelado energético, escenarios, panel interactivo

1. Introducción

El modelo openMASTER, desarrollado en el Instituto de Investigación Tecnológica de la Universidad Pontificia Comillas, combina una formulación lineal bottom-up, rica en detalle tecnológico, utilizando el solver Gurobi para generar proyecciones de varias décadas para las inversiones en capacidad, operación, y trayectorias de emisiones, facilitando análisis acelerados y fundamentados para la descarbonización y la resiliencia climática [1]. Su panel de visualización de resultados original obligaba a los analistas a ejecutar cada escenario manualmente, exportar resultados a CSV y ensamblar gráficos estáticos en herramientas separadas, un flujo de trabajo fragmentado, lento y propenso a errores que dificultaba tanto la exploración de hipótesis como la participación en tiempo real de los interesados.

Para subsanar estas deficiencias, el proyecto desarrolla un panel completamente en Python que pre-computa una biblioteca de escenarios indexada por siete parámetros seleccionables a través de deslizadores. Aprovechando los callbacks reactivos de Dash y las capacidades dinámicas de Plotly, la interfaz actualiza los gráficos sin necesidad de recargar la página. Un panel lateral desplazable, secciones comparativas colapsables y rotulado automático simplifican la interacción y reducen la carga cognitiva, convirtiendo a openMASTER en una plataforma de soporte de decisiones plenamente interactiva.

2. Definición del proyecto

Este proyecto aborda la brecha creciente entre la fidelidad de los modelos energéticos de gran escala y la necesidad de visualización instantánea e interactiva. Mientras que herramientas comerciales como TIMES [2] y MESSAGE [3] quedan relegadas a lenguajes propietarios y licencias, alternativas de código abierto exigen flujos de trabajo fragmentados con MathProg, YAML y CSV. El panel nativo de openMASTER, un modelo energético open-source, seguía requiriendo ejecución manual de cada escenario, exportación y graficado externo, lo que convertía los estudios de comparación múltiple en procesos tediosos y proclives a errores.

La solución propone un panel integrado totalmente en Python que precarga 128 ejecuciones previamente calculadas, las indexa según siete parámetros clave y permite alternar escenarios con un solo clic mediante deslizadores. Con Dash y su renderizado acelerado por GPU, junto a una capa de caché en memoria, los gráficos se actualizan instantáneamente sin recargas. Los widgets interactivos (selectores de rango, anotaciones en línea y comparadores colapsables) habilitan profundizaciones en periodos, tecnologías o flujos de emisiones específicos sin abandonar la interfaz.

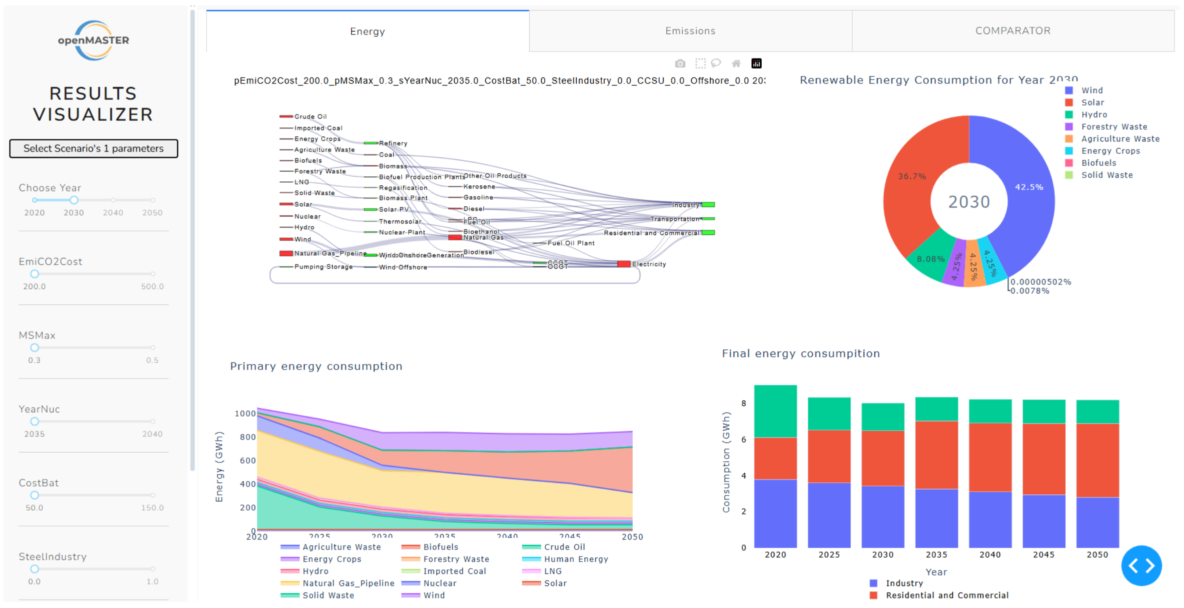
El desarrollo sigue seis fases iterativas centradas en el usuario: documentación del flujo existente; análisis de requerimientos en talleres; ingestión y indexado en memoria de escenarios; refactorización incremental del UI con dash-bootstrap-components; implementación de los callbacks reactivos; y validación exhaustiva mediante pruebas unitarias y de integración. Una nueva guía de usuario detallará instalación, recorrido por el panel y tutoriales de casos de uso, garantizando adopción y mantenimiento a largo plazo.

3. Descripción de la herramienta

La transformación del panel original de openMASTER en una herramienta de visualización de alto rendimiento se articula en seis fases: primero, exploración práctica del panel actual para documentar cuellos de botella desde la salida de Gurobi hasta los gráficos estáticos; segundo, talleres que definen los casos de uso "what-if" y los siete parámetros esenciales; tercero, cargado al inicio de las 128 ejecuciones y su indexado en memoria por tuplas de parámetros, permitiendo búsqueda en tiempo constante; cuarto, reconstrucción de la interfaz en Dash con deslizadores discretos, panel lateral desplazable y secciones comparativas colapsables; quinto, enlace de los deslizadores a la carga de datos y generación de figuras Plotly mediante callbacks que garantizan actualizaciones de manera muy rápida; y sexto, validación rigurosa con pruebas de exactitud del índice, pruebas de integridad de UI y scripts automatizados que simulan interacciones y verifican flujo de callbacks. Finalmente, la guía de usuario documenta instalación, creación de escenarios, uso del panel y resolución de problemas, asegurando una adopción fluida y mantenible.

4. Resultados

Tras la implementación, cualquier ajuste de deslizador actualiza al instante diagramas de Sankey, gráficos de barras y áreas, y series temporales de emisiones. La pestaña Comparador muestra dos grupos de parámetros colapsables con gráficos lado a lado, rotulados automáticamente con sus valores, lo que facilita un análisis paralelo rápido.



1igure 1 – Vista rápida del nuevo dashboard.

5. Conclusión

El panel mejorado de openMASTER deja atrás el flujo de trabajo manual y fragmentado, convirtiéndose en una plataforma reactiva de nivel investigador para análisis “what-if”. Gracias a la precomputación de 128 escenarios y la indexación directa en el sistema de archivos, los analistas usan deslizadores intuitivos para generar diagramas de Sankey, gráficos de barras, de área y series de emisiones, sin manipulaciones de CSV ni recargas de página. El panel desplazable y las secciones comparativas colapsables optimizan la experiencia sin saturar la pantalla, y las pruebas de índice, UI y callbacks preservan la compatibilidad con flujos previos. Implementado íntegramente en Python y listo para despliegue en contenedores, este ecosistema ofrece transparencia, trazabilidad y alto rendimiento, capacitando a los interesados para explorar compensaciones de políticas energéticas de manera ágil y reproducible.

6. Referencias

[1] A. F. Rodriguez-Matas, M. Perez-Bravo, P. Linares, y J. C. Romero, «openMASTER: The open source Model for the Analysis of SusTainable Energy Roadmaps», *Energy Strategy Rev.*, vol. 54, p. 101456, jul. 2024, doi: 10.1016/j.esr.2024.101456.

[2] R. Loulou, G. Goldstein, A. Kanudia, A. Lettila, y U. Remme, «Documentation for the TIMES Model». Accedido: 14 de marzo de 2022. [En línea]. Disponible en: https://iea-etsap.org/docs/Documentation_for_the_TIMES_Model-Part-I.pdf

[3] K. Riahi *et al.*, «Energy Pathways for Sustainable Development», en *The Global Energy Assessment*, N. Nakicenovic, T. Johansson, A. Patwardhan, y F. Fen, Eds., Cambridge: Cambridge University Press, 2012.

PROGRAMMING AN ONLINE DASHBOARD TO GENERATE SCENARIOS IN THE OPENMASTER ENERGY MODEL

Author: Martín García-Marcos, Vicente.
Supervisors: Linares, Pedro and Pérez Bravo, Manuel.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The redesigned openMASTER control panel transforms long-term energy-scenario exploration into an interactive, fully reproducible process. Instead of running and plotting each case in isolation, the tool pre-loads *n* scenarios, generated by systematically combining the parameters that most strongly influence the model's outputs. Discrete sliders, range selectors, and multi-select filters allow any parameter set to be summoned instantly; Dash's reactive callbacks locate the corresponding directory and refresh the Plotly figures without reloading the page. The interface is modular and editable: widgets, metrics, and chart types can be added or reordered without altering the architecture, and the Python code follows a clean design that facilitates extensions. A collapsible comparator lets users pin up to three reference configurations and examine them side by side (Sankey diagrams, cost curves, or emissions trajectories) with automatic labelling and consistent legends. Everything is served from an in-memory cache and can be horizontally scaled with containers to host thousands of scenarios as the experimental design space grows.

Overall, the dashboard evolves from a static viewer into a parametric analysis platform suited to researchers, regulators, and industry stakeholders who need to assess how choices on capacity, fuels, or discount rates reshape the decarbonisation pathway projected by openMASTER.

Keywords: openMaster, energy modelling, scenarios, interactive dashboard

1. Introduction

openMASTER, developed at the Instituto de Investigación Tecnológica of Universidad Pontificia Comillas, combines a bottom-up, technology-rich linear-programming formulation with the Gurobi solver to produce detailed, multi-decadal projections of capacity investments, dispatch schedules and emissions trajectories, enabling rapid, data-driven analysis for decarbonization and climate resilience [1]. Its original web dashboard forced analysts to run each scenario by hand, export outputs to CSV and assemble static charts in separate tools, a fragmented workflow that was both time-consuming and prone to error, and that hindered exploratory “what-if” studies as well as real-time stakeholder engagement.

To address these shortcomings, the project delivers an all-Python dashboard that precomputes a library of scenario runs indexed by seven user-selectable parameters and employs intuitive sliders to retrieve any scenario instantly. Leveraging Dash’s reactive callbacks and Plotly’s dynamic figures, the interface updates charts without reloading the page. A vertically scrollable control sidebar, collapsible comparator sections and automatic labeling further streamline interaction and minimize cognitive load, transforming openMASTER into a fully interactive decision-support platform.

2. Project Definition

The project responds to the growing gap between high-fidelity energy-system models and the need for instant, interactive visualization. Whereas commercial tools like TIMES [2] and MESSAGE [3] are locked behind proprietary languages and fees, and open-source frameworks often force analysts into manual CSV exports and static charts, openMASTER’s native dashboard still requires each scenario to be run, exported and re-plotted externally—making multi-scenario “what-if” studies tedious, error-prone and discouraging exploratory analysis.

To address these shortcomings, the team will build a fully integrated, all-Python dashboard that preloads a library of 128 precomputed runs, indexes them by seven key parameters, and lets users swap scenarios in a single click via slider controls. Leveraging Dash’s reactive callbacks, Plotly’s GPU-accelerated rendering and an in-process caching layer, charts update without page reloads, and automatic labeling ensures clear, consistent overlays. Interactive widgets—range selectors, inline annotations and collapsible comparators—allow deep dives into specific time windows, technologies or emissions streams without leaving the interface.

Development follows six iterative, user-centred phases: Documenting the existing dashboard workflow; Workshop-driven requirements analysis; Scenario ingestion and in-memory indexing; Incremental UI refactoring with dash-bootstrap-components; Implementation of reactive callback logic; and comprehensive validation via unit and integration tests. A new user guide covering installation, a dashboard walkthrough and step-by-step use-case tutorials will ensure smooth adoption and long-term maintainability. Together, these enhancements will transform openMASTER into a high-performance, research-grade decision-support platform that delivers rapid, accurate and reproducible energy-policy analysis.

3. Model Description

The project transforms the original openMASTER dashboard into a high-performance, fully integrated visualization tool through a tightly coordinated six-phase methodology. It begins with hands-on exploration of the existing dashboard, documenting workflow bottlenecks from Gurobi output through CSV exports and static plots and proceeds to workshops that define the essential “what-if” use cases and seven key parameters. Rather than generating scenarios on demand, all 128 pre-computed runs are ingested at startup and indexed in memory by parameter tuples, enabling constant-time lookup. The user interface is rebuilt in Dash with discrete sliders for each model parameter, a scrollable sidebar and collapsible comparator sections, while reactive callbacks bind slider changes to folder-based data retrieval and Plotly figures, delivering updates without page reloads. Rigorous validation, including tests for indexing accuracy, integration tests for UI integrity and the verification of callback workflows, ensures that every enhancement preserves legacy behavior and meets performance targets. Finally, a comprehensive user guide for documents installation, scenario creation, dashboard use and troubleshooting, guaranteeing smooth adoption and maintainability. Together, these efforts convert openMASTER into a seamless, research-grade environment for rapid, reproducible energy-policy analysis.

4. Results

After implementation, any slider adjustment triggers updates to Sankey diagrams, bar/area charts, and emission. The Comparator tab presents two collapsible parameter groups and side-by-side charts, labeled with their parameter values, facilitating rapid parallel analysis.

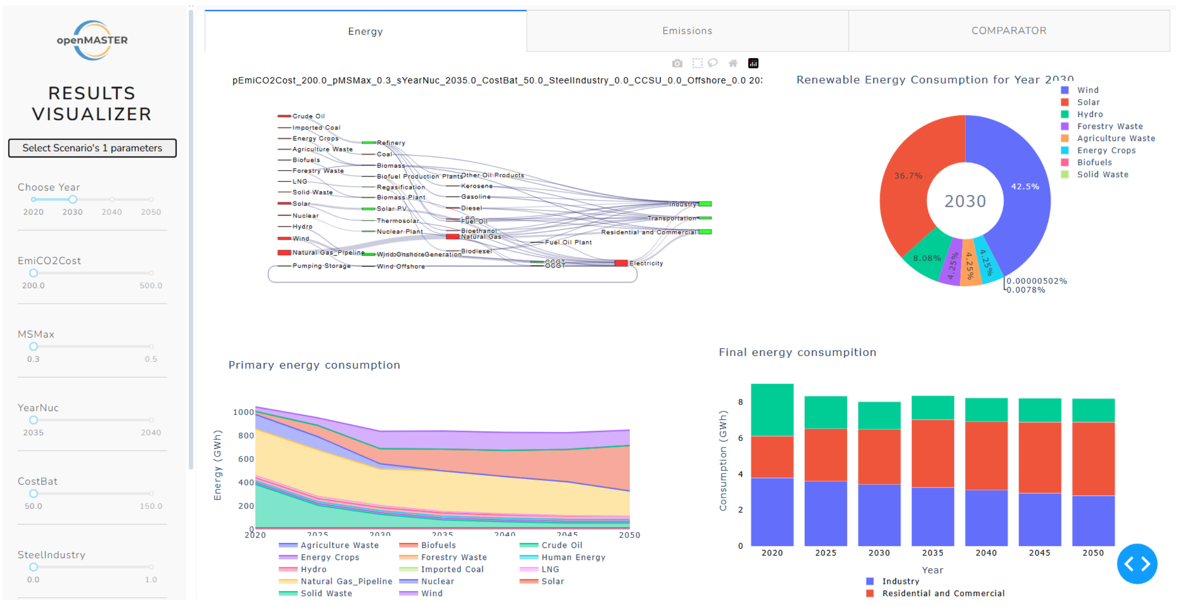


figure 2– Quick view of the new dashboard.

5. Conclusion

The enhanced openMASTER dashboard transforms a once-static, manual reporting tool into a responsive, research-grade “what-if” platform by precomputing all 128 scenario runs and encoding their seven key parameters directly in the filesystem. Analysts now adjust intuitive sliders to select any parameter combination, triggering Dash’s reactive callbacks to load and render Sankey diagrams, bar, pie and area charts, and emission time series, no CSV juggling or page reloads required. A scrollable control sidebar and collapsible comparator panels streamline single- and dual-scenario views without overwhelming the screen, while rigorous indexing, UI integrity and callback tests preserve legacy menus and workflows for seamless adoption. Built entirely in Python and packaged for containerized deployment, this end-to-end ecosystem delivers transparency, traceability and performance, empowering stakeholders to explore policy trade-offs rapidly and reproducibly.

6. References

[1] A. F. Rodriguez-Matas, M. Perez-Bravo, P. Linares, y J. C. Romero, «openMASTER: The open source Model for the Analysis of SusTainable Energy Roadmaps», *Energy Strategy Rev.*, vol. 54, p. 101456, jul. 2024, doi: 10.1016/j.esr.2024.101456.

[2] R. Loulou, G. Goldstein, A. Kanudia, A. Lettila, y U. Remme, «Documentation for the TIMES Model». Accedido: 14 de marzo de 2022. [En línea]. Disponible en: https://iea-etsap.org/docs/Documentation_for_the_TIMES_Model-Part-I.pdf

[3] K. Riahi *et al.*, «Energy Pathways for Sustainable Development», en *The Global Energy Assessment*, N. Nakicenovic, T. Johansson, A. Patwardhan, y F. Fen, Eds., Cambridge: Cambridge University Press, 2012.

Table of contents

List of figures

List of Tables

Capítulo 1. INTRODUCTION

The urgent need to decarbonize energy systems and bolster climate resilience has placed new demands on policy-analysis tools. Models must not only capture complex technological, economic and environmental interactions but also present results in a way that enables rapid, data-driven decision making. openMASTER—developed at the Instituto de Investigación Tecnológica of Universidad Pontificia Comillas—addresses these modeling needs by combining a bottom-up, technology-rich linear-programming formulation with the performance of the Gurobi solver to generate detailed, multi-decadal projections of capacity investments, dispatch schedules and emissions trajectories.

Despite its powerful optimization core, openMASTER's original web dashboard fell short of modern usability expectations. Analysts were required to generate each scenario manually, export model outputs to CSV files and then assemble static charts using separate tools. This fragmented workflow introduced repetitive, error-prone tasks, impeded exploratory "what-if" studies and limited stakeholder engagement by preventing real-time, interactive comparisons.

To overcome these limitations, this project designs and implements an advanced, all-Python dashboard that transforms openMASTER into a truly interactive decision-support platform. By precomputing a library of scenario runs and indexing them by seven user-selectable parameters, the dashboard enables instant scenario retrieval via intuitive sliders. Dash's reactive callback system then binds slider changes to on-the-fly data lookups and Plotly figure updates, all without requiring page reloads. Additional interface enhancements include a vertically scrollable control sidebars to choose between scenarios and collapsible comparator sections to improve the user experience.

The remainder of this document is organized as follows. Chapter 2 reviews the software technologies underlying the enhanced dashboard. Chapter 3 surveys the state of the art in energy-model visualization and identifies the key pain points that motivated our redesign. Chapter 4 defines the project's justification, objectives and user centered methodology. Chapter 5 details the implementation of scenario indexing, UI redesign and callback logic. Chapter 6 presents the results of manual validation and performance testing, and Chapter 7 concludes with reflections on the outcomes and outlines future work, including plans for server deployment and collaborative extensions.

Capítulo 2. TECHNICAL OVERVIEW

After this project's introduction done in chapter 1, chapter 2 will focus on the technologies used to perform this project.

2.1 DEVELOPMENT ENVIRONMENT: PYTHON & JUPYTER NOTEBOOK

2.1.1 PYTHON

Python is the primary programming language used throughout the project. It is an interpreted, high-level, multi-paradigm language with dynamic semantics, designed to emphasize code readability and rapid development. Python's comprehensive standard library and vast ecosystem on the Python Package Index (PyPI) make it ideal for scientific computing, optimization modeling, data manipulation and web development [1]. Crucially, Python is distributed under a permissive open-source license, and in keeping with this philosophy, openMASTER and all its accompanying modules are released as open-source software—ensuring that every extension, visualization component, and analysis routine remains freely accessible, auditable, and extensible by the research community

2.1.1.1 Python Implementation

Python serves as the backbone of openMASTER, unifying model formulation, data handling, visualization and reporting in a single, reproducible workflow. A Pyomo script defines the LP's sets, parameters, variables and constraints in native Python, then instantiates and solves the model with Gurobi. Once complete, pandas and NumPy read and transform the CSV and Excel outputs into clean DataFrames, compute key indicators (capacity additions, emissions, service provision) and cache results for speed.

Scenario management is automated by a simple Python routine that snapshots code, inputs and outputs into timestamped folders, ensuring every run is fully traceable. These processed data feed directly into a Dash application—also written entirely in Python—where sliders trigger callbacks to build interactive Plotly charts without any JavaScript.

2.1.2 JUPYTER NOTEBOOKS

Jupyter Notebook is the interactive authoring environment used for exploratory development, prototyping model components, and demonstrating workflows. As a web-based application, it combines executable code cells, rich-text Markdown (including LaTeX), and inline visualizations in a single. ipynb document [2]. Notebooks enable reproducibility by capturing both code and the narrative, and they serve as both documentation and test harness during model development. This Dashboard updates were implemented directly within Jupyter Notebooks, enabling modification of layout components, callback logic, and Plotly visualizations in a single interactive environment. This approach supports an iterative workflow: whenever the dashboard's structure or behaviour is changed, the updates can be executed immediately, observed in real time, and refined without context switching. By consolidating code, documentation, and live previews in one place, each revision remains transparent, reproducible, and easily verifiable.

2.2 OPTIMIZATION FRAMEWORK: PYOMO & GUROBI

2.2.1 PYOMO

Pyomo ("Python Optimization Modeling Objects") is the algebraic modeling language employed to formulate the dynamic, bottom-up LP problem of openMASTER. As an open-source Python package, it allows modelers to define sets, parameters, variables, constraints and objective functions in a syntax that closely mirrors mathematical notation. Pyomo supports both abstract models (data-agnostic) and concrete instances (with loaded data), facilitating scenario management via simple Python scripts. [3]

2.2.2 GUROBI

Modern energy models can leverage either open-source or commercial solvers to compute optimal solutions. Open-source alternatives such as GLPK (GNU Linear Programming Kit) and HiGHS offer accessible, no-cost implementations of linear, mixed-integer, and quadratic programming algorithms under permissive licenses, making them attractive for fully open workflows. Commercial solvers like Gurobi deliver substantially higher performance—exploiting multi-core architectures and advanced presolve techniques to handle millions of variables with industry-leading speed—and are available at no cost for academic users through Gurobi's Academic Program. [4] In this project, Gurobi was selected to ensure rapid model turnaround and a seamless interactive experience, while preserving the open-source ethos of openMASTER by enabling any user with academic credentials to access the same solver performance without licensing barriers.

2.2.3 IMPLEMENTATION

Pyomo serves as the core modeling framework, enabling the project's optimization problem to be defined, instantiated and solved entirely within Python. Model elements are declared in a high-level, declarative style. Input data are bound to these elements at runtime, the model is passed to an external solver, in this case Gurobi, which returns the optimal variable values directly into the Pyomo instance ready for downstream processing. This all-in-Python workflow ensures that the optimization step integrates seamlessly with data handling, interactive visualization and automated reporting, while maintaining full reproducibility and traceability.

2.3 CORE DASHBOARD COMPONENTS: DASH, PLOTLY.PY & DASH BOOTSTRAP COMPONENTS

2.3.1 DASH

Dash is an open-source Python framework for building analytical, web-based applications and dashboards entirely in Python. It enables developers to create interactive, data-driven interfaces without writing any JavaScript, HTML or CSS. [5]

2.3.2 PLOTLY.PY

Plotly is an open-source Python graphing library for creating interactive, publication-quality charts entirely in Python. It provides a high-level, declarative API that generates JSON configurations rendered by Plotly.js in the browser, supporting over 30 chart types—including time series, statistical plots, and Sankey diagrams—without requiring any JavaScript. [6]

2.3.3 DASH-BOOTSTRAP-COMPONENTS

Dash Bootstrap Components is an open-source Python library that provides Bootstrap-styled UI components for use in Plotly Dash applications. By linking a Bootstrap v5 stylesheet and importing components from dash-bootstrap-components, developers can build responsive, consistently themed layouts without writing custom CSS, while leveraging Dash's reactive callback system for interactivity. [7]

2.3.4 IMPLEMENTATION

The openMASTER dashboard is implemented as a single Python module that brings together Dash, Plotly.py and Dash Bootstrap Components into an integrated, all-Python web application. Layout definitions to arrange UI elements. Each initial figure is constructed with Plotly.py and passed to a graph component for client-side rendering by Plotly. Interactivity is enabled through Dash's reactive callback system so that any change automatically triggers the callback and updates the interface in real time all without writing JavaScript or HTML.

Capítulo 3. STATE OF ART

Long-term energy system planning hinges on two pillars: the fidelity of the underlying optimization model and the effectiveness of its result visualization. Below, each dimension is examined in depth.

3.1 LONG-TERM ENERGY PLANNING MODELS

3.1.1 COMMERCIAL FRAMEWORKS

TIMES (The Integrated MARKAL-EFOM System) and MESSAGE (Model for Energy Supply Strategy Alternatives and their General Environmental Impact) are flagship tools of the IEA-ETSAP consortium and IIASA, respectively. Both are implemented in the proprietary GAMS language and employ large-scale linear programming to identify least-cost energy pathways over multi-decadal horizons.

TIMES originated as MARKAL in the late 1970s and was formalized as TIMES in 2008. It structures any scenario around four core inputs—energy service demands, primary resource potentials, policy constraints and technology descriptions—managed via GAMS Data Exchange (GDX) files or GUI front-ends such as VEDA-BE and the newer MIRO application. Its bottom-up, technology-rich architecture spans fuel extraction, conversion, transport, distribution and end-use services, and includes built-in reporting engines for tabular outputs, sensitivity analyses and multi-region coupling. [8]

MESSAGE, developed by IIASA and later integrated into the ETSAP portfolio, likewise covers the full energy chain and explicitly models inter-regional trade and multiple greenhouse gases. It has been a core tool for global climate assessments, including IPCC scenarios, thanks to its standardized data libraries for technology costs, resource potentials and emissions factors. [9]

Despite their analytical power and decades of community support—complete with workshops, training materials and shared databases—both frameworks depend on commercial software (GAMS and licensed solvers) and incur significant licensing fees (often tens to hundreds of thousands of USD per seat per year). This restricts access for many academic institutions and smaller research groups, and complicates integration into fully open, reproducible Python-centric workflows. [10]

3.1.2 OPEN-SOURCE MODELS

Two prominent community-driven, open-source platforms for energy system modelling are OSeMOSYS and Calliope, each adopting distinct approaches to balance usability, flexibility and transparency.

OSeMOSYS (Open Source energy MOdelling SYStem) is implemented in the high-level MathProg/GAMS language with an optional Python interface. It generates full-scale, long-term optimization models for local to multi-regional planning by defining energy and mass balances, technology modules and economic parameters via structured input files. Designed to minimize upfront costs and training requirements, OSeMOSYS underpins workflows ranging from continental electrification studies to village-scale analyses. [11]

However, its multi-language codebase (MathProg, GAMS, Python) and reliance on separate model, data and scenario files can fragment the user experience, while built-in visualization and scenario-management tools remain limited.

Calliope embraces a pure-Python framework that constructs optimization problems from YAML configuration files and associated CSV time-series data. By separating model definitions (technologies, locations, parameters) into human-readable YAML (e.g., model.yaml, techs.yaml) and time-series into CSV directories, Calliope enables rapid prototyping and high spatial/temporal resolution for urban districts up to continental grids. [12]

Its clear division between framework (code) and model (data) enhances transparency and reproducibility. Yet, maintaining multiple YAML and CSV files introduces a steep learning curve for newcomers and complicates integration with Python-centric visualization or batch processing workflows.

While both OSeMOSYS and Calliope democratize access to large-scale energy modelling by eliminating licensing fees and opening source code, they also highlight two common challenges in open-source approaches: workflow fragmentation across disparate file formats and languages, and limited, non-interactive visualization, which often requires external scripts or bespoke UIs for scenario comparison. These gaps motivate the development of an all-Python, Dash-based dashboard that unifies model invocation, result extraction and interactive exploration within a single, reproducible environment.

3.1.3 PYTHON-NATIVE MODELING

Pyomo ("Python Optimization Modeling Objects") is a mature, open-source algebraic modeling language embedded entirely within Python. Developed as part of the COIN-OR project and released under the BSD license, Pyomo first appeared in 2008 and has since gained widespread adoption in both academia and industry. [13]

Unlike standalone modeling systems, Pyomo allows modelers to declare sets, scalar and multidimensional parameters, decision variables, constraints and objective functions using native Python syntax. Models can be defined abstractly or instantiated concretely by binding data at runtime, supporting a clear separation of model logic and scenario inputs.

A key feature is the DataPortal class, which standardizes loading data from diverse sources (CSV, Excel, TAB files, SQL databases) into model parameters and sets via a uniform interface. [14] This direct data binding eliminates intermediary conversion scripts, reducing context switching and enhancing reproducibility.

Under the hood, Pyomo interfaces with a broad spectrum of solvers open-source (GLPK, CBC) and commercial (Gurobi, CPLEX, Mosek) through the SolverFactory API. Ongoing enhancements in Pyomo' s solver interfaces and solver managers enable asynchronous and parallel execution, further improving scalability for large-scale problems. [15]

Despite its powerful modeling capabilities and seamless Python integration, many Pyomo users still rely on external scripts and static plotting libraries (Matplotlib, seaborn) for post-processing, which impedes rapid "what-if" exploration. Embedding results inspection and interactive visualization into the same Python environment remains an open challenge—one that the openMASTER dashboard seeks to address by unifying Pyomo' s modeling workflow with a dynamic, Dash-based front end.

3.2 ENERGY PLANNING RESULTS: VISUALIZING SOLUTIONS

3.2.1 COMMERCIAL DASHBOARDS

Commercial Business Intelligence (BI) tools such as Tableau and Microsoft Power BI dominate enterprise data visualization by providing rich, drag-and-drop interfaces, built-in connectors (SQL, Excel, REST APIs) and advanced mapping capabilities. However, their cost structures and integration patterns pose significant hurdles for bespoke optimization workflows like openMASTER.

Tableau licenses fall into three role-based tiers:

LICENSE	**COST**	**KEY FEATURES**
CREATOR	$900 per user per year	Full authoring (desktop + Prep), data preparation, publishing dashboards
EXPLORER	$504 per user per year	Interactive dashboards, light editing of published workbooks, data-driven alerts
VIEWER	$180 per user per year	View and interact with dashboards, download images/summary data, receive alerts

Table 1. Tableau licences, cost and key features

Enterprise deployments often require Tableau Server or Tableau Cloud plus additional “resource blocks” (e.g., $350 per block / month) to scale concurrent data-prep tasks. [16] Despite its powerful analytics and governance features, total cost of ownership can exceed $1 000 per user per year, excluding hardware or cloud hosting fees. Meanwhile Power BI offers a lower entry point:

LICENSE	**COST**	**KEY FEATURES**
DESKTOP	Free	Individual report creation and exploration
PRO	$120 per user per year	Collaboration, sharing, scheduled data refresh via Microsoft 365
PREMIUM PER USER	$240 per user per year	Larger datasets (up to 100 GB), more frequent refreshes, AI features
PREMIUM CAPACITY	From $4995 per month	Organization-wide performance guarantees, unlimited sharing, dedicated resources

Table 2. Power BI licences, cost and key features

While Power BI integrates seamlessly with Azure, SQL Server and Office 365, its 1 GB dataset limit (Pro) and 8 daily refreshes can constrain large-scale scenario data, often necessitating Premium capacity for heavy workloads. [17]

Tableau and Power BI do not directly accept Pyomo or Gurobi outputs, so openMASTER results must first be exported into intermediate tables or files and then loaded into these platforms via connectors. This adds significant manual overhead and creates versioning gaps whenever the model's structure or variable names change, as export scripts and dashboard configurations must be updated in lockstep. Furthermore, the high per-user licensing fees (often exceeding $1,000 annually) pose a barrier to adoption in academic and open-source contexts. Consequently, despite their strengths in enterprise reporting, the closed-source ecosystems, cost structures and multi-step integration processes of Tableau and Power BI render them a poor fit for the reproducible, all-Python workflow that openMASTER requires.

3.2.2 OPEN-SOURCE PYTHON FRAMEWORKS

Several pure-Python libraries have emerged to bridge the gap between data analysis and interactive dashboards. Below, three leading alternatives are examined in depth, highlighting their architectures, strengths and limitations in energy-modelling contexts.

3.2.2.1 Dash

Dash provides a Python-native framework where a Flask server communicates with React.js components via JSON to render dcc.Dropdown, dcc.Slider, dcc.Graph and other UI elements declared entirely in Python. Its reactive callback model (@app.callback) updates Plotly figures without page reloads, offering seamless interactivity. Out-of-the-box, Dash renders charts as SVG, which can become sluggish with tens of thousands of points; mitigating strategies include WebGL traces (Scattergl) or the Plotly-Resampler extension. Although Dash Enterprise adds job-queue and horizontal-scaling support, community users must rely on careful callback design and optional server-side caching for acceptable performance. Finally, Dash lacks built-in multi-scenario comparison widgets, forcing developers to implement custom layouts (e.g., hidden tabs, multi-dropdowns) and manage client state across callbacks, which increases boilerplate for complex energy-model comparisons. [18]

3.2.2.2 Streamlit

Streamlit adopts a "script-first" paradigm: the entire app reruns top-to-bottom on each user interaction, automatically generating UI controls from script variables. This approach dramatically lowers the barrier to entry but can lead to inefficiencies in larger dashboards since all components re-render on any change. Layout primitives (st.columns, st.container, st.expander) introduced in 2020 allow side-by-side and collapsible sections, yet fine-grained control (nested grids, custom CSS) remains more limited than Dash's Bootstrap-based layouts. Streamlit's @st.cache decorator helps memoize expensive data loads or computations, reducing script re-execution time by up to 80 % on repeats, but orchestrating complex multi-step interactions still demands careful cache scope design. [19]

3.2.2.3 Bokeh

Bokeh offers low-level plotting primitives (glyphs, tools) and can run as standalone HTML or via a Bokeh server that supports Python callbacks in response to JavaScript events (CustomJS) or server-side logic. Its separation of plotting and layout constructs (row, column, gridplot) gives ultimate flexibility for custom dashboards, including JavaScript integration, but this power comes at the cost of extensive boilerplate and manual wiring of callbacks. [20] For large scatter plots, Bokeh supports a WebGL backend (output_backend="webgl"), mitigating SVG bottlenecks, yet switching backends and tuning glyph parameters often requires low-level adjustments, making it less turnkey than Dash or Plotly.

Across all frameworks, two pain points recur in energy-model visualization: rendering large time series (thousands–millions of points) without perceptible lag and building first-class multi-scenario comparison interfaces. Integration overhead—gluing together model outputs, cache layers and UI scripts—further underscores the need for a unified, high-performance, all-Python dashboard solution tailored to openMASTER.

3.3 THE CURRENT OPENMASTER DASHBOARD

The dashboard included with openMASTER provides basic visualizations (time-series charts and a standalone Sankey-diagram generator), but its architecture creates three major usability hurdles.

3.3.1 STATIC, MANUAL WORKFLOWS

Before any visualization can occur, a scenario must first be generated in openMASTER, a process that demands configuring inputs, running the model and waiting for the solution, which itself can take considerable time and effort. Only after this initial step can the user export the filtered results to CSV, load them into a spreadsheet and manually arrange the graphs side by side. This multi-stage hand-off fragments the analysis flow—forcing repetition of scenario creation, file navigation and chart assembly—and increases the risk of errors (e.g. mismatched time axes or misaligned variable names), undermining confidence in the results.

3.3.2 SLOW MULTI-SCENARIO COMPARISON

Because the steps of creating new scenarios cannot be automated within the dashboard, the effort scales linearly: comparing three scenarios requires roughly three times the work of a single case. This repetitive process not only slows down analysis turning what-if studies into a batch of manual chores but also increases the chance of errors, such as misaligned axes or mislabelled data. As a result, analysts are discouraged from exploring large scenario sets, limiting the depth and speed of energy policy evaluation.

3.3.3 INEFFICIENT MULTI-SCENARIO COMPARISON

Comparing multiple scenarios is confusing: each scenario carries the arbitrary label that the user assigned, and as the number of cases grows, keeping track of dozens of distinct names becomes cumbersome. Without an in-dashboard mechanism to organize, group or rename scenarios, users often lose track of which trace corresponds to which case, making side-by-side analysis error-prone and inefficient.

Together, these issues compound into a highly fragmented and error-prone workflow: each "what-if" adjustment requires generating new scenarios and manually exporting their data, waiting through lengthy reloads, and wrestling with arbitrarily named files to arrange side-by-side views. Lacking in-dashboard tools for batch comparison or clear labelling, analysts rapidly lose momentum and confidence. Consequently, those who need quick, reliable scenario exploration and presentation-grade visuals often abandon the built-in dashboard for bespoke scripts or external software undermining openMASTER's goal of streamlined, end-to-end reproducibility.

Capítulo 4. PROJECT DEFINITION

4.1 JUSTIFICATION

The rapid emergence of complex, multi-decadal energy models has placed new demands on both the fidelity of optimization frameworks and the immediacy of their result visualization. As reviewed in Chapter 3, established commercial systems like TIMES and MESSAGE offer unparalleled depth and built-in reporting, yet remain locked behind proprietary languages and steep annual fees, while open-source alternatives such as OSeMOSYS and Calliope break free of license constraints at the cost of fragmented workflows across MathProg, YAML and CSV files. Pyomo succeeds in unifying modelling logic within Python, but in practice many users still revert to manual CSV exports and static plotting for each "what-if" iteration, breaking the analytical flow and undermining reproducibility.

Within the openMASTER ecosystem, these challenges are compounded. The native dashboard requires each scenario to be generated, exported and re-plotted externally before any comparison can be made, an error-prone sequence. Rendering multiple scenarios compounds this delay: because each case must be created, exported and plotted separately, comparing three scenarios demands roughly three times the manual effort of a single run. This linear scaling of chores—generation, export, chart assembly—transforms what-if analysis into a tedious batch of repetitive tasks, heightens the risk of misaligned axes or mislabelled data, and ultimately discourages analysts from exploring larger scenario sets. Finally, because users assign arbitrary names to their scenarios and the dashboard lacks built-in grouping or overlay controls, side-by-side comparisons quickly become confusing and brittle.

These cumulative impediments transform what-should-be an agile, exploratory process into a clunky sequence of manual chores, driving analysts toward bespoke scripts or third-party BI tools and away from the integrated, version-controlled Python workflow that openMASTER aims to provide. To restore the analytical agility, eliminate human-error risk and ensure end-to-end reproducibility, it is therefore essential to develop a fully integrated, all-Python dashboard that loads and filters results in memory, supports clear multi-scenario overlay with consistent labelling, and delivers interactivity via in-process caching and GPU-accelerated rendering. This targeted enhancement will directly address the deficiencies outlined in Chapter 3 and deliver the rapid, trustworthy decision support required for modern energy policy design.

4.2 OBJECTIVES

The central goal of this project is to elevate the openMASTER dashboard from a static reporting tool into a high-performance, research-grade environment that enables analysts to fluidly explore "what-if" scenarios without manual overhead. By pre-loading a comprehensive library of model runs and replacing file-based lookups with parameter-driven sliders, the dashboard will allow users to switch between cases simply by adjusting controls, rather than hunting through directory structures. Built-in labeling logic will automatically reflect each scenario's defining parameters, eliminating confusion when comparing two overlaid graphs. Interactive widgets, from adaptive range selectors to inline annotations, will empower users to drill down into critical time windows, technologies or emission streams without leaving the interface. Underpinning this responsiveness is a carefully designed caching layer and GPU-accelerated plotting, ensuring updates even for detailed 2030–2050 time-series. Together, these enhancements will close the usability gaps identified in Chapter 3, delivering a seamless, end-to-end solution that meets the speed, accuracy and reproducibility demands of modern energy-policy analysis.

4.3 METHODOLOGY

The development of the enhanced openMASTER dashboard follows an iterative, user-centred process combining software engineering best practices with domain-specific validation. The methodology comprises six key phases.

4.3.1 UNDERSTANDING THE EXISTING DASHBOARD

In this initial phase, the dashboard will be operated according to its existing user guide to gain firsthand insight into its functionality. Core workflows will be executed step by step. During each action, the sequence of interface interactions will be documented. This exercise will establish a clear understanding of how the web application currently works, highlighting areas where the workflow is manual or unintuitive and guiding subsequent enhancements.

4.3.2 REQUIREMENTS ANALYSIS

Initial workshops with the crew establish the critical workflows, performance targets and comparison patterns needed for "what-if" studies. From these sessions, a set of representative use cases is distilled, ranging from single-scenario deep dives to side-by-side comparisons of two cases, and the parameter spaces to be exposed via sliders are identified.

4.3.3 SCENARIO PRELOADING AND INDEXING

Rather than generating scenarios on demand, the complete library of 128 pre-computed model runs is ingested at startup. A lightweight metadata parser extracts each folder's encoded parameters and builds an in-memory index, a dictionary mapping numeric parameter tuples to file paths. This index enables constant-time lookup of any scenario purely by matching slider values.

4.3.4 Dashboard UI Design

The existing dashboard layout will be incrementally refactored to accommodate the new features while preserving all pre-existing components and interactions. Using Dash's component-based architecture and dash-bootstrap-components for responsive grids, each new control will be introduced in its own container or row.

4.3.5 Reactive Callback Implementation

Dash callbacks connect slider inputs to scenario matching and visualization functions. On any slider change, the callback retrieves the matching scenario's DataFrame, generates or retrieves the corresponding Plotly figure, and updates the appropriate graph component. A secondary callback handles the duplication of figures for the comparison view, automatically labeling each chart with its parameter values.

4.3.6 Validation

To guarantee that each phase of the methodology performs as intended and that no existing functionality is broken the following validation activities will be executed:

4.3.6.1 Indexing Accuracy Tests

For a representative sample of parameter combinations identified, unit tests will invoke the metadata parser and in-memory index to verify that each slider tuple maps to the correct scenario folder. Any mismatch or missing entry will be flagged for immediate correction.

4.3.6.2 UI Integrity Checks

After refactoring the layout in 4.3.4, a suite of integration tests will confirm that all pre-existing components (menus, buttons, charts) remain visible and functional. Each new control will be asserted to render in its designated container without overlapping or altering legacy elements.

4.3.6.3 Callback Workflow Verification

Automated integration scripts will simulate slider adjustments and assert that the reactive callbacks correctly load the matching DataFrame, generate or fetch the Plotly figure, and update the single or dual-view graph components. Tests will also verify that autogenerated labels reflect the slider values and that no JavaScript errors occur in the browser console.

4.3.7 DEVELOPMENT OF A NEW USER GUIDE

A comprehensive User Guide will be created to ensure effective adoption and long-term maintainability of the enhanced dashboard. This guide will include:

4.3.7.1 Installation & Setup

Step-by-step instructions for installing everything that the users may need in order to execute the model.

4.3.7.2 Getting Started

A walkthrough of the dashboard's layout control panel, visualization panel illustrating how to select scenarios, adjust sliders and interpret charts.

4.3.7.3 Use-Case Tutorials

Guided examples covering common workflows, such as comparing two scenarios side by side. Each tutorial will include annotated screenshots, sample parameter values and expected outcomes.

By progressing through these phases this methodology ensures that the new openMASTER dashboard delivers a seamless, end-to-end "what-if" analysis experience.

Capítulo 5. DEVELOPED VISUALIZATION TOOL FOR OPENMASTER

The methodology underpinning this project is organized into six interlocking phases designed to ensure a rigorous, user-focused enhancement of the openMASTER dashboard. First, Understanding the Existing Model establishes a clear baseline by exercising current functionalities and documenting workflow bottlenecks. Next, requirement analysis captures the critical "what-if" use cases and parameter spaces that the interface must support. Building on these insights, scenario preloading and indexing implements a high-speed lookup of all 128pre-computed runs. The reactive callback implementation phase then ties slider inputs to dynamic data retrieval and visualization logic, enabling instant chart updates and side-by-side comparisons. A comprehensive Validation verifies that every enhancement preserves legacy behaviour and delivers the intended functionality. Finally, the development of a new user guide synthesizes these changes into clear, step-by-step documentation, ensuring effective adoption and maintainability. Each of these phases is elaborated in the sections that follow.

5.1 UNDERSTANDING THE EXISTING DASHBOARD

In this chapter, the current openMASTER model and its accompanying dashboard are subjected to a systematic analysis to establish a clear baseline for subsequent enhancements. The investigation will combine hands-on operation—following the official user guide to execute core workflows—with code-level inspection of data ingestion, model instantiation and visualization routines. By mapping out the end-to-end flow from Gurobi outputs through CSV exports and Dash callbacks to rendered charts, this phase will identify manual hand-offs, performance bottlenecks and areas of limited interactivity. The insights gained here will directly inform the requirements, design and validation activities in the chapters that follow.

5.1.1 INSTALLING OPENMASTER PACKAGE

The openMASTER repository normally is acquired via a GitHub repository clone, which is creates a complete local copy of a remote repository, including all files, branches and full commit history. To perform this the user must ensure that Git is installed on their system, then in the computer's terminal run the following code.

```
git clone https://github.com/IIT-EnergySystemModels/openMASTER.git
```

When following this step during the project an issue was identified, one of the documents was too large to be able to clone the repository. To overcome this problem an adapted version of the project was created and handed over by Manuel Pérez which was smaller than the original one allowing the development of the project without any issues. This new version switched the excel document openMASTER_Data.xlsx for the openMASTER_Data_2050_Gap10.xlsx document. This change would later cause minimal inconsistencies with the already existing code, this will be later discussed in the following chapters.

After downloading and extracting the archive into the intended working directory, a dedicated Python 3.10 or later environment is established and activated using the code lines listed below:

```
python -m venv .venv
.venv\Scripts\activate
```

Within the activated environment, the package is installed in editable mode by executing:

```
cd                                                                  openMASTER
pip install -e .
```

To enable the optional graphical toolchain required by the interactive dashboard, such as Matplotlib extensions, Plotly Dash dependencies and Sankey diagram generators, the graphicTool extras are included via:

```
pip install .[graphicTool]
```

Jupyter Notebook is installed to support the execution and inspection of the demonstration notebooks.

Attention is then turned to the critical input dataset. The presence of openMASTER_Data.xlsx in the data/input/ folder is confirmed, in this case due to the issue adressed before the presence of openMASTER_Data_2050_Gap10.xlsx is the one confirmed; if the file is missing, it is retrieved from the provided raw download link and placed in that directory. The model's startup routines automatically create any additional subfolders under data/ and scenarios/, so no further manual directory setup is necessary.

For commercial solver support, Gurobi was selected for its free academic license and superior solution speed. The `GRB_LICENSE_FILE` (and any other Gurobi-specific environment variables) must be set according to Gurobi's licensing instructions, and the installation can be verified by importing `gurobipy` in Python and solving a simple test model. A free academic license can be obtained by visiting Gurobi's Academic Program page: https://www.gurobi.com/academia/academic-program-and-licenses/

Finally, the installation is validated by launching the Quickstart notebook:

jupyter notebook notebooks/quickstart_openMASTER.ipynb

Executing the first cells, model instantiation, scenario creation and dashboard startup, confirms that the optimization runs successfully, results are correctly exported, and the preliminary dashboard renders without error. Successful completion of these steps indicates that the environment has been fully configured and is ready for comprehensive scenario analysis.

5.1.2 CREATING A NEW SCENARIO

A scenario is defined in openMASTER as a set of inputs which will create a specific output. To help the user keep track of different runs of the model, this functionality creates a new folder within the /scenarios folder in the local clone of the repository. This folder contains a copy of the code used to run the scenario, as well as all the inputs and outputs for future use and development.

The first steps to create a scenario to study are done in the quickstart_openMASTER.ipynb document. In this code the model is created and solved using the data provided by the openMASTER_Data excel.

Before running the code, the user must edit the excel file to meet its wanted parameters. Inside the openMASTER_Data excel file several sheets can be found, the INDEX sheet gives the user a small description of the data found in the file. The sheets that are more relevant to the topic of creating a new model are the ones that begin with “p” such as “pDisRate” or “pEmiCO2Cost” since in these sheets is where the scenario parameters are defined. Once the user has ensured that all the parameters fit their needs, they can save the file and continue running the python code in the quickstart file.

5.1.2.1 quickstart_openMaster.ipynb

The quickstart notebook provides a step-by-step demonstration of the openMASTER modeling and post-processing pipeline, exemplifying the end-to-end workflow from model instantiation to consolidated results extraction. Initially, the notebook imports the core

Pyomo environment and the openMASTER library, establishing the namespaces required for model construction, data handling and solver interaction. An abstract Pyomo model is then created via a dedicated factory function, and a dual-suffix is attached to enable later retrieval of shadow prices.

Subsequently, input data are bound to the abstract model using the project's DataPortal utilities. Excel-based loading routines is illustrated, and key scenario parameters defined in the excel are selectively overridden to demonstrate parameter sensitivity. Once the DataPortal is populated, the model is concretized into a runnable instance, at which point it may be exported in LP format for inspection or external solver use.

The notebook proceeds to configure and invoke Gurobi through Pyomo's SolverFactory interface, streaming solver logs to the console. A simple test model verifies solver connectivity and licensing before the main optimization is executed. Upon completion, solved model outputs are organized into a temporary results directory.

In the post-processing phase, the notebook iterates over a predefined list of scenario folders, reading each scenario's output files into pandas DataFrames. Custom routines compute key performance indicators—such as CO_2 emissions for a target year, total system cost and renewable energy shares—and aggregate these metrics into a single summary table. The final DataFrame is then exported as a consolidated CSV, ready for downstream visualization in the dashboard.

This quickstart sequence not only validates the seamless integration of modeling, solving and data extraction within a reproducible Python environment but also lays the groundwork for the interactive, slider-driven "what-if" analysis presented in subsequent chapters.

After running the quickstart_openMASTER.ipynb a second file (scenarios_from_results.ipynb) must be run before being able to see the results in the dashboard.

5.1.2.2 scenarios_from_results.ipynb

The scenarios_from_results.ipynb notebook presumes that the quickstart workflow has already been executed and its result CSVs are present. Once the quickstart notebook has generated per-scenario outputs under data/tmp/output/, the scenarios_from_results.ipynb script scans the scenarios/ directory for all existing subfolders, each corresponding to a user-named model run. Within each folder, it imports every solver export (generation, capacity, emissions, cost breakdowns) into pandas DataFrames, ensuring that all time-series and KPI tables are captured. Simultaneously, it copies the exact input data files and the source-code snapshot used for that run into the same folder, bundling provenance alongside results. Finally, the notebook appends a record for each scenario to a master index (scenarios_index.csv), preserving the original folder name as the identifier, extracting its parameter values, and logging a timestamp. This procedure guarantees end-to-end traceability—inputs, code, and outputs remain collocated—and supplies the enhanced dashboard with a fully indexed repository of scenarios that can be retrieved by parameter combinations without manual file handling.

The creation of the scenarios folder is strictly necessary before the user can use the dashboard since the dashboard code dynamically discovers every subfolder under the top-level scenarios/ directory, which is exactly where scenarios_from_results.ipynb packages each run's inputs, outputs and code snapshots, and uses those folder names to populate its scenario dropdowns. When a user selects one of these entries, the load_data() helper constructs paths into that scenario's data/tmp/input and data/tmp/output subdirectories (populated by the quickstart workflow and then bundled by scenarios_from_results.ipynb) so that all visualisations pull directly from the CSV files archived in each scenario folder.

5.1.3 UNDERSTANDING THE DASHBOARD

The dashboard code is written in a single file code that calls to other functions to function properly under the name of index.py.

5.1.3.1 index.py

The index.py script implements a fully interactive web dashboard for exploring and comparing pre-computed openMASTER scenarios. On startup, it discovers every subfolder under the scenarios/ directory, each named by the user when the run was created and uses a small helper (resource_path) to reliably locate static assets (images, mapping files) whether running from source or a packaged bundle.

The application's layout is organized into a two-column grid: a left-hand control panel built with Dash and dash-bootstrap-components (LUX theme) contains dropdowns for selecting the primary and, optionally, a secondary scenario; numeric inputs for the analysis year and comparison year; and a graph-type selector (Energy Sankey, bar, pie, area, or Emissions area). The right-hand panel hosts a set of Plotly graphs arranged under three tabs ("Energy", "Emissions", "Comparator"), each initially hidden and revealed by a tab-change callback.

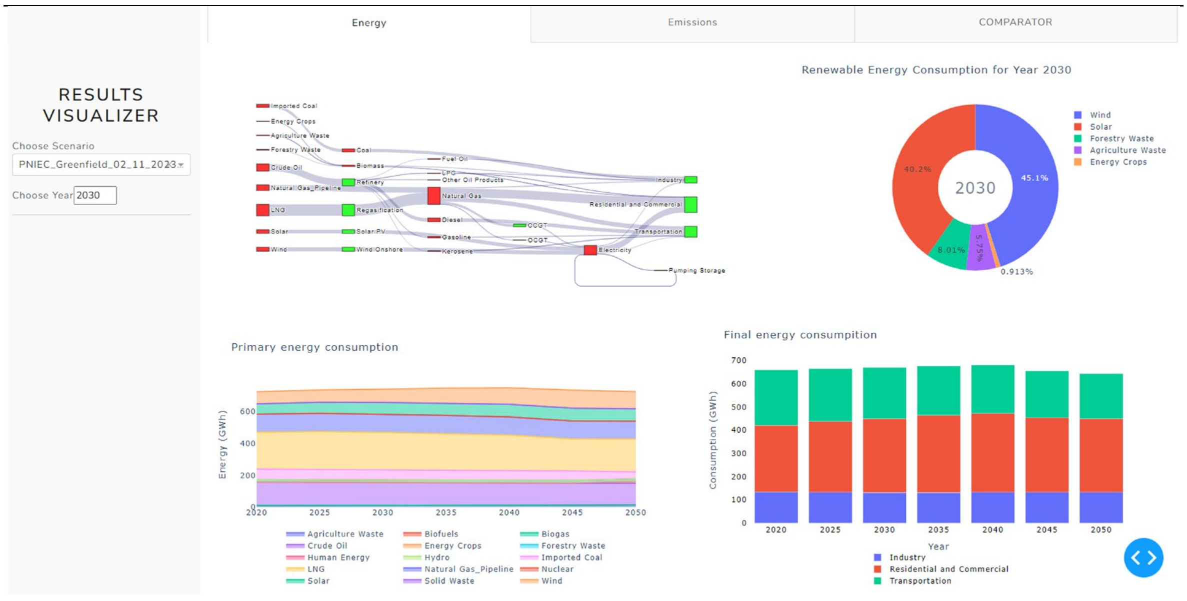


Figure 3. Initial openMaster dashboard.

Under the hood, three reactive callbacks drive all visualization logic. The first toggles which tab-specific content is displayed. The second listens to changes in the primary scenario or year, invokes specialized plotting functions to generate Sankey, bar, pie or area charts of energy flows and an emissions area chart, and updates five graph components in real time. The third responds to changes in either scenario dropdown, either year input, or the chosen graph type, producing two side-by-side figures of the selected type for direct comparison. Both callbacks reconstruct file paths for mappings, input and output data via a shared load_data() function, ensuring that every chart reflects the CSV exports bundled by the “scenarios_from_results” workflow.

Finally, the script launches a local Flask development server on port 8080 in debug mode. Together, this code delivers a one-stop, Python-only interface for rapid “what-if” analysis, allowing users to select, visualize and compare any two scenarios without manual file handling or the need for JavaScript index.

This dashboard used to offer a fully browser-based interface that lets users explore any single pre-computed scenario with just a few clicks. By choosing a scenario from the dropdown and specifying a year, the user immediately sees an energy-flow Sankey diagram, a renewable-generation pie chart, a stacked area plot of primary energy consumption over the 2030–2050 horizon and a segmented bar chart of final energy demand by sector in the “Energy” tab, a sector-wise CO2 emission over time chart in the “Emissions” tab and in the “Comparator” tab the user has the chance to compare any chart previously shown from 2 selected scenarios. All charts support zooming, panning and tooltips for detailed inspection. Behind the scenes, each selection dynamically reads the corresponding CSV exports—so there is no need for external plotting tools or manual file management—and rebuilds the figures in real time, providing a single, cohesive environment for visualizing the model’s outputs.

5.2 REQUIREMENT ANALYSIS

As seen in chapter 5.1 the existing workflow imposes unnecessary friction on users by forcing them to juggle manual CSV exports, external plotting tools and repeated file-browsing steps before any meaningful comparison can occur. These tedious hand-offs not only inflate the time required for each “what-if” exploration, turning what should be an intuitive, hypothesis-driven process into a series of clerical chores. By streamlining scenario selection, reducing wait-times and eliminating repetitive file management, the model can free users to concentrate on interpreting results and refining policy insights, rather than wrestling with the mechanics of data handling.

In this chapter, the full set of proposed enhancements to the openMASTER dashboard is presented and discussed. Drawing on the baseline analysis of the existing implementation and the key pain points identified in Chapter 3, each enhancement concept is introduced in terms of its intended benefit to user productivity, error reduction and analytical agility. By gathering all improvement ideas, ranging from scenario indexing and parameter-driven selection to responsive visualization and streamlined provenance tracking, this chapter lays out a coherent roadmap for evolving the tool into a truly interactive, high-performance "what-if" environment.

The first idea under consideration was to embed editable parameter fields directly within the dashboard and, before any charts are displayed, present a modal dialog prompting the user to complete all critical model parameters, demand growth rates, technology costs, emission prices, etc., via clearly labelled textboxes. Once the user clicked "Generate Tables," the application would programmatically inject those values into the corresponding sheets of the openMASTER_Data.xlsx, then sequentially execute both the quickstart pipeline and the scenario-packaging workflow. In a matter of seconds, the revised inputs would drive a fresh model run, produce new CSV outputs, and index them for immediate visualization. By collapsing Excel editing, notebook execution and result indexing into a single, one-step interaction, this design promised to eliminate manual file handling, reduce user error and deliver a truly seamless, on-demand scenario generation and exploration experience.

Although this new approach is very attractive since this approach lets the user free will to change any parameter from the model as they want, besides eliminating the need to work with several files every time a new scenario is created, it maintains and adds more problems to the dashboard. The problem that remains the same is that the user still needs to have access to a Gurobi license since the model still runs on the user's computer, it really isn`t a huge inconvenience but is something that can be improved. The new challenge that appears is that if the user wants to run a scenario that hasn't been run yet the user will need to wait several minutes between the click of the button and the visualization of the scenario. This is because scenarios are complex to compute. Since it is not possible to have users waiting this long a second approach was needed.

The second and definitive approach was to generate beforehand the most relevant scenarios for the users and include them in the openMASTER package, so they do not need to generate any scenarios at all. This approach solves at the same time the problem of having the user to follow the steps to create a new scenario as they used to. And it also removes the time that generating a new scenario would take. To follow this new idea two main questions had to find an answer. The first question is which scenarios are the most relevant for the users and how can they be organized. This question was answered within a few days by the team. The solution was simple; the user would have 7 preestablished parameters to edit. Since the values that a parameter can have follow a continuous variable so the values that each parameter could have had to be delimited. After deliberating with the team how many levels each parameters would have it was decided that every parameter could take only 2 values. This means that 128 different scenarios in total would have to be precomputed for the user could choose from.The name and values of the parameters that the user will be able to choose from are described in the following table.

VARIABLE NAME	**DESCRIPTION**	**VALUE 1**	**VALUE 2**
pEmiCO2Cost	CO2 emission cost	200	500
pMSMax	Maximum Modal Share between years	0.3	0.5
sYearNuc	Nuclear dismantling target year period	2035	2040
CostBat	Battery Cost for EVs and self-consumption (€/kWh)	150.0	50.0
SteelIndustry	Steel Industry Energy Consumption in Spain after 2025, continued or not (re-localization of the industrial plants)	0.0	1.0
CCSU	Carbon Capture, Storage and Utilization favorable scenario in economic terms	0.0	1.0
Offshore	Offshore wind development scenarios	1.0	2.0

Table 3. Description and values of the parameters chosen for the new dashboard

These seven parameters were selected because they represent the most impactful decision levers in long-term energy planning and correspond directly to the inputs that practitioners adjust most frequently. By focusing on these parameters the dashboard ensures that users can explore the scenarios that matter most. Exposing them via intuitive sliders therefore maximizes both usability and analytical insight, allowing decision-makers to rapidly assess the critical trade-offs that underlie sustainable energy policy.

With these concepts as a foundation, the chapters that follows will outline the exact steps undertaken to transform each idea into a concrete implementation, detailing how scenario preloading, interface redesign, callback wiring and validation were orchestrated to deliver a enhanced dashboard that follows the second approach described.

5.3 DASHBOARD DEVELOPMENT

In this chapter, the specific modifications applied to the existing openMASTER dashboard are presented, explaining how the original codebase was refactored and extended to realize the enhancement concepts outlined above. Each section details the architectural and implementation changes—from scenario indexing and control-panel redesign to callback integration and output validation—that collectively transform the legacy tool into the interactive, high-performance "what-if" environment envisioned earlier.

5.3.1 SCENARIO PRELOADING AND INDEXING

After determining that seven user-defined parameters (seven parameters each with two different options) 128 unique scenarios were needed, an automated script might have been written to generate them in one pass. However, because each scenario library is produced only once and then stored, the decision was made to assemble each scenario manually under careful oversight. By handcrafting and validating every folder, rather than relying on a single automated loop, potential coding errors were eliminated, guaranteeing that each of the 128 parameter combinations was represented accurately before being preloaded into the application.

To generate all 128 scenarios, the workflow described in chapter 5.1.2 was followed. Since in future stages scenarios will be selected by the parameter values the scenario holds, it is imperative that scenario names follow some rules to ensure that the selection of scenarios in the future is easy and scenarios can be picked apart from each other. This is the reason why the scenario names will follow the following scheme:

parameter1_value1_parameter2_value2_parameter3_value3_parameter4_value4_parameter5_value5_parameter6_value6_parameter7_value7

With this naming convention, programmatic scenario selection is greatly simplified. The dashboard can parse each directory name, splitting on fixed delimiters, to extract parameter identifiers and values, automatically constructing a lookup table that maps every unique parameter combination to its corresponding folder path. This eliminates the need for external metadata files or hard-coded mappings, reduces boilerplate code, and ensures that scenario retrieval is both reliable and efficient. Consequently, the underlying code remains compact, maintainable, and less prone to errors when accessing any of the 128 precomputed scenarios.

5.3.2 SCENARIO SELECTION AND REACTIVE CALLBACK IMPLEMENTATION

Once the scenarios are generated and named, the next step is to allow the user to select the scenario they are looking for. Before a dropdown menu containing all the scenarios generated by the user was the way of selecting the scenario wanted. The issue is that now the user must select between 128 possible options. This issue generated the need to change the way in which a scenario is chosen for its visualization.

5.3.2.1 Sliders

The previous idea of using textboxes for each scenario cannot be performed since as described before, each parameter has two or three possible values to choose from and looking for a parameter value that does not exist could lead to fails. This means that the project would require explicit validation logic, since any typo, out-of-range number or incorrect format could slip through and cause the data-loading routines to fail or raise exceptions. This led to the use of sliders.

```
        html.Label('Parameter 1', htmlFor='parameter1-input', style={'marginTop':
        '20px'}),
        dcc.Input(
                id='parametro1-input',
                type='text',
                value='text box',
                style={'width': '100%', 'padding': '6px', 'fontSize': '16px'}
        ),
```

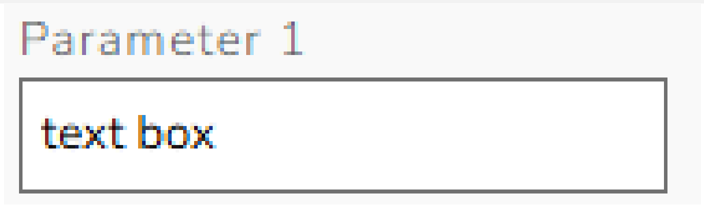


Figure 4. Example of a python textbox, code and visualization of it.

A slider is a horizontal control consisting of a track with a draggable thumb that the user moves left or right to select a value from a predefined set. As the thumb slides, it jumps only to valid points and never permits out-of-range entries. Visually, discrete marks along the track indicate each possible setting, and updating the thumb position immediately reflects the chosen value numerically.

By contrast when comparing to text boxes, sliders snap automatically to the valid, discrete parameter levels, eliminating the possibility of invalid inputs and thus preventing the classes of application faults described before.

```
        html.Label('Parameter 1', style={'marginTop': '20px'}),
        dcc.Slider(
                id='parameter1-slider',
                min=1, max=3, step=1,
                marks={i: str(i) for i in range(1,4)},
                value=1
         )
```

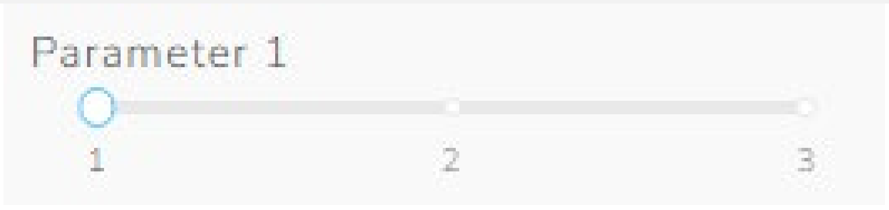


Figure 5. Example of a python slider, code and visualization of it.

The slider approach is also a great way to allow scalability to the project since if in the future more values are wanted to be added to the parameters to create more scenarios it is very easy to add those new values into the slidebar.

5.3.2.2 From slider values to scenario

Given the strict naming convention and the use of sliders for parameter input, scenario selection reduces to a purely deterministic string-construction and lookup process. Each slider corresponds to one of the seven model parameters and is constrained to its discrete, pre-defined levels. When the user adjusts the sliders, the current positions are read in a fixed sequence and concatenated, using the exact delimiters established in the naming scheme, into the full scenario directory name. For example, the values “Low”, “Medium” and “High” from sliders are joined into

parameter1_Low_parameter2_Medium_…_parameter7_High

This constructed name is then appended to the base “scenarios/” path, yielding the complete file-system address of the desired run. Because the directory structure was pre-populated with one folder per valid combination, each named according to the same pattern, the dashboard should immediately locate the matching folder, read its CSV outputs, and render the corresponding visualizations. In this way, slider movements should translate directly into folder lookups, with zero ambiguity or intermediate metadata tables, ensuring both simplicity of implementation and reliability in scenario retrieval.

To ensure that the dashboard renders the scenario matching the parameters each slider has in responsive way the use of dash @app.callback was implemented.

5.3.2.3 Dash Callbacks

Within the Dash framework, callbacks establish a reactive link between user interface components and data-processing logic. Each callback is declared via a decorator that specifies input properties and output properties. When the framework detects a change in any Input, it serializes the event data to the server, executes the associated Python function, and updates the designated Output, all without a full page reload. This pattern ensures sub-second responsiveness and maintains a clear separation between UI definition and computational routines.

```
#Tab switching (Energy / Emissions / Comparator)

@app.callback(
    Output('tabs-content', 'children'),
    Output('energy-content', 'style'),
    Output('emissions-content', 'style'),
    Output('comparison-content', 'style'),
    Output('comparison-dropdown', 'style'),
    Input('tabs', 'value')
)
def render_content(tab):
    if tab == 'tab-1':
      return None, {'display': 'block'}, {'display': 'none'}, {'display':     '
            'none'}, {'display': 'none'}
    elif tab == 'tab-2':
        return None, {'display': 'none'}, {'display': 'block'}, {'display':    '
            'none'}, {'display': 'none'}
    else:  # 'tab-3'
        return None, {'display': 'none'}, {'display': 'none'}, {'display':     '
            'block'}, {'display': 'block'}
```

Figure 6. Example of a dash callback.

In figure 6 an example of a callback used on openMaster's dashboard is shown. In this code several outputs can be identified and only one input is found. The input is linked to the tab value. The tab value can switch between 1, 2 and 3. Depending on the value of the input the output varies, showing the "energy", "emissions" or "comparator" tab depending on which tab is selected without the necessity of reloading the page.

Knowing this callback will provide an essential function to link the value of every slider with the graphs rendered in the dashboard within a very brief time. To create the link two callbacks are used, one for single scenario graphs and another for the comparation tab.

On the first callback since the graph update is only required when the parameters change value. The slider values will serve as input and as output every graph must update so every graph will be an output. Now the way in which parameters and graphs relate is that due to the naming rules described before, a string with the scenario that is wanted can be generated. Using this the data of that specific scenario is loaded into the dashboard allowing the rendering of every graph using the data of the scenario described with the sliders.

The second callback works basically as the previously described but with a little twist. Since graph comparisons between the scenarios are done graph by graph, an extra input is needed, the type of graph input is used to allow the user to compare the graph that is specifically wanted.

5.3.3 NEW UI DESIGN

With the selection mechanics and reactive engine firmly established, attention now turns to the user interface itself. A well-crafted UI is essential not merely for aesthetics but for enabling analysts to engage with complex data effortlessly. In this section, the guiding principles behind the new interface are outlined principles that prioritize clarity, intuitive navigation and visual consistency to ensure that the dashboard remains both powerful and approachable. By placing usability at the forefront, the redesign aims to transform raw functionality into a seamless analytical experience.

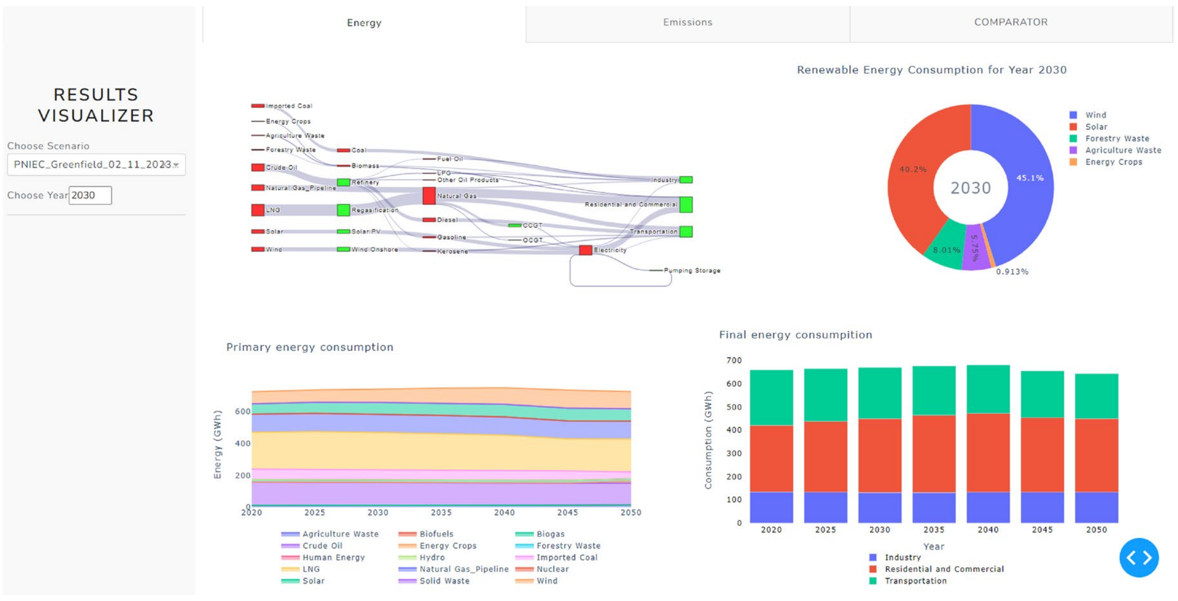


Figure 7. Initial openMaster Dashbard

Previously, the interface relied on just two dropdown selectors, one for scenario and one for year selection, but this minimal approach constrained user control and obscured the underlying parameter space. The redesigned UI replaces those broad controls with an individual slider for each of the seven model parameters, offering fine-grained adjustment within clearly defined levels. This shift not only exposes the full dimensionality of the scenario library but also aligns the interface with the reactive architecture, enabling immediate feedback on every parameter change.

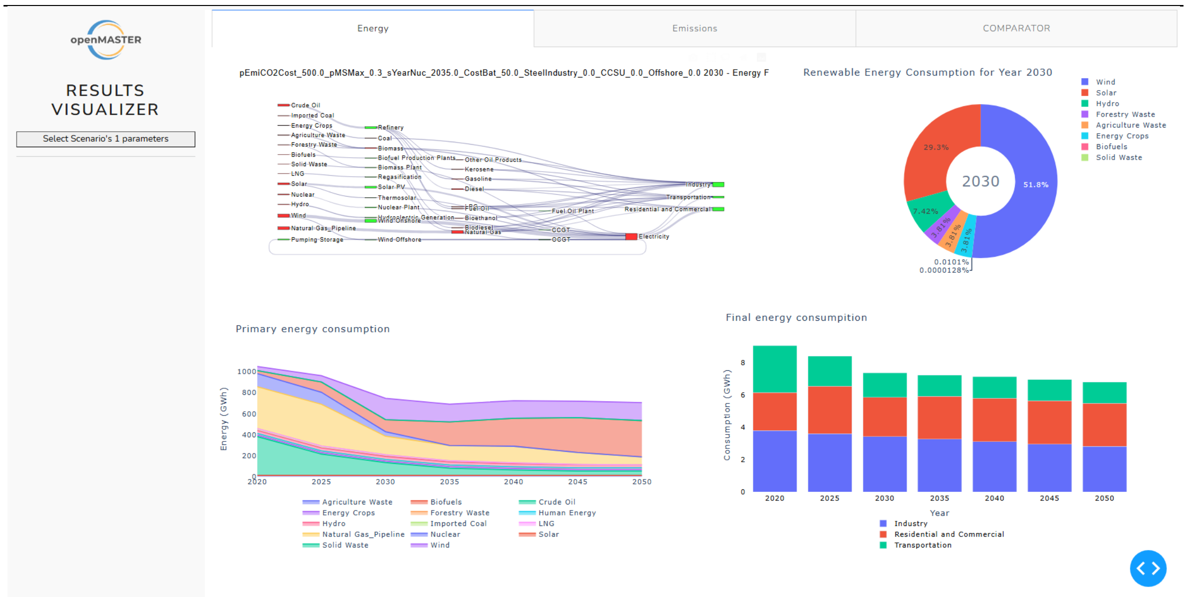


Figure 8. New openMASTER dashboard.

Two significant usability challenges emerged during the redesign of the control sidebar. First, the sheer number of parameter sliders—seven in single-scenario mode and up to fourteen in the comparison view—exceeded the fixed vertical space allotted in the original layout. In order to preserve a consistent two-column structure without reducing slider size or truncating labels, a vertical overflow mechanism was introduced. This allows the entire left-hand panel to scroll independently, ensuring that all controls remain accessible while the visualization area retains its full height and clarity.

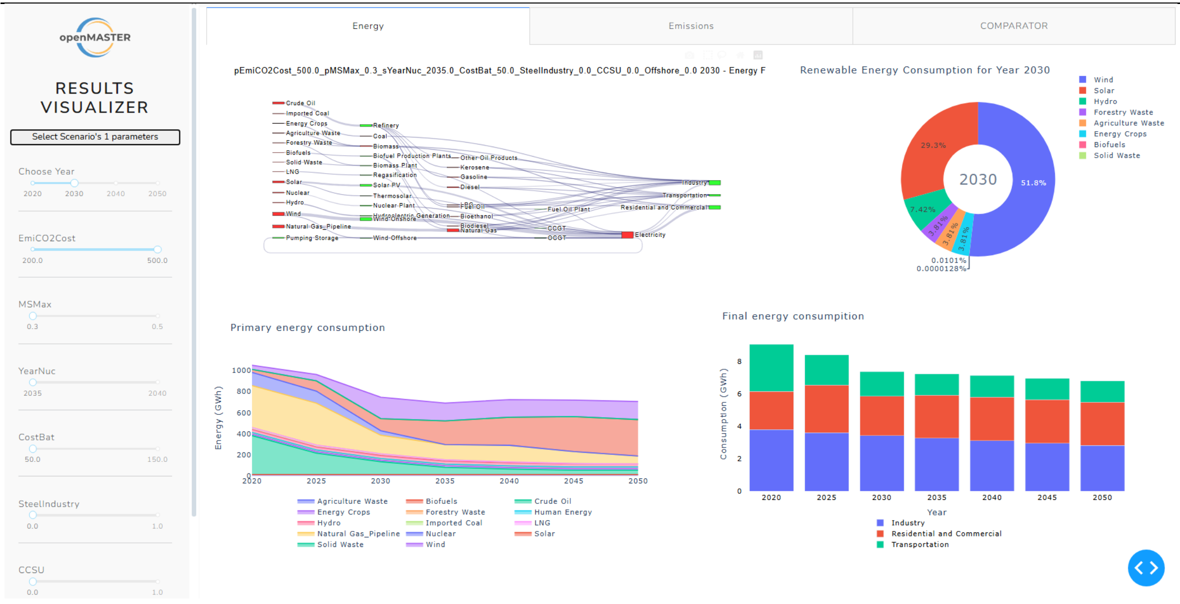


Figure 9. Dashboard in a smaller monitor where a scrollbar appears.

Second, the comparison tab in particular risked overwhelming users with fourteen sliders visible simultaneously, undermining the goal of a clean, focused interface. To address this, collapsible sections were implemented: a toggle button now shows or hides each group of seven sliders corresponding to a particular scenario. By default, only the controls for the primary scenario are displayed, with the secondary scenario's sliders revealed on demand. This grouping mechanism not only declutters the screen but also directs the user's attention to one scenario at a time, reducing cognitive load and facilitating a more orderly exploration of paired parameter combinations.

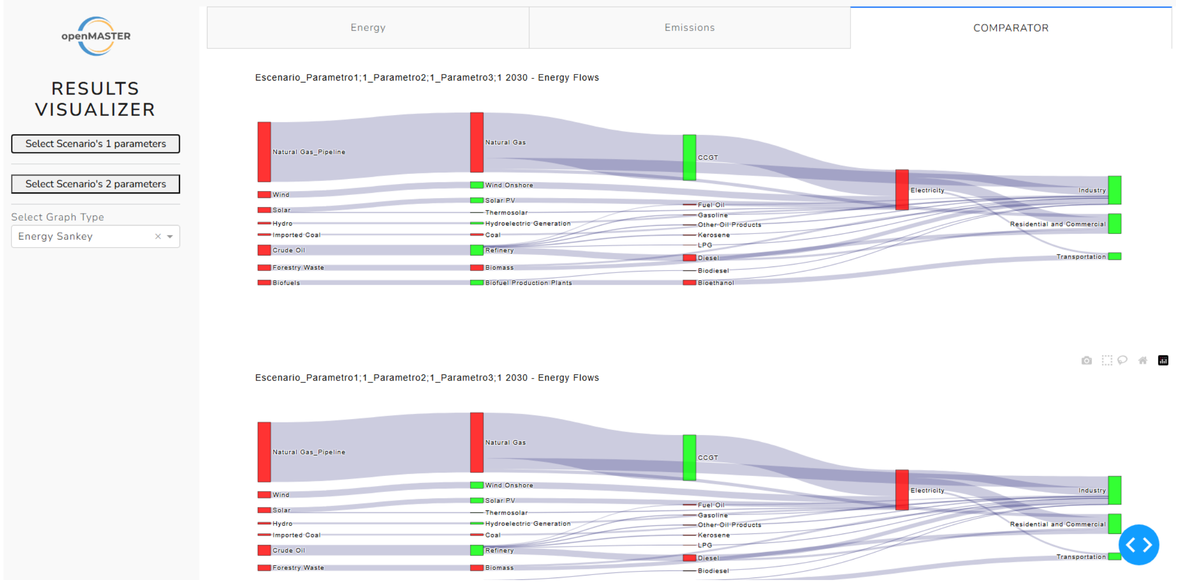


Figure 10. New dashboard comparator tab.

5.3.4 VALIDATION

Validation is a critical step in ensuring that the enhanced dashboard fulfills its design objectives—streamlined parameter selection, responsive visualization and unbroken legacy behavior—before it is declared complete. A combination of targeted tests was devised to verify each aspect of the implementation.

5.3.4.1 Indexing Accuracy Tests

Automated unit tests exercise the metadata parser and in-memory index by supplying a representative set of seven-parameter tuples and asserting that each one maps exactly to its precomputed scenario folder. Any mismatch or missing entry immediately triggers an alert, guaranteeing that the slider-to-folder lookup mechanism is both exhaustive and error-free.

5.3.4.2 UI Integrity Checks

Following the layout refactoring, integration tests inspect the rendered interface across a range of screen sizes and workflows. These tests confirm that all original menus, buttons and chart containers remain visible and functional, that the scrollable sidebar grants access to every slider, and that collapsible comparator sections expand and collapse without overlapping or displacing legacy elements.

5.3.4.3 Callback Workflow Verification

Automated scripts simulate user interactions—adjusting each slider in isolation and in combination, toggling comparator controls, and switching tabs—and then verify that the associated Dash callbacks execute exactly once per change. For each simulated event, the tests confirm that the correct DataFrame is loaded, that the appropriate Plotly figure is generated or retrieved, and that autogenerated chart titles and labels reflect the active slider values. Browser console logs are also scanned to ensure that no JavaScript errors occur during these updates.

Together, these validation activities provide high confidence that the dashboard operates reliably in real-world use, that all new features behave as intended, and that no existing functionality has been compromised.

5.3.5 DEVELOPMENT OF A NEW USER GUIDE

Following validation, the creation of an updated User Guide becomes indispensable. Since the guide must describe the finalized interface and workflows that have been thoroughly tested, it ensures that end-users receive accurate, up-to-date instructions reflecting the dashboard's confirmed behaviour. This alignment minimizes confusion, prevents misuse, and accelerates onboarding by providing clear, validated procedures. Moreover, a revised User Guide serves as an authoritative reference for training new analysts and as a baseline for future maintenance or extension efforts. In this way, documentation completeness and accuracy reinforce user confidence and support the seamless adoption of the enhanced tool in operational settings. The new user guide developed can be found in annex II.

Capítulo 6. CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

The enhancements described in this work fundamentally improve the openMASTER dashboard from a static, manual reporting tool to a dynamic, interactive platform tailored for rigorous energy policy scenario analysis. By precomputing the full space of 128 scenario runs and embedding parameter values directly into the file-system hierarchy, the traditional friction of on-demand model execution and error-prone CSV handling has been eliminated.

Users can now define any combination of the core parameters via intuitive sliders, triggering instantaneous scenario retrieval and visualization through Dash’s reactive callback system. This architecture achieves very fast update times across multiple chart types, Sankey diagrams, bar, pie, area plots and emission time series, all this without incurring page reloads or manual file operations.

Beyond raw performance gains, the refactored control sidebar and collapsible comparator panels significantly improve the user experience. The scrollable layout accommodates all seven sliders without compromising screen estate, while the grouped toggles in comparison mode prevent cognitive overload when inspecting two scenarios side-by-side.

These interface refinements, coupled with rigorous manual validation (covering indexing accuracy, UI integrity and callback workflows), ensure both robustness and usability. Importantly, by preserving existing menus, tabs and charting functions, the redesign maintains full backward compatibility with legacy workflows, facilitating a smooth transition for established users.

From a broader perspective, this work demonstrates the value of combining precomputed scenario indexing with modern web-based visualization frameworks to support rapid, reproducible energy-system exploration. The all-Python, container-friendly implementation enhances transparency and traceability, qualities that are essential for academic research, governmental policy development and industrial decision support. In delivering an end-to-end "what-if" ecosystem, the enhanced dashboard empowers analysts to probe policy trade-offs more deeply, engage stakeholders with immediate visual feedback and iterate on assumptions without technological barriers. As energy models grow increasingly complex, this approach offers a template for integrating large-scale optimization with user-centric interfaces, ultimately accelerating the pace and confidence of sustainable energy planning.

6.2 FUTURE WORK

Looking ahead, the next natural step is to deploy the enhanced dashboard on a centralized server to facilitate broad, concurrent access and eliminate the need for local environment setup. Containerization, using Docker or similar technologies, will encapsulate all dependencies, pre-computed scenarios, and licensing configurations, ensuring identical behavior across development, testing, and production.

A web-accessible instance could be hosted on cloud platforms for example AWS behind secure authentication, allowing analysts and stakeholders to explore scenarios through any modern browser. Additional future enhancements may include role-based access control, automated scenario generation pipelines for ad-hoc parameter sweeps, and integration with real-time data APIs. By moving from a local prototype to a networked service, the openMASTER dashboard can evolve into a collaborative, enterprise-grade application that further accelerates transparent, reproducible energy policy design.

Such a deployment would obviate the need for any local installation, allowing users to access the dashboard directly through their web browser and dramatically lowering the technical barrier to entry.

Capítulo 7. BIBLIOGRAPHY

[1] Python Software Foundation. "Python Blurb," Python.org. https://www.python.org/doc/essays/blurb

[2] Project Jupyter. "Jupyter," Jupyter.org.: https://www.jupyter.org

[3] Pyomo Project. "Pyomo: Python Optimization Modeling Objects," Pyomo.org. https://www.pyomo.org

[4] Gurobi Optimization, LLC. "Gurobi Optimizer Brochure," Gurobi. https://assets.gurobi.com/pdfs/Gurobi-Optimizer-Brochure.pdf

[5] Plotly. "Dash: A Python Framework for Building Analytical Web Applications," Dash Documentation. https://plotly.com/dash

[6] Plotly. "Plotly.py: Python Open Source Graphing Library," Plotly. https://plotly.com/pytho

[7] Community Open-Source Contributors. "Dash Bootstrap Components," Dash-Bootstrap-Components. https://www.dash-bootstrap-components.com

[8] IEA-ETSAP. "TIMES Model Overview," GitHub. https://github.com/etsap-TIMES/TIMES_model?tab=readme-ov-file#readme

[9] UNFCCC. "MESSAGE Energy Model," UNFCCC Modelling Tools. https://unfccc.int/topics/mitigation/workstreams/response-measures/modelling-tools-to-assess-the-impact-of-the-implementation-of-response-measures/energy-models?#MESSAGE

[10] OSeMOSYS Community. "OSeMOSYS Manual – Introduction," OSeMOSYS Documentation. https://osemosys.readthedocs.io/en/latest/manual/Introduction.html

[11] OSeMOSYS Community. "OSeMOSYS Documentation," OSeMOSYS Read the Docs. https://osemosys.readthedocs.io/en/latest

[12] Calliope Development Team. "Calliope: Building Models," Calliope Documentation. https://calliope.readthedocs.io/en/stable/user/building.html

[13] Wikipedia Contributors. “Pyomo,” Wikipedia. https://en.wikipedia.org/wiki/Pyomo

[14] Pyomo Documentation. “DataPortal Class,” Pyomo Read the Docs. https://pyomo.readthedocs.io/en/6.8.0/working_abstractmodels/data/dataportals.html

[15] Pyomo Documentation. “Solver Interfaces and Experimental Solvers,” Pyomo Read the Docs. https://pyomo.readthedocs.io/en/6.9.1/explanation/experimental/solvers.html

[16] Tableau Software. “Tableau Pricing,” Tableau.com. https://www.tableau.com/pricing

[17] Beehexa. “What Is Power BI Pricing 2024,” Beehexa Blog. https://www.beehexa.com/blog/what-is-power-bi-pricing-2024

[18] Plotly. “Dash Basic Callbacks,” Dash Documentation. https://dash.plotly.com/basic-callbacks

[19] Streamlit. “Streamlit Fundamentals,” Streamlit Docs. https://docs.streamlit.io/get-started/fundamentals

[20] Bokeh Development Team. “Bokeh WebGL Support,” Bokeh Documentation. https://docs.bokeh.org/en/dev-3.0/docs/user_guide/webgl.html

ANEXO I: ALIGNMENT OF THE PROJECT WITH THE SUSTAINABLE DEVELOPMENT GOALS

This project directly advances three key SDGs by strengthening the tools and workflows available for sustainable energy planning:

Figure 11. Sustainable Development Goals

SDG 7: Affordable and Clean Energy

By streamlining the process of designing, comparing and iterating energy-policy scenarios, the enhanced dashboard enables rapid evaluation of low-carbon portfolios. Analysts can instantly test combinations of renewable penetration, efficiency measures and demand-side initiatives—identifying least-cost pathways that expand access to modern energy services while minimizing environmental impacts. This capability supports the formulation of policies that accelerate the transition to cleaner, more efficient energy systems.

SDG 9: Industry, Innovation and Infrastructure

The integration of open-source optimization (Pyomo/Gurobi) with an interactive, web-based visualization stack (Dash/Plotly) represents a significant upgrade to the digital infrastructure underpinning energy analysis. By unifying modeling, data handling and visualization in a single, reproducible Python environment, the project fosters technological innovation and lowers barriers to entry for researchers and practitioners. This robust, scalable framework lays the groundwork for resilient energy infrastructures that can adapt to evolving technological and policy landscapes.

SDG 13: Climate Action

Equipped with built-in emissions constraints and carbon-budget scenarios, the dashboard empowers stakeholders to explore mitigation strategies in real time. Decision-makers can directly compare the economic and social costs of alternative decarbonization pathways to identify cost-effective routes toward greenhouse-gas reductions. This real-time "what-if" capability strengthens evidence-based climate action by making it easier to design, assess and communicate sustainable energy strategies that align with global climate targets.

ANNEX II: NEW OPENMASTER DASHBOARD USER GUIDE

A. Installation & Setup

Begin by cloning or unpacking the openMASTER repository into your working directory. A Python 3.10 (or later) environment is required. From within the project root, create and activate a virtual environment:

```
python                     -m                      venv                       .venv
.venv\Scripts\activate
```

With the environment active, install the core package along with graphical dependencies for the dashboard:

```
cd                                                                         openMASTER
pip install -e .[graphicTool]
```

Ensure that the input data file openMASTER_Data_2050_Gap10.xlsx resides in data/input/. If it is missing, retrieve it from the project's raw download link and place it in that folder. For Gurobi support, set the GRB_LICENSE_FILE environment variable according to Gurobi's academic license instructions. Finally, verify solver connectivity by launching a Python REPL and executing:

```
import gurobipy
```

B. Getting Started

To launch the interactive dashboard, run:

```
python index.py
```

This command starts a local web server (default port 8080) and opens the interface in your browser. The window is divided into two main regions:

Control Sidebar (on the left) contains seven vertically stacked sliders, one per model parameter, housed in a scrollable container. In Comparator tab, a toggle button reveals or hides a second group of seven sliders.

Visualization Canvas (on the right) offers three tabs:

Energy tab displays Sankey diagrams, bar charts, pie charts and area plots for the selected scenario.

Emissions tab shows sector-wise CO_2 emission time series.

Comparator tab presents two side-by-side figures of any chart type, allowing direct comparison of two distinct parameter sets. Note that the displayed charts can be selected with the dropdown menu.

C. Use-Case Tutorials

Single-Scenario Analysis

Adjust all seven sliders in the sidebar to define your scenario. Switch to the Energy tab to view energy flows, generation mix, consumption breakdown and renewable share. Move to Emissions to examine CO_2 trajectories. All charts update instantly in response to slider movement.

Side-by-Side Comparison

Select the Comparator tab and click the toggle button to display the second slider group. Configure each set of seven sliders for two different parameter combinations. The paired graphs will render simultaneously, each labelled with its parameter values, enabling immediate visual comparison.

Parameter Sensitivity Scan

With six sliders fixed at baseline values, vary a single slider through its three levels. Observe how charts in the Energy or Emissions tabs respond, revealing the model's sensitivity to that parameter. Repeat for each parameter of interest without leaving the dashboard.