MAGPIE

SMART GREEN PORTS

Digital Twin Platforms and services(initial version)

Funded by the European Union

This project has received funding from the European Union's Horizon 2020 (MFF 2014-2020) research and innovation programme under Grant Agreement 101036594

Digital Twin Platforms and Services (initial version) D4.5

Grant Agreement No.	101036594
Start date of Project	1st October 2021
Duration of the Project	60 months
Deliverable Number	D4.5
Deliverable Leader	INESC
Dissemination Level	PU
Status	1.1
Submission Date	13-5-2024
Author	Yves-Marie Bourien, CEA, yves-marie.bourien@cea.fr Eric Francois, CEA, eric.francois@cea.fr Racem, CEA André Lisboa, EDP, andre.lisboa@edp.pt Gloria Goncalves, EDP, gloria.goncalves@edp.pt Gonçalo Calado, EDP, goncalo.calado@edp.pt João Megre, EDP, joao.megre@edp.pt Zenaida Mourão, INESC TEC, zenaida.mourao@inesctec.pt Karol B. Gonçalves, INESC TEC, karol.b.goncalves@inesctec.pt Tomás Rocha, INESC TEC, tomas.rocha@inesctec.pt Adrian Galvez, INESC TEC, adrian.c.galvez@inesctec.pt Ilia Ponomarev, INESC TEC, ilia.ponomarev@inesctec.pt

Jorrit Harmsen, TNO, jorrit.harmsen@tno.nl
Verônica ghisolfi, TNO, veronica.ghisolfi@tno.nl
Cornelis Bouter, TNO (cornelis.bouter@tno.nl)
Wouter Korteling, TNO
(wouter.korteling@tno.nl)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101036594.

Modification Control

Version #	Date	Author	Organisation
V1.0	14-03-2024	All Authors	CEA, INESC, EDP, TNO
V1.1	26-04-2024	All Authors	CEA, INESC, EDP, TNO

Release Approval

Name	Role	Date
Zenaida Mourao	WP Leader	26-04-2024
Gunnar Platz	Peer Reviewer	04-04-2024
Jorrit-Jan Serraris	Peer Reviewer	28-03-2024
Arne-Jan Polman	Project Coordinator	13-5-2024
Maarten Flikkema	Scientific Coordinator	29-04-2024

History of Changes

Section, page number	Change made	Date
		DD-MM-YYYY
		DD-MM-YYYY
		DD-MM-YYYY
		DD-MM-YYYY

Table of Contents

Executive Summary

This document details the current advancements in Task T4.5 within Work Package 4 (WP4) of the MAGPIE project, focusing on the development and deployment of digital tools, models, and services aimed at enhancing sustainability in port energy use and the transport supply chain. As outlined, Task T4.5 will unfold its deliverables across two key stages:

- Deliverable 4.5: Deployment of the initial digital tools and services at Month 30 (M30).
- Deliverable 4.6: Improved version of the digital tools that compose the digital twin and support the respective use-case at Month 48 (M48), concluding T4.5 and WP4.

Currently, at M30, the emphasis is on Deliverable 4.5, featuring:

- Detailed functionalities, structural design, dependencies, use cases, user interfaces, deployment status, and development timelines for critical tools such as the GHG Tooling, Energy Matching Tool, and Smart and Green Logistics Tool.
- Back-end model implementations supporting these tools, including development status, model structure, software/hardware dependencies, access to codes/models, installation instructions, and example use cases.
- An update on the deployment and development timelines of the data-sharing infrastructure, digital twin, and associated services.
- The definition of an initial set of use cases related to the implementation of the ontology and data-sharing architecture, as key elements of the digital twin and digital services, including:
    - Using real-time data provided by the digital twin to inform and provide added value for updating the charging strategy of electric trucks (linked to Demo 9);
    - Providing an automatic generation framework capable of generating a core ontology that represents the principal high-level concepts related to transport and logistics activities in the MAGPIE port, or support specific digital services and tools (linked to the implementation of digital tools in HAROPA port);
    - Facilitating the integration of the backend models (e.g., renewable forecast, demand estimation, flexibility) and port logistics information with the Energy Matching Tool (EMT), through their connection to the data sharing infrastructure, applied to MAGPIE ports (linked to the development of the EMT tool, with potential application to the Port of Sines and DeltaPort).

1 Introduction

This document provides an overview of the current progress of Task T4.5 in WP4 which involves the development of various digital tools, models, and digital services. It complements the T4.5 deliverables, including the code, apps, platforms, data-sharing infrastructure, and executables. As per the original objectives of the task (for a full description, refer to Annex 1), the outputs of Task T4.5 will be rolled out in two stages:

- Deliverable 4.5: The initial version of the digital tools and services in M30,
- Deliverable 4.6: Improved version of the digital tools that compose the digital twin and support the respective use-case. to be delivered in M48 at the end of T4.5 and WP4.

This document focuses on the current state of implementation (M30) of the digital tools and services (D4.5). It includes:

- A description of the functionalities, tool structure, dependencies, use cases, user interfaces, c4.5urrent state of deployment, and development timeline for the GHG tooling, Energy Matching Tool, and Smart and Green Logistics Tool,
- Implementation of the back-end models that support the three main tools. This section summarises the state of development, model architecture, software and hardware dependencies, access to the code and models, installation instructions, example use cases and development timeline,
- An overview of the current deployment status and development timeline for the data sharing infrastructure, digital twin, services, and respective use cases.

1.1 Context and objectives

The **implementation of the digital tools** in MAGPIE is planned as a **3-tiered approach**. The **digital sharing infrastructure** described in deliverables D4.2 and D4.3 constitutes the **first tier**, providing the main infrastructure that supports the collection and storage of data, produces meaningful insights for decision-making, creates the protocols to share data between stakeholders and tools and standardises how information is shared and used. The **second tier** includes the **intelligence and analysis layer** that takes raw data to produce the inputs and models that have been described in detail in deliverable D4.4. These tools and back-end models produce inputs and support the **third tier digital tools and services** under development in task T4.5 to enable sustainable port energy use and operations, and a greener transport supply chain.

The main objectives of task T4.5, according to the description of the task as presented in Annex 1, are the following:

- Implement digital twin platforms and services for different ports.
- Develop modelling and prediction capabilities to facilitate emission reduction related to efficiency of operations, fuel, and modal shift at operational and strategic levels (GHG tooling, Subtask 4.5.1.1).
- Deliver an energy matching platform to balance the need for supplying loads with existing renewable energy sources in different segments of ports (Energy Matching Tool, Subtask 4.5.2).
- Deliver an integrated decision support system that enables the port to deploy and manage more efficient, reliable, environmentally sustainable, and less energy-consuming operations (Green and Smart logistics tool, Subtask 4.5.3).

- Implementation of WP4 backend models that support MAGPIE digital tools, as outlined in deliverable D4.4 and shown in Figure 1.

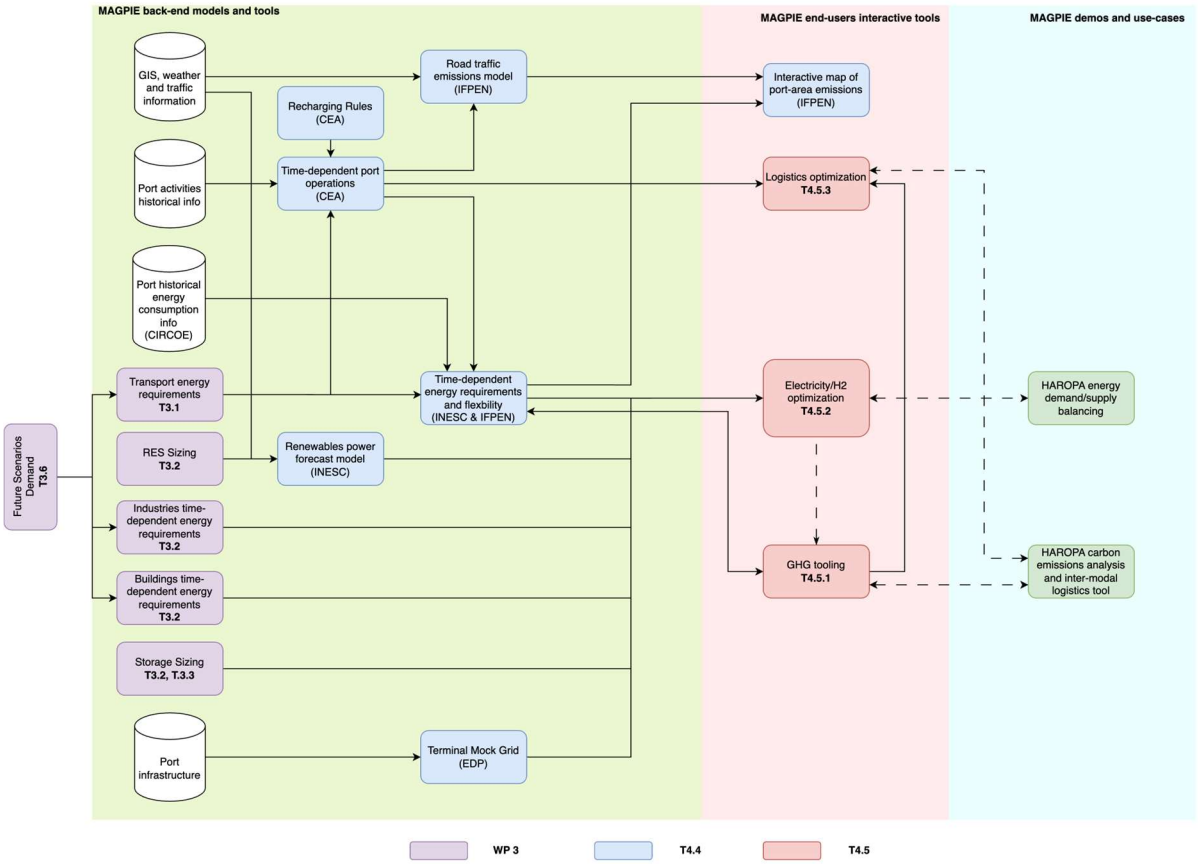


Figure 1 - Overall MAGPIE back-end models and tools architecture

1.2 Work Package Dependencies

The activities carried out generally within WP4 interact with several activities and outputs in other MAGPIE work packages. For the work developed in Task 4.5, the following links are relevant:

- WP4 is closely connected with WP3, as mapped in Figure 1. Particularly,
  - The work carried out in Task 3.1, which quantified the current and future energy demand of transport modalities, provides inputs on emission factors of different technologies for oceangoing vessels, inland water shipping, trains, and road freight. This data can be used as input for the GHG tooling and the Smart and Green Logistics tool. As deliverable D3.1 also covered future technology shifts, the data can be used for strategic planning and testing decarbonisation scenarios.
  - The work carried out in tasks T3.2-T3.3, which quantified the electricity and hydrogen supply and demand of transport modalities, buildings, and industries in port areas, will be used within the Energy Matching Tool. This work has been described in deliverables D3.2 and D3.3. The models and data generated within WP3 will be used to generate time-dependent demand

models for buildings, industries, and transport modalities that complement the terminal-oriented back-end models under development. One of the main differences is that while the energy supply models in WP3 focus mainly on planning and sizing of future low-emissions and renewable energy production, the digital tools, and back-end models under development in WP4 focus also on operational decision-making and optimization. This means, for example, that the renewable energy forecast models and the energy demand models described in this deliverable focus on the forecast for periods of days or weeks, with high temporal resolution and not years as do the modelling of supply and demand in WP3.
    - Models for sizing future renewable capacity (e.g., PV, wind) are used as input for the renewable generation forecast, especially since the physical-based models proposed in D3.2 are being used to generate synthetic data for training the data-based forecast models in the absence of site-specific real-time data.
    - The future demand scenarios under development in T3.6 provide information on the decarbonisation pathways of transport supply chains and port activities. These can be used to run additional scenarios in the terminal simulator tool to generate the time-dependent demand profiles, emissions, and flexibility potential with and without EES. These will be used as input for the digital tools under development in T4.5.
    - Finally, the WP3 demos, especially demo 3 (Shore Peak Power shaving) and demo 2 (Smart Energy Systems) could provide data and model parameters relevant for the back-end models presented in this document to generate the time-dependent demand profiles, flexibility modelling (with and without EES), and the terminal (mock) grid, that provide input for the Energy Matching Tool.
- Demos and models under development in WP5 and WP6, especially for ongoing work in demo 7 (Green energy container), demo 9 (Green Connected Trucking) and demo 10 (Spreading road traffic) provide use cases of the implementation of the back-end models and for the implementation of the data-sharing infrastructure. Also, it would be beneficial to compare some of the models from WP5 and WP6 with the ones in development T4.5, to ensure consistency in the use of input parameters and mathematical formulation of e.g., storage, energy demand and energy supply of specific systems/areas – e.g., trucks and charging in demo 9, battery storage in IWT and demand of IWT in demo 7, strategies for reduction of emissions through management of road freight in demo 10. Finally, the links to the transport model under development in task T6.4 will also be explored.
- In terms of the non-technical barriers, work of WP7 strategies for differentiated and dynamic tariffs for electricity for OPS systems or to encourage shifting consumption to fit local renewable generation could be relevant to the uptake and modelling of energy demand and supply in the back-end models and the Energy Matching tool. Similarly, additional use cases and scenarios resulting from the outputs of WP7 could be run in the logistics, energy, and emissions back-end models.
- Some of the data being collected and used for KPI monitoring in WP8 could be used as input to the back-end models that support the digital tools. Additionally, some of the KPIs could be used as additional analytics implemented in the digital twin
- The outputs and use cases of implementing the digital tools under different scenarios of decarbonisation of ports and transport supply chains may be relevant for the Master Plan under development in WP9. Additionally, these could inform the current work on the Vision Elements, particularly those in Groups "New World Energy", "Smart and Efficient", "Future Proof Business Models", and "Nature Positive".

- Finally, the continuing relationship with WP10 involves exchanging practices and exploring synergies with other sister projects in terms of the contribution of port digitalisation to decarbonising operations.

1.3 Outline

The status and next steps of development of the digital tools, backend models, the data-sharing infrastructure and digital twin are described in Chapters 2, 3 and 4, respectively.

Chapter 2 briefly outlines the current state of development of the digital infrastructure, which includes the data models and data-sharing infrastructure that will support the implementation of digital tools and services. It also provides an overview of some of the implementation use cases.

Chapter 3 describes the development of the digital tools by presenting the context, state of the art, objectives, functionalities, model structure, data inputs and outputs, the user interface, and the use cases/applications of the tools. A development timeline is also provided for each tool, as well as any major milestones.

Chapter 4 follows from the detailed description of the backend models presented in deliverable D4.4 and is intended to complement the implementation of these models in code. This chapter provides information on the structure of the platforms, apps or code resulting from the implementation of the backend models, including the hardware and software dependencies, a brief explanation of the functionalities and purpose, the structure of the code implementation, instructions for installation and use of the code/app/platform, examples of application and, in some cases, synthetic that can be used to train the models. This section also explains how end users can access the code, platforms or apps.

Finally, Chapter 5 outlines next steps in the development of the digital tools and services.

2 Digital Twin & Services

Deliverables D4.2 and D4.3 have specified a data-sharing architecture, core language specification and methodology for an extension that can be reused in use cases to share data in a trusted and semantically interoperable way. Specifically, D4.2 sets out a data sharing architecture that facilitates trusted data sharing between stakeholders by incorporating International Data Space (IDS) connectors. In D4.3, a generic modular ontology for the port was presented that includes details on various logistical assets and activities and the methodology for extending it.

This work is intended as generic, use-case-independent. The design choices were informed by the domain use cases reported in D4.1, but the architectural digital tools can also be applied to other use cases and demos that have yet to be identified. The MAGPIE description of work explicitly offers this extensibility to future demos and tools.

Therefore, any digital service can be supported in the digital tools environment, but the service owner has to decide whether integrating with the digital tools is useful for the service. Adopting the data sharing infrastructure and the ontology takes some additional work, which can be most profitable if the service involves sharing and alignment between various data sources.

The so-called "digital twin platform" would then comprise the integration of semantic data sharing among various data sources, consisting of data spaces for trust and semantic interoperability for data sharing. The services can then plug into the infrastructure to gather the data they need for their digital twin services.

The work still to be done within MAGPIE is reusing the core language specification and extending it in a modularized way to accommodate the data requirements in the tools in WP4 and, if relevant, demo use cases in other WPs. To do this, the data requirements need to be specified by the tool builders and demo owners. The current deliverable (D4.5) can partially accommodate that. With this initial delivery of the tools and backend models, the next step consists of further delineating the exact requirements. The tools and use cases that are currently on the radar for this extension include an application for demo 9, the implementation of the data sharing architecture and ontology to facilitate the integration of energy and GHG emission tools and models for HAROPA port and, finally, the implementation of the Energy Matching Tool, to facilitate the integration with backend models and port logistics databases. Other use cases may extend the core modules as well, and may be implemented, especially as outlined in deliverables D4.1, D4.2 and D4.3.

By the end of M48, an extended version of the modular language specification based on the requirements set out in these use cases should be available as a major output of this work.

The MAGPIE use cases that are currently envisioned to provide input for extending the MAGPIE core language specification are described below.

2.1 Use case 1: Implementing the data sharing architecture and ontology within the scope of Demo 9

The charging of heavy-duty electric trucks is likely to require both planned and unplanned stops at public charging stations during daily operations. It is expected that companies will reserve charging slots in advance for planned stops. However, due to the dynamic nature of logistics, deviations from planned routes often occur, leading to missed reservations and the need for unplanned charging, where no prior slot bookings are made.

To address these challenges, the proposed digital twin, and especially the data sharing infrastructure as developed in WP4, aims to provide a solution by harnessing real-time data that will be collected as part of Demo 9. This data encompasses the current routing plans, battery charge status, and the availability of charging slots at public stations. By integrating this real-time information, the digital twin can facilitate an adaptive charging strategy for electric trucks, significantly enhancing operational flexibility and efficiency.

The focus within this use case is to delineate specific services described by a dedicated ontology and utilize the data sharing infrastructure to deliver these real-time updates. This effort builds on top of the Demo 9 operation where heavy duty electric trucks are being monitored, analysed, and evaluated in detail.

The anticipated outcome is that the real-time data provided by the digital twin will not only inform but can also provide added value for updating the charging strategy of electric trucks. Depending on the outcomes, follow-up actions will be determined. It is expected that the real-time data shared via the digital twin will support the output of real-time data planning, battery management and identifying public charging locations.

2.2 Use case 2: Implementing the data sharing architecture & ontology to facilitate the sharing of data between CEA tools

As explained in the deliverable D4.2 and D4.3, the operationalization of a Digital Twin (DT) of the port requires three components: (1) a data sharing infrastructure, (2) a language specification of the data that is available and shared and (3) (intelligent) tools and systems that produce and consume data.

A language specification is required for the operationalization of a Digital Twin (DT) that provides a format for stakeholders to share and integrate data. This language specification ensures the interpretability and interoperability of the different systems and applications that are integrated into the DT.

To present a language specification for the DT of the port, TNO follows the approach by the Port of Rotterdam Digital Twin vision presented in section 2 of D4.2 and uses semantic models, also known as ontologies. In a nutshell, ontologies are formal representations of knowledge within a specific domain as a set of concepts and the relationships that hold between these concepts. They are suitable to achieve the process of exchanging and integrating data from various sources. From this perspective, ontologies can be viewed as a data language that stakeholders can use to communicate.

To be developed, TNO proposes a generic overarching core model that describes the principal high-level concepts related to transport and logistics activities in the port. This core model is extended modularly by reusing a wide range of more domain-specific models that describe knowledge in sub-domains within the transport and logistics field, such as information on particular transport modalities. The core model and its modular extensions can be reused and possibly further extended by parties possessing expertise on the domain of their specific use case or tool. As a starting point, the FEDeRATED [1] ontology is used to support interoperability between a broad range of domain-specific ontologies in the transport and logistics sector and it is aligned with several models for the various logistic modalities such as SAREF4AUTO [2], the ERA vocabulary [3], SAREF4ENER [4] etc. as presented in Figure 2.

We detail as example:

- **The FEDeRATED ontology** is developed by the Digital Transport and Logistics Forum (DTLF) and contains several modules covering conceptually separate sub-domains of information, namely a Digital Twin, Physical Infrastructure, Business Service, Classifications and Event module.
- **The Smart Applications REFerence (SAREF)** ontology has been developed in accordance with standardization frameworks from ETSI. Specialization of this ontology such as SAREF4AUTO provides a detailed model for the automotive domain and SAREF4ENER focuses on the energy domain.
- **ERA Vocabulary KG** is governed by the European Union Agency for Railways. It models the European railways infrastructure as well as the vehicles that operate over it.
- **EU KG and Kadaster** [5] is developed by the Netherlands' Kadaster Land Registry and Mapping Agency (in short "Kadaster") and integrates various public data sources on administrative and spatial information in the Netherlands.

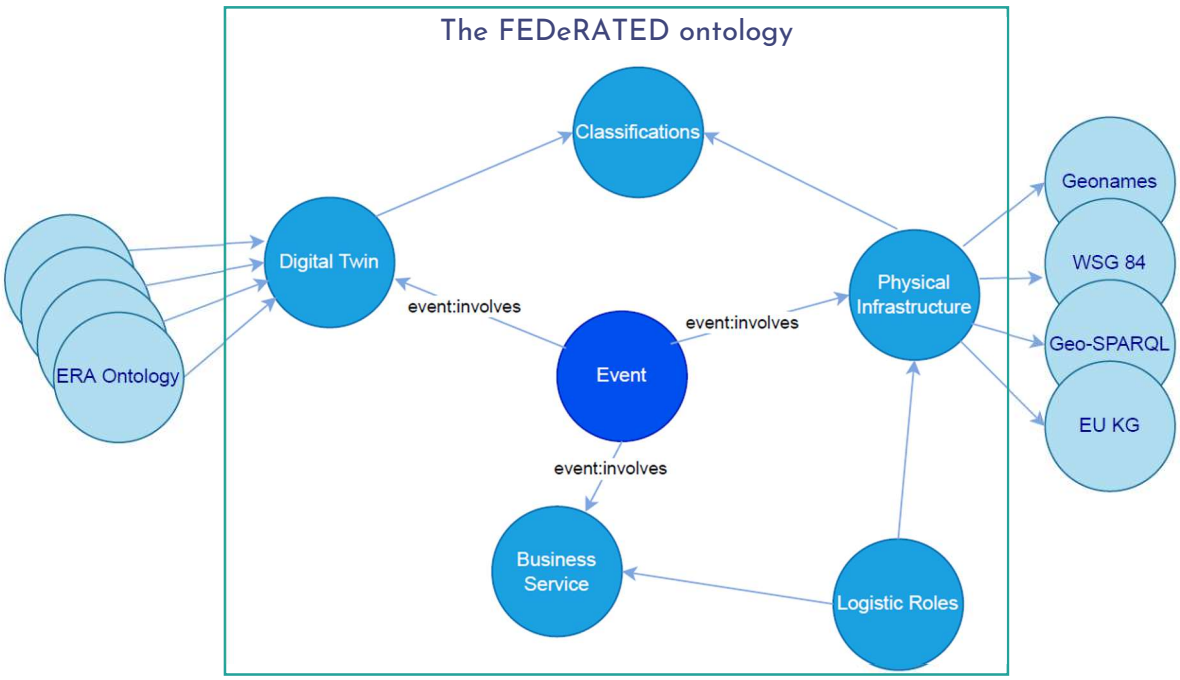


Figure 2 - Overview of modules of the core ontology and alignment with external modules

This modular multimodal logistics core ontology and TNO methodology provide a main language specification for the MAGPIE Digital Twin of the port and should cover the core functionality of the tools currently envisioned within MAGPIE as well as those that will be developed in the future. This methodology should facilitate the collaboration between application developer, domain expert, and ontology expert on extending the ontology when the tools are further specified or when new use cases and tools emerge. However, from this context, a first issue is identified while this multimodal logistics core ontology is of high complexity and difficult to use.

This issue is stemming from the fact that for a user not familiar with ontologies, this methodology is considered as theoretical and non-functional. Particularly, to model a system

domain, the user needs to know all input ontologies contents and how to reuse them. Otherwise, he should pass throw an ontology specialist who is the only one specialised in leveraging semantic technologies, such as ontologies, for data interoperability. Adding to that, each time a conceptual model is needed, the various related ontologies must be taken into consideration to find the appropriate concept.

A second issue identified is the non-existence of a domain specific ontology, which allow MAGPIE systems interoperability, through ontology exploration. In fact, input ontologies encompass multiple concepts that could be redundant in several ontologies with similar or different representations, which confuse the user use. Here, as ontologies are not necessarily compatible, they may in turn need to be reconciled. Ontology reconciliation requires most of the time to find the correspondences between entities (e.g., classes, objects, properties) occurring in the ontologies. We call a set of such correspondences an alignment.

In order to aim with these issues, we propose a model based approach for an automatic generation of a modular multimodal logistics core ontology based on the FEDeRATED core model and its extension with domain-specific models from the transport and logistics field.

The proposed approach

The aim of the proposed approach is to provide an automatic generation of a core ontology that represent the principal high-level concepts related to transport and logistics activities in the MAGPIE port. The framework of this approach is adapted to complex modular multimodal logistics core ontology where several kinds of ontologies should be taken into consideration. Thus, the method aims at defining syntactic and semantic alignment between the several ontologies. Therefore, it is based on Mode-Driven-Engineering approach and composed of a two-step process as shown in Figure 3.

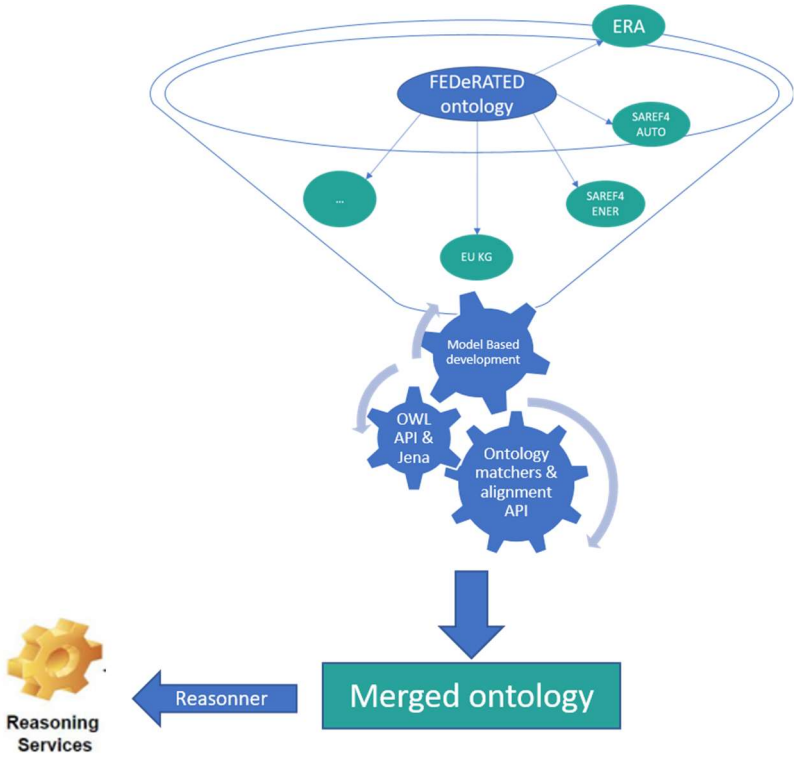


Figure 3: Core Ontology Automatic Generation

The first step named "Ontologies Alignment" consists in representing correspondences between entities from a set of ontologies. These correspondences are generated by ontology matchers and they can be used for merging ontologies, transforming queries or linking data sets. During their life cycle, alignments may be written in files and further read, applied various thresholds, merged together and finally transformed into a operational format. To perform this step, the FEDeRATED core model and its extension with the ontologies proposed by TNO such as SAREF4AUTO, the ERA vocabulary, SAREF4ENER etc. are taken as inputs and an interplay of three mechanisms is presented:

1. Model-based development to develop the ontology
2. OWL API & Jena to manage ontologies
3. Ontology matchers and Alignment API to establish alignment and correspondence between ontologies concepts.

To process ontologies alignment a set of alignment algorithms are used:

- **NameEqAlignment** Simply compares the equality of class and property names (once downcased) and align those objects with the same name;
- **EditDistNameAlignment** Uses an editing (or Levenstein) distance between (downcased) entity names. It thus has to build a distance matrix and to choose the alignment from the distance;
- **SubsDistNameAlignment** Computes a substring distance on the (downcased) entity name;
- **StrucSubsDistNameAlignment** Computes a substring distance on the (downcased) entity names, uses, and aggregates this distance with the symmetric difference of properties in classes.
- **Semantic similarity** Relates words to one another in terms of synonyms, hypernyms, hyponyms, and more.
- **NLP semantic similarities** computes semantic similarities between two concepts based on the WordNet dictionary (a large lexical database of English).

Outputs

After application of this set of algorithms one ontology is automatically generated called the "Merged Ontology" which encompasses all the concepts from the inputs ontologies with alignment application between matching ones. Behind this alignment, a set of metadata is also generated which contains a set of cells representing the correspondences: they relate two entities with a Relation. The entities may be any identified element of an Ontology. Relation represents the relation between two entities. The set and type of relations are extensible in the Alignment API and its implementation. These classes provide access to the information in instances. They also provide local methods for manipulating this information: adding correspondences to alignments, cutting correspondences under a confidence threshold, etc.

Once this first step of alignment is achieved, a second step named "Merged Ontology Reasoning" about reasoning services application is triggered to verify the consistence and correctness of the generated "Merged Ontology".

This "Merged ontology" will be available to other partners and tools of MAGPIE to be used in establishing Interoperability between different systems In the MAGPIE Port.

Illustration of the proposed approach

To illustrate the approach, we propose to apply it on the the FEDeRATED ontology while it contains several modules covering conceptually separate sub-domains of information, namely a Digital Twin, Physical Infrastructure, Business Service, Classifications and Event module as shown in Figure 4.

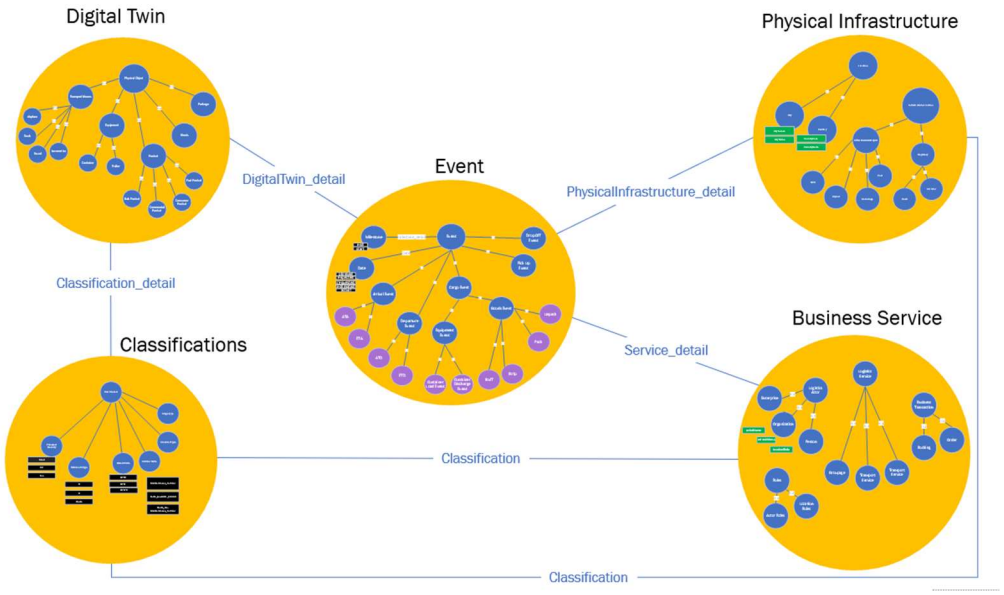


Figure 4 – Architecture of the FEDeRATED ontology.

The output result, the "Merged Ontology," is generated automatically, and reasoning services are applied to prove its consistency. Figure 5 shows an extract of the output.

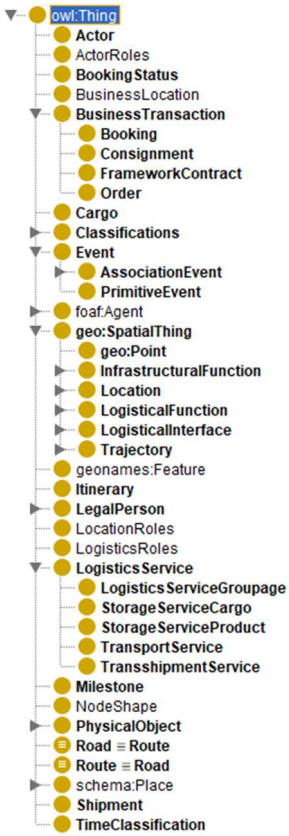


Figure 5 – Extract of Merged Ontology

Timeline of development

At the time of submitting this deliverable (M30), a draft version of the developed code around this approach exists and a first version of the "Merged Ontology" is available generated from the FEDeRATED Inputs. A final version of the code is expected by the end of May 2024 and a new version of the "Merged Ontology" will be generated once TNO provides the final versions of the used input ontologies.

After that, a second approach will be developed, which allows for the reduction of the complexity of the "Merged Ontology" by defining a refinement to represent and associate MAGPIE domain models and systems. The benefit of this step is twofold: first, it allows the generation of a complete and consistent MAGPIE ontology limited only to concepts used by MAGPIE systems. Second, it provides links and associations between the MAGPIE systems and the MAGPIE ontology. A first version of this second approach is expected by the end of 2024. A first version of this second approach outputs is expected by May 2025. In July 2025, the outcomes on aligning new systems with MAGPIE systems using the created "MAGPIE ontology" and reconfiguring the MAGPIE simulators and tools with new parameters using the MAGPIE ontology will be delivered. By the end of WP4, a methodology encompassing these two approaches and a usage report on executing and using it will be delivered.

2.3 Use Case 3: Implementing the data sharing architecture and ontology to facilitate the interaction of the back-end models and the EMT tool

The main aim of this use case is to facilitate the integration of the backend models (e.g., renewable forecast, demand estimation, flexibility) and port logistics information with the Energy Matching Tool (EMT), through their connection to the data sharing infrastructure,

as outlined in deliverable D4.2 and D4.3. Much like use case 2, this will require a clear definition of the services supported by the ontology – e.g., renewable power forecast, demand load forecast, flexibility, and the clear identification of all the data exchanges between the logistics databases, platforms, applications and tools.

One possibility that is currently under discussion, is the application of the tool to the Port of Sines, which would then also require a detailed characterization of the available platforms and databases that contain the relevant logistics information. Additionally, as the Port of Sines owns the electricity distribution network, the possibility of acquiring information directly from the electricity consumption and local renewable production could also be explored. The main analysis could then focus on the strategic and operational decisions that different decarbonization levels of the terminals in the port would entail. The possibility of applying the EMT tool to DeltaPort as part of this use case will also be explored.

The use case is just kicking off, with the first inventory of the inputs, outputs, and interlinkages between the EMT and the different backend models and databases, as shown in Figure 11. The next step corresponds to the clear identification of the storyline and services to be supported by ontology and data-sharing infrastructure. This will be followed by the development of the relevant branches of the MAGPIE ontology and the development of the connectors to link the EMT, backend models and logistics platform to the data-sharing infrastructure. Finally, several case studies and scenarios will be run to validate the implementation of the digital infrastructure and services. A first preliminary version with mock logistics data is expected for M42, with the final version validated and available in M48.

3 MAGPIE Digital Tools

This chapter outlines the functionalities, tool structure, dependencies, use cases, user interfaces, current deployment status, and development timeline for the GHG tooling (3.1), Energy Matching Tool (3.2), and Smart and Green Logistics Tool (3.3).

3.1 Greenhouse Gas Tooling

The greenhouse gas (GHG) emissions tooling aims to implement modelling and prediction capabilities to facilitate emission reduction related to the efficiency of operations, fuel, and modal shifts at operational and strategic levels. At the operational level and in compliance with international standards (EN 16258 [6], ISO 14083 [7]), emission information will be used in situational decision-making allowing evidence-based selection of the least polluting transport options (e.g. synchro-modality, carrier choice, routing). At the strategic level, it will assess the impact of decarbonization measures and provide evidence-based assessment functionality for analysis of trends, innovations, and policy measures.

To identify the data requirements and modeling needs, a detailed scope had to be defined. As stated before, the main aim of the tool is to analyze transport chains that go via the port and collect and organize carbon footprint data to establish GHG emissions along the transport chains on the origin-destination level.

The emission data gathering and analysis process will be approached in a structured way. Transport chains related to the port will be split into uniform transport chain elements, for which the data will be gathered per modality and cargo type. The tool will deliver for each transport chain element, GHG emissions per ton-km or ton. For those transport chain elements where a reasonable effort cannot yield the necessary data, estimation methods, or relevant default emission factors will be used. The tool will perform computations following the EN16258 [6] standard and will comply with the ISO 14083 [7] standard on quantification and reporting of greenhouse gas emissions arising from the operation of transport chains.

Regarding the scope of the logistics chain, we will include direct European transport (linked to Port of Rotterdam as the MAGPIE lighthouse port) and also pre-haulage transport in other continents. Moreover, transshipment emissions from loading/unloading operations taking place within terminals will also be considered. The transport modes will include road, rail, inland waterways, and maritime transport. The goods will be considered in an aggregated level of classification such as dry bulk, liquid bulk, containers, etc.

Based on the defined scope, the GHG emissions and performed ton-km per mode in the transport chain are going to be calculated. Besides the business-as-usual scenario in the base year, the tool will provide forecasts up to 2050 taking into account demand growth factors and possible developments in emissions legislation in the European Union. In this sense, decarbonization measures will be considered for the scenario analysis, including logistics measures such as modal shift and optimization of consolidation rates, as well as technological measures such as energy efficiency improvement and alternative fuels/energy sources.

The geographic scope of the tool comprehends firstly the Port of Rotterdam as the lighthouse port, but of course, the tool can be used for the other MAGPIE ports since the necessary data is provided.

The next section provides a review of the state of the art of existing toolsets and their current calculation methods, including toolsets and data developed by TNO such as the Decarbonization model for continental freight Decamod [8] and models developed for the Dutch National Emission Registration; available Carbon Toolsets and frameworks, such as GLEC Framework [9], BigMile [10] tool, and other CO_2 calculation tools like the EmissionInsider [11] by PortXchange and Routescanner [12], besides the CO_2 modelling toolsets developed by the Netherlands Environmental Assessment Agency (PBL). This review also examines which data can be accessed from other sources/parties.

3.1.1 Review of existing tools, models and datasets

By reviewing models, tools, and datasets within the context of emissions in the freight transport sector, our examination has been structured around key parameters, including transport segments, geographical scopes, and primary focuses. The objective is to offer a comprehensive understanding of the current state of knowledge in the field and to guide the work for the Green House Gas emission tool.

Table 1 gives an overview of the analysis. The literature has revealed diversity in models, tools, and datasets related to various transport segments within the freight industry. From maritime shipping and air cargo to road transportation and rail logistics, we observed a nuanced landscape of tools designed to address specific challenges. The geographical context plays a crucial role in shaping the dynamics of freight transport. The tools and datasets often encompass a regional perspective, probably due to context-dependent dynamics and data availabilities. A final aspect of our review involved identifying gaps in the existing body of knowledge to support decisions around the GHG tool. These gaps encompass areas where current models or tools may be insufficient, datasets are limited, or specific transport segments and geographical scopes are underrepresented. In the subsequent sections, we delve into each of these dimensions, providing a detailed examination of the literature's findings and their implications for the freight transport sector.

Transport segments and focus

The study revealed a variety of approaches concerning the main focus of models and tools. Notably, some models adopted an aggregate perspective, providing a holistic overview of the transport sector. These comprehensive tools offered insights into consumptions and emissions across multiple transport modes, reflecting a generalized understanding of the sector's environmental impact. Others were developed for consultancy needs within logistics companies. These tools aim to assist companies in carbon footprint reduction and logistics chain optimization. In general, many tools focused on specific needs within each transport segment, with a considerable variation in their scopes. However, a commonality emerged – a widespread lack of emphasis on carbon emissions, and the operational effects of policies were not explicitly addressed in most models. This observation highlighted an overarching gap that warrants attention in future research endeavors. For example, Decamod [8] allows the user to see the effect of different reduction policies (and their combinations) on emissions, which was lacking in the other models.

The majority of models embraced a multi-modal approach. However, a notable gap was identified – the absence of a comprehensive tool providing an integrated overview of all transport modalities. While existing tools were diverse and covered various segments, a unified tool addressing the entire spectrum of transport modes was notably absent.

Several tools targeted different aspects or perspectives, including energy consumption, electricity management, fuel supply, and more. This diverse range of tools indicated a

recognition of the multifaceted challenges posed by port operations, demonstrating a nuanced approach to addressing specific needs within this crucial transport segment. A notable gap emerged in the representation of transshipment points, other than ports. This underscored the need for more tools tailored to the unique operational requirements of transshipment points.

Within the realm of seagoing vessels, the focus predominantly revolved around emissions. However, a limitation surfaced concerning the scope of data: existing tools primarily addressed the final segment of a vessel's journey within territorial waters or the continental shelf. Some other modules focused on optimizing the consumption of individual ships, presenting a need for more comprehensive tools to capture the entire voyage. Tools concerning inland water shipping mainly concentrated on current and future energy demands and emissions. This segment demonstrated a forward-looking approach, addressing environmental considerations and energy efficiency in anticipation of future developments.

Geographical scope

The investigation of this section focused on understanding the spatial dimensions of these tools, shedding light on their applicability and limitations. Interestingly, most models were developed by European-based actors, with the majority focusing on specific European countries or maintaining a scope within EU member states. A concentration of models, particularly from the Netherlands, further shaped the landscape, offering valuable insights into freight transport in this region. On the one side, the substantial availability of toolsets and data centered around the Netherlands gives a positive signal for developing more comprehensive tools. On the other hand, it raises concerns about the applicability of these tools on a global scale, which represents a limitation not to be underestimated. While some of the models maintained a global focus, they were primarily services provided by companies to advise shipping entities, which made it challenging to access in-depth explanations of the methodologies employed. Conversely, when models adopt a more geographically contained perspective, they lose specificity in terms of the logistic chain. For example, a model studying emissions in Dutch national waters may lack insights into the individual logistics chain of each ship causing those emissions, and vice versa. These findings suggest that achieving a balance between geographical scope and specificity is a crucial consideration for the development of comprehensive tools in the freight transport sector. Striking this balance is essential to ensure that tools provide meaningful insights into both the broader global context and the intricacies of specific logistics chains. This consideration becomes particularly pertinent as the field progresses toward more integrated and universally applicable models.

Table 1 - Models, transport segment, and geographical scope

MODEL	AIMS	Transport segments	Focus	Geographical scope	Source
BasGoed	Map the economic development and policy measures on freight transport	Road, rail, inland shipping and sea shipping	Freight transport forecast	Netherlands	[13]
BigMile	Calculate and analyze transport-related emissions offering insights into the carbon footprint and identifying areas for improvement.	Transport-related carbon emissions	Transport-related carbon emissions	Not mentioned	[10]
Decamod	Provide information about the effects of CO_2 reduction measures; Provide insight into the impact of decarbonization measures in logistics	Road, rail, inland shipping	Transport-related carbon emissions	Netherlands	[8]
EmissionInsider	Track and analyze emissions in and around the port to develop an actionable decarboniza-tion strategy	Transport-related carbon emissions)	Transport-related carbon emissions	Not mentioned	[11]
EWI global PtX Cost Tool	Calculate supply costs based on	-	Supply cost of green ammonia	Global	[14]

	RES potentials, technology costs, and weighted average costs of capital				
Fritzy and friends	Balance energy supply and demand automatically to ensure energy security; Reduce costs; Increase energy safety and infrastructure efficiency	Port infrastruc-ture	Port energy consump-tions	Netherlands (Port of Amsterdam)	[15]
HESP	Calculate shipping emissions based on available AIS data	Ships	Ship emissions	Zone of 80 km around the Port of Rotterdam	[16]
NEXUS	Digital and green transition of the whole logistics chain	Ports	Port value chain	Portugal (Port of Sines)	[17]
OPERA model	Minimize the societal costs of the Dutch energy system; Search for an optimal fuel mix portfolio for the Dutch transport sector	Transport sector	Fuel mix portfolio	Netherlands	[18]
Port Energy Consumption Tool (PECMT)	Track energy use of all port operations	Ports	Energy consump-tions in ports	Germany (JadeWeser-Port)	[19]

	and stakeholders				
POSEIDON	Forecast energy use and emissions from shipping	Maritime	Forecasting energy use and emissions	Netherlands (continental shelf)	[20]
REff Tool®	Measures and Monitor the resource consumption and emissions efficiency at logistics sites; Monitor resource efficiency and develop further KPIs in the storage and transhipment sector	Logistic sites	Resource consump-tion and emissions	Not mentioned	[21]
Route-Scanner	Calculate carbon footprint of shipments; Visualize intermodal networks; Show faster, cheaper and more sustainable trade lanes	Road, rail, inland shipping and sea shipping	Container logistics	Global	[12]
Study into Transport Emissions from All Modalities (STREAM)	Create an overview of the emission figures of the transport modes in freight transport	All modes of freight transportation	Energy consumptions	Netherlands	[22]
The Small Emitters Tool (SET)	Estimate the fuel burn and CO2 emissions for flights under	Air transport	Fuel consumptions and emissions	Europe	[23]

	the EU emissions trading scheme (EU ETS)				
VECTO Tool	Determine CO2 emissions and Fuel Consumption from Heavy Duty Vehicles	Trucks, buses and coaches	Energy requirements	Europe	[24]
WITCH Tool	Assess climate change mitigation and adaptation policies	-	Energy demand	Global	[25]

3.1.2 Conclusion and knowledge gaps

The exploration of models, tools, and datasets within the freight transport sector has provided a comprehensive overview of the current state of knowledge and identified critical gaps that necessitate further research. The geographical scopes of the models highlighted their strong connection with European countries or EU member states. Some knowledge gaps emerged from the review, pointing towards areas that deserve further investigation and development as highlighted as follows:

- Tools designed for emissions in logistics chains at a national and international levels.
- Tools covering all the transport chain elements of a logistics chain, including the road, rail, inland shipping, and maritime modes, as well as transshipment of cargo as hub operations.
- Tools capable of calculating the effect of emissions reduction measures to facilitate informed decision-making.

In conclusion, the literature review has provided a panoramic view of the current landscape of models, tools, and datasets in the freight transport sector. It has identified strengths, such as the diversity of tools catering to specific transport segments, but has equally shed light on critical gaps regarding emissions of supply chain operations. The development of the GHG tool aims to close such gaps by including all transport modes related to the logistics chain beyond national borders (road, rail, inland shipping and maritime) and transshipment of cargo as hub operations. Such a tool will facilitate the development of policies targeting specific supply chain elements. Moreover, the GHG tool will also include the calculation of the effects of emission reduction measures. By addressing these knowledge gaps, it will contribute to a more nuanced understanding of emissions in supply chains as a whole and pave the way for the development of targeted and effective strategies to mitigate the environmental impact of logistics operations.

3.1.3 Methodology

This section describes how the GHG tool is structured and which sources are used as a basis for calculating the GHG emissions for the different logistics activities. In addition, it describes how the impact of decarbonization measures is calculated.

Base scenario: Current emissions and the development for the period 2018-2050

To gain insight into the effect of logistics decarbonization measures, a base scenario is first drawn up with goods flows from and to the Port of Rotterdam for the projected years 2018-2050. Various sources of information are used in drawing up the base scenario. From these sources, 2018 was chosen as the base year. For this year data regarding Dutch freight transport is available at the desired level of detail.

The base case has two applications. Firstly, it is used to provide insight into the business-as-usual situation in terms of, among other things, tonnes, vehicle kilometers and CO_2 emissions. In addition, the base scenario is used as the starting situation against which the effects of the CO_2 saving measures are calculated. Decamod [8] takes into account future developments as a result of established policy. The effects and potential CO_2 reduction of the additional measures can then be mapped out.

Data sources used to determine the base scenario

The basic scenario is recorded in a database. Various external data sources were used to construct the most accurate possible base scenario. These data sources have been combined into a source dataset for the toolbox that underlies the base scenario. The source dataset concerns the total flows of goods going in and out of the port of Rotterdam for the reference years 2018-2050, including tonnes, vehicle kilometers, tonne-kilometers, and the related CO_2 emissions.

Transport flows

For the transport flows (tonnes and vehicle kilometers), Freight Transport data was made available by the Dutch infrastructure manager, Rijkswaterstaat, and Statistics Netherlands. These files contain detailed data on observed freight flows within, through, to, and from the Netherlands for seagoing and hinterland transport (road, inland shipping, rail) per origin destination. In particular, the breakdown per OD and good types (NST 2007 classification for hinterland [26]) makes the combined dataset suitable for the desired level of detail in the Greenhouse Gas Tool.

CO_2 emissions

The CO_2 emissions for the different modalities are based on the methodology set out for the Dutch Pollutant Release and Transfer Register [27] Based on calculations of a dedicated model suite per modality, dedicated emission factors are used for different subclasses within the modalities. The CO_2 emissions in the source only relate to the emissions released when using the vehicles/vessels, namely the Tank-To-Wheel (TTW) emissions. The Well-to-Tank emissions are considered important when regarding the energy transition, and therefore have been added based on [28] and additional sources.

Growth scenario

For the growth forecast for freight transport from 2014 to 2050, we will use the BasGoed forecast carried out on behalf of the Netherlands Environmental Assessment Agency (PBL), based on the principles of the Climate and Energy Outlook – KEV [29] These results have been made available at an aggregated level by PBL to TNO for this project. Of the KEV calculations with BasGoed, the autonomous development (established policy) in transported weight per origin and destination relationship (SMILE zone), NSTR category, and (non) container transport was specifically used. The growth for the intervening years has been derived in the GHG Tool using linear interpolation between 2018 and the KEV BasGoed forecast for 2030. The same growth has been assumed for vehicle kilometers as for the weight transported. This means that in the GHG tool's base scenario, no improvement in logistics efficiency has been included in the development of freight transport until 2030. The reason for this is that such logistics efficiency improvements are not part of established policy and it is therefore not certain that these developments are to be realized. Since logistics processes will likely improve in the future, they can be included as part of a future scenario in a Decamod analysis.

The source for the growth for the period after 2030 still needs to be decided upon. Either the scenarios for 2050 by Port of Rotterdam will be taken as a base [30] or the long-term National scenarios which are going to be published in 2024 by PBL.

Decomposition of data

- Modalities
- Freight categories
- Origin destination

3.1.4 Methodology for calculation of GHG reduction measures

This section explains the methodology used for calculating measures. The implementation of technical and operational decarbonization measures affects several performance variables (tonnes and vehicle kilometers). In addition, the fleet composition is influenced and in the case of road transport, the distribution across road types may change.

Figure 6 shows the methodology used in the GHG tool to quantify the potential CO_2 savings from different decarbonization measures. This methodology follows a step-by-step plan in which it is examined for each performance variable whether and to what extent the measures affect it.

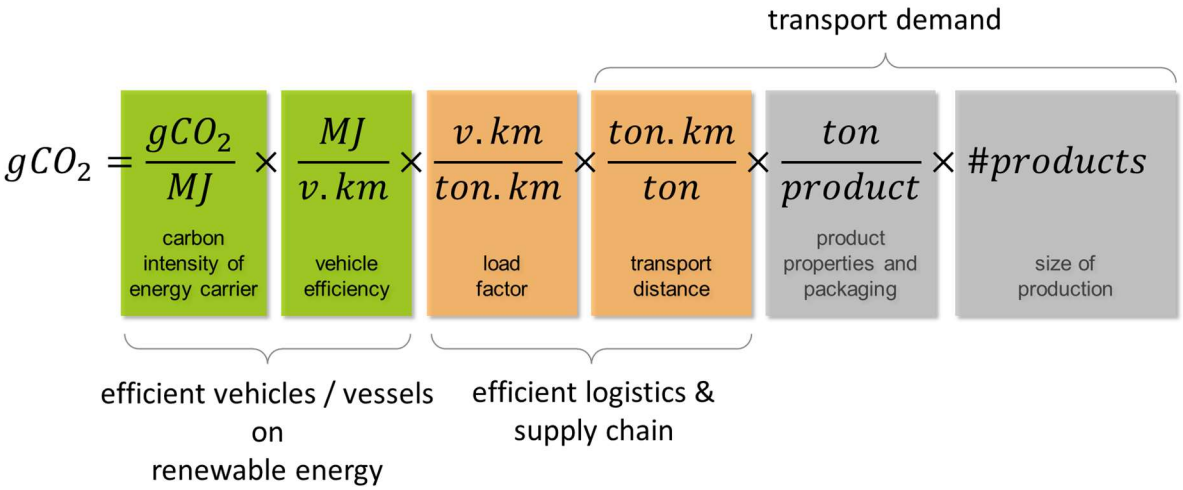


Figure 6 - Environmental performance of emission reduction measures.

Quantity of goods transported (tonnes)

The starting point for the calculations is the amount of freight (in tons) that is transported. With the GHG Tool, it should be possible to map scenarios in a future sustainable world in such a way that the demand for goods can always be met. Measures aimed at reducing the volume transported are therefore outside the scope of the tool.

Vehicle or vessel kilometers

The number of vehicle kilometers is related to logistic efficiency; the number of tons transported per vehicle kilometer (or vessel nautical mile). Based on the historical vehicle kilometers per ton of goods (depending on the vehicle type), the number of vehicle kilometers is calculated in parallel with a change in tons. Logistics measures can also have a direct effect on the number of vehicle kilometers, for example by using a vehicle with a larger loading capacity. In that case, more freight is transported per vehicle kilometer if the preconditions in the entire logistics chain allow this.

Fleet composition

The next step is to determine how vehicle kilometers are distributed in terms of fleet (and road types in case of road transport). For example, there may be a shift of goods between modalities or the use of vehicle/ vessel types may change (e.g. by use of larger ships, or by use of decoupling points in road transport).

CO_2-emissions

The previous steps reveal how the tons and vehicle kilometers are distributed across the modalities, vehicle types, and road types. The CO_2-emissions from freight transport in the outlined scenario are then calculated using emission factors per vehicle kilometer. The emission factors depend on the vehicle and road type.

3.1.5 Interaction effects for combinations of different measures

With the model, a subset of different measures can be calculated. This section explains how the tool deals with combining measures and what happens if the effects of different measures overlap. The effect of a measure is determined by the reduction potential and the application area of the measure, as can be seen in Figure 7.

Figure 7 - The impact of a measure is calculated by multiplying the reduction potential by the application potential of the measure.

Reduction potential

Measures have an effect that leads to a certain reduction or increase in a logistics performance variable. This is called the reduction potential of a measure. If two separate measures target the same performance variable, they are dependent on each other and the reduction potential of both measures is influenced. For example, there may be an interaction between fuel-saving measures. A first measure could be to improve the driving style of drivers through training, which reduces CO_2 emissions per kilometer. A second measure, for example, is an improvement in engine efficiency, which also leads to a reduction in CO_2 emissions per vehicle kilometer. By improving engine efficiency, the improvement in driving style has less effect; the reduction potential of the latter measure is therefore less significant. The combined effect of the individual measures is generally not equal to the sum of these effects.

Application potential

The reduction potential of a measure may apply to a smaller part of the logistics system, namely to different elements of the decomposition. This makes it possible to investigate the impact of measures, for example, only for one logistics segment or a selection of good types and vehicle types. The selection of the logistics system on which a measure has an effect is the application potential.

Dependency between measures can also occur in the area of the application potential of measures. For example, an initial measure could lead to a 10% reduction in vehicle kilometers for parcel services. A second measure improves engine efficiency, reducing CO_2 emissions per vehicle kilometer by 5%. Due to the reduction in the number of vehicle kilometers of the first measure, the application potential of the second measure has been reduced by 10%. The 5% decrease in CO_2 emissions per vehicle kilometer will only be applied to the remaining vehicle kilometers after the introduction of the first measure.

Adhering to the methodology described ensures that dependence between measures in the field of application potential is implicitly included in the modeling of the GHG tool.

Second-order effects

Logistics and operational measures usually influence the design of the logistics system. As a result, the direct effect that a measure has on one performance variable can have an impact

on other performance variables. This is called a second-order effect. This occurs, for example, with an increase in the load factor. The initial point of action for this change is the ratio of vehicle kilometers per ton. With an increase in the load factor, fewer vehicle kilometers are needed to transport the same weight. But as a result of the increase in the load factor, a second-order effect takes place: the increase in weight per vehicle also leads to an increase in energy consumption per kilometer and therefore also in CO_2 emissions per vehicle kilometer. Another second-order effect that could occur is that the measure opens up capacity which can be used for additional cargo movements (more demand). These second-order effects of measures are not implicitly included in the calculation of the effects of measures in the toolbox.

Process of the model

Figure 8 shows the schematic representation of the toolbox. The base scenario (reference scenario) and a scenario in which a decarbonization measure is implemented (measure scenario) form the input for the toolbox. If necessary, other models are consulted or a preliminary study is done to determine the impact of a measure. The impact on CO_2 emissions and costs is then calculated by comparing the outcome of the future scenario with the base case.

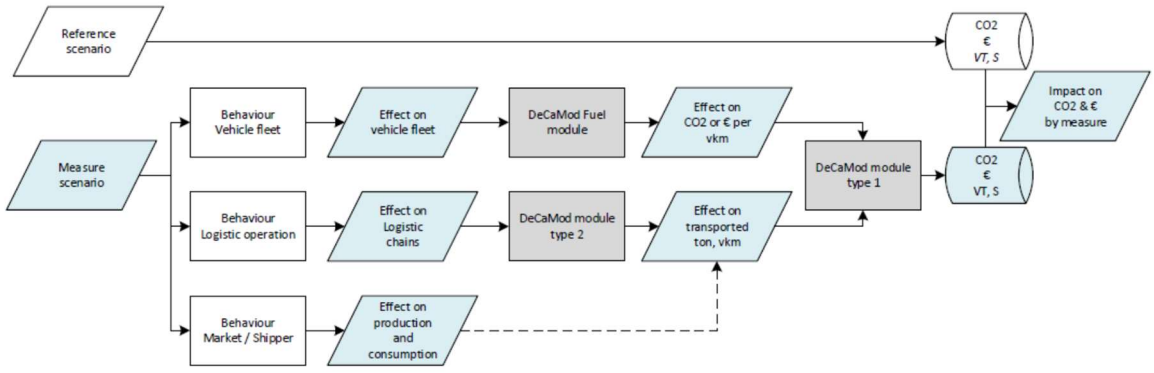


Figure 8 - Schematic representation of the Decamod toolbox.

3.1.6 Expected outcomes

The tool will deliver for each transport chain element, GHG emission intensities, and per tonne-km or tonne throughput measured in CO_2 equivalents. The presented emissions will be compliant with the ISO 14083 [7] standard (published in March 2023) on quantification and reporting of greenhouse gas emissions arising from the operation of transport chains. The results will furthermore be aligned with the CountEmissionsEU proposal, which will serve as a common framework for quantifying the greenhouse gas emissions of transport services across different modes.

Figure 9 presents a schematic overview of the different chain elements, the required input, and the link to ISO 14083 [7].

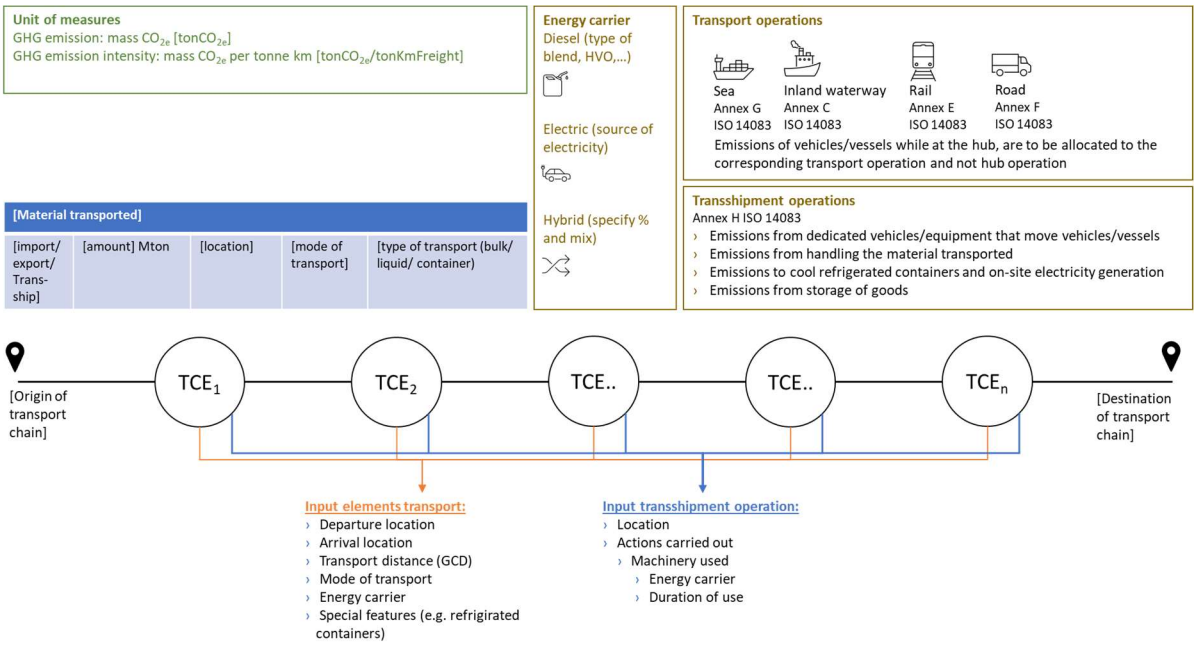


Figure 9 - Overview of the different chain elements considered in the GHG emissions tool.

The scope of the tool will be:

- The tool will provide insight into the emissions for freight transport on the level of :
    - individual supply chains,
    - freight corridors (both maritime and hinterland corridors), and
    - port as a whole
- All elements of the logistics supply chain linked to the ports will be included:
    - Maritime transport and hinterland transport (road, rail, and IWT) on both sides of the maritime leg.
    - Emissions of transshipment
- All types of goods will be included in an aggregated level (e.g. dry bulk, liquid bulk, general cargo, containers, Ro/Ro)
- The tool will give insight into the current GHG emission levels and forecast scenarios up until 2050
- The tool will furthermore give insight into the effect of GHG reduction measures, including:

- Application of alternative energy carriers (such as electric, bio-fuels, hydrogen),
- Other technical options (wind assist for ships, truck platooning)
- Efficiency and logistics measures (slow steaming, modal shift)

- The tool will first be applied to the Port of Rotterdam. Depending on the data availability of traffic and throughput, the tool might be extended to the other MAGPIE ports. Data on maritime transport seems to be available, data on the hinterland transport less so.

3.1.7 Time Planning

Table 2 presents a high-level planning of the further development of the tool. For the year 2024, the focus will be on the development of three modules, focusing on maritime transport, hinterland (road, rail, inland shipping), and transshipment. In 2025, the focus will be on the integration of the different modules and first applications with MAGPIE use cases.

Table 2 - Development timeline of the tool

	2023												2024												2025											
	mrt	apr	mei	jun	jul	aug	sep	okt	nov	dec	jan	feb	mrt	apr	mei	jun	jul	aug	sep	okt	nov	dec	jan	feb	mrt	apr	mei	jun	jul	aug	sep	okt	nov	dec		
	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36	M37	M38	M39	M40	M41	M42	M43	M44	M45	M46	M47	M48	M49	M50			
Task 1: Scoping																																				
Task 2.1: Tool Design Back-end																																				
- Data gathering																																				
- Maritime module																																				
- Changes in Decamod																																				
- Changes multimodal assignment module																																				
- Transshipment module																																				
- Non Magpie port module																																				
- module integration																																				
Task 2.2: Tool Design front-end																																				
Task 3: Applying the toolkit in use cases																																				
Task 4: Roll out and possible extension																																				

3.2 Energy Matching Tool (EMT)

3.2.1 Overview

Ports are characterized by having a diverse number of stakeholders, encompassing industries, warehouses, and various types of terminals such as container or LNG terminals. Each of these entities possesses distinct assets, operational constraints, and energy carriers. Consequently, Ports currently stand as complex, energy-intensive hubs with significant CO2 emissions. In response, Ports are actively pursuing ambitious decarbonization and digitalization initiatives. These initiatives involve the electrification of loads and a growing use of alternative energy carriers like hydrogen. Additionally, there is an effort to invest in storage and local production technologies, e.g., PV systems and onshore/offshore wind power.

While these additional assets present opportunities for both CO2 emissions reduction and cost savings, to realize its full potential their utilization requires optimal operational management. This entails aligning local production of energy, leveraging flexibility from storage, and exploring the potential of flexible demand within the Port energy system. By doing so, Ports can enhance their overall energy efficiency and sustainability.

Within this context, the Energy Matching Tool (EMT) focuses on optimally managing local distributed energy resources (DERs) and flexible loads, to promote self-consumption and self-sufficiency relative to the main electrical grid, within the next 24-48 hours. The tool can be applied to individual players, aggregators' portfolios, or micro-grids.

Employing a mixed-integer linear programming (MILP) optimization algorithm, the EMT is developed using PyPSA [31], an open-source Python software toolbox for simulating and optimizing power and energy systems. Leveraging the features of this library, the EMT includes multiple carriers, i.e., electricity, hydrogen, heat, and the grid topology, ensuring the feasibility of grid operational constraints (as described in D4.4). Furthermore, a P2P market module is set to be developed on top of the EMT to facilitate energy exchange between different players, for instance, in an energy community, thereby further promoting local grid self-sufficiency.

3.2.2 Tool Description

The EMT primarily focuses on modelling the main elements within the complex and dynamic Ports environment. These elements encompass common renewable energy production sources, storage solutions (e.g., batteries), as well as both flexible and non-flexible electricity and/or hydrogen demand (e.g., barge swap containers, crane operations, smart reefers climatization, and vessels being charged with onshore power supply - OPS). Through collaborative discussions with project partners and leveraging insights from the work conducted in other work packages (WPs), a comprehensive list of elements to include and model was identified. Figure 10 succinctly presents these components, offering a holistic representation for modelling the Port's energy landscape.

While certain elements mentioned may not currently be present within the Port's energy system, more futuristic elements, inspired by MAGPIE's demos (e.g., barge swap containers) are being included. To apply the EMT effectively in both the current and future Port energy systems, Task 3.6 will generate scenarios. These scenarios will outline the port dimension, type, and number of elements to consider (created with the assistance of respective WP3 models) and include more general parameters such as the energy demand per energy carrier.

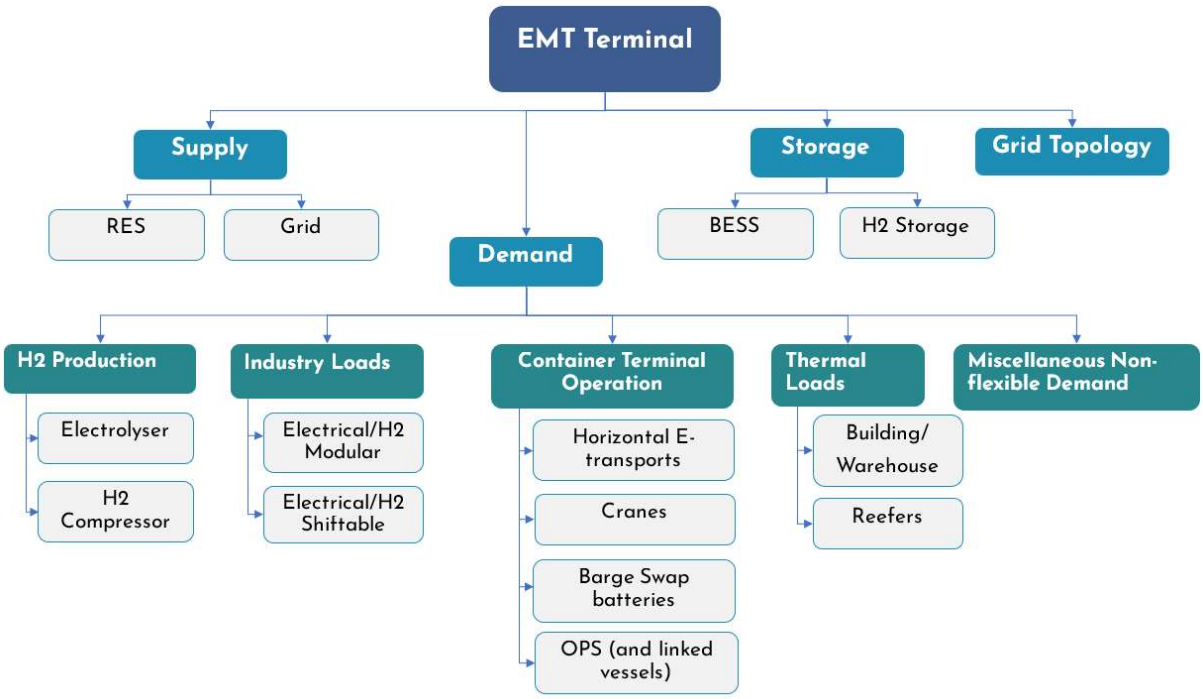


Figure 10 - Port elements addressed and modelled in the Energy Matching Tool.

Inputs

To accurately model each component, it is essential to incorporate operational and technical data. For example, when considering barge swap containers, operational data includes available time slots for battery management and the initial and final state of charge. On the technical side, it involves parameters like the swap container capacity, charge/discharge power, and charger efficiency.

Although the detailed list of inputs is being refined, a preliminary overview of the requisite data is presented in Table 3.

Table 3 - Overview of Energy Matching Tool Inputs

Domain	Component	Expected Inputs
Supply	Grid	Grid Prices
	Renewable Energy Source (RES)	Production profile
Storage	Battery Energy Storage System (BESS)	Technical parameters, i.e., capacity, power, charge/discharge efficiency
	H2 Storage	Technical parameters, i.e., capacity, power
Demand	H2 Production	- Electrolyser technical parameters, i.e., capacity, power, efficiency (electricity to H2)

		- H2 Compressor technical parameters, i.e., capacity, power, specific consumption - H2 demand profile
	Industry Loads	- Baseline demand profile - Modular Loads flexibility margins, i.e., upward/downward flexibility amount - Shiftable Loads operation constraints, i.e., time interval allowed to shift load
	Container Terminal Transports (i.e., trucks, reach staker, and barge swap batteries)	- Transport baseload demand and charging profile - Transports' batteries technical parameters, i.e., capacity, power, charge/discharge efficiency - Chargers technical parameters - Transport and connected charger at each charging - Transport initial and end SoC at each charging
	Thermal Loads	- Baseline Demand profile - Technical parameters, i.e., thermal mass, power, efficiency (electric to heat) - HVAC Technical parameters, e.g., power, efficiency - Operation climatization constraints, e.g., max. and min. temperature
	Miscellaneous Non-flexible Demand	Baseline Demand profile
Grid Topology	Buses, Lines, Transformers	- Technical parameters, e.g., lines and transformers capacity - Location - Linking elements (e.g., buses and lines) require starting and ending element (e.g., line_1 links bus_1 to bus_2)

To showcase the practical application of the EMT, input parameters for the characterization/modelling will be drawn from models and simulators of MAGPIE partners, as specified in D.4.4. The models and data flow to be implemented with the EMT are presented in Figure 11. These inputs can, nonetheless, be directly provided by the user.

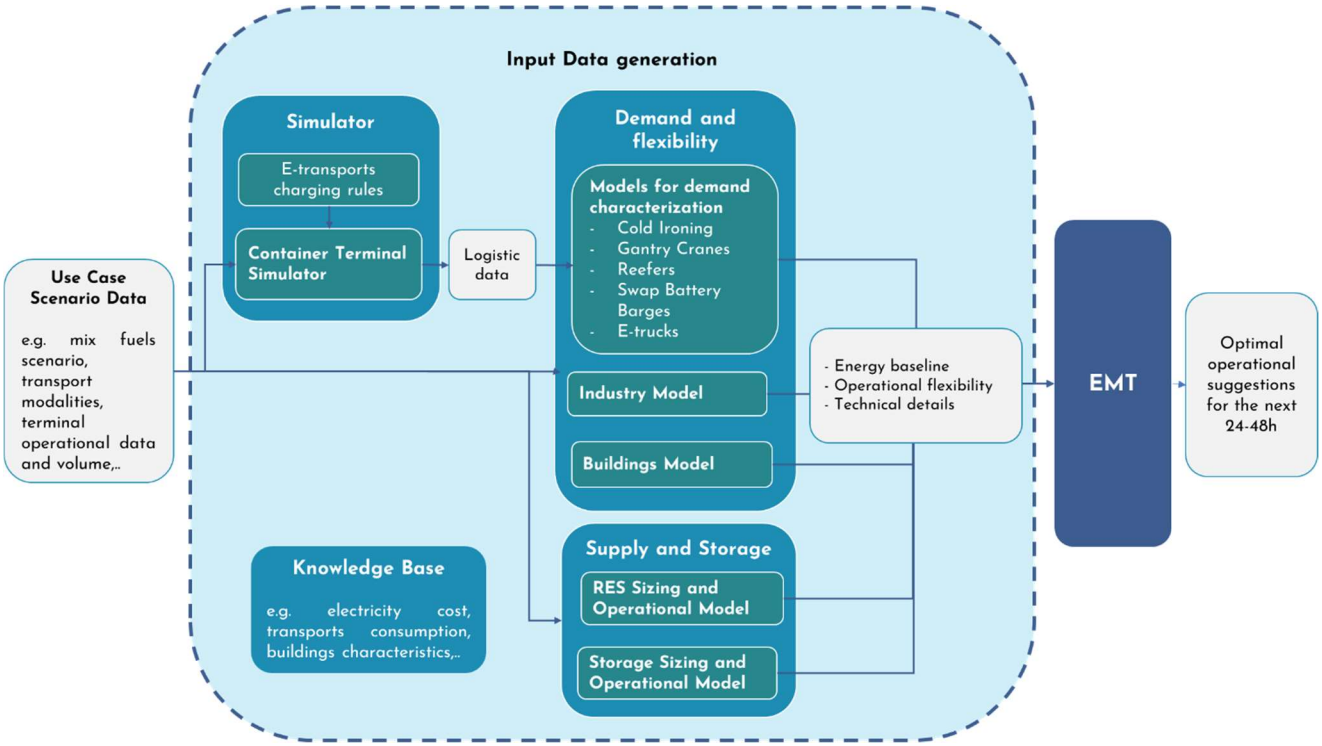


Figure 11 – Models and data flow of the Energy Matching Tool.

Optimization Model

The main objective of the EMT is to reduce energy dependency from the main electrical grid, thus maximizing self-consumption. The objective function comprises two elements, (i) purchase of energy from the grid and local RES, and (ii) the cost of activating demand-side flexibility. Minimizing the first element means increasing renewable penetration, as it is considered a residual value for RES local production when compared to the grid prices. The second element incorporates a penalization related to activating flexible loads. This introduces a dynamic aspect to the system, taking into account the willingness of users to modify specific loads energy consumption patterns, e.g., an industry might only be willing to move the consumption if it represents considerable savings. The objective is then given by,

$$min \sum_{t}^{t} (E_{grid,t} * P_{grid,t} + E_{res,t} * P_{res,t} + P_{flex,t})$$

where, $E_{grid,t}$ is the energy consumed from the grid at each time step, and $P_{grid,t}$ the corresponding timestep price. Similarly, $E_{res,t}$. and $P_{res,t}$ correspond to local RES. $P_{flex,t}$, represents the cost of activating demand-side flexibility.

To formulate the remaining optimization problem, the EMT will build upon the inputs in Table 3 to perform the parametrization and definition of constraints governing the operation of all elements. A concise overview of the anticipated constraints within each domain is shown in Table 4 and in accordance with Figure 10.

Table 4 - Energy Matching Tool constraints overview

General
• Overall energy balance for all carriers, ensuring a match between supply, storage, and demand.
Grid Topology
• Grid topology restrictions, e.g., respect lines maximum capacity
Supply
• RES throughput limited by production profile (provided as input). • Restrict the grid energy supply based on interconnections with the main grid (related to grid topology).
Storage
• Definition of technical characteristics which affect storage operation, i.e., max. capacity, max. charge/discharge power, standing loss, and charger efficiency. • Storage energy or mass balance
Demand
Container Terminal
• For battery-swapping barges and horizontal transport (e.g., e-trucks, reach stackers), to incorporate technical details of batteries, i.e., maximum capacity, charge/discharge power, and charger efficiency. In terms of operation, to consider, initial/final state of charge (SoC), and available time slots for charging/discharging. Moreover, technical details of chargers are required, i.e., capacity, and efficiency. • The Onshore Power Supply (OPS), operating as a charger, and the corresponding berthed vessels incorporate similar constraints as outlined for battery-swapping barges and horizontal transport. Additionally, regulatory requirements for the minimum electricity demand of vessels during berthing are considered, also requiring the constraint definition on vessel loads modulation. • Cranes due to logistic restrictions, are modelled as non-flexible loads, thus only requiring load profile.
H2 Production
• Define technical characteristics for the electrolyser and compressor, which affect operation, i.e., capacity, charge/discharge power, specific consumption
Industries

- Industries operation is made by either modular or shiftable loads, which require different constraints: (i) modular loads characterized by upward/downward flexibility margins for each timestep, and total energy required for the load; (ii) load shifting, considers load profile and available time slots for shifting.

Thermal Load

- The thermal energy balance equation of reefers and buildings, including thermal inertia, and heat gains/losses occurring both internally and externally to the envelope.
- Define technical characteristics for associated electricity-to-heat equipment (e.g., chiller), specifying capacity and efficiency/COP

Market Module

The EMT can be seen as centralized tool that manages all elements of a single or multiple players. In a Port environment, involving numerous stakeholders, managing diverse assets and operations under a singular entity, to achieve an optimal and cooperative management may not precisely mirror reality. This is because each individual player operation is autonomous to others.

Consequently, an analysis was conducted on relevant literature to explore potential scenarios for a market structure that could incentivize the investment and consumption of local RES by multiple individual players organized in an energy community. Two prospective approaches were identified for further analysis as an add-on to the EMT.

A first approach focusing on implementing a P2P trading market [32] within the energy community, after players determine their optimal operations according to wholesale grid prices. This approach would aim to facilitate energy buying/selling between players within the community at a more favourable price than the real-time grid price/feed-in-tariff. Sellers stand to obtain higher selling prices than the grid feed-in tariff, while buyers can reduce their energy bills, ensuring access to green energy at lower price than the grid price. This approach, as a result, incentivize players to invest in RES, as they can achieve decarbonization target faster while selling excess production for a higher selling price to the grid tariff, translating to more attractive compensation for their investments, thus promoting even further self-sufficiency and self-consumption.

In contrast to the above solution, a second approach involves establishing an iterative process between a central entity, i.e., Port authority, and the energy community players, i.e., port's stakeholders, as highlighted in the literature [33], [34]. In this approach, the central entity introduces grid price incentives aligned with the energy community's demand and local production at each time-step. For example, the internal energy community price would vary for each timestep based on the local production and demand ratio, varying between grid and feed-in-tariff prices. At each iteration, new (de)incentives are provided by the central entity to encourage players to independently adjust their operations to increase local RES consumption.

The objective of this additional market layer will be developed to promote local energy exchange within the energy community, benefiting both energy sellers and buyers, and incentive local RES investment and production. The outcomes of implementing a market structure will be provided as Key Performance Indicators (KPIs) as presented in Table 5.

Outputs

After solving the optimization problem, the EMT will produce two main outputs: (i) detailed suggestions including necessary adjustments to the player's asset set points and operation, and (ii) a comprehensive comparison between the baseline scenario and the computed optimal scenario. This comparative analysis will be executed by evaluating the KPIs outlined in Table 5.

Table 5 - Energy Matching Tool KPIs

KPIs	Description
Energy Cost Savings	The overall energy costs saved from optimal management
CO2 Savings	The overall CO2 emission saved from optimal management
Self-Consumption	The ratio of consumed produced energy to total production
Self-Sufficiency	The ratio of consumed produced energy to total demand.
Activated Flexibility	The amount of activated demand-side flexibility

3.2.3 Access to Use

Regarding access and usage of the tool, the EMT will be made available for MAGPIE partners through a locally used library. To employ the tool, users will need to load an input file specifying the parameters of the respective elements they wish to analyse in the optimization. Once the EMT is executed, it will generate a file containing the outputs as elaborated in the preceding section. A more in-depth analysis of the user's library utilization, environment dependencies, along with an examination of the output file, will be provided in the next deliverable.

3.2.4 Development Timeline

The expected timeline for the EMT is detailed in Table 6, categorized into four main phases:

(i) **Scoping of the tool**: This involves the definition of EMT's role and focus, conducting an assessment on available data on ports, as well as defining partners' models developments and contributions;

(ii) **Model Development and Integration:** This phase not only encompasses EMT development but also includes partners' models, such as characterizing demand/supply and updating the container simulator. As a milestone, a working version with all integrated models is expected to be running by the end of 2024.

(iii) **Use Case definition and application:** This phase involves the practical application of the EMT and its associated models in a defined Port Use Case, and an analysis of the outputs. For this use case, scenarios, including both current and futuristic perspectives, will be created for one of the MAGPIE ports (currently under assessment) to showcase the tool's application.

(iv) **Library Development:** The final phase focuses on developing a library that facilitates the tool's accessibility for MAGPIE Partners. This ensures that the tool becomes user-friendly and readily available for utilization by partners involved in the MAGPIE project.

Table 6 - Development timeline of the Energy Matching Tool

	2023										2024											2025								
Task	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Scope and definition																															
Modelling EMT elements																															
Third party Models integration																															
Market implementation																															
Use Case Definition																															
Use Case Test/Validation																															
Library Development																															
Library Deployment																															

3.3 Ports Smart and Green Logistics Tool

3.3.1 Description of the tool

The Smart and Green Logistics tool being developed under MAGPIE 4.5.3 will be a decisions support system (DSS) that can be used by Port Authorities to evaluate their hinterland network for container transport. It takes the schedules of different modes of hinterland transport and the demand for container shipments as an input and finds a shipment plan that balances cost, time and emissions. Even though the tool will also provide a module for operational planning, the main use-case of the tool is at the tactical or strategic level of decision making. With this module, the user can evaluate the impact of changes in the hinterland transportation network on the modal split of the container transport. For this, it takes a given hinterland network and one year of demand data for container shipment and uses a rolling horizon scheduling module to evaluate the modal split under the given hinterland network. By varying the provided network by, for example, adding services, increasing capacities, or introducing new sustainable transportation modes, the tool can show the impact on modal split and emissions.

The tactical module of the tool makes use of an operational scheduling module to find a shipment plan for individual days within the period under study. This module can also be used as a stand-alone tool in which a shipment plan can be made for the next 48 hours. In this module, decision on the mode of transport will be used collectively by trading off the cost of the shipment, the delivery time and the resulting emissions. How to balance these objectives is a parameter that can be set by the user. An overview of the interaction between the operation module and the tactical module is given in Figure 12.

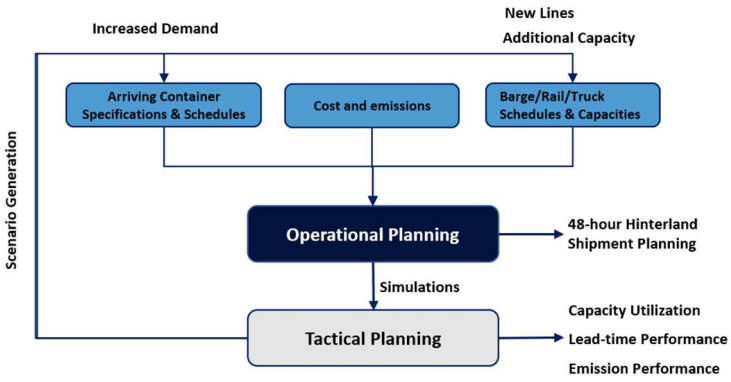


Figure 12 - Overview of interactions between Tactical and Operational module.

The tool makes a few assumptions with respect to the scope: a) it only considers shipment of containers across the hinterland network of the given port and b) operations undertaken within the port (i.e., in-port) operations are not considered in detail. The tool will use the results of the GHG tool (developed in 4.5.1) to determine the emissions of the different transport modes. This information will be static and therefore does not require direct communication between the two tools. The output of the GHG tool will affect parameters in the back-end optimization of the Smart and green Logistics tool. This information provided by the GHG tool helps drive the decisions of selecting the greenest and cheapest transport chain from the port to the final container delivery destination.

3.3.2 Tool comparison

An exploration of the state-of-the-art has identified two tools with a similar purpose as the one developed here: Routescanner and Circoe. Both do, however, have a different perspective (see Figure 13). Where our tool focuses on the development of a collective shipment plan for the entire hinterland transport of containers from a given port, Routescanner focuses on the entire shipment plan for a single shipment. This is not limited to the hinterland transport of a single port but does also include deep sea transport. The main purpose of Routescanner is to identify different routes for shippers and to directly request a quote. The Circoe tool also takes the perspective of a single shipment but does narrow down to the hinterland container transportation from a single port. In their case that is the HAROPA port. This perspective is more similar to ours and Circoe is heavily involved in the development of the Smart and Green Logistics tool. The main difference is that Circoe provides options for individual shipments, whereas the Smart and Green Logistics tool makes a collective shipment plan. The input of the two tools will be aligned so that the individual shipment plan resulting from Circoe can be compared with the collective shipment plan. This provides insight into the potential benefit of collaboration between different hinterland shipments. As part of the collaboration, Circoe provides data on the hinterland network of the HAROPA port, which will be the first to be included in the Smart and Green Logistics tool.

Metrics	routescanner	CIRCOE	RSM Leading
Relevance to the Shippers	●●●	●●	●
Relevance to the Operators	●●	●●	●●
Relevance to the Port Authorities	●	●●	●●●
Network	Global	HAROPA Hinterland	HAROPA Hinterland
Planning Level	Operational	Operational	Operational & Tactical
Tool Value-Added	Emission & Lead Time Minimization	Emission & Cost Minimization	Increasing the Efficiency of Hinterland Operations
Integration w. Port Operations	N/A	New modes, new regular services	Potential Integration with the Digital Twin

Figure 13 - Comparison of tools

3.3.3 Tool structure

The tool consists of two modules, each focusing on a different level of decision making. The operational module takes a list of container shipments that is currently ready for shipment and creates a hinterland shipment plan for all these shipments together. This shipment plan is based on the existing connections in the hinterland network and balances three objectives: time, cost and emissions. The tactical tool will take a year of shipment data and run the operational model for each day to get insight in the long-term modal split under a given hinterland network. This allows for evaluation of the impact of modifications to the hinterland network.

The version of the tool that will be developed will not directly communicate with the digital twin or any of the other tools. Instead, it reads data from a database that contains the shipment information. The way this is set up allows for a connection with the digital twin. However, as we do not expect the digital twin to have access to real-time shipment data, there is no value in building this connection. Our tool will be a stand-alone tool that ports can use to evaluate the benefit of a collective shipment plan compared to a shipment plan based on decision made by individual shippers. This aligns with the concept of a Hinterland Master in a port environment that would have the authority to make shipment plans. The GHG tool will provide input to our tool in static manner as it provides values for parameters used in the optimization engine.

Identification of inputs and outputs

The main inputs for the tool consist of the available hinterland transportation services and the demand for hinterland transportation. We distinguish three main modes of transportation for the hinterland: barge, rail and truck. For the first two, the tool requires detailed schedules for all relevant services from the port to the hinterland. The schedule should provide the timing, intermediate stops, capacities and costs. For truck transportation this is not needed as we assume there is enough supply of truck transportation services. We do, however, need expected costs for this service as an input. Besides the schedules, we need emission estimates for the different modes of transportation.

The second important input relates to the demand for container shipment. For the operational module the demand for shipment for the next 48 hours should be provided by the user. This includes the origin, destination, size, weight, required delivery date amongst

others. For the tactical tool, we need the demand data for a longer period of time. We envision a one-year horizon for the tactical module. Instead of actual data on historical shipments, we could also work with aggregate data that allows us to generate realistic data.

The output of the operational model will be a shipment plan for the containers that need to be shipped in the next 48 hours. Each shipment will be assigned to a service transporting it to the final destination. On top of the details of the shipment plan, the tool will also provide key performance indicators related to the shipment plan. This includes, among others, total emissions, on-time performance and total cost. On the tactical level, the main outputs relate to aggregate metrics, such as modal split, total emissions, resource utilization. A detailed data model is provided in Figure 14.

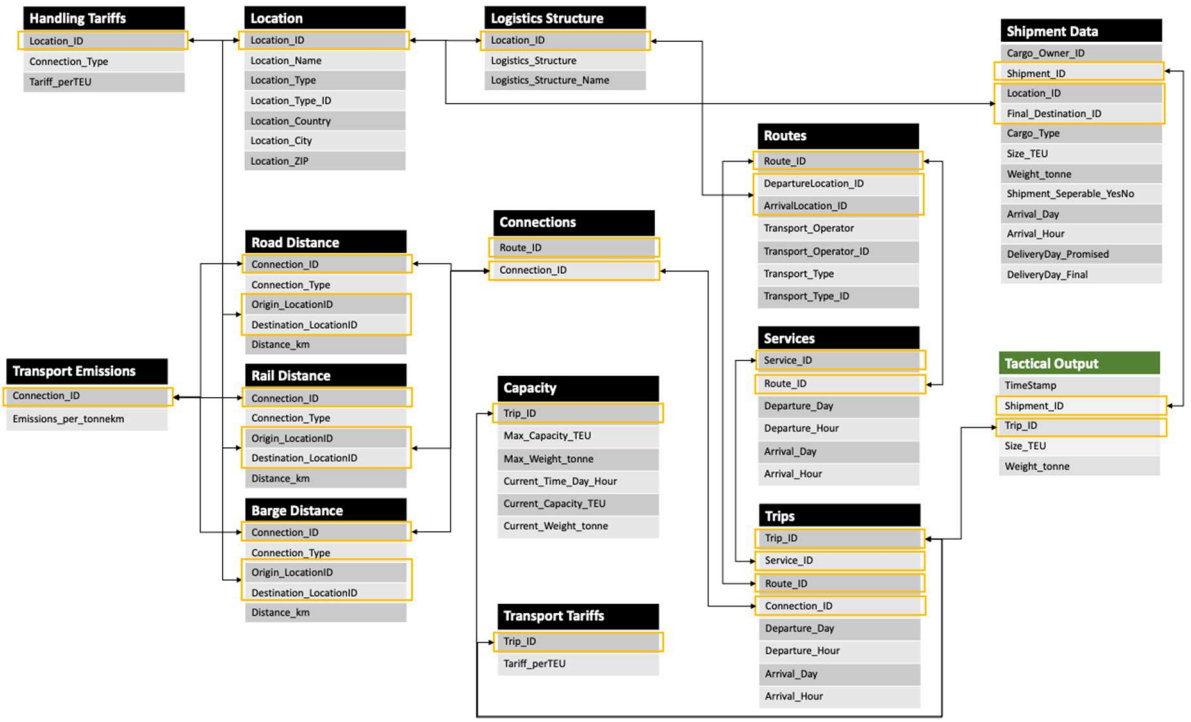


Figure 14 - Detailed data model Smart and Green Logistics Tool

Functionalities

The intended user of the tool is a port authority that is interested in evaluating the hinterland transportation from its port. Example use-cases include:

Operational module:

- Port Authority evaluates the **optimal hinterland transportation planning** of the arriving container on a given day
- Port Authority **compares** the system optimal hinterland transportation with **the selected modes of the terminal operators**
- Port Authority incentivizes the terminal operators / cargo owners to opt for the 'Port Authority' suggested planning

Tactical module:

- Port Authority evaluates the potential expansion of services at new or existing routes
- Port Authority evaluates the long-term effect of **increasing demand** or the short-term effect of a **demand shock**
- Port Authority evaluate the effect of **increasing penetration of electric trucks** on the hinterland transportation network

Description of User-Interface

The user interface of the tool consists of four main pages:

1. Main page for module and port selection
2. Interface for operational tool
3. Interface for tactical tool
4. Results page

Main page for module and port selection

The first page any user will see is a page on which the user can select the module they want to use and the port for which the tool will be used. Figure 15 gives a sketch of what this page will look like.

Figure 15 - Interface of main page of Smart and Green Logistics Tool

The user can select either the operational or tactical module of the tool. On top of that, the user should select the port for which the tool will be used. Which ports will be available in

the tool depends on data availability. Ports within MAGPIE that are interested in the use of the tool should work with the tool developers to make the data for their port available. What data is required from a port in order to make the tool available is described in the "input and output" section above. Depending on the selections on the main page, the correct next page will load.

Interface for operational tool

The use case of the operational tool is that it takes the demand for container shipment for the next 48 hours and makes a shipment plan using the available resources for hinterland transport. These resources include all modes of hinterland transport, including at least barge, rail and truck. For barge and rail we assume that the schedules and capacities are known. These can be loaded into the tool from the interface page (Figure 16). This data should be available in the back-end of the tool and is assumed to not change at the operational level. A map of the hinterland network connections will be visible on the page. No schedule is required for the truck transport as the model assumes that sufficient capacity is available for direct truck shipment from the port (or any other inland terminal) to the final destination. On this page, the user can load the demand for container shipment for the next 48 hours from the port. Details of the required input for each shipment is given in Section "input and output". After loading the data, the tool will show the provided demand data on the page. Finally, the page will give the user the opportunity to set the weights of the different objectives. We expect to include cost, emission and time-related parameters. These weights will influence the decision made by the optimization model and thereby will affect the modal split. Once the calculations are completed, the results page will open.

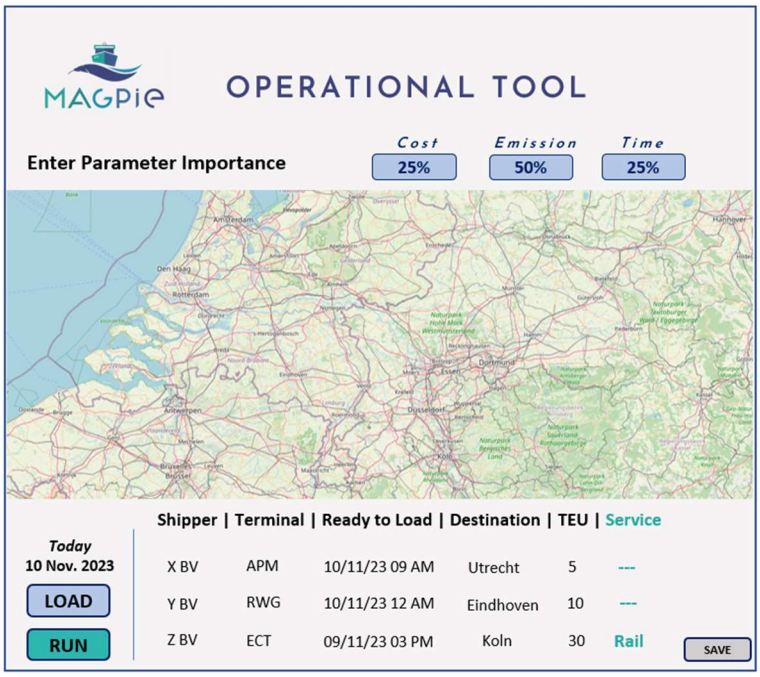


Figure 16 - Interface of Operational Module of Smart and Green Logistics Tool

Interface for tactical tool

The main module of the tool will be the tactical module that evaluates the impact of changes to the hinterland transportation network on the modal shift of container transport. The User Interface of the tactical tool (Figure 17) will first show the status quo network for hinterland transport. This should be available in the back-end of the tool. Then, the user can make changes to the network by modifying certain connections. For example, a service can be added, the frequency or capacity of a service can be modified. Alternatively, the demand scenario can be changed to evaluate the impact of alternative demand scenarios. The scenarios that can be evaluated are port-specific and can be developed as part of the data preparation process. As in the operation module, the user can set the relative importance of the different objectives. Finally, a simulation horizon can be set. This determines the length of the simulation, for which the default value will be one year. Pressing run will start the simulation in which a shipment plan will be made for each day of the simulation horizon, using the operational tool as a subroutine. Once the calculations are completed, the results page will open.

Figure 17 - Interface of Tactical Module of Smart and Green Logistics Tool

Results page

This page will give an overview of the results of the simulation. For the operational tool it will give a shipment plan for each shipment and an overview of the key performance indicators cost, emissions, time and modal choice. It might give additional information on the utilization of the different available services. For the tactical tool, the results page will focus on the modal split and the resulting cost and emissions.

3.3.4 Availability of the tool

The tool will be made available to end-users via a web interface to users in ports for which the data has been included in the tool. Depending on data limitations, there might be a log-on requirement for calculations based on data with restricted access. For at least one port,

the tool will be fully available for interested users so that the capabilities of the tool can be demonstrated.

For most users, there will be no need to access the code of the tool. Nevertheless, a link to download the code can be provided upon request.

3.3.5 Expected timeline of development

A mock version of the tool is available at the time of submitting this deliverable (March 2024) upon request. After that, two parallel phases of development will take place: 1) adding port data to the tool, 2) improving the back-end optimization model. A first realistic port case is expected to be available in the tool by September 2024, which allows for testing the optimization model on realistic instances. Around that time, the first version of the optimization model will also be ready. In February 2025, the final version of the optimization will be available and additional use cases will be available. After that, the main focus will be on obtaining insights from the use cases of the tool. By the end of the work package (M48 of the MAGPIE project), we expect to deliver a report describing how the tool can be used to support tactical decision making in a port.

Milestones

Data for first port available in tool	September 2024
First version of optimization model	September 2024
Final version of tool available	February 2025
Second real case available in tool	February 2025
Final version of optimization model	February 2025
Report on insights from use case	September 2025

4 Implementation of back-end models

This chapter outlines the implementation status of the back-end models that support the three main tools described in Chapter 3, especially the Energy Matching Tool. Further connections of the backend models with the other tools described in Chapter 3 are currently being explored and will be implemented as required for the final version of the digital tools and models to be delivered in M48.

This section describes the state of development, model architecture, software and hardware dependencies, access to the code and models, installation instructions, example use cases, and development timeline for the following backend models: energy demand of terminal assets (4.1), flexibility modelling of terminal assets and buildings (4.2), renewable resources sizing and power production estimation (wind and solar PV), and forecast of renewable power production (from solar PV and wind) (4.3).

4.1 Energy demand model for port terminal assets

This section outlines the implementation of the energy demand backend model for terminal assets, which is described in detail in deliverable D4.4. The backend model has been implemented in a set of Python modules that are available as stand-alone or through a Dash interactive Python framework. An app has been developed for the MAGPIE project that can be used to analyse the energy demand of different port terminals. The modules consider the missions and assets scheduled for logistics operations to estimate the electricity demand of each type of asset.

The current version of the Python modules and the app can estimate the energy demand from reefers, cranes, and containerships that are connected to On-shore Power Supply systems (OPS). To estimate the electricity demand, these modules source the logistics information from the outputs of the Container Terminal Simulator provided by CEA. In future versions of the tool, additional modules will be developed for assets such as charging stations and buildings. Enhanced estimation algorithms and models will also be implemented. Furthermore, the models for energy demand estimation of assets in other types of terminals described in Deliverable 4.4 will also be implemented in later versions of the app and additional modules. Examples of the additional assets include compressors and pumps for liquid bulk terminals and belt conveyor systems for dry bulk terminals. The purpose of these models and the app is to process logistics information and understand the type, structure, and measurement units required to estimate the electricity demand in ports. Another main goal of the demand modules is to generate energy demand time series that are necessary as input for the Energy Matching Tool.

The tool has been implemented using an open-source perspective and will be hosted in a public repository. To learn more about the app structure and how it can be modified, visit https://dash.plotly.com/tutorial.

4.1.1 Instructions for setting up the Plotly Dash needed for the app

Before installing Dash, Python should already be running. To avoid compatibility issues, the tool was built using the packages and versions described in Table 7.

Table 7 - Python packages used in the implementation of the terminal energy demand app.

Python environment	
spyder	IDE 5.4.3
Python	3.10.12
dash	2.7.0
pandas	1.5.3
flask	2.2.2

To install Plotly Dash, *pip* (pip3 install dash) or *conda* (conda install conda-forge::dash) can be used. These commands should be download and install *dash*, *plotly*, and the other required packages (except perhaps *pandas*). The app also needs several python libraries, but normally these are already included in the python distribution, but if not, please also install: *numpy, pandas, itertools, ast, datetime* and *os*.

4.1.2 Backend models and electricity demand estimation

This section provides a brief overview of the backend models and strategies used to estimate the electricity demand for cranes, ships connected to OPS, and reefers. For more detailed information, please refer to Deliverable 4.4. It is important to note that the models already implemented in Python can be utilized to estimate the electricity demand of these assets independently of the app use. In other words, the code presented in this section can be easily extracted and used elsewhere. This is one of the main outcomes of the work done until now.

Energy Demand of Cranes

Eq. (1) is used to estimate the hourly electricity demand of cranes (P_{c_t}). In this equation NC represents the number of cranes in the terminal. First, the committed cranes during each hour are identified by introducing the state variable $s_{i_t} \in \{0,1\}$, which is equal to zero if the crane *i* is not working during hour *t*. In the span of one hour, a crane can complete multiple cycles of loading or unloading containers. To estimate the energy demand of each crane during a specific hour, the maximum power required during any of these complete cycles is considered. This maximum power corresponds to the power required when lifting a container from the ship side or the berth side.

For simplicity, a simple logical rule has been implemented. During the analysed time period of an hour, the lifting power of each individual crane $P_i(\bar{x}_{i_t})$, expressed in kW, is estimated based on historical data and the weight of the heaviest container to be lifted by each crane $\bar{x}_{i_t}$ in tons, using Eq. (2):

$$P_{c_t} = \sum_{i=1}^{NC} s_{i_t} \cdot P_i\left(\bar{x}_{i_t}\right) \quad (1)$$

$$P_i(x) = \begin{cases} P_{i_1}, & 2 \le x < 15, \\ P_{i_2}, & 15 \le x < 32.5, \\ P_{i_3}, & x \ge 32.5, \end{cases} \quad (2)$$

The ranges for container weight are determined by three factors: (1) the mass of an empty container or a container filled up to about half of its maximum gross weight, (2) the maximum gross weight for general purpose containers, which is 32500 kg, and (3) the possible weight of special containers, such as flatracks or reefers. It's important to note that that more ranges can be created if there is more detailed information available about the relationship between container gross weight and the power required to move them. Also, it's worth mentioning that the current version of the container terminal simulator developed by CEA does not provide weight of containers as an output, thus the current version of the energy demand app and modules assume a weight of 32500 kg or more for all containers (third case in Eq. (2)).

For each crane, the parameters P_1, P_2 and P_3 should be introduced by the user in the cranes' input file. An additional parameter P_{mean} is also considered. This value is the mean demand during an entire cycle and is used for estimating the energy consumed for one hour, by analyzing the number of completed cycles of each crane.

Energy demand of Ships and OPS at berth

The first step in the estimation of the hourly electricity demand of ships connected to OPS systems (P_{OPS_t}) [kW], is the calculation of the occupancy of each berth (and consequently the occupancy of each OPS system, note that NO is the number of installed OPS systems) by using the state variable $o_{i_t} \in \{0,1\}$, which is equal to zero if the OPS *i* is not being used during hour *t*. This is followed in a second step by the calculation of the minimum between each OPS capacity ($\overline{OPS_i}$) [kW] and the estimated electricity demand of each connected ship (S_i), as shown in Eq. (3):

$$P_{OPS_t} = \sum_{i=1}^{NO} o_{i_t} \cdot \min\{\overline{OPS_i}, S_i\} \quad (3)$$

The relationship between the gross tonnage of a containership (GT) [tons] and the electricity demand at berth is discussed in detail in Deliverable 4.4 thus this section provides only a summary of the model used. The electricity demand of ships at berth is obtained by using the rated nominal power of the onboard auxiliary engines (P_{AE}) [kW] and considering a load factor (lf) that reflects the average power required at berth, as a fraction of the peak (or rated) power. However, as the power of the auxiliary engines is not always available, the current implementation of the model uses information that is easily accessible on the relationship between the gross tonnage (GT) [tons] of the ships and the electricity demand of containerships at berth, given in Eq. (4):

$$S_i = 2.9165 \cdot GT_i^{0.8719} \cdot \rho_\alpha \cdot lf_i \quad (4)$$

In Eq. (4), the first two terms are used to estimate the installed rated power of the main engines and ρ_α is a coefficient that estimates the ratio between the main and auxiliary engines installed power. For containerships a value of 0.25 is considered for ρ_α. For each ship, the user must introduce the gross tonnage and load factor, as well the capacity of each installed OPS available at berth. Note that, if the P_{AE_i} is already available for a particular vessel, the user should provide as gross tonnage, GT_i, the following value instead:

$$GT_i = \left(\frac{P_{AE_i}}{\rho_\alpha \cdot 2.9165}\right)^{1/0.8719}$$

Energy Demand of Reefers

Currently a simplified model is implemented in the app for estimating the electricity demand of reefers. The model considers a typical power requirement of 12 kW for each reefer and assumes a worst-case scenario that this power is required whenever the reefers are connected at the yard. In further versions of the app and backend model, an enhanced model will be implemented to consider additional variables, such as the specific type of cargo being stored in the reefer and external temperature.

Inputs

Table 8 summarises the type of input data required for demand models of the assets in the current implementation of the app and backend demand models. A use case has been implemented as an illustrative example, and the demand was estimated using the parameters in Table 9. It is important also to understand that information coming from the operational side, is also relevant. For the cranes, it is mandatory to know when a cycle started and when it ended; for the ships, the berthing time must be known; and for the reefers, when a particular reefer was plugged in and plugged out when in the yard.

Table 8 - Input data required for the demand estimation.

Equipment	Data	Description	Unit
Crane	P_1	Average power required for lifting an empty container	kW
	P_2	Average power required for lifting a full container	kW
	P_3	Average power required for lifting a special weight container	kW
	P_{mean}	Average power required during a complete loading/unloading cycle	kW
	x	Container weight	tons
Ship/OPS	$\overline{OPS}$	OPS capacity	kW
	GT	Gross tonnage of the ship	tons
	lf	Load factor of the auxiliary engines of the ship when at berth	-
Reefer	P	Typical power requirement of a reefer container (considered 12 kW)	kW

Table 9. Use case input data.

Equipment Type	Equipment	Data	Description
Crane	'Portique_NA_1'	(290,200,100,150)	$(P_1, P_2, P_3, P_{mean})$
	'Portique_NA_3'	(300,200,100,170)	

	'Portique_NA_5'	(295,205,100,160)	
	'Portique_NA_6'	(270,160,100,140)	
OPS	'N1'	2000	$\overline{OPS}$
	'N2'	2500	
	'N3'	3000	
Ships	'NA_1'	(50084, 0.5)	(GT, lf)
	'NA_2'	(24257, 0.6)	
	'NA_3'	(33604, 0.33)	
	'NA_4'	(49000, 0.78)	
	'NA_5'	(45000, 0.45)	
	'NA_6'	(60000, 0.33)	
	'NA_7'	(236078, 0.4)	

4.1.3 Dash Code Structure

The current implementation of the energy demand models of the terminal assets is a Plotly Dash app (a web analytic application) with multiple pages. The structure of the app is shown in Figure 18.

```
CTS_DASH
|   app.py
|
└── assets
|   |   img.png
|   |   pre_processing.py
|
└──pages
|   |   __init__.py
|   |   assets_control.py
|   |   energy.py
|   |   home.py
```

Figure 18 – Structure of the Plotly Dash app that implements the backend energy demand models of terminal assets.

The main components of the app are:

1. The `app.py` is the main app file, which is the entry point to the multi-page app (and in which it is included three pages (`home.py`, `energy.py`, `assets_control.py`) in the `pages` directory.
2. The directory `assets` includes the files needed for the app execution. The directory contains the file `img.png` (the illustrative image used in the home page of the app); and the file `pre_processing.py` (a file where different classes, methods and functions are defined for reading the input files and obtaining the energy and activity levels of the assets);

3. Finally, the pages of the app are defined in the `pages` directory. The empty `__init__.py` file is required to manage the import from the `pages` directory.

The structure and main characteristics of each of these Python modules are described below.

app.py

This is the entry point to the app. The current version is only deployed locally. To launch it, users should run the Python `app.py` and open the given link. Once the app is running, the user should access the homepage defined in the file `home.py`. From here, the different pages can be accessed by clicking on the respective links.

***Home** page (home.py)*

A brief description of the app's objectives is provided on the initial page of the app (`home.py`). On this page, all the information required for the estimation of the demand should be introduced under the **ASSETS INFO** section as shown in Figure 19.

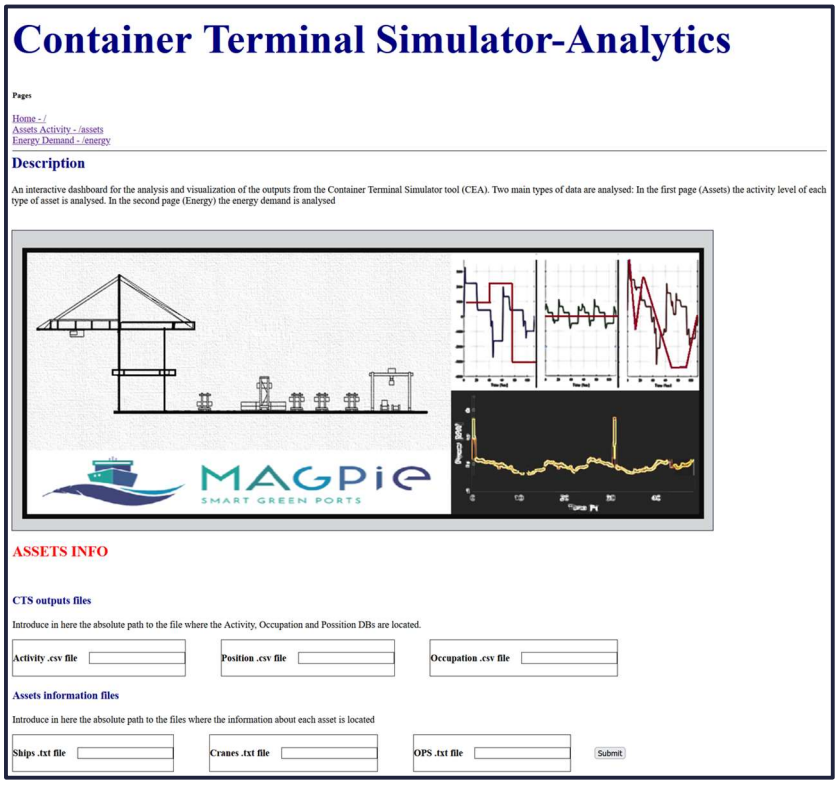


Figure 19. Screenshot of the app's homepage.

The required inputs are divided in two groups:

1. Container Terminal Simulator (CTS) outputs files: the absolute paths to where the **Activity, Position** and **Occupation** databases (.csv) to be analysed are locally stored. The input required from each file is as follows:
    a. Activity file: Relevant information about each crane is extracted from this file, i.e., the exact moment when the loading and downloading operations start and end is identified.
    b. Occupation file: This file provides information on the exact moment when each reefer is plugged in and out.

c. Position file: This file provides information on the exact position at berth and the berthing duration of each ship calling at the port. This information is used for estimating the energy consumption of the OPS systems.

2. Assets information files: The user must provide the absolute paths to the files (.txt) containing the parameters for estimating the energy demand of the ships and cranes. The structure of these inputs is the following:
   a. Ships input file: In this input file, each row characterizes a particular ship. The first column contains the identifier of the ship, the second column contains the gross tonnage (in tons) and the load factor is shown in the last column. Note that the identifier of the ship should be the same as provided in the CTS output files.
   b. Cranes input file: Each row of the file characterizes a particular crane. In the first column should contain the identifier of the crane; in the second column the lifting maximum power (P_3 - a fully loaded container); the third column the lifting medium power (P_2); the fourth the minimum lifting power (P_1 - an empty container); and the last column should contain the average required power (P_{mean} - in a cycle). Note that the identifier of the crane should be the same as provided in the CTS output files.
   c. OPS input file: Each row corresponds to an OPS system located on a specific berthing position. The first column identifies the berthing position (linking each ship with a specific OPS system during the berthing time), and the second column corresponds to the maximum capacity of the OPS (expressed in kW).

After providing the required paths (if the input files are correctly formatted) and once the **Submit** button is clicked, a message should be displayed in the app explaining that three files were created at the root directory where the app is running (***.../CTS_DASH***):

1. A .csv file with the hourly demand of the assets (see Table 10);
2. A .csv file with the hourly activity of the assets (see Table 11);
3. A .txt file named *initializing.txt* containing several paths to files that the app requires for the visualization of the results on the other pages.

Table 10. Structure of the output file (.csv) with the estimation of the hourly electricity demand.

	Demand_ Vessel	Demand_ Cranes	Demand_ Reefer	Demand_ Total
2022-01-02 00:00:00	0	0	6000	6000
2022-01-02 01:00:00	0	0	6000	6000
2022-01-02 02:00:00	0	0	6000	6000
2022-01-02 03:00:00	0	0	6000	6000
2022-01-02 04:00:00	0	0	6000	6000
2022-01-02 05:00:00	0	0	6060	6060
2022-01-02 06:00:00	0	0	6072	6072
2022-01-02 07:00:00	0	0	6120	6120
2022-01-02 08:00:00	0	0	6060	6060
2022-01-02 09:00:00	0	2127	6084	8211
2022-01-02 10:00:00	295	5038	6156	11489
2022-01-02 11:00:00	565	5038	6216	11819

Table 11. Structure of the output file (.csv) with the estimation of the hourly electricity demand.

	Activity_Vessel	Activity_Cranes	Activity_Reefer
2022-01-02 00:00:00	[]	[]	500
2022-01-02 01:00:00	[]	[]	500
2022-01-02 02:00:00	[]	[]	500
2022-01-02 03:00:00	[]	[]	500
2022-01-02 04:00:00	[]	[]	500
2022-01-02 05:00:00	[]	[]	505
2022-01-02 06:00:00	[]	[]	506
2022-01-02 07:00:00	[]	[]	510
2022-01-02 08:00:00	[]	[]	505
2022-01-02 09:00:00	['NA_3']	[]	507
2022-01-02 10:00:00	['NA_2', 'NA_3']	['Portique_NA_5']	513
2022-01-02 11:00:00	['NA_2', 'NA_3']	['Portique_NA_5', 'Portique_NA_6']	518

These output files are obtained under the hood using a callback function triggered only after clicking the button. The file with the estimated hourly energy demand can then be used by the Energy Matching Tool (EMT) as an input to the optimisation problem that must be solved. Figure 20 provides a general schematic summary of the input files required and the output obtained when running the app.

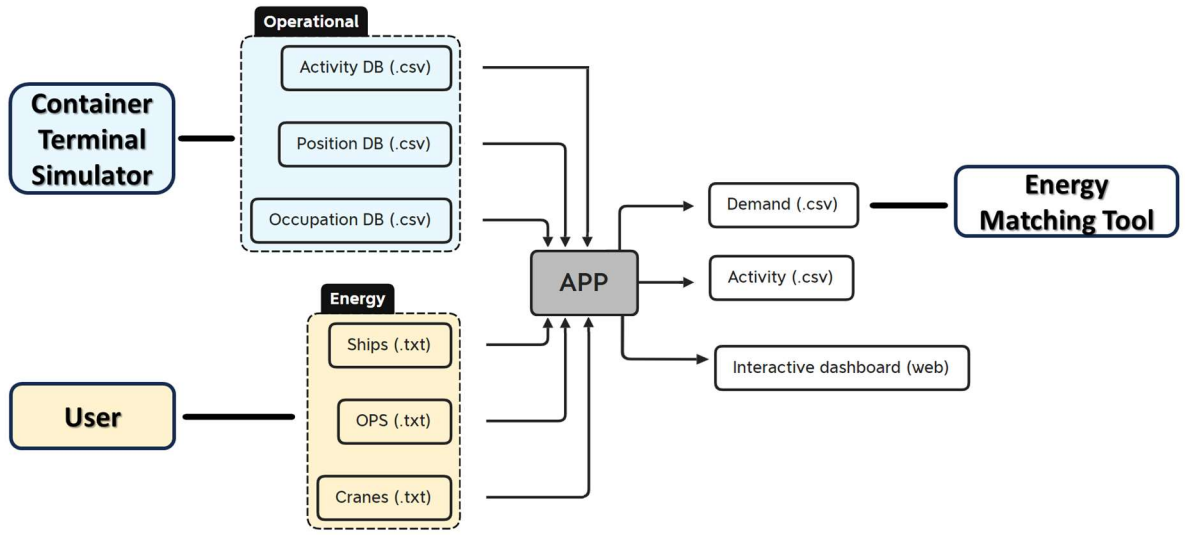


Figure 20. Input and outputs from the app.

Assets Activity page (assets_control.py)

This page analyses the activity of each type of asset. A basic dropdown menu allows users to select which asset type to visualise: ships, cranes, or refeers. Once selected, two types of graphs are shown as shown in Figure 21.

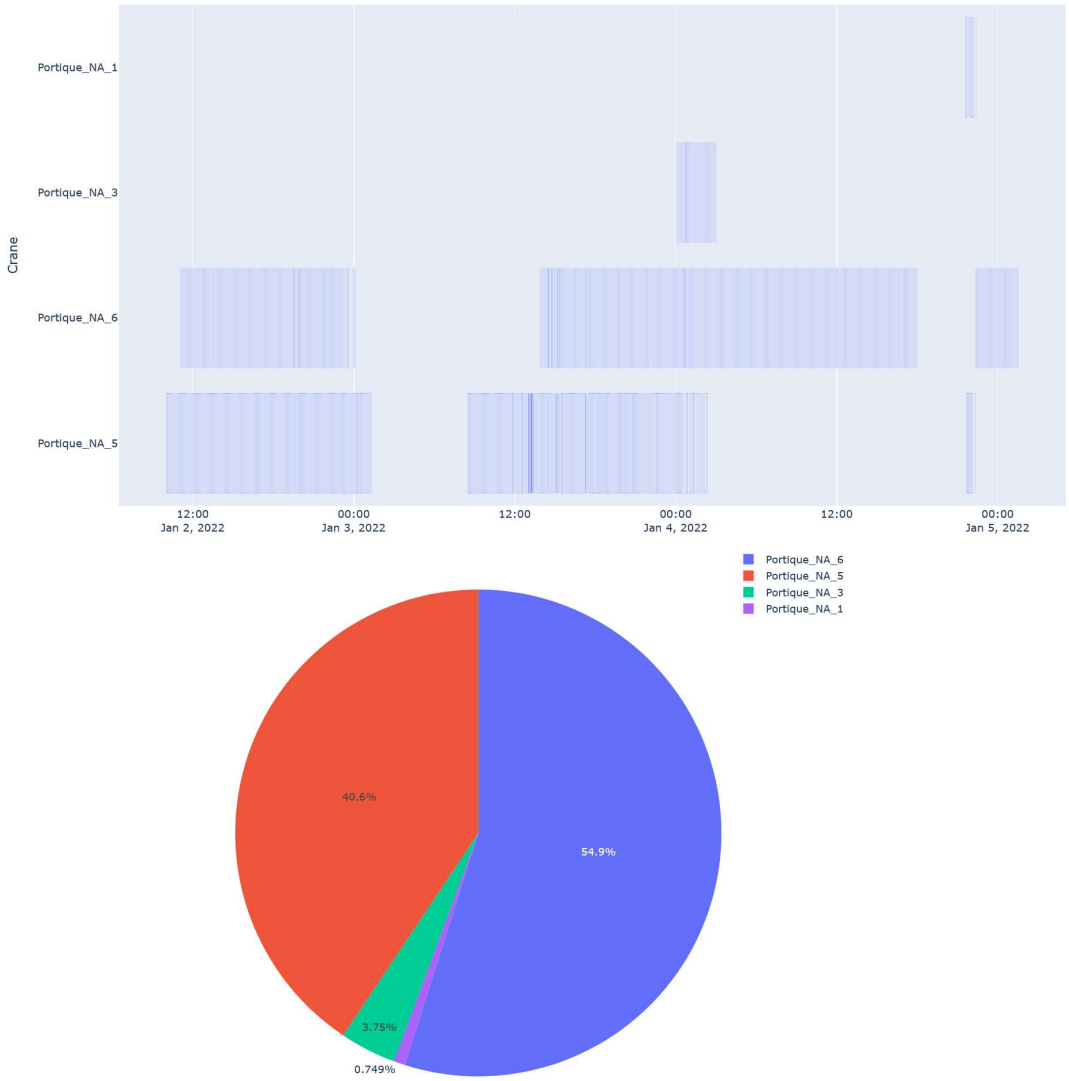


Figure 21. Selecting the crane assets to visualise its activity.

Figure 21 shows the visualisation of the results of the cranes' activity. The top graph depicts all the active cranes during the analysed period and the exact moment when these are being used for loading or unloading operations. The second graph shows the proportion of the operation time of the cranes during the analysed period. Figure 22 shows the results for the selected ships. These include the berthing position of each ship; in case of the use case, these are N1, N2 or N3. The positions can be directly associated with a specific OPS system if the location of each OPS is known. Other typical parameters that can be obtained from the app include each ship's berth occupancy or berthing time. As in the case of the cranes, the second graph is a pie chart showing the proportion of the operation time of the OPS systems during the analysed period.

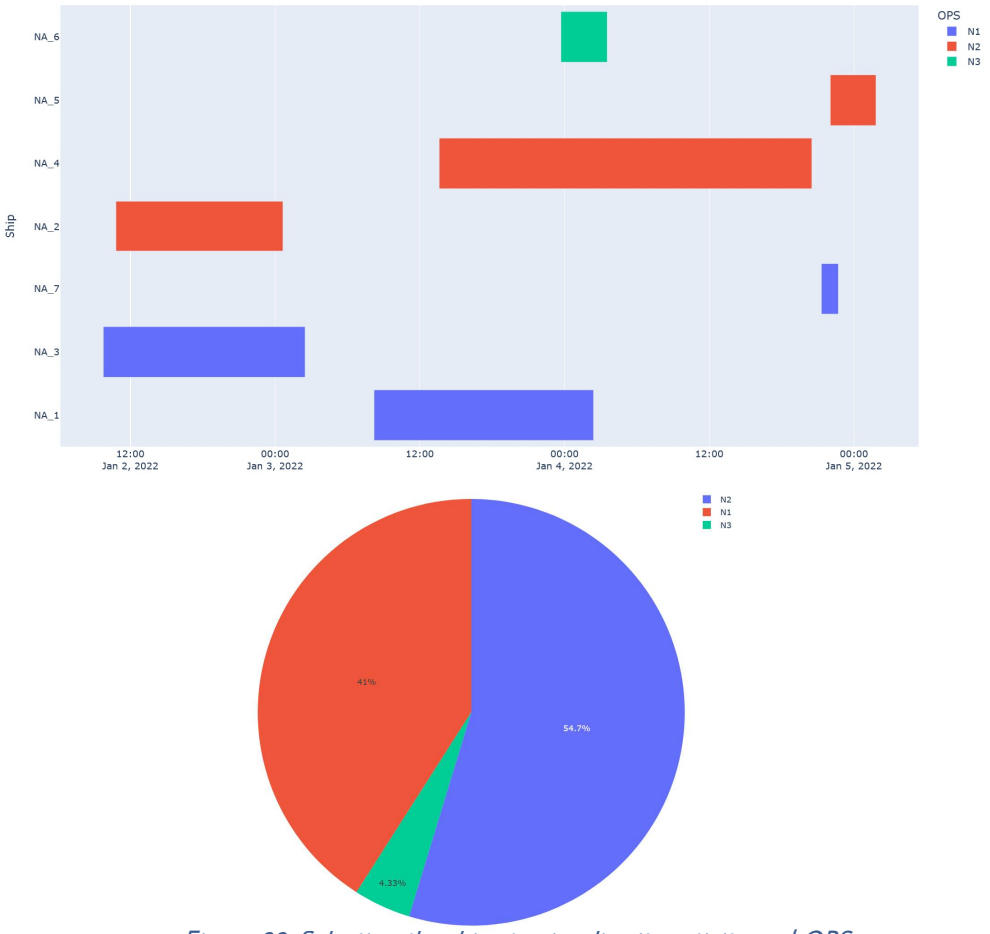


Figure 22. Selecting the ships to visualize its activity and OPS use.

When reefers are selected in the dropdown menu, the app shows the number of reefers plugged in during each hour, as shown in Figure 23. Additionally, the app shows all entries of a specific asset selected from the original databases through a second dropdown menu.

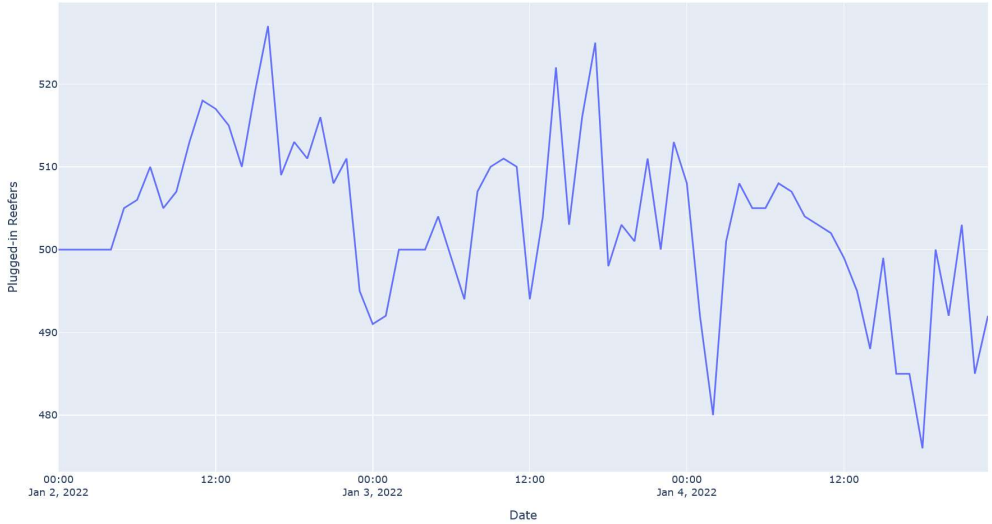


Figure 23. Selecting the refeers to visualize the quantity plugged in.

Energy Demand *page (energy.py)*

In this page, the energy demand is analyzed. A dropdown allows to select if the user wants to visualize the total demand (considering the sum of the demand of cranes, ships and reefers) or only the demand of each type of asset. In Figure 24 we show the total energy demand and the contribution of the different group of assets, note also that each individual asset can be visualized as well.

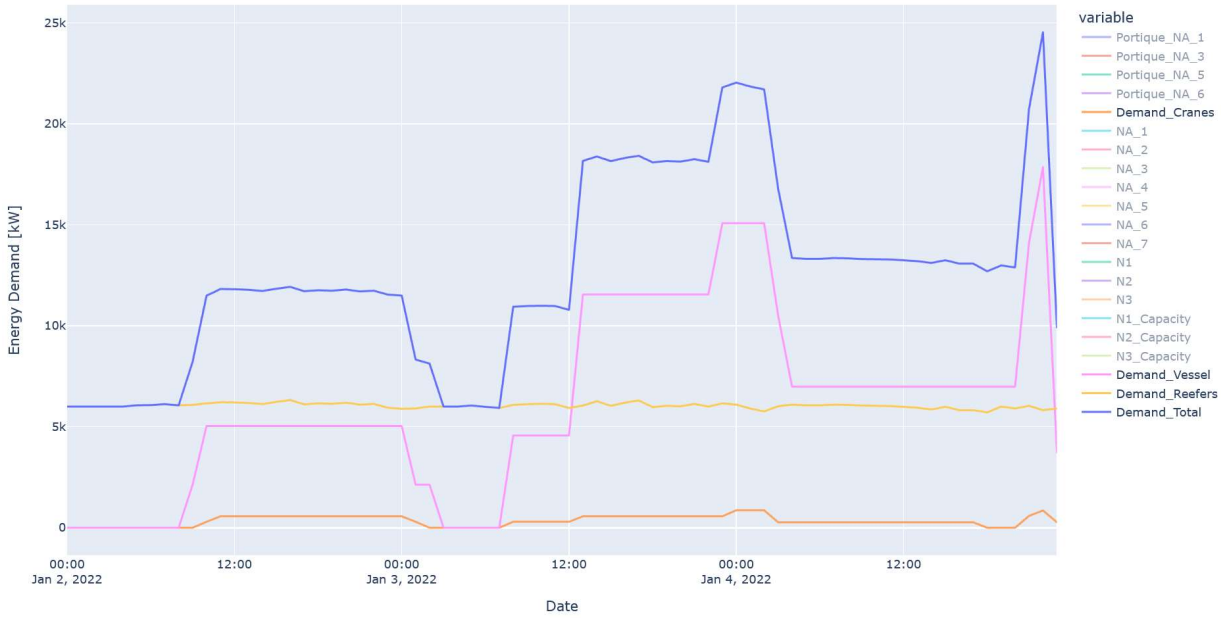


Figure 24. Energy demand visualization.

When the ships or cranes are selected, an additional table is deployed (e.g., Table 12), showing the parameters used for the estimation (i.e., the parameters from the input files).

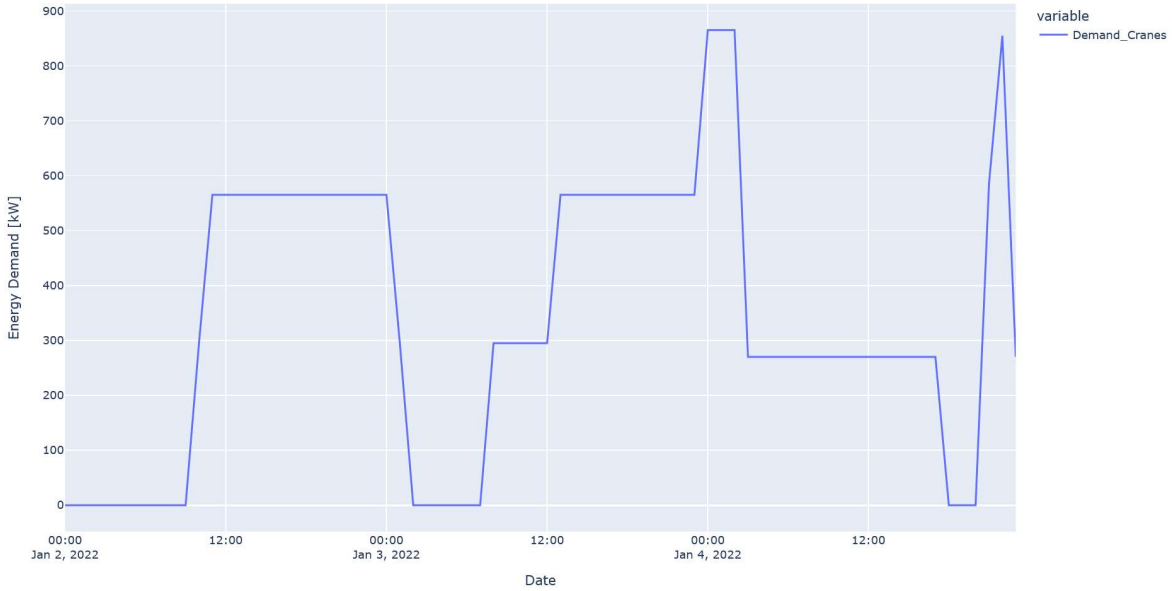


Figure 25. Energy demand from cranes.

Table 12 - Energy parameters used for estimating the energy demand of cranes, shown in the app UI.

Crane	Max Lift Power [kW]	Med Lift Power [kW]	Min Lift Power [kW]	Average Lift Power [kW]
Portique_NA_1	290	200	100	150
Portique_NA_3	300	200	100	170
Portique_NA_5	295	205	100	160
Portique_NA_6	270	160	100	140

In Figure 25 we show the results when the cranes are selected. It can be observed both, the visualization of the energy demand and the parameters used for the estimation; whereas in Figure 26 we show the results when the OPS option is selected, instead.

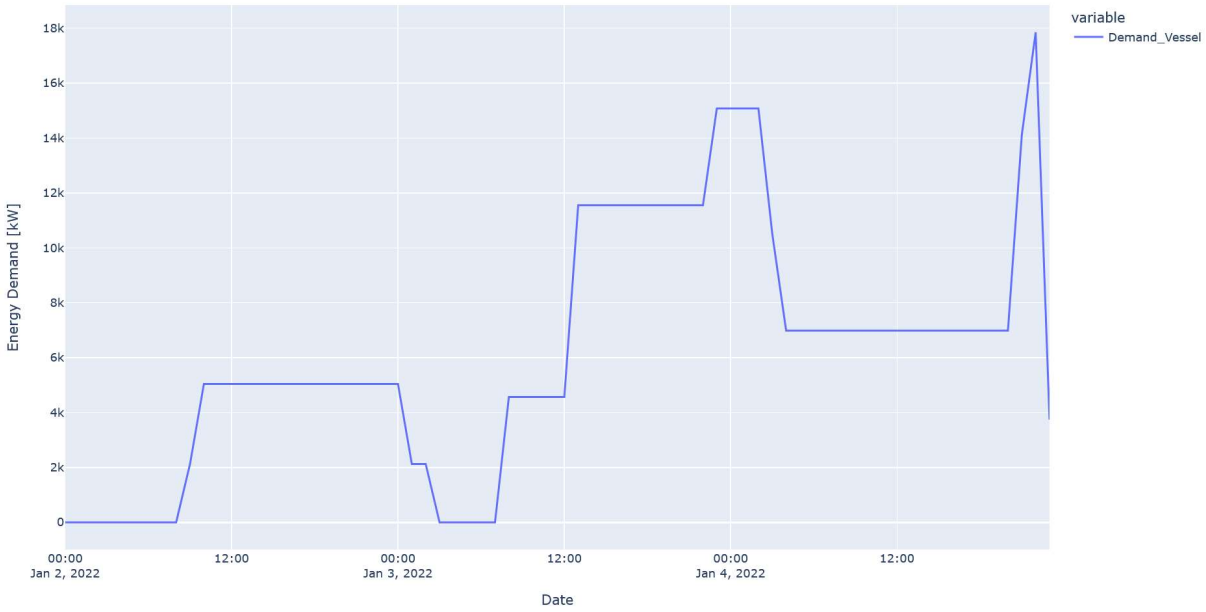


Figure 26. Energy demand from ships/OPS.

Table 13 - Parameters used to estimate the energy demand of OPS systems, shown in the app UI.

Ship	AE Power [kW]	Load Factor	OPS	OPS Capacity
NA_1	4565	0.5	N1	2000
NA_2	2911	0.6	N2	2500
NA_3	2128	0.33	N1	2000
NA_4	6987	0.78	N2	2500
NA_5	3742	0.45	N2	2500
NA_6	3527	0.33	N3	3000
NA_7	14113	0.4	N1	2000

4.1.4 Development Timeline

The modules and app will be further developed to improve the complexity of the models (e.g., reefers) and cover additional assets (e.g., charging stations). These developments are expected by the end of 2024. The integration of the demand models with the EMT tool is

ongoing, with a first implementation expected by the end of March 2024 (M30). The final version of the modules and app will be available in M48. The open-source calculation modules will be publicly available through a public repository, such as GitHub.

4.1.5 Access to app and modules

The first version of the app and modules can be downloaded via a link that can be provided upon request. The final version of the modules and apps will be available open source via a public repository, such as GitHub.

4.2 Flexibility modelling

4.2.1 Flexibility of Buildings

Buildings are huge energy consumers and important GHG emitters: in 2022, 30% of world global energy consumption and 26 % of the global energy-related emissions are due to buildings' operation regarding IEA statistics [35]. In the context of the climate change with global warming increasing conditions, the consortium of the MAGPIE European project is striving to reduce strongly the use of fossil fuels and to replace them by renewable energy sources onto the European harbours. For a few years, buildings of all types (offices, industries, accommodations, collective or individual housing) switch from pure energy consumers to also energy producers and more recently, to energy flexibilities providers. The buildings connected to various networks (electrical grids, district heating …), can behave as batteries do, enabling to store energy (real charge or increase the consumption) whenever the network energy providing capacity is bigger than demand or to discharge energy to the grid (real discharge or decrease of consumption) whenever the demand overpasses the supply capacity of the network. This is true for both electrical grid and district heating. A kind of BEMS (Building Energy Management System) controls the energy management of the building, anticipating the building needs but also local energy generation and supply capacities over the upcoming hours or days, and forecasting potential energy congestions. BEMS adapts the energy flow in an optimized way, shifting or shedding the building loads and even, anticipating the loads according to the grids necessity while deliberately ensuring the functionalities and services required for the building: occupants comfort (lighting, temperature, IAQ), systems and process running operation depending on the use of the buildings. This becomes possible thanks to the inertia of the buildings, which can be thermal inertia according to the building's structure, or flexibility of use as far as the building's activities allow it.

Numerical tool

The first step to integrate the flexibilities provided by buildings within the Energy Matching Tool (EMT) is to create a model of the building's thermal behaviour, taking into account all energy flows impacting the building, solar radiation, energy emission inside the building by dedicated emitters or other appliances linked to the activities hosted. This building model is intended for use in PMC (Predictive Model Control) to forecast congestion and energy needs in order to optimize the supply from renewable energy sources, and then from low carbon energy sources.

As described in the MAGPIE deliverable D3.2, the selected building model relies on physical equations defined in the deprecated, but still relevant, standard ISO 13790:2008 [36] the revision of this standard is the ISO 52016-1:2017 [37]). It is an electric-like model with four resistances and one capacity. The model must be fed by various inputs: the solar radiation, the internal heat gain from the various activities and devices, occupancy also in terms of set

point temperatures to follow. It is mainly constructed to calculate the resulting operative temperature resulting from the heat supply into the interesting area of the building.

The building model developed by CEA and proposed for MAGPIE uses this framework and is based on an acausal algebraic modelling approach, i.e. a MILP (Mixed Integer Linear Programing) code. This approach enables a wide range of problem-solving capabilities using the same model, simply by tuning the set of variables to be fixed and those to be found as model outcomes.

Another option would have been to draw a reverse ISO 13790:2008 [36] model as a usual sequential programing approach intended to calculate the time dependent energy required to ensure set point temperature inside the building. This method was dropped by CEA as it was deemed time-consuming and less efficient.

Amon other possibilities, the building model developed by CEA is suitable for determining supply energy needs as a function of the requested setpoint temperature. Moreover, the MILP model is ideally suited to finding out the solution to a linear problem under constraints. For the development of the MAGPIE generic EMT, the linopy Python library has been selected to enable linear programming.

Computer configuration for using the Building model:

Table 14 – Software and hardware requirements

Python environment	
Spyder	IDE 5.4.3
Python	3.9.18 (main, Sep 11 2023, 14:09:26) [MSC v.1916 64 bit (AMD64)]
Xarray	2024.1.1
Numpy	1.26.3
Pandas	2.1.4
Linopy	0.3.3
PC features	
Processor	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
Installed RAM	32,0 Go (31,9 Go usable)
System type	Operating system 64 bits, processor x64

Structure of building model

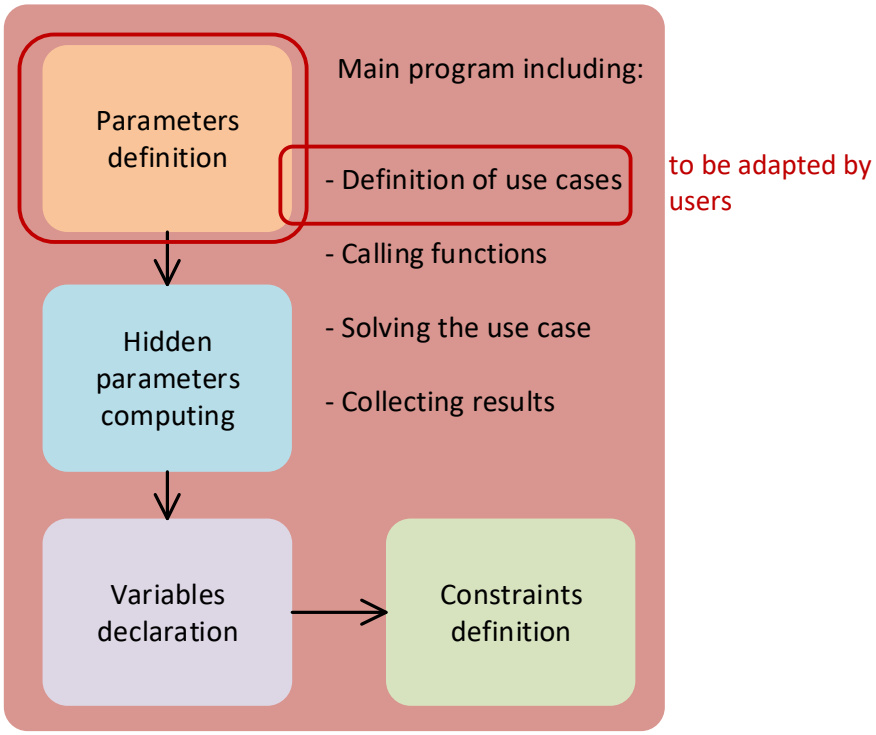


Figure 27 : Architecture of the building model developed by the CEA

Definition of the parameters

The so-called "*BuildingModel3_ParamsDeclaration*" block function includes the declaration and the definition of the material needed to feed the model, input data, thermal features of the building.

Since the linopy library doesn't take the parameters as properties into the linopy model class, input data and various parameters are stored in a specific dictionary structure: *BuildingModel3_Params*, for easy calling by the other block functions of the model.

Among the parameters to be declared, there are also the boundaries that limit the space of freedom of the various variables that have been declared in the "*BuildingModel3_VariablesDeclaration*" function (see below).This block function must be customized by the users in order to make the all parameters and input data fit the study case.

Computing the hidden parameters

This second block function "*BuildingModel3_ComputeParams*", calculates the hidden parameters of the building model, depending of the user's parameters declared previously. Nothing must be modified herein.

These parameters are also embedded into the dictionary structure: *BuildingModel3_Params*.

Variables declaration

As requested by the linopy model "bm", the variables of the model are declared in this block.

Nothing must be modified herein.

Constraints definition

In this block, the physical equations steering the building behaviour are written as constraints to be met by the model.

The building model is intended to be used to participate into the optimization EMS tool.

Nothing must be modified herein.

Main program

The main program drives the building model by executing step by step the different blocks before-mentioned.

It includes the definition of the time horizon over which to compute the building model and the number of buildings that are simulated with their own dimensional and structural features and that are exposed to the same environmental conditions.

The main program also entails the trigger of the building model in term of objective function. This is another part to be customized by the users within the EMT.

Testing the model

CEA has simulated a virtual building to illustrate the use of the model coded with linopy and to validate it. Features and environment context are mentioned hereafter.

Input data are mentioned with the prefix i_ and the parameters linked to the building study case are named like BM_p_

```
########################################
##          TIME HORIZON CONSIDERED          ##
########################################

# *** DEFINITION OF THE TIME HORIZON TO BE ADDRESSED: # TO BE COSTUMIZED BY USERS! ***
# ************************************************************************

TimeStep                    = 1/2                                          # time step, in hours

Nbr_Day_to_get              = 2                                            # number of days for the horizon considered,
in #

N_TimeStep                  = int( Nbr_Day_to_get * 24 / TimeStep )        # number of time steps for the horizon considered,
in #

mainParam_TimeStep      = TimeStep * np.ones((N_TimeStep))          # array of time steps along the horizon considered


#########################################################
##      NB OF BUILDINGS CONSIDERED WITHIN THE USE-CASE      ##
#########################################################

# *** DEFINITION OF THE NB OF BUILDINGS CONSIDERED: # TO BE COSTUMIZED BY USERS! ***
# ************************************************************************************
```

Nbr_Buildings_entities = 3 # number of buildings to be simulated over the periode

Input data

A set of input data is given hereafter as an example to allow the checking of the program. The input data to be put into the building model are the followings.

Table 15: input data to the BuildingModel3 - example

Input data	Designation	Coding	Unit	Value
i_Theta_e	outdoor temperature around the buildings	xr.DataArray(10 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	°C	(10, 10, 10, …)
i_Theta_sup	supply air temperature to the buildings	xr.DataArray(10 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	°C	(10, 10, 10, …)
i_Theta_op_mean	set point operative temperature of the first building	xr.DataArray(np.array((np.full([Nbr_Buildings_entities, N_TimeStep], np.nan))), dims = ["buildings", "time"], coords = [buildings, time]) i_Theta_op_mean[0, 0:16] = 18 i_Theta_op_mean[0,16:36] = 20 i_Theta_op_mean[0,36:48] = 18 i_Theta_op_mean[0,48:64] = 18 i_Theta_op_mean[0,64:84] = 20 i_Theta_op_mean[0,84:96] = 18	°C	(18, 18, … , 18, 20, 20, …, 20, 18, 18,…, 18, …, 18, 20, 20, … 20, 18, 18, …, 18)
	set point operative temperature of the second building	i_Theta_op_mean[1,:] = i_Theta_op_mean[0,:] + 2	°C	(20, 20, … , 20, 22, 22, …, 22, 20, 20,…, 20, …, 20, 22, 22, … 22, 20, 20, …, 20)
	set point operative temperature of the third building	i_Theta_op_mean[2,:] = i_Theta_op_mean[0,:] - 2	°C	(16, 16, … , 16, 18, 18, …, 18, 16, 16,…, 16 …, 16, 18, 18, … 18, 16, 16, …, 16)
i_Phi_sol	solar radiation profile received through the building's effective collecting area	xr.DataArray(100 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W	(100, 100, 100, …)
i_Phi_int	internal heat gain dissipated by occupants and all appliances in the building	xr.DataArray(5 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time]	W	(5, 5, 5, …)

In the ISO 13790:2008 [36] standard, the **operative temperature** was defined as a weighted average of the air and mean radiant temperatures, weighted by the internal surface convective (3/10) and radiative coefficients (7/10). This temperature is assumed to be more representative of the human feelings, i.e. his comfort perception, than air temperature.

Table 15 shows input data for only one building, except in the case of input data where the name specifies otherwise.

Input data could be time series from various meteorological web servers, e.g. meteonorm. These data would be computed to feed the building model: i_Theta_sup, i_Phi_sol.

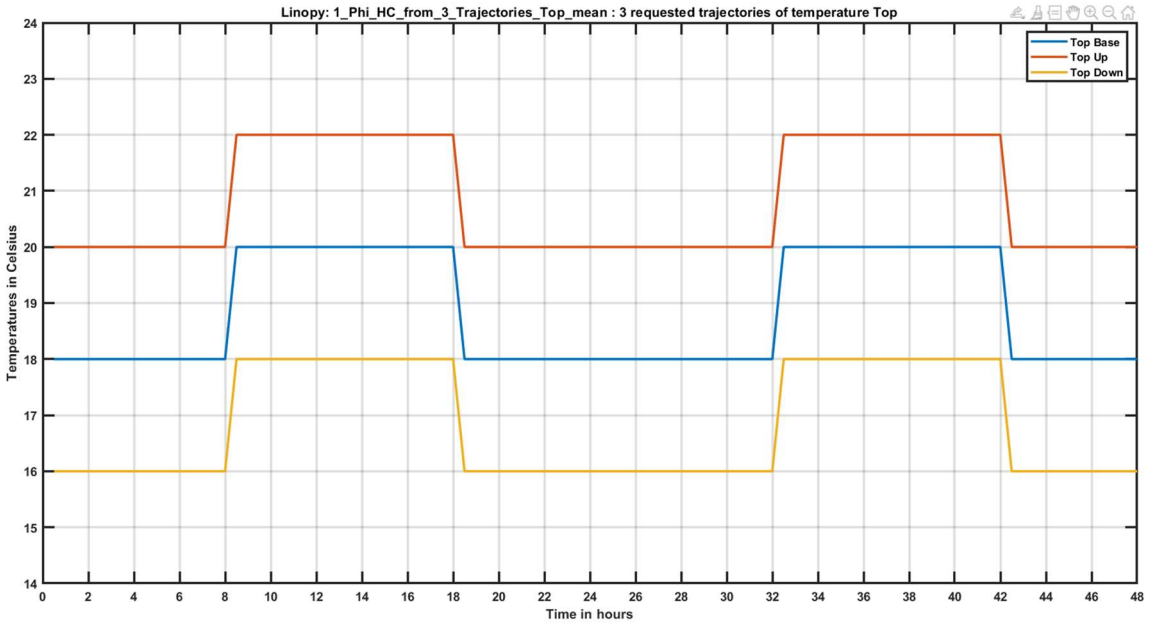


Figure 28 : Set point temperature in the three simulated buildings over the time horizon considered.

The set point temperatures for the three different buildings along the time horizon of 2 days are shown in the Figure 28. Three different set point operative mean temperature time series for as many buildings.

Parameters linked to Building Model

Table 16 : Parameters set up for the BuildingModel3 - example

Parameter	Designation	Coding	SI Unit	Value per building
BM_p_Af	conditioned floor of the building	xr.DataArray(100 * np.ones((Nbr_Buildings_entities)), dims = ["buildings"], coords = [buildings])	m²	100
BM_p_Atot	total area of all surfaces facing the building	xr.DataArray(450 * np.ones((Nbr_Buildings_entities)), dims = ["buildings"], coords = [buildings])	m²	450
BM_p_Am	effective mass area of the building	xr.DataArray(250 * np.ones((Nbr_Buildings_entities)), dims = ["buildings"], coords = [buildings])	m²	250
BM_p_cm	heat capacity of the effective mass of the building	xr.DataArray(165000 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	J/(m².K)	(165e3, 165e3, 165e3, ...)

BM_p_his	convective heat transfer coefficient of the building	xr.DataArray(3.45 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W/(m².K)	(3.45, 3.45, 3.45, …)
BM_p_hms	conductive heat transfer coefficient throughout the buildings effective mass area	xr.DataArray(9.10 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W/(m².K)	(9.10, 9.10, 9.10, …)
BM_p_H_em	heat transmittance between outdoor air at the temperature, Te, and indoor air volume at the temperature, Tair	xr.DataArray(90 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W/K	(90, 90, 90 …)
BM_p_H_ve	heat transmittance between supply air at the temperature, Tsup, and indoor air volume at the temperature, Tair	xr.DataArray(100 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W/K	(100, 100, 100, …)
BM_p_H_w	heat transmittance between outdoor air at the temperature, Te, and star node surface at the temperature, Ts	xr.DataArray(108 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	W/K	(108, 108, 108, …)
BM_p_C1_ ConvRadia	coefficient of convective part, for calculating the indoor air temperature, Tair, depending on the emitters	xr.DataArray(0.5 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	-	(0.5, 0.5, 0.5, …)
BM_p_C1_ Theta_op	coefficient of convective part, for calculating the operative air temperature, Top, from the indoor air temperature, Tair	xr.DataArray(0.3 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings", "time"], coords = [buildings, time])	-	(0.3, 0.3, 0.3, …)
BM_p_ Theta_m_ini	initial temperature of the building mass (walls and floors)	xr.DataArray(18 * np.ones((Nbr_Buildings_entities, N_TimeStep)), dims = ["buildings"], coords = [buildings])	°C	18

Most of the parameters accept variations over the time horizon. Time horizon is considered here for all the parameters except for the dimensional parameters as areas and for the initial temperature of the effective building mass. As we cannot rule out the idea that the initial temperature can be recalled over the time horizon each time a new optimization is requested, this parameter must be adapted by the user with the latest calculated value.

BM_p_H_em, BM_p_H_ve, BM_p_H_w parameters calculation are not described herein. It would be useful to refer to the various standards currently applicable, or failing that the ones deprecated.

The variable boundaries need also to be defined at this stage.

The following variable bounds have been left free. Thus, the lower and upper limits are set to -inf and +inf respectively:

- BM_p_Phi_HC_lb/ub: heat power supply to the conditioned space of the building, in W
- BM_p_Theta_air_lb/ub: indoor air temperature, in Degree Celsius
- BM_p_Theta_air_mean_lb/ub: indoor air mean temperature, in Degree Celsius
- BM_p_Theta_op_lb/ub: operative temperature inside the building, in Degree Celsius

- BM_p_Theta_op_mean_lb/ub: operative mean temperature inside the building, in Degree Celsius

The further variable bounds are fixed as follows for the example:

- BM_p_Theta_e_lb/ub = i_Theta_e: outside ambient air temperature, in Degree Celsius
- BM_p_Theta_sup_lb/ub = i_Theta_sup: supply air temperature, in Degree Celsius
- BM_p_Phi_int_lb/ub = i_Phi_int: internal heat gain dissipated into the building, in W
- BM_p_Phi_sol_lb/ub = i_Phi_sol: solar power received by the building through the glazed area, in W

Results

The following figures represent the solution of the simulation with the set of parameters described previously. They reveal the curves of heating power required to ensure the set point operative mean temperatures.

The results were reached in 0.02 seconds:

- Running HiGHS 1.5.3 [date: 2023-05-16, git hash: 594fa5a9d]
- Copyright (c) 2023 HiGHS under MIT licence terms
- Presolving model
- 2874 rows, 2874 cols, 7470 nonzeros
- 0 rows, 0 cols, 0 nonzeros
- Presolve : Reductions: rows 0(-3168); columns 0(-4609); elements 0(-11517) - Reduced to empty
- Solving the original LP from the solution after postsolve
- Model status : Optimal
- Objective value : 0.0000000000e+00
- HiGHS run time : 0.01

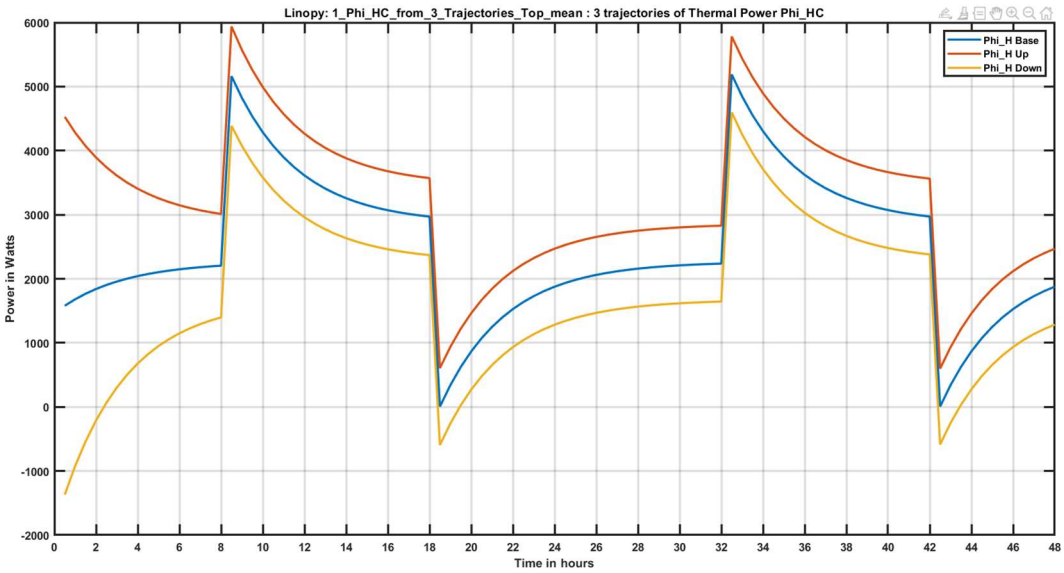


Figure 29 : Heat power requirements to ensure the set point operative temperatures.

Considering the performance coefficient of the heating system (COP = 3) selected for this example, it induces the followings power needs over the time horizon.

The electrical energy consumed to supply the thermal energy using any heat pump is not included into the BuildingModel3 provided by CEA but is simply calculated from the following formula:

$$ElectricalPower = \frac{BM\_v\_Phi\_HC}{COP}$$

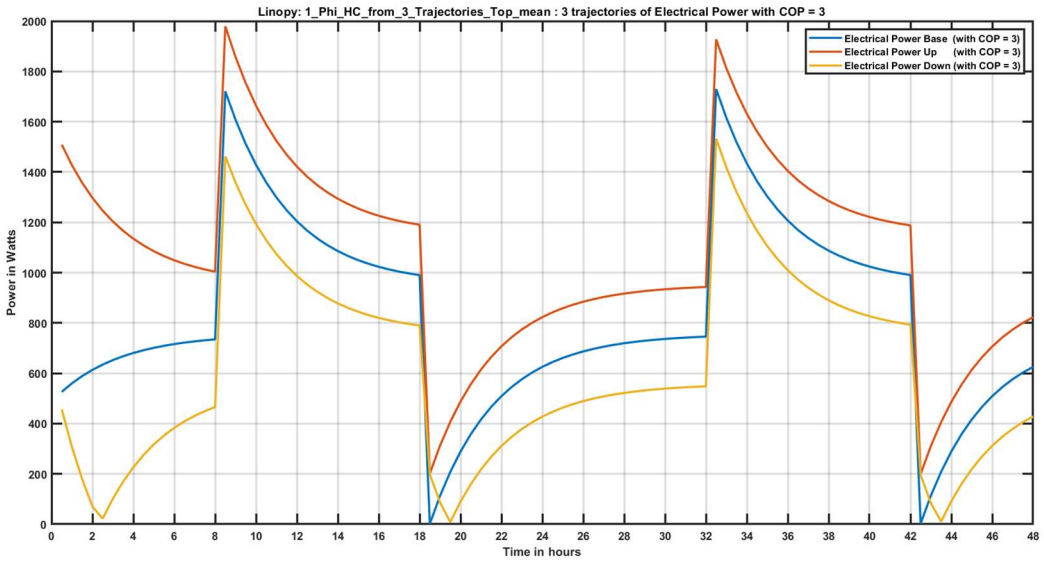


Figure 30 : Electrical power consumption to ensure the set point operative temperatures.

Development Timeline

The current state of work is as follows: CEA has developed, tested and already sent the BuildingModel3 to the EMT developer. CEA stands ready to provide support in getting the model up and running by the end of the year 2024. Furthermore, in the context of HAROPA use-case, CEA will gather and create input data to feed in the BuildingModel3: solar radiations through selected buildings, occupancy, heat internal gain, by the end of the year 2024.

Access to the building flexibility model implementation

The *BuildingModel3* Python model and the installation instructions can be accessed through a link that can be provided upon request.

4.2.2 Flexibility of Terminal Assets

To effectively leverage the load flexibility potential of a port, it is imperative to integrate the operation of its assets with the energy management tool (EMT). This relationship ensures that the port's assets can dynamically respond to demand response (DR) signals, thereby optimising energy usage while maintaining operational efficiency and security.

There are various approaches to incorporating DR into port management systems. In the *centralized* approach, all relevant optimisation problems are formulated into a single comprehensive model. While this approach theoretically offers the most efficient solution, it often faces practical challenges due to the high complexity of the problem and the need for many assumptions.

Conversely, the *distributed* approach splits the optimisation problems into separate components, each addressed individually. While this simplifies the problem-solving process, it may lead to suboptimal solutions as not all constraints are fully considered.

Considering the strengths and limitations of these approaches, a hybrid solution was chosen within the project. This approach involves a sequential process:

Scheduling of Port Assets: Initially, the scheduling of the port's assets is determined without considering demand response requirements. This ensures that the fundamental operational logistics of the port are prioritized and established.

Computing Flexibility Potential: Once the initial schedule is set, the flexibility potential of the port's load is computed. This calculation assumes that rescheduling activities will not significantly disrupt the established logistics of the port.

Integration with EMT: Finally, the computed flexibilities are integrated with the EMT. This integration allows the EMT to optimise the port's operation by leveraging the identified flexibilities, thereby achieving the most cost-effective solution.

After reviewing all port operations, four types of loads were defined as flexible: OPS, yard cranes, reefer containers, and e-vehicles. These loads could be rescheduled without affecting the port's logistics. The following sub-sections will provide a short description of these types of loads, flexibility parameters provided by the tool, and required inputs.

This section of the deliverable focuses on modelling and developing a special tool to compute these flexibilities without disrupting the port's logistical operations.

Description of the Tool

The developed tool is a Python-based code that efficiently processes logistics data and provides load flexibility parameters to the EMT. The tool enables a seamless integration into existing workflows. A user interface or any other visualization of the processes are not provided by the tool. Upon processing the input logistics data, the tool generates output in JSON format, providing a set of calculated load flexibility parameters for all nodes within the port network. The generated file serves as an input for the EMT, enabling it to make decisions regarding energy optimization and demand response strategies. The input data accepts csv format. The Python code comprises a modular structure consisting of nine files, organized around object-oriented principles. Port's assets are represented in 6 files. These files contain the class of the corresponding asset.

Structure of Load Flexibility Tool

The general structure of the tool is presented in Figure 31. A more detailed description is presented below.

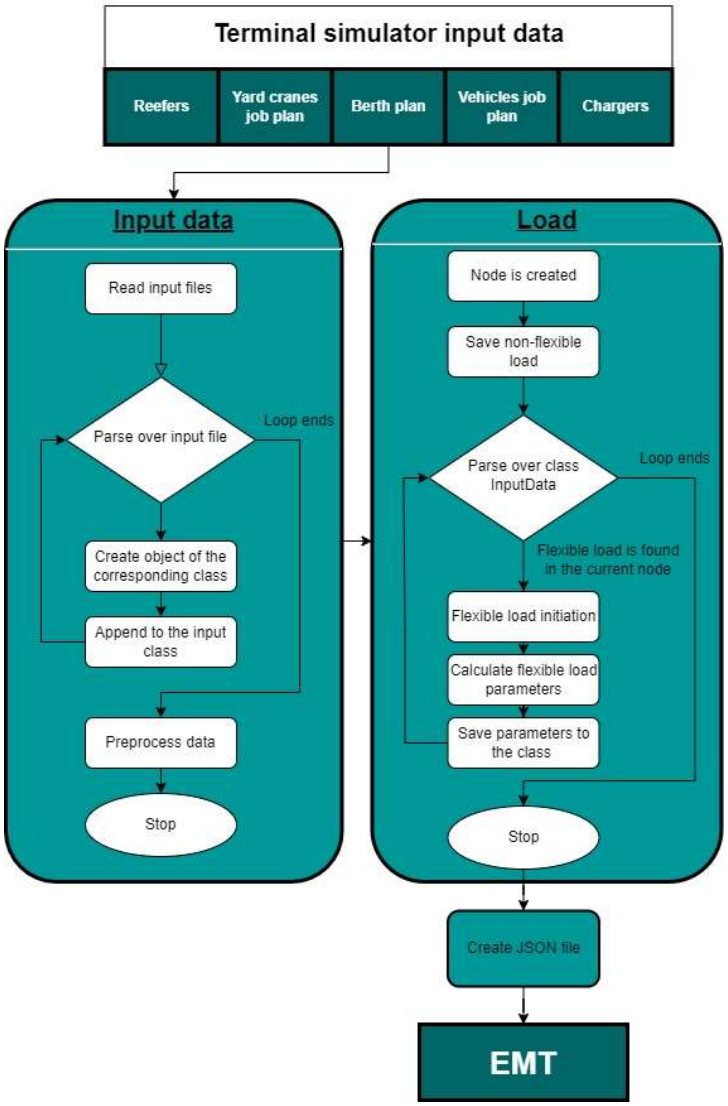


Figure 31 : Architecture of the flexibility modelling tool.

Computer configuration:

Table 17 – Software and hardware requirements

Python environment	
spyder	IDE 5.4.3
Python	3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)]
PC features	
Processor	12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz
Installed RAM	16.0 GB (15.6 GB usable)
System type	64-bit operating system, x64-based processor

Class Definitions for Port Assets

a. "ops.py": This class includes information about onshore power supply (OPS) systems installed in the port. The properties of the class correspond to the parameters of the OPS system. These parameters must be presented in the input file.

Table 18 - Properties of the class OPS (input data)

Input data	Designation	Type of data	Unit
name	Name of the	String	-
max_OPS_power	Maximum power that could be provided by the OPS	Float	kW
berth	Name of the berth where the OPS is located	String	-
bus_indx	Number of the node that supplies the OPS	Integer	-

b. "reefer.py": Represents the refrigerated containers within the port. The class defines properties of a reefer container that must be obtained from the input file.

Table 19 - – Properties of the class Reefer (input data)

Input data	Designation	Type of data	Unit
name	Name of the reefer container	String	-
bus_indx	Number of the node that supplies the reefer container	Integer	-
content_mass	Total mass of a reefer content	Float	kg

heat_capacity	Heat capacity of the reefer container	Float	J/k
area	Total surface area of the reefer container	Float	m^2
heat_transition_coeeficient	Heat transition coefficient of a reefer container	Float	W/(m^2K)
initial_temperature	Initial internal temperature of the reefer container	Float	°C
upper_bound	The maximum allowed internal temperature of the reefer container	Float	°C
lower_bound	The minimum allowed internal temperature of the reefer container	Float	°C
max_power	Maximum power of the supplier	Float	kW
min_power	Minimum power of the supplier	Float	kW

c. "yardcranes.py": Represents the yard cranes within the container terminal. The properties of the class correspond to the parameters of the yard crane and must be presented in the input file.

Table 20 – Properties of the class YardCrane (input data)

Input data	Designation	Type of data	Unit
name	Name of the yard crane	String	-
bus_indx	Number of the node that supplies the yard crane	Integer	-
Pmax	Maximum power of the yard crane	Float	kW
job_plan	List of all jobs that the yard crane performs within the interval of calculation	List	-

Each job performed by the yard crane is described by the parameters of class "YCJob" that exists in the file "yardcranes.py". The parameters are also extracted from the input file.

Table 21 – Properties of the class YCJob (input data)

Input data	Designation	Type of data	Unit
type	Type of yard crane job. There are three types of job: "Loading", "Unloading", and "Rearrangement"	String	-
energy	Energy that is needed to finish the job	Float	kWh

start	Start time of the job	datetime	-
duration	The duration of the job	datetime	-

d. "vehicle.py": Focuses on the electric vehicles utilised within the port, covering attributes necessary for their efficient utilisation and charging management.

Table 22 – Properties of the class Vehicle (input data)

Input data	Designation	Type of data	Unit
name	Name of the vehicle	String	-
capacity	Capacity the vehicles' battery	Float	kwh
charging_efficiency	Charging efficiency of the vehicles' battery	Float	pu
discharging_efficiency	Discharging efficiency of the vehicles' battery	Float	pu
max_charging	Maximum charging power of the vehicles' battery	Float	kW
max_discharging	Maximum discharging power of the vehicles' battery	Float	kW
max_SOC	Maximum allowed state of charge of the vehicles' battery	Float	pu
min_SOC	Minimum allowed state of charge of the vehicles' battery	Float	pu
job_plan	The list of all jobs performed by the vehicle within the period of calculation	List	-

Each job performed by the vehicle is described by the parameters of class "VehicleJob" in the file "vehicle.py". The parameters are also extracted from the input file.

Table 23 – Properties of the class VehicleJob (input data)

Input data	Designation	Type of data	Unit
type	Type of job. There are two types of job: "Charging" and "Operation"	String	-
start	Time when the vehicle starts performing the job.	Datetime	-
duration	Duration of the job	Datetime	-

initial_SOC	State of charge of the vehicle when it starts the job.	Float	pu
final_SOC	State of charge of the vehicle when it finishes the job	Float	pu
location	Indicates the location of the job. If type of the job is "Charging", this parameter must indicate a particular charging station.	String	-

e. "chargers.py": Corresponds to the charging infrastructure for electric vehicles. This class provides information about all port charging stations.

Table 24 – Properties of the class Charger (input data

Input data	Designation	Type of data	Unit
name	Name of the charger	String	-
bus_indx	Number of the node that supplies the charger	Integer	-
max_power	Maximum power that could be provided by the charger	Float	kW
min_power	Minimum power of the charger	Float	kW

f. "ship.py": Represents the vessels visiting the port. The set of all ships that arrive at the port within the simulation period is a berth plan that must be provided in the input file.

Table 25 – Properties of the class Ship (input data)

Input data	Designation	Type of data	Unit
name	Name of the ship	String	-
allocated_berth	The name of berth that was allocated for the ship	String	-
arrival	The time when the ship is arrived at allocated berth	Datetime	-
min_ops_energy	Minimum energy that must be consumed by the ship when it is at berth	Float	kWh
consumption	Hourly power consumption of the ship. The dictionary includes time when the ship is at berth.	Dictionary	kW

Input Data Handling

The file "inputdata.py" has a class "InputData." The input data class consolidates functionalities related to parsing, processing, and storing input data for various port assets. It encapsulates methods for reading data from external sources and pre-processing and organizing the inputs into appropriate data structures for further use by other classes.

Output Data Representation

The "Load" class in file "load.py" serves as a container for output data, particularly focusing on information related to each bus or operational load within the port. Each object of the class properties is indicated in Table 26.

Table 26 – Properties of the class Load (output data)

Output data	Designation	Type of data	Unit
bus_indx	Bus index of the node in electrical network	Integer	-
number_of_interval	Number of intervals within the simulation period (typical value is 24)	Integer	-
timestep	Timestep of the simulation	Datetime	-
non_flexible_load	Scheduled consumption of non-flexible loads	Dictionary	kWh
flexible_load	List of all non-flexible load objects that are connected to the current node	List	-

For non-flexible loads, a baseload profile and a load type are indicated. The load type could be the following: "OPS", "YC", "EV", and "Reefer". Depending on the load type, the output parameters may vary. The output parameters for all types of loads are presented in Table 27 - Table 30. For reefers, the output parameters are those given in the equation below that describes the relationship between the internal temperature of reefers and the cooling power:

$$T_{i,t+1} = C_1 \cdot T_{i,t} - P_{i,t}^{cool} \cdot C_2 + C_3 \quad (5)$$

where $T_{i,t}$ is an internal temperature of a reefer container;

$P_{i,t}^{cool}$ is a reefer's power consumption;

i is an index of reefers;

t is an index of time step;

C_1, C_2, C_3 are coefficients that defined by reefer's parameters and ambient temperature.

Table 27 – Output parameters of OPS systems

Output data	Designation	Type of data	Unit
upward_flex	Difference between maximum power that could be provided by the OPS and	Dictionary	kW

	scheduled OPS power. The dictionary includes time when the ship is at berth.		
downward_flex	Difference between scheduled OPS power and minimum power that could be provided by the OPS. The dictionary includes time when the ship is at berth.	Dictionary	kW
minimum_energy	Minimum energy that must be consumed by the ship when it is at berth	Float	kWh

Table 28 – Output parameters of yard cranes

Output data	Designation	Type of data	Unit
upward_flex	Upward flexibility of the yard crane	Dictionary	kW
downward_flex	Downward flexibility of the yard crane	Dictionary	kW
shift	This parameters show a time range within which the operation could be rescheduled	Dictionary	-

Table 29 – Output parameters of reefers

Output data	Designation	Type of data	Unit
initial_temperature	Initial internal temperature of the reefer container	Float	°C
max_temperature	The maximum allowed internal temperature of the reefer container	Float	°C
min_ temperature	The minimum allowed internal temperature of the reefer container	Float	°C
max_power	Maximum power of the supplier	Float	kW
min_power	Minimum power of the supplier	Float	kW
c1	Coefficient from equation 1	Float	-
c2	Coefficient from equation 1	Float	kW/°C
c3	Coefficient from equation 1. This coefficient varies over time depending on the ambient temperature. The dictionary includes time steps within the period of calculation.	Dictionary	°C

Table 30 – Output parameters of e-vehicles

Output data	Designation	Type of data	Unit
upward_flex	Difference between maximum power that could be provided by the	Dictionary	kW

	charging station and scheduled power. The dictionary includes time when the vehicle is at charging station.		
downward_flex	Difference between scheduled power of the charging station and minimum power that could be provided by the charging station. The dictionary includes time when the vehicle is at charging station.	Dictionary	kW
initial_SOC	State of charge of the vehicle when it arrives at the charging station	Float	pu
final_SOC	State of charge of the vehicle when it leaves the charging station	Float	pu
charging_efficiency	Charging efficiency of the vehicles' battery	Float	pu
discharging_efficiency	Discharging efficiency of the vehicles' battery	Float	pu
max_SOC	Maximum allowed state of charge of the vehicles' battery	Float	pu
min_SOC	Minimum allowed state of charge of the vehicles' battery	Float	pu

Main File Execution

The main file serves as the entry point for the program execution. It enables the interaction between different classes, calling relevant functions to initiate data processing, perform computations, and generate output as required. This file parses over all nodes in an electrical network and adds flexibility parameters to each of them.

Testing the Model

The model was tested with the typical parameters. As an example, a calculation for an OPS system Is presented in this section for one ship.

Input Data

The input data of an OPS system is presented in Table 31. An example berth plan for one ship is shown in Table 32.

Table 31 – OPS parameters used in the example.

Input data	Value	Unit
name	OPS 1	-
max_OPS_power	20	kW
berth	Berth 1	-
bus_indx	1	-

Table 32 – Berth plan for one ship

Input data	Value		Unit
name	Ship 1		-
allocated_berth	Berth 1		-
arrival	2023-01-01 02:00		-
min_ops_energy	36		kWh
consumption	2023-01-01 02:00	12	kW
	2023-01-01 03:00	12	
	2023-01-01 04:00	12	
	2023-01-01 05:00	12	
	2023-01-01 06:00	12	

Output Data

Using the input data presented in Table 31 and Table 32, the flexibility tool was run to produce the output parameters for OPS's flexibility presented in Table 33.

Table 33 – OPS flexibility parameters

Output data	Value		Unit
upward_flex	2023-01-01 02:00	4.8	kW
	2023-01-01 03:00	4.8	
	2023-01-01 04:00	4.8	
	2023-01-01 05:00	4.8	
	2023-01-01 06:00	4.8	
downward_flex	2023-01-01 02:00	7.2	kW
	2023-01-01 03:00	7.2	
	2023-01-01 04:00	7.2	
	2023-01-01 05:00	7.2	
	2023-01-01 06:00	7.2	
minimum_energy	36		kWh

Development Timeline

In the current implementation, the flexibility modelling tool provides flexibility parameters according to the specified data structure but will be further developed to accommodate additional inputs as they become available. So far, the tool has been tested with a small set of sample data. Additional tests will be performed using the outputs of the Terminal Simulator Tool. The next steps also include the integration with the EMT tool. Table 34 shows the expected timeline of the asset flexibility model development.

Table 34 - Expected timeline of model development

Task	2024		2025	
	S1	S2	S3	S4
Model validation				

Establishing assumptions				
Testing the tool with terminal simulator data				
Testing within the EMT				
Implementation of adjustments				
Integration with MAGPIE digital tools				
Integration with the EMT				
Implementation of adjustments				

Access to the asset flexibility model implementation
The terminal assets' flexibility model and the installation instructions can be downloaded via a link that can be provided upon request. The final version of the model will be available open-source via a public repository, such as GitHub.

4.3 Renewable energy sizing and forecasting tools

This section describes the implementation of the back-end models that provide the sizing of renewable resources and the forecast of renewable electricity for the Energy Matching Tool (Task 4.5). The models are an integral part of a digital framework that leverages electricity generation from local renewable sources (namely offshore wind and solar photovoltaic) to supply the forecasted power demand within the port.

The focus on offshore, rather than onshore, wind energy is aligned with the expected contribution of ports as logistic hubs for the installation of offshore wind farms, as well as their role as integrators of the generated power in the mainland power grid and the prospect of on-site production of hydrogen for energy storage. Figure 32 shows the integration of the offshore wind and solar photovoltaic prospecting and power forecasting tools within the Energy Matching Tool. The maximum installed capacity of each technology estimated by the prospecting tools is used by the respective power forecasting tool (as described in detail in Deliverable D4.4) to generate short-term power generation forecasts, which are then employed by the Energy Matching Tool for an optimized energy management within the port.

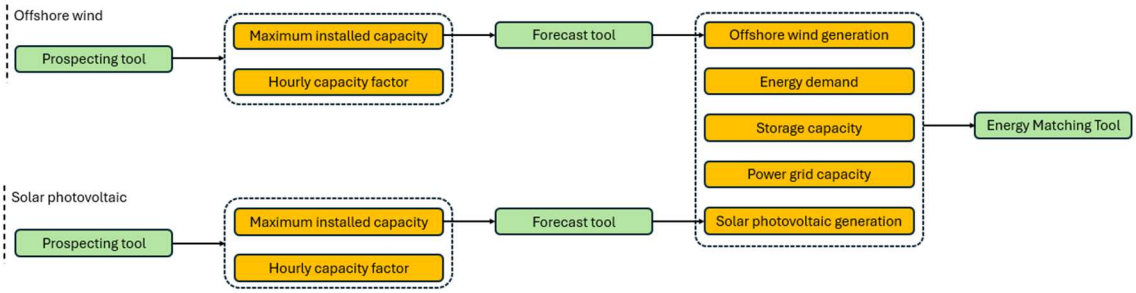


Figure 32 - Integration of offshore wind and solar PV tools within the Energy Matching Tool.

This section provides the technical knowledge to operate the offshore wind and solar photovoltaic prospecting and power forecast tools. For each tool, the architectural solution of the implemented model is presented, along with a detailed guide on model customization and a description of a sample case study whose results are also included in the repository. The expected timeline of model development is also provided for each tool.

4.3.1 Offshore wind prospection and forecast

Offshore wind farm prospection tool

The offshore wind farm prospection tool employs state-of-the-art weather data and modelling techniques to estimate the geographical potential within a user-defined area and the associated wind farm productivity considering the local wind resource. The modelling process starts with the downscaling of ERA5 reanalysis data to a representative point within the potential wind farm area, by performing both horizontal and vertical interpolation of wind and climate data to the desired coordinates and wind turbine hub height. Air density is computed and later used for wind turbine power curve correction. In parallel, a siting algorithm goes over the potential wind farm area and tightly packs wind turbines considering pre-defined minimum inter-turbine distance criteria. Slight variations of the algorithm are run to generate different layouts. The wind farm layout with the most wind turbines is simulated with a wake model, and other power losses are computed to generate the wind farm power matrix. The power matrix is then subject to additional adjustments to better account for the temporal and spatial variability of the wind resource within the wind farm area. The time-series of wind speed/direction is combined with the wind farm power matrix to generate an hourly time-series of wind farm power output.

Model implementation

The model pipeline is divided into different blocks categorized into five main types, as shown in Figure 33:

- *input* blocks: required data to feed the model.
- *calculation* blocks: main structure of the pipeline.
- *structural* blocks: ancillary data structures that facilitate data flow.
- *output* blocks: results of the simulation.
- *ancillary* blocks: pre/postprocessing of *input*/*output* blocks.

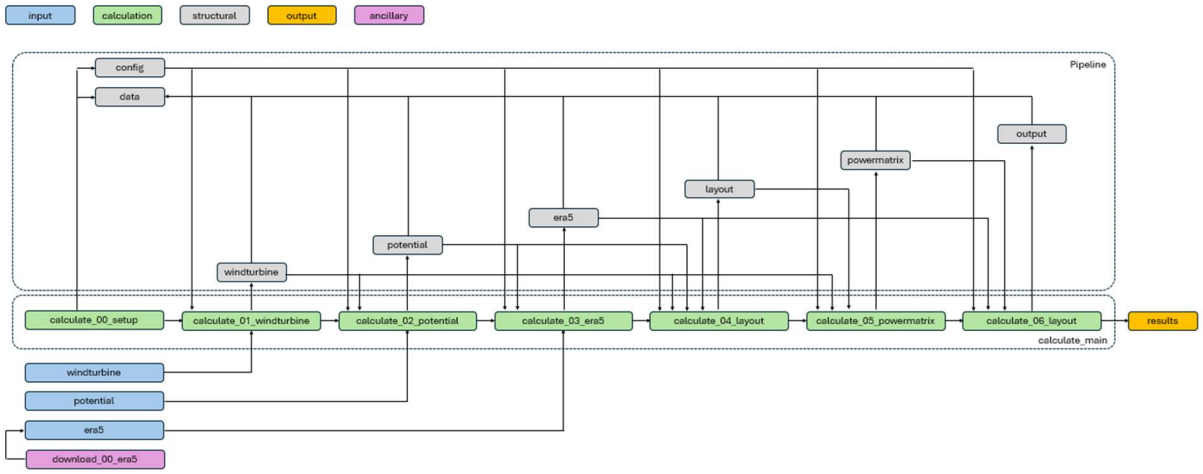


Figure 33 - Model pipeline and data flow.

A general description of a typical use-case of the model is as follows:

1. The user gathers all the necessary *input* blocks (***windturbine***, ***potential***, ***era5***) before running the model itself. The *ancillary* block (***download_00_era5***) aids in preparing/preprocessing some of the *input* data (***era5***) and must be configured according to the desired parameterization, whereas the remaining *input* blocks (***windturbine***, ***potential***) are filled in manually.
2. The user configures the simulation by filling in all the required parameters in ***calculate_00_setup***. This user-defined parameterization is assigned as a dictionary (structural block ***config***) to a purpose-built ***Pipeline*** class object, which is instanced once the user runs the aggregator *calculation* block ***calculate_main***. The rationale for this implementation solution is that it allows for the model parameterization to be accessed by the different *calculation* blocks without the need for explicit declaration of function inputs nor global variables.
3. As the simulation progresses along the *calculation* blocks, each intermediate output is assigned to the ***Pipeline*** object as a component of the *structural* block ***data***, also coded as a dictionary, making the intermediate outputs easily accessible by *calculation* blocks further downstream for additional calculations.
4. Once the simulation is complete, the relevant intermediate results are aggregated into a final *output* block ***results*** and stored in a user-defined path.

The usage instructions are described later in greater detail.

Input blocks

The *input* blocks form the database required to run the simulation and must be gathered beforehand. The model requires input data of distinct nature which are also obtained from different sources. Each *input* block has its own requirements regarding data types and structure, with which the user must comply to avoid errors when running the simulation. Table 35 describes each *input* block in detail. An example of each *input* block is provided in the sample case-study.

Table 35 - Characterization of input *blocks*

Block	Description	Format	Requirements	Application

windturbine	Technical characteristics of the wind turbine(s).	*csv*	The power and thrust coefficient curves are defined strictly for the operating range of the wind turbine (i.e., the first and last values of wind speed correspond to cut-in and cut-off wind speeds, respectively). All units are in the SI system (rated power in Watt [W], diameter in metres [m], wind speed in metres per second [m/s], power in Watt [W], thrust coefficient is dimensionless [-]). The power curve may be defined directly as capacity factor (dimensionless [-]) – in this case, the rated power must be set to 1.	***calculate_01_windturbine***: uses the wind turbine rated power to normalize its power curve (the model works with capacity factors). ***calculate_02_potential***: uses the wind turbine diameter to apply buffers to the search areas and spatial restrictions. ***calculate_04_layout***: uses the wind turbine diameter to account for inter-turbine distances. ***calculate_05_powermatrix***: uses the wind turbine diameter and power and thrust coefficient curves to run the wind farm wake model and estimate the wind farm power matrix.
potential	Potential locations for the installation of wind turbines (search areas and spatial restrictions).	*kml*	Search areas may be provided as a single file (e.g., single, continuous area) or as multiple files (e.g., multiple, discontinuous areas) – the same is applicable to spatial restrictions. Google Earth is recommended for the generation of this input data by drawing polygons that define both the search areas and spatial restrictions.	***calculate_02_potential***: uses the search areas and spatial restrictions to compute the available wind farm area. ***calculate_03_era5***: downscales ERA5 data to a representative point within the available wind farm area. ***calculate_04_layout***: deploys siting within available wind farm area to generate a layout.
era5	Data on wind speed and direction, and other climate parameters (e.g., temperature).	*netcdf*	The ancillary script *download_00_era5* is recommended for data download/processing, as the data must be provided in a very specific format for compatibility with downstream blocks in the pipeline.	***calculate_03_era5***: downscales the input reanalysis data to a time-series representative of the wind farm area (horizontal and vertical interpolation). ***calculate_04_layout***: determines predominant wind direction for optimal wind turbine alignment in the layout. ***calculate_06_output***: uses time-series of wind speed and direction to determine

				hourly wind farm capacity factor.

Calculation blocks

The *calculation* blocks integrate the main framework of the model and apply various types of mathematical operations to compute different intermediate outputs. Each block takes a ***Pipeline*** object as input and estimates a relevant output, *i.e.*, produces a tangible result with a clear physical meaning in the simulation context. They are coded as stand-alone scripts and integrated within the aggregator block ***calculate_main***, which ensures a sequential calling of the individual blocks. Table 36 describes each *calculation* block in detail, namely its communication with the *input*, *structural*, and *output* blocks of the model.

Table 36 - Characterization of calculation *blocks*

Block	Description	Input	Structural	Output
calculate_00_setup	Instancing of ***Pipeline*** object to host *structural* blocks (see Structural blocks). Model parameterization (see Config). Ancillary function to write simulation results.	-	***config*** Assigns ***user***, ***potential***, ***era5***, ***layout***, ***powermatrix***	-
calculate_01_windturbine	Reading and preprocessing of wind turbine input data.	***windturbine***	***config*** Reads ***user***, ***windturbine data*** Assigns ***windturbine***	-
calculate_02_potential	Reading and preprocessing of geographical potential input data. Definition of available wind farm area.	***potential***	***config*** Reads ***user***, ***potential data*** Reads ***windturbine*** Assigns ***potential***	-
calculate_03_era5	Reading and downscaling of ERA5 input data (horizontal and vertical interpolation). Calculation of air density.	***era5***	***config*** Reads ***user***, ***era5 data*** Reads ***potential*** Assigns ***era5***	-
calculate_04_layout	Generation of potential wind farm layout.	-	***config*** Reads ***user***, ***layout data*** Reads ***windturbine***, ***potential***, ***era5*** Assigns ***layout***	-

calculate_05_powermatrix	Estimation of wind farm wake losses and power matrix.	-	***config*** Reads ***user***, ***powermatrix*** ***data*** Reads ***windturbine***, ***layout*** Assigns ***powermatrix***	-
calculate_06_output	Calculation of wind farm power output (capacity factor time series).	-	***data*** Reads ***era5***, ***powermatrix*** Assigns ***output***	-
calculate_main	Aggregator script that integrates the main *calculation* blocks.	-	***data*** Reads ***layout***, ***output***	***results***

Structural blocks

The *structural* blocks are dictionary-type variables which store information that is accessed by multiple *calculation* blocks. Assigning the *structural* blocks directly to the ***Pipeline*** object (which is then given as input to the *calculation* blocks) greatly simplifies the number of function arguments that are passed on to each *calculation* block since all the information is stored within the object itself.

Config

The *structural* block ***config*** stores the user-defined parameterization. This information is filled in by the user, prior to running the simulation, in the *calculation* block ***calculate_00_setup*** and assigned to the ***Pipeline*** object by the same block. During the simulation, it is accessed by most other *calculation* blocks to define the respective inner workings in terms of mathematical procedures. Figure 34 shows the structure and data types of this block and Table 37 describes each of its components in detail.

```
config_user = {'years': {'start': int,
                         'end': int},
               'windturbine': {'name': str,
                               'height': float},
               'potential': {'searchareas': [str],
                             'restrictions': [str]},
               'paths': {'ERA5': str,
                         'windturbines': str,
                         'potential': str,
                         'results': str}}

config_potential = {'buffer': {'searcharea': bool,
                               'restrictions': bool}}

config_era5 = {'airdensity': {'horizontal': str,
                              'limitoffshore': bool,
                              'vertical': str},
               'wind': {'horizontal': str,
                        'limitoffshore': bool,
                        'vertical': {'speed': str,
                                     'direction': str}}}

config_layout = {'direction': str,
                 'resolution':  float,
                 'buffer': float,
                 'distances': {'streamwise': {'m': float,
                                              'RD': float},
                               'spanwise': {'m': float,
                                            'RD': float}}}

config_powermatrix = {'windspeed': float,
                      'winddirection': float,
                      'wakesmoothing': {'apply': bool,
                                        'sigma': float},
                      'others': float,
                      'convolution': {'base': float,
                                      'scale': float}}
```

Figure 34 – Structure and data types of structural **config** *block.*

Table 37 - Characterization of structural **config** *block*

Component	Parameters	Application
user	***years***: *dict* • ***start***: first year of the simulation. • ***end***: last year of the simulation. ***windturbine***: *dict* • ***name***: name of *csv* file with wind turbine data. • ***height***: wind turbine hub height- ***potential***: *dict* • ***searchareas***: name(s) of *kml* file(s) defining search area(s).	Case-study parameterization (see 0). ***calculate_01_windturbine*** ***calculate_02_potential*** ***calculate_03_era5*** ***calculate_05_powermatrix***

	• ***restrictions***: name(s) of *kml* file(s) defining spatial restriction(s). ***paths***: *dict* • ***ERA5***: relative path to directory with ERA5 data (*input* block ***era5***). • ***windturbines***: relative path to directory with wind turbine data (*input* block ***windturbine***). • ***potential***: relative path to directory with data search area(s) and spatial restriction(s) data (*input* block ***potential***). • ***results***: relative path to directory in which to store simulation results (*output* block ***results***).	
potential	***searcharea***: if *True*, a buffer of 1 rotor diameter is subtracted from the combined search area(s) to avoid wind turbine flyover beyond search area bounds. ***restrictions***: If *True*, a buffer of 1 rotor diameter is added to the combined spatial restriction(s) to avoid wind turbine flyover into restriction area bounds.	Handling of search areas and spatial restrictions (advanced parameterization – see 0). ***calculate_02_potential***
era5	***airdensity***: *dict* • ***horizontal***: horizontal interpolation of climate parameters (*'inversedistance'* or *'nearest'*). • ***limitoffshore***: if *True*, horizontal interpolation is limited to offshore grid points. • ***vertical***: vertical interpolation of climate parameters (*'linear'* or *'logarithm'*). ***wind***: *dict* • ***horizontal***: horizontal interpolation of wind data (*'inversedistance'* or *'nearest'*). • ***limitoffshore***: if *True*, horizontal interpolation is limited to offshore grid points. • ***vertical***: *dict* ○ ***speed***: vertical interpolation of wind speed (*'linear'* or *'logarithm'*) ○ ***direction***: vertical interpolation of wind direction (*'linear'* or *'logarithm'*)	Downscaling of ERA5 data (advanced parameterization – see 0). ***calculate_03_era5***
layout	***direction***: rationale for determining the predominant wind direction (*'energy'*, *'frequency'*, or *'mean'*)	Turbine siting algorithm (advanced parameterization – see 0).

	resolution: resolution of availability matrix in wind turbine rotor diameters. ***buffer***: buffer applied to final available area to increase flexibility of siting algorithm (in rotor diameters). ***distances***: *dict* • ***streamwise***: *dict* ○ ***m***: minimum streamwise inter-turbine distance in metres. ○ ***RD***: minimum streamwise inter-turbine distance in rotor diameters. • ***spanwise***: *dict* ○ ***m***: minimum spanwise inter-turbine distance in metres. ○ **RD**: minimum spanwise inter-turbine distance in rotor diameters.	***calculate_04_layout***
powermatrix	***windspeed***: wind speed step for wind farm wake simulation. ***winddirection***: wind direction step for wind farm wake simulation. ***wakesmoothing***: *dict* • ***apply***: if *True*, a Gaussian kernel is applied to the wind farm wake losses (over wind direction). • ***sigma***: width of the Gaussian kernel (only relevant if *apply* is *True*). ***others***: fixed factor to account for other wind farm power losses (e.g., electrical). ***convolution***: *dict* • ***base***: fixed term of Gaussian kernel applied to wind farm power matrix (over wind speed). • ***scale***: wind-speed dependent term of Gaussian kernel applied to wind farm power matrix (over wind speed).	Wake modelling and calculation of wind farm power matrix (advanced parameterization – see 0). ***calculate_05_powermatrix***

Data

The *structural* block ***data*** stores the intermediate model outputs. This information is assigned to the ***Pipeline*** object by each *calculation* block. During the simulation, it is accessed by downstream *calculation* blocks to perform additional calculations. Figure 35 shows the structure and data types of this block and Table 38 describes each of its components in detail.

```
data = {'windturbine': {'manufacturer': str,
                        'model': str,
                        'ratedpower': float,
                        'diameter': float,
                        'curves': xarray.Dataset},
        'potential': {'area': geopandas.GeoDataFrame,
                      'coordinates': {'latitude': float,
                                      'longitude': float},
                      'crs': str},
        'ERA5': xarray.Dataset,
        'layout': geopandas.GeoDataFrame,
        'powermatrix': xarray.Dataset,
        'output': {'capacity': float,
                   'capacityfactor': xarray.Dataset}}
```

Figure 35 - Structure and data types of structural **data** *block.*

Table 38 - Characterization of structural **data** *block*

Component	Parameters	Source

windturbine	***manufacturer***: wind turbine manufacturer.
	model: wind turbine model.
	ratedpower: wind turbine rated power.
	diameter: wind turbine rotor diameter.
	curves: wind turbine performance data.
	• *coords*:
	  ○ *windspeed*: *float*
	• *data_vars*:
	  ○ *capacityfactor (windspeed): float*
	  ○ *thrustcoefficient (windspeed): float*
		Technical information on the wind turbine used in the simulation.
		calculate_01_windturbine

potential	***area***: available area for the installation of wind turbines (considering both search areas and spatial restrictions). • *Name*: *pandas.Series* of *str* • *Description*: *pandas.Series* of *str* • *geometry*: *geopandas.GeoSeries* of *shapely.Polygon* ***coordinates***: *dict* • ***latitude***: latitude of centroid of available area. • ***longitude***: longitude of centroid of available area. ***crs***: EPSG corresponding to the UTM zone of the available area.	Geographical potential (i.e., available area for the installation of wind turbines). ***calculate_02_potential***
era5	*coords*: • *longitude*: *float* • *latitude*: *float* • *height*: *float*	Site-representative (i.e., downscaled) time-series of wind and climate parameters. ***calculate_03_era5***

	• *time: numpy.datetime64* *data_vars:* • *offshore (longitude, latitude): bool* • *windspeed (longitude, latitude, height, time): float* • *winddirection (longitude, latitude, height, time): float* • *temperature (longitude, latitude, height, time): float* • *humidity (longitude, latitude, height, time): float* • *pressure (longitude, latitude, height, time): float*	
layout	*geometry: geopandas.GeoSeries of shapely.Point*	Wind farm layout (*i.e.*, wind turbine coordinates). ***calculate_04_layout***
powermatrix	*coords:* • *windspeed: float* • *winddirection: float* *data_vars:* • *capacityfactor (windspeed, winddirection): float*	Wind farm capacity factor as a function of wind speed and direction. ***calculate_05_powermatrix***

output	***capacity***: maximum installed capacity in the available area.
	capacityfactor: time-series of wind and climate data and wind farm power output.
	• *coords:*
	  ○ *time: numpy.datetime64*
	• *data_vars:*
	  ○ *windspeed (time): float*
	  ○ *winddirection (time): float*
	  ○ *airdensity (time): float*
	  ○ *capacityfactor (time): float*
		Wind farm power output as a time-series of hourly capacity factor.
		calculate_06_output

oOutput blocks

The *output* blocks are a combination of intermediate model outputs from different *calculation* blocks stored in the *structural* block ***data***, which are aggregated into a single structure once the simulation is complete. Table 39 describes each *output* block in detail. An example of each *output* block is provided in the sample case-study.

Table 39 - Characterization of output *blocks*

Block	Description	Intermediate outputs	Format
results	Simulation results (*i.e.*, maximum installed capacity and hourly time-series of wind farm capacity factor and wind/climate parameters).	***calculate_04_layout*** ***calculate_06_output***	*csv*

Ancillary blocks

The *ancillary* blocks are intended to aid the user in the pre- and postprocessing of *input* and *output* blocks, respectively. On the *input* side, they ensure that the *input* data gathered from whichever source complies with the expected format for the simulation, especially when the data source is not easily accessible (*e.g.*, submit requests through an API) and/or requires significant preprocessing. On the *output* side, they allow the user to transform simulation results into any desired format without the need for re-running the simulation. Overall, they detach the task of handling input/output data from the main simulation, simplifying the core of the model and increasing its flexibility for different data sources and applications. The configuration of these stand-alone blocks is also separated from that of the *calculation* blocks, the user having to parameterize each *ancillary* block individually (within each script itself) rather than in an equivalent *structural* block.

Download_00_era5

The purpose of this *ancillary* block is to facilitate the retrieval of ERA5 reanalysis data from its API by establishing a standard framework for submitting the download requests. Thus, the user is only required to adjust a few parameters to align the downloaded data with the application in hands. Moreover, it performs all the necessary preprocessing of the raw data to meet the requirements of the *input* block ***era5***. Figure 36 shows the structure and data types of this block and Table 40 describes each of its components in detail.

```
config_user = {'coordinates': {'latitude': float,
                               'longitude': float},
               'years': {'start': int,
                         'end': int},
               'path': str}

config_download = {'levels': list(int),
                   'heights': list(float),
                   'a': {'above': list(float),
                         'below': list(float)},
                   'b': {'above': list(float),
                         'below': list(float)},
                   'resolution': float}
```

Figure 36 - Structure and data types of ancillary **download_00_era5** block.

Table 40 - Characterization of the ancillary *block* **download_00_era5**

Component	Parameters	Application
config_user	***coordinates***: *dict* • ***latitude***: latitude of the representative point within the search area for which to download the data (<0 for the Southern hemisphere and >0 for the Northern hemisphere. Ranges between -90 – South pole – and 90 – North pole). • ***longitude***: longitude of the representative point within the search area for which to download the data (<0 for the Western hemisphere and >0 for the Eastern hemisphere. Ranges between -180 – West – and 180 – East).	Case-study parameterization (see 0).

	years: *dict* • ***start***: first year for which to download the data. • ***end***: last year for which to download the data (equal to or greater that the start year for yearly and multi-yearly periods, respectively). ERA5 data is available from 1939 to the present day with a delay of 2-3 months, so the download request must be submitted only for yearly/multi-yearly periods with complete data coverage (check https://apps.ecmwf.int/data-catalogues/era5/?type=an&class=ea&stream=oper&expver=1). ***path***: relative path to directory in which to store downloaded data (*input* block ***era5***).	
config_download	***levels***: model levels from which to download ERA5 data. ERA5 data is available in 137 model levels corresponding to heights ranging from 10 metres to over 80 kilometres above ground level (check https://confluence.ecmwf.int/display/UDOC/L137+model+level+definitions). The height range of the user-specified levels must include the hub height of the user-specified wind turbine to ensure vertical interpolation (rather than extrapolation). *heights*: geometric heights corresponding to the specified model levels (heights and levels must be ordered coherently to ensure proper matching). ***a***: *dict* • *above*: pressure parameter *'a'* for the model levels above the user-specified model levels (must be ordered coherently with user-specified model levels to ensure proper matching – e.g., the first value corresponds to the pressure parameter 'a' for model level above the first user-specified model level). • *below*: pressure parameter *'a'* for the model levels below the user-specified model levels (must be ordered coherently with user-specified model levels to ensure proper matching – e.g., the first value corresponds to the pressure parameter 'a' for model level below the first user-specified model level). ***b***: *dict* • *above*: pressure parameter *'b'* for the model levels above the user-specified model levels (must be ordered coherently with user-specified model levels to ensure proper matching – e.g., the first value corresponds to the pressure parameter 'a' for model level above the first user-specified model level). • *below*: pressure parameter *'b'* for the model levels below the user-specified model levels (must be ordered coherently with user-specified model levels to ensure proper matching – e.g., the first value corresponds to the pressure parameter 'a' for model level below the first user-specified model level). ***resolution***: spatial resolution for data downscaling. A regular latitude/longitude resolution of 0.25 degrees is recommended to approximate the latitude/longitude equivalent of 0.28125 degrees of the spherical harmonics (T639) native ERA5 grid. A coarser resolution reduces precision, whereas a higher resolution oversamples the data without any substantial increase in accuracy (https://confluence.ecmwf.int/display/CKB/ERA5%3A+What+is+the+spatial+reference).	Advanced parameterization – see 0.

Usage instructions

Dependencies

The dependencies listed below may be easily installed by running the following command from the main project folder:
pip install -r requirements.txt

Package	Version
cdsapi	0.6.1
fiona	1.9.5
geopandas	0.14.1
geopy	2.4.0
numpy	1.25.2
pandas	2.1.0
python-dateutil	2.8.2
requests	2.31.0
scikit-learn	1.3.2
scipy	1.11.4
shapely	2.0.2
simplekml	1.3.6
sklearn	0.0.post11
utm	0.7.0
xarray	2023.8.0

Customization

The model is structured for maximum flexibility and customizability across two distinct dimensions: case-study parameterization and advanced parameterization. The configuration of the *calculation* blocks is centred in the *calculation* block ***calculation_00_setup***, whereas each *ancillary* block is configured separately. This rationale simplifies user-interaction by removing hardcoded variables and reducing the number of interaction points with the user and facilitates the definition of case-studies and the tweaking of model parameters (*e.g.*, for model validation).

Case-study parameterization

Case-study parameterization allows the user to define their own case-study by adjusting only the necessary configurations on a case-by-case basis (*i.e.*, without altering the parameters of

the *calculation* blocks). This requires only minimal knowledge on the physical phenomena being modelled and the mathematical framework of the model itself. Table 41 indicates which blocks the user may configure for case-study parameterization.

Table 41 - Blocks subject to case-study parameterization

Block	Description	Component
download_00_era5	The user specifies the spatial bounds and the time horizon for the downloaded data without altering the structure of the API request.	***config_user***
calculate_00_setup	The user defines the case-specific constraints (e.g., wind turbine, spatial availability) without altering the mathematical framework of the model.	***config_user***

Advanced parameterization

Advanced parameterization allows the user to alter the mathematical formulation of the *calculation* modules, effectively adjusting the structure of the model itself. This requires robust knowledge on the physical phenomena being modelled and the mathematical framework of the model itself and is not recommended for inexperienced users as it may cause incompatible data formats and/or generate inaccurate results. Table 42 indicates which blocks the user may configure for advanced parameterization; Figure 37 and Figure 38 show the respective default advanced parameterization.

Table 42 - Blocks subject to advanced parameterization

Block	Description	Component
download_00_era5	The user alters the structure of the API request, leading to data of an entirely different nature (e.g., different variables) being downloaded.	***config_download***
calculate_00_setup	The user alters the structure of the model itself, by adjusting the parameterization of one or more of its *calculation* blocks.	***config_potential*** ***config_era5*** ***config_layout*** ***config_powermatrix***

```
config_download = {'levels': [131, 133, 135],
                   'heights': [169.51, 106.54, 53.92],
                   'a': {'above': [302.476563, 122.101563, 22.835938],
                         'below': [122.101563, 22.835938, 0]},
                   'b': {'above': [0.975078, 0.984542, 0.991984],
                         'below': [0.984542, 0.991984, 0.997630]},
                   'resolution': 0.25}
```

Figure 37 – Default advanced parameterization of ancillary *block* **download_00_era5**.

```
config_potential = {'buffer': {'searcharea': True,
                               'restrictions': True}}

config_era5 = {'airdensity': {'horizontal': 'inversedistance',
                              'limitoffshore': False,
                              'vertical': 'linear'},
               'wind': {'horizontal': 'inversedistance',
                        'limitoffshore': True,
                        'vertical': {'speed': 'logarithm',
                                     'direction': 'linear'}}}

config_layout = {'direction': 'frequency',
                 'resolution':  0.1
                 'buffer': 1,
                 'distances': {'streamwise': {'m': None,
                                              'RD': 10},
                               'spanwise': {'m': None,
                                            'RD': 4}}}

config_powermatrix = {'windspeed': 1,
                      'winddirection': 1,
                      'wakesmoothing': {'apply': True,
                                        'sigma': 5},
                      'others': 0.05,
                      'convolution': {'base': 0.5,
                                      'scale': 0.04}}
```

Figure 38 – Default advanced parameterization of calculation *block* **calculate_00_setup**.

Example case-study

For illustration purposes, an example application of the tool to a generic case-study is included in the *example* directory of the repository. The premises of the case-study are as follows:

- 1-year period (2010).
- IEA 10-MW Reference Wind Turbine with a hub height of 120 metres.
- 4 search areas and 3 different spatial restrictions.

The *input* blocks ***windturbine***, ***potential***, and ***era5*** are available in the corresponding directories (*example/database_windturbines*, *example/database_potential*, and *example/database_era5*, respectively). Nonetheless, the procedure for downloading the ERA5 data necessary for *input* block ***era5*** is also described in this guide. Simulation results are stored in ***results.csv*** in the *example* directory.

1. Configure ***download_00_era5*** (see Figure 39).

```
config_user = {'coordinates': {'latitude': 40.1,
                               'longitude': 6.9},
               'years': {'start': 2010,
                         'end': 2010},
               'path': 'example/database_ERA5'}
```

Figure 39 – Example case-study parameterization of ancillary *block* **download_00_era5**.

2. Configure ***calculate_00_setup*** (see Figure 40).

```
config_user = {'years': {'start': 2010,
                         'end': 2010},
               'windturbine': {'name': 'IEA_10-MW Reference Wind Turbine',
                               'height': 120},
               'potential': {'searchareas': ['searcharea_1',
                                             'searcharea_2',
                                             'searcharea_3',
                                             'searcharea_4'],
                             'restrictions': ['restriction_1',
                                              'restriction_2',
                                              'restriction_3']},
               'paths': {'ERA5': 'example/database_ERA5',
                         'windturbines': 'example/database_windturbines',
                         'potential': 'example/database_potential',
                         'results': 'example/results.csv'}}
```

Figure 40 – Example case-study parameterization of calculation *block* **calculate_00_setup**.

3. Run ***calculate_main***.

4. Analyze ***results.csv***: maximum installed capacity of 420 MW (42 10 MW wind turbines) and mean capacity factor of 0.43.

Access to the offshore wind sizing tool implementation

The offshore wind sizing tool can be downloaded via a link that can be provided upon request. The final version of the model will be available open-source via a public repository, such as GitHub.

Timeline

The tool is already operational, although some of the planned features have not yet been implemented. As such, the next step in the development schedule regards validation of the underlying model, which culminates in the final version of the tool. For integration with the Energy Matching Tool and other Digital Tools within the MAGPIE project, the architecture of the implemented tool may suffer some adjustments. Lastly, the tool will be migrated into a purpose-built graphical interface for stand-alone applications. Table 43 details the timeline in greater detail.

Table 43 - Expected timeline of model development

Task	2024		2025	
	S1	S2	S3	S4
Model validation				
Data gathering				
Definition of methodology				
Extraction of results				
Implementation of adjustments				
Integration with MAGPIE digital tools				

Definition of IT architecture				
Implementation of adjustments				
Development of graphical interface				
Assembly of database				
Definition of IT architecture				
Design of interface				
Migration of model and database				
Testing				

Offshore wind power forecast tool

This offshore wind power forecast tool employs state-of-the-art machine learning techniques to estimate future short-term power generation from an existing/prospective offshore wind farm based on Numerical Weather Predictions (NWP). The modelling process starts with the gathering of historical data on local weather conditions and power generation. This data is pre-processed and subject to feature engineering to eliminate noise and extract as much information as possible. Afterwards, Gradient-Boosting Trees are used to derive an empirical relationship between weather conditions and wind farm power generation. This correlation model may then be applied to short-term weather forecasts to estimate future power generation of the offshore wind farm (not featured in the current version).

Model implementation

The model pipeline is divided into different modules categorized into four main types, as shown in Figure 41:

- *input* blocks: required data to feed the model.
- *calculation* blocks: main structure of the pipeline.
- *structural* blocks: ancillary data structures that facilitate data flow.
- *output* blocks: results of the simulation.

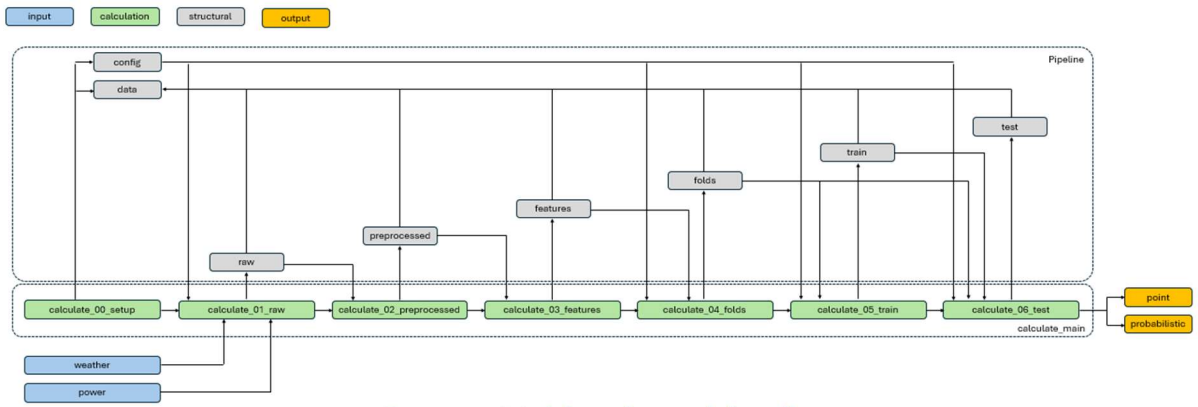


Figure 41 - Model pipeline and data flow.

A general description of a typical use-case of the model is as follows:

1. The user gathers all the necessary *input* blocks (***weather***, ***power***) before running the model itself. These *input* blocks are filled in manually.
2. The user configures the simulation by filling in all the required parameters in ***calculate_00_setup***. This user-defined parameterization is assigned as a dictionary (structural block ***config***) to a purpose-built ***Pipeline*** class object, which is instanced once the user runs the aggregator *calculation* block ***calculate_main***. The rationale for this implementation solution is that it allows for the model parameterization to be accessed by the different *calculation* blocks without the need for explicit declaration of function inputs nor global variables.
3. As the simulation progresses along the *calculation* blocks, each intermediate output is assigned to the ***Pipeline*** object as a component of the *structural* block ***data***, also coded as a dictionary, making the intermediate outputs easily accessible by *calculation* blocks further downstream for additional calculations.
4. Once the simulation is complete, the relevant intermediate results are aggregated into a final *output* block ***results*** and stored in a user-defined path.

The usage instructions are described later in greater detail.

Input blocks

The *input* blocks form the database required to run the simulation and must be gathered beforehand. The model requires input data of distinct nature which are also obtained from different sources. Each *input* block has its own requirements regarding data types and structure, with which the user must comply to avoid errors when running the simulation. Table 44 describes each *input* block in detail. An example of each *input* block is provided in the sample case-study.

Table 44 - Characterization of input *blocks*

Block	Description	Format	Requirements	Application
weather	Historical weather data.	*csv*	The first column is the timestamp in *dd/mm/yyyy hh:mm:ss*. The following columns are the predictor variables (wind speed and wind direction).	***calculate_01_raw***: reads the raw historical data. ***calculate_02_preprocess***: preprocesses the raw historical data (e.g., filtering, outlier removal, timestamp alignment). ***calculate_03_features***: applies feature engineering to the observations (weather data) to extract as much information as possible (not featured in the current version). ***calculate_04_folds***: divides dataset into train and test subsets. ***calculate_05_train***: trains the estimator on the train folds. ***calculate_06_test***: tests the estimator on the test folds.
power	Historical generation data.	*csv*	The first column is the timestamp in *dd/mm/yyyy hh:mm:ss*. The second column (wind farm power output) is the predicted variable.	

Calculation blocks

The *calculation* blocks integrate the main framework of the model and apply various types of mathematical operations to compute different intermediate outputs. Each block takes a ***Pipeline*** object as input and estimates a relevant output, *i.e.*, produces a tangible result with a clear physical meaning in the context of the simulation. They are coded as stand-alone scripts and integrated within the aggregator block ***calculate_main***, which ensures a sequential calling of the individual blocks. Table 45 describes each *calculation* block in detail, namely its communication with the *input*, *structural*, and *output* blocks of the model.

Table 45 - Characterization of calculation *blocks*

Block	Description	Input	Structural	Output
calculate_00_setup	Instancing of ***Pipeline*** object to host *structural* blocks (see 00). Model parameterization (see 0). Ancillary function to write simulation results.	-	***config*** Assigns ***user, preprocessed, folds, train***	-
calculate_01_raw	Reading of raw historical weather and power generation data.	***weather power***	***config*** Reads ***user*** ***data*** Assigns ***raw***	-
calculate_02_preprocessed	Preprocessing of historical data (e.g., filtering).	-	***config*** Reads ***preprocessed*** ***data*** Reads ***raw*** Assigns ***preprocessed***	-
calculate_03_features	Applies feature engineering to weather data to extract more information.	-	***data*** Reads ***preprocessed*** Assigns ***features***	-
calculate_04_folds	Separates historical data into train and test folds.	-	***config*** Reads ***folds*** ***data*** Reads ***features*** Assigns ***folds***	-
calculate_05_train	Trains the estimator on each train fold.	-	***config*** Reads ***user, train*** ***data*** Reads ***folds*** Assigns ***train***	-
calculate_06_test	Tests the estimator on each test fold.	-	***config*** Reads ***user*** ***data*** Reads ***folds, train*** Assigns ***test***	-
calculate_main	Aggregator script that integrates the main *calculation* blocks.	-	***data*** Reads ***train, test***	***point probabilistic***

Structural blocks

The *structural* blocks are dictionary-type variables which store information that is accessed by multiple *calculation* blocks. Assigning the *structural* blocks directly to the ***Pipeline*** object (which is then given as input to the *calculation* blocks) greatly simplifies the number of function arguments that are passed on to each *calculation* block since all the information is stored within the object itself.

Config

The *structural* block ***config*** stores the user-defined parameterization. This information is filled in by the user, prior to running the simulation, in the *calculation* block ***calculate_00_setup*** and assigned to the ***Pipeline*** object by the same block. During the

simulation, it is accessed by most other *calculation* blocks to define the respective inner workings in terms of mathematical procedures. Figure 42 shows the structure and data types of this block and Table 46 describes each of its components in detail.

```
config_user = {'forecasts': {'point': bool,
                             'probabilistic': bool},
               'paths': {'weather': str,
                         'power': str,
                         'results': {'point': str,
                                     'probabilistic': str}}}

config_preprocessed = {'power': {'window': int}}

config_folds = {'start': pd.Timestamp,
                'train': pd.DateOffset,
                'test': pd.DateOffset}

config_train = {'point': {'learning_rate': tuple,
                          'max_depth': tuple,
                          'min_samples_split': tuple,
                          'min_samples_leaf': tuple,
                          'max_features': str,
                          'n_estimators': tuple,
                          'subsample': float},
                'probabilistic': {'alpha': np.linspace,
                                  'learnin_grate': tuple,
                                  'max_depth': tuple,
                                  'min_samples_split': tuple,
                                  'min_samples_leaf': tuple,
                                  'max_features': str,
                                  'n_estimators': tuple,
                                  'subsample': float}}
```

Figure 42 – Structure and data types of structural **config** *block.*

Table 46 - Characterization of structural **config** *block*

Component	Parameters	Application
user	***forecasts***: *dict* • ***point***: if True, trains point forecast model. • ***probabilistic***: if True, trains probabilistic forecast model. ***paths***: *dict* • ***weather***: relative path to directory with historical weather data (*input* block ***weather***). • ***power***: relative path to directory with historical generation data (*input* block ***power***). • ***results***: *dict* ○ ***point***: relative path to directory in which to store point forecast model (only applicable if *forecasts>point* is *True*). ○ ***probabilistic***: relative path to directory in which to store probabilistic forecast model (only applicable if *forecasts>probabilistic* is *True*).	Case-study parameterization (see 0). ***calculate_01_raw*** ***calculate_05_train*** ***calculate_06_test***

preprocessed	***power***: *dict* • ***window***: size of sliding time window to compute capacity factor.	Preprocessing applied to raw historical data (advanced parameterization – see 0). ***calculate_02_preprocessed***
folds	***start***: start of the first train fold. ***train***: time window of the train folds. ***test***: time window of the test folds.	Separation of historical data into train and test folds (advanced parameterization – see 0). ***calculate_04_folds***
train	***point***: *dict* • ***learning_rate***: lower and upper bound in point forecast model. • ***max_depth***: lower and upper bound in point forecast model. • ***min_samples_split***: lower and upper bound in point forecast model. • ***min_samples_leaf***: lower and upper bound in point forecast model. • ***max_features***: criterion for maximum number of features in point forecast model. • ***n_estimators***: lower and upper bound in point forecast model. • ***subsample***: reference value. ***probabilistic***: *dict* • ***alpha***: quantiles in probabilistic forecast model. • ***learning_rate***: lower and upper bound in probabilistic forecast model. • ***max_depth***: lower and upper ***bound*** in probabilistic forecast model. • ***min_samples_split***: lower and upper bound in probabilistic forecast model. • ***min_samples_leaf***: lower and upper bound in probabilistic forecast model. • ***max_features***: criterion for maximum number of features in probabilistic forecast model. • ***n_estimators***: lower and upper bound in probabilistic forecast model. ***subsample***: reference value.	Configuration of estimator (advanced parameterization – see 0). ***calculate_05_train***

Data

The *structural* block ***data*** stores the intermediate model outputs. This information is assigned to the ***Pipeline*** object by each *calculation* block. During the simulation, it is accessed by downstream *calculation* blocks to perform additional calculations. Figure 43 shows the structure and data types of this block and Table 47 describes each of its components in detail.

```
data = {'raw': xarray.Dataset,
        'preprocessed': xarray.Dataset,
        'features': xarray.Dataset,
        'folds': {'train': [xarray.Dataset],
                  'test': [xarray.Dataset]},
        'train': {'point': [sklearn.ensemble.GradientBoostingRegressor],
                  'probabilistic': [(float, [sklearn.ensemble.GradientBoostingRegressor])]},
        'test': {'point': {'predictions': [xarray.DataArray],
                           'errormetrics': {'mae': [float],
                                            'rmse': [float]}},
                 'probabilistic': {'predictions': [(float, [xarray.DataArray])],
                                   'errormetrics': {'crps': [float]}}}}
```

Figure 43 - Structure and data types of structural **data** *block.*

Table 47 - Characterization of structural **data** *block*

Component	Parameters	Source
raw	*coords*: • *time*: *numpy.datetime64* *data_vars*: • *power* (*time*): *float* • *m_speed* (*time*): *float* • *m_dir* (*time*): *float*	Raw historical weather and power generation data. ***calculate_01_raw***
preprocessed	*coords*: • *time*: *numpy.datetime64* *data_vars*: • *capacityfactor* (*time*): *float* • *m_speed* (*time*): *float* • *m_dir* (*time*): *float*	Preprocessed historical weather and power generation data. ***calculate_02_preprocessing***
features	*coords*: • *time*: *numpy.datetime64* *data_vars*: • *capacityfactor* (*time*): *float* • *m_speed* (*time*): *float* • *m_dir* (*time*): *float*	Features extracted from historical weather and power generation data (to be implemented). ***calculate_03_features***

folds	***train***: list of train data folds.
	• *coords*:
	  ○ *time*: *numpy.datetime64*
	• *data_vars*:
	  ○ *capacityfactor* (*time*): *float*
	  ○ *m_speed* (*time*): *float*
	  ○ *m_dir* (*time*): *float*
	test: list of test data folds.
	• *coords*:
	  ○ *time*: *numpy.datetime64*
	• *data_vars*:
	  ○ *capacityfactor* (*time*): *float*
	  ○ *m_speed* (*time*): *float*
	  ○ *m_dir* (*time*): *float*
		Normalized features separated into train and test folds.
		calculate_04_folds

train	***point***: trained point forecast models. ***probabilistic***: trained probabilistic forecast models for each quantile.	Point and/or probabilistic forecast models trained on train folds. ***calculate_05_train***
test	***point***: *dict*	Point and/or probabilistic forecast models tested on test folds.

	• ***predictions***: predictions of capacity factor by trained point forecast models applied to test folds.
	  ○ *coords*:
	  ▪ *time*: *numpy.datetime64*
	• ***errormetrics***: *dict*
	  ○ ***mae***: Mean Absolute Error of point forecast model.
	  ○ ***rmse***: Root Mean Square Error of point forecast model.
	probabilistic: *dict*
	• ***predictions***: predictions of capacity factor by trained probabilistic forecast models applied to test folds.
	  ○ *coords*:
	  ▪ *time*: *numpy.datetime64*
	• ***errormetrics***: *dict*
	  ○ ***crps***: Continuous Ranked Probability Score of probabilistic forecast model.
		calculate_06_test

Output blocks

The *output* blocks are a combination of intermediate model outputs from different *calculation* blocks stored in the *structural* block ***data***, which are aggregated into a single structure once the simulation is complete. Table 48 describes each *output* block in detail. An example of each *output* block is provided in the sample case-study.

Table 48 - Characterization of output *blocks*

Block	Description	Intermediate outputs	Format
point	Trained point forecast model. The tool trains/tests several models (one for each train/test fold) – the model with the highest performance is selected.	***train***	*pkl*
probabilistic	Trained probabilistic forecast models. For each quantile, the tool trains/tests several models (one for each train/test fold) – for each quantile, the model with the highest performance is selected.	***train***	*pkl*

Usage instructions

Dependencies

The dependencies listed below may be easily installed by running the following command from the main project folder:
pip install -r requirements.txt

Package	Version
pandas	2.1.0
numpy	1.25.2
sklearn	0.0.post11

scikit-learn	1.3.2
xarray	2023.8.0
properscoring	0.1

Customization

The model is structured for maximum flexibility and customizability across two distinct dimensions: case-study parameterization and advanced parameterization. The configuration of the *calculation* blocks is centred in the *calculation* block ***calculation_00_setup***. This rationale simplifies user-interaction by removing hardcoded variables and reducing the number of interaction points with the user, and facilitates the definition of case-studies and the tweaking of model parameters (*e.g.*, for model validation).

Case-study parameterization

Case-study parameterization allows the user to define their own case-study by adjusting only the necessary configurations on a case-by-case basis (*i.e.*, without altering the parameters of the *calculation* blocks). This requires only minimal knowledge on the physical phenomena being modelled and the mathematical framework of the model itself. Table 49 indicates which blocks the user may configure for case-study parameterization.

Table 49 - Blocks subject to case-study parameterization

Block	Description	Component
calculate_00_setup	The user defines the case-specific constraints (e.g., train/test split) without altering the mathematical framework of the model.	***config_user***

Advanced parameterization

Advanced parameterization allows the user to alter the mathematical formulation of the *calculation* modules, effectively adjusting the structure of the model itself. This requires robust knowledge on the physical phenomena being modelled and the mathematical framework of the model itself and is not recommended for inexperienced users as it may cause incompatible data formats and/or generate inaccurate results. Table 50 indicates which blocks the user may configure for advanced parameterization; Figure 44 shows the respective default advanced parameterization.

Table 50 - Blocks subject to advanced parameterization

Block	Description	Component
calculate_00_setup	The user alters the structure of the model itself, by adjusting the parameterization of one or more of its *calculation* blocks.	***config_preprocessed*** ***config_folds*** ***config_train***

```
config_preprocesssed = {'power': {'window': 480}}

config_folds = {'start': pd.Timestamp(year=2014, month=1, day=1, hour=0),
                'train': pd.DateOffset(years=0, months=12, weeks=0, days=0, hours=0),
                'test': pd.DateOffset(years=0, months=5, weeks=0, days=0, hours=0)}

config_train = {'point': {'learning_rate': (0.01, 0.05),
                          'max_depth': (5, 9),
                          'min_samples_split': (150, 350),
                          'min_samples_leaf': (20, 80),
                          'max_features': 'sqrt',
                          'n_estimators': (500, 800),
                          'subsample': 0.80},
                'probabilistic': {'alpha': np.linspace(0.05, 0.95, 19),
                                  'learnin_grate': (0.01, 0.05),
                                  'max_depth': (5, 9),
                                  'min_samples_split': (150, 350),
                                  'min_samples_leaf': (20, 80),
                                  'max_features': 'sqrt',
                                  'n_estimators': (500, 800),
                                  'subsample': 0.80}}
```

Figure 44 – Default advanced parameterization of calculation *block* **calculate_00_setup**.

Example case-study

For illustration purposes, an example application of the tool to a generic case-study is included in the *example* directory of the repository. The premises of the case-study are as follows:

- Generation of point and probabilistic forecast models.

The *input* blocks ***weather*** and ***power*** are available in the corresponding directories (*example/database_weather* and *example/database_power*, respectively). Simulation results are stored in ***point*** (point forecast model) and ***probabilistic*** (probabilistic forecast model) in the *example* directory.

1. Configure ***calculate_00_setup*** (see Figure 45).

```
config_user = {'forecasts': {'point': True,
                             'probabilistic': True},
               'paths': {'weather': 'example/database_weather',
                         'power': 'example/database_power',
                         'results': {'point': 'example/point',
                                     'probabilistic': 'example/probabilistic'}}}
```

Figure 45 – Example case-study parameterization of calculation *block* **calculate_00_setup**.

2. Run ***calculate_main***.
3. Analyze ***point.h5*** and ***probabilistic.h5***.

Access to the offshore wind forecasting model implementation

The offshore wind forecasting model implementation can be downloaded via a link that can be provided upon request. The final version of the model will be available open-source via a public repository, such as GitHub.

Development Timeline

The tool is already operational, although some of the planned features have not yet been implemented. As such, the next step in the development schedule regards data architecture, namely gathering the required weather data from an adequate data source and extraction of the relevant features. Afterwards, model validation will provide an indication on forecast accuracy. Lastly, the architecture of the implemented tool may suffer some adjustments for

integration with the Energy Matching Tool and other Digital Tools within the MAGPIE project. Table 51 details the timeline in greater detail.

Table 51 - Expected timeline of model development

Task	2024		2025	
	S1	S2	S3	S4
Definition of data architecture				
Selection of data source				
Integration of API				
Implementation of feature engineering				
Model validation				
Data gathering				
Definition of methodology				
Extraction of results				
Implementation of adjustments				
Integration with MAGPIE digital tools				
Definition of IT architecture				
Implementation of adjustments				

4.3.2 Solar photovoltaic

Photovoltaic power output prospection tool

Assessing the technical and geographical potential of photovoltaic (PV) power within the port infrastructure is a foundational step in the strategic integration of renewable energy sources. This assessment involves a comprehensive analysis of various factors to determine the feasibility and optimal deployment of PV energy systems.

Technical potential assessment encompasses evaluating available PV technologies, their efficiency rates, and compatibility with existing port infrastructure. This includes examining the specific PV panel models suitable for the port environment, considering corrosion resistance, structural integrity under maritime conditions, and space utilization efficiency. The assessment also involves analyzing the capacity for energy generation given the port's physical layout, including available land for ground-mounted systems and roof space for PV installations.

Geographical potential assessment focuses on the climatic and environmental conditions specific to the port's location. For PV systems, this includes analyzing solar irradiance levels, sun hours, and the impact of seasonal variations. The assessment must also consider potential

shading or blocking effects from port structures and equipment, which could impact energy generation efficiency.

Furthermore, the integration of PV energy sources in ports requires consideration of grid connectivity and potential for energy storage, which are influenced by the technical and geographical assessments. The feasibility of connecting new PV installations to existing power grids, the need for grid upgrades, and the potential for on-site or near-site energy storage solutions are critical components of the overall assessment.

This comprehensive assessment aids in identifying the most viable and efficient PV solutions for the port, ensuring that the selected systems maximize energy generation, contribute to the port's sustainability goals, and offer economic benefits through reduced energy costs and potential revenue from excess energy production. It also supports regulatory compliance and alignment with environmental standards, reinforcing the port's commitment to sustainable and responsible operations.

Numerical tool

The potential assessment is a tool implemented in Python that utilizes geospatial data to evaluate the potential for solar power generation in a designated area. This document serves as the technical documentation for the tool, detailing the methodologies and functionalities employed to provide an accurate and reliable assessment of photovoltaic (PV) power output potential.

Core Functionalities

The tool encapsulates a range of functionalities designed to interpret geographical data and translate it into actionable insights regarding PV installation potential. Herein, we detail the key components of the tool's architecture:

Library Integration

At the foundation of the tool's capabilities is the integration of Geopandas, a Python library that extends pandas to allow spatial operations on geometric types. Geopandas is pivotal in processing the Geographic Information System (GIS) files, providing the framework within which the spatial analysis is conducted.

Parameterization

The tool's flexibility is facilitated through parameterization, allowing users to define specific inputs that guide the analysis. These parameters include:

file_path: Denotes the location of the GIS file within the file system.

label_column: Identifies the column in the GIS data that differentiates distinct geographic regions.

label: Specifies the region within the GIS data for which the assessment is to be conducted.

target_crs: Indicates the Coordinate Reference System to be applied for precise area calculations.

Function Definition

A dedicated Python function is crafted to encapsulate the process of calculating the area of a labeled region. This function is the heart of the tool, orchestrating the workflow from data ingestion to area computation.

Data Handling and Analysis

The tool's analytical engine begins with the loading of the GIS file, supported by the robust capabilities of Geopandas. It processes the data to isolate the region of interest based on

the provided label parameters. A built-in validation mechanism ensures the region's presence in the dataset, raising a ValueError if the specified label does not exist.

Upon successful validation, the data undergoes a transformation to the user-specified CRS to ensure the accuracy of subsequent area calculations. The tool then employs Geopandas' geometry.area to compute the physical area designated for potential PV installation.

Technical Assessment of PV Power Output

Post area calculation, the tool embarks on a technical assessment of the solar PV potential. It integrates additional considerations such as panel dimensions, inter-panel spacing, panel inclination, and the topography factor, which collectively refine the estimate of usable panel area.

With these parameters accounted for, the tool applies solar irradiance data, system efficiency metrics, and a performance ratio to yield an estimate of the potential power output. This process synthesizes geographical constraints with technical specifications to provide a comprehensive view of the PV system's capabilities.

Outputs

The output of the tool's operation is the production of an estimated annual energy yield, expressed in kilowatt-hours (kWh). This yield serves as a metric for subsequent analyses, including but not limited to economic feasibility, environmental impact, and integration with existing power grids.

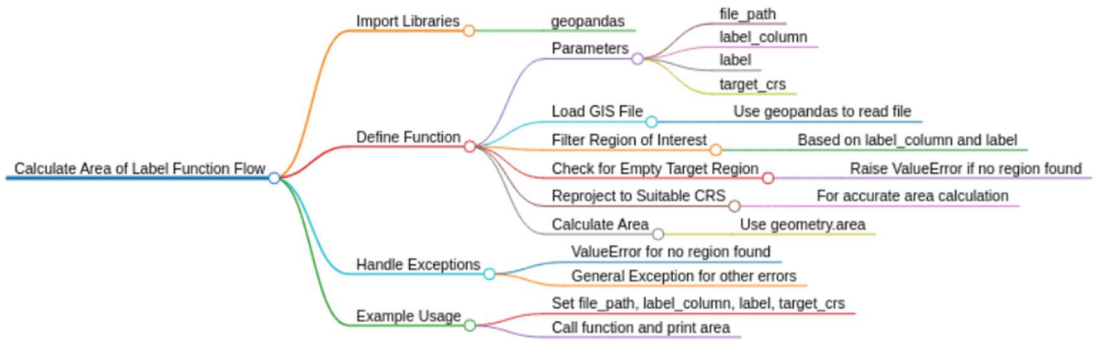


Figure 46. Overall flux to get the area from the GIS data, if this is the input case.

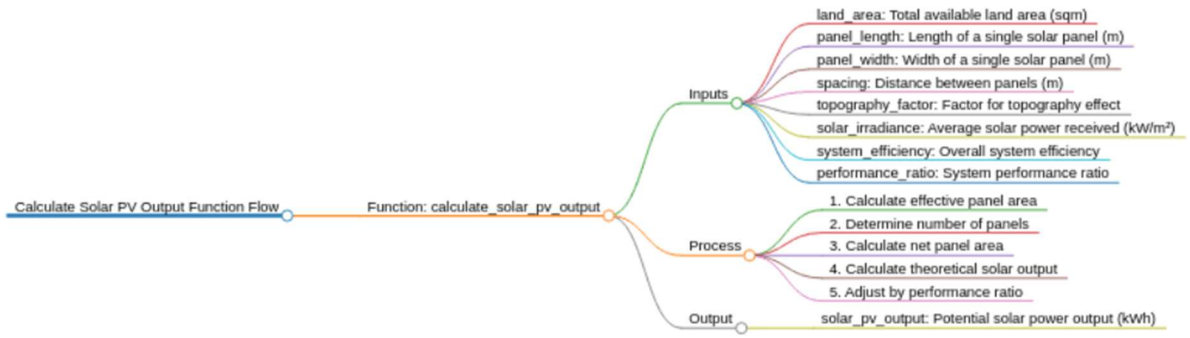


Figure 47. Overall flux to calculate solar photovoltaic resource potential.

Computer configuration

To effectively run the geospatial assessment tool for estimating the photovoltaic (PV) power output, it is essential to have a computer system configured with the necessary hardware and software capabilities. The tool, developed in Python and utilizing libraries such as Geopandas, requires computational resources sufficient for processing geospatial data and performing numerical simulations. Adequate computer configuration is crucial for the optimal operation of the geospatial assessment tool for PV power output. The specifications outlined here are designed to ensure smooth performance, quick data processing, and an overall efficient workflow for users working on solar energy potential assessments. It is also essential to maintain the system regularly, with updates to both the operating system and the Python libraries, to safeguard against vulnerabilities and to benefit from improvements in processing capabilities.

Below is the detailed system configuration required to operate the tool efficiently.

Hardware Requirements

- Processor (CPU): A modern multi-core processor (Intel i5, i7, or equivalent AMD processor) with a minimum clock speed of 2.5 GHz is recommended to handle the computational tasks effectively.

- Memory (RAM): Geospatial data processing is memory-intensive, especially when working with large datasets. A minimum of 8 GB RAM is recommended, with 16 GB or more being ideal for more extensive or complex datasets.

- Storage (Hard Drive): While the tool itself is not particularly large, geospatial datasets can be sizeable. Solid-state drives (SSDs) with at least 256 GB of storage are recommended for their faster read-write speeds, which significantly improve data processing times. Adequate space should be allocated for the GIS files and the output data.

- Graphics (GPU): Although not a strict requirement for geospatial data processing, a dedicated graphics card can accelerate rendering if the tool includes visualization features.

Software Requirements

- Operating System: The tool is platform-independent but is commonly deployed on Windows 10/11, macOS, or a modern Linux distribution such as Ubuntu 20.04 LTS or later. The chosen OS should be 64-bit to utilize the full capabilities of the hardware.

- Python Version: Python 3.7 or later must be installed. It's advisable to use the most recent Python version that's compatible with all the required libraries to ensure the best performance and security.

- Python Libraries:

  Geopandas: For reading and processing GIS data.

  Pandas: A dependency of Geopandas, used for handling data structures.

  Shapely: For manipulation and analysis of planar geometric objects, a dependency of Geopandas.

  Fiona: For reading and writing GIS files, also a dependency of Geopandas.

  GDAL: Required by Fiona for handling multiple geospatial data formats.

  Rtree: A library that provides spatial indexing features for better performance.

Numpy: For numerical computations.

- GIS Data: The GIS data files must be in a compatible format such as .shp, .geojson, or .gpkg as supported by the Geopandas library.

Recommended Development Environment

- Integrated Development Environment (IDE): An IDE such as PyCharm, Visual Studio Code, or Jupyter Notebooks offers a user-friendly interface for writing and testing Python code. These IDEs provide features such as code completion, syntax highlighting, and debugging tools that can enhance the development experience.

- Version Control: Git, along with hosting services like GitHub or GitLab, is recommended for version control, allowing for incremental development and collaboration.

Network Requirements

While the tool itself may not require an internet connection for processing local data, an active internet connection is beneficial for accessing online datasets, updating libraries, and technical support.

Model structure

The model for calculating solar photovoltaic (PV) power output is a numerical tool that transforms geographical data input, specifically the available area for solar panel installation, into an estimate of the potential energy output. This process involves several computational steps, each incorporating different aspects of PV system design and environmental considerations.

Model Workflow

After the input of the available area data, the model performs the following steps to calculate the solar PV power output:

- Panel Layout Configuration: The model starts by determining the optimal layout of the solar panels within the given area. This includes calculating the number of panels that can be placed based on their dimensions and the spacing between them to reduce shading and allow for maintenance access. The layout also accounts for service areas, access paths, and buffer zones around the installation.

- Topographical Adjustment: The effective area for panel installation may be reduced by topographical features such as slopes or uneven terrain. The model adjusts the layout configuration based on a topographical factor derived from the GIS data. This factor quantifies the extent to which the topography diminishes the usable area.

- Solar Irradiance and Inclination Factor: The model estimates the average solar irradiance using the location's latitude and historical meteorological data. It then adjusts this value based on the optimal inclination angle for the panels, maximizing exposure to sunlight throughout the year.

- System Efficiency and Performance Ratio: The model incorporates the system's efficiency, reflecting the conversion efficiency of the solar panels and losses in other system components like inverters and cables. The performance ratio further adjusts the theoretical

output to account for real-world conditions such as temperature effects, dust, and manufacturing tolerances.

- Energy Output Calculation: With all factors considered, the model calculates the theoretical energy output in kilowatt-hours (kWh). This figure represents the potential annual energy production from the installed PV system.

Future Improvements and Enhancements

To improve the accuracy and flexibility of the model, the following enhancements are proposed:

- Alternative Calculation Methods: Introduce the capability to select from different solar PV power output calculation methods. This could include methods that vary in complexity, such as simple empirical models for quick estimates or detailed simulation-based approaches for more precise assessments.

Integration of Additional GIS Data:

- Shading Analysis: Incorporate GIS data on nearby structures and vegetation to perform a shading analysis, which can significantly impact the actual energy production.
- Solar Path Analysis: Use GIS to analyze the solar path across different seasons to optimize panel orientation and inclination dynamically.
- Land Cover Data: Integrate land cover data to identify the types of surfaces within the area (e.g., vegetation, water, buildings), which can influence the suitability for solar panel installation.
- Infrastructure Proximity: Include data on existing electrical infrastructure to assess the feasibility of connecting the solar installation to the grid and potential transmission losses.

Also to consider in the following developments:

- Weather Patterns and Climate Data: Implement the use of long-term climate data to improve the estimation of the solar irradiance variability, taking into account seasonal changes and weather patterns such as cloud cover and precipitation.
- Economic and Environmental Impact Models: Augment the tool with modules that estimate the economic returns of the solar installation and its environmental impact, considering factors like carbon offset and land use changes.
- Machine Learning for unsupervised classification of available areas: Employ machine learning algorithms to predict solar power output based on historical data and identify patterns that may affect future performance.

End-to-End use example

This is an exemple of the function call (calculate_solar_pv_output) for the resource assessment potential:

```
# Example of function call
output = calculate_solar_pv_output(
    land_area=10000,  # in square meters
    panel_length=1.6,  # in meters
    panel_width=0.8,  # in meters
    spacing=0.5,  # in meters
    topography_factor=0.9,  # No unit
    solar_irradiance=5.0,  # in kW/m²
    system_efficiency=0.15,  # No unit (15% efficiency)
    performance_ratio=0.75  # No unit (75% performance ratio)
)
```

Figure 48. PV resource assessment function call example.

The result will then be displayed as follows:

```
Potential solar PV output for the given parameters: 2373.12 kWh
```

Figure 49. Example of result representation for PV resource assessment.

Access to the PV sizing model implementation

The solar PV sizing model implementation can be downloaded via a link that can be provided upon request. The final version of the model will be available open-source via a public repository, such as GitHub.

Development Timeline

This tool will be further developed during the next months of the task activities according to the development plan presented in Table 52.

Table 52 - Expected timeline of model development

Task	2024		2025	
	S1	S2	S3	S4
Model validation				
Data gathering				
Definition of methodology				
Extraction of results				
Implementation of adjustments				
Integration with MAGPIE digital tools				
Definition of IT architecture				

Implementation of adjustments				
Development of graphical interface				
Assembly of database				
Definition of IT architecture				
Design of interface				
Migration of model and database				
Testing				

Photovoltaic power output forecasting tool

The integration of photovoltaic (PV) systems into port infrastructure requires the implementation of sophisticated forecasting methods to enhance energy matching, which is the process of aligning the supply of energy from PV sources with the demand profiles of port operations. Accurate forecasting of PV power output is crucial for several reasons. It significantly improves operational efficiency by enabling the scheduling of energy-intensive tasks during periods of high availability of renewable energy. This forecasting becomes even more critical when considering the variability of solar resources, which can fluctuate more widely over short periods compared to traditional energy sources. The accurate forecasting of PV power output is indispensable for enhancing the operational efficiency, grid stability, economic performance, and sustainability of port infrastructures. It enables strategic planning and real-time management of energy resources, ensuring that renewable energy production is closely aligned with consumption patterns and regulatory requirements, thereby optimizing the overall efficiency and sustainability of port operations.

This section describes the current state of implementation, the structure, main inputs and outputs, an application to a use case and the expected development timeline of the PV forecasting tool.

Numerical tool

The developed numerical tool is a data processing and predictive modelling pipeline designed to forecast photovoltaic (PV) power output based on historical data. The tool is structured into three main components, each encapsulated in a separate Python file, facilitating a modular and sequential data processing flow. Here is a technical breakdown of the tool's components and workflow:

- Data Pre-processing (data_input.py): This module ingests raw time-series datasets related to PV power output, solar irradiation, and ambient temperature. Each dataset undergoes resampling to a uniform time interval specified by the user, with interpolation used to fill in missing values. The resampled datasets are then merged into a single DataFrame, with an additional feature 'Hour of the Day' extracted from the timestamp. The processed DataFrame is saved as a CSV file for subsequent steps.

- Feature Engineering (lagging_variables.py): This module takes the preprocessed data and enhances it by adding lagged variables for each feature, based on user-defined lag values. The inclusion of lagged variables aims to capture temporal dependencies in the data, which are crucial for time-series forecasting. Rows with incomplete data due to the lagging process are removed, and the augmented dataset is saved as a new CSV file.

- Predictive Modeling (lstm_prediction.py): The final component utilizes Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN) well-suited for time-series forecasting. The module loads the feature-augmented dataset, divides it into training and test sets, and constructs an LSTM model based on user-defined parameters such as the number of input and output timesteps, epochs, and batch size. The model is trained on the training set, and its performance is evaluated on the test set using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) as metrics. The module generates plots to visually compare the true and predicted values over the entire test set and within a user-specified detailed time interval.

- The main function orchestrates the execution of these components, ensuring the output from one step serves as the input for the next. Parameters such as the base path, file names, timestep for resampling, lag values, and LSTM model configurations are defined at the beginning of the main function and passed through the pipeline. This structured approach allows for a flexible, modular tool that can be adapted to various datasets and forecasting requirements.

Hardware and Software configuration

For running the described predictive model, especially in an online environment where real-time or near-real-time data processing and forecasting are crucial, it's essential to consider both hardware and software configurations. The complexity and size of the dataset, the frequency of data updates, and the expected response time for generating predictions will significantly influence the required computer configuration. These are recommendations, although weaker hardware configurations can also be used - the computer configuration for running this predictive model online should balance computational power, storage, and network capabilities while ensuring the system is secure, scalable, and maintainable.

Hardware Configuration

- CPU: A multi-core processor (e.g., Intel Core i7 or Xeon, AMD Ryzen) with a high clock speed is recommended to handle multiple threads efficiently, especially for data pre-processing and feature engineering steps.

- GPU: For training LSTM models, a dedicated Graphics Processing Unit (GPU) is highly recommended due to its parallel processing capabilities. NVIDIA GPUs with CUDA support (e.g., NVIDIA GeForce RTX or Tesla series) can significantly accelerate the training of deep learning models.

- RAM: Sufficient Random Access Memory (RAM) is crucial for handling large datasets in memory. A minimum of 16GB RAM is recommended, but 32GB or more may be required for larger datasets to avoid swapping data to disk, which can significantly slow down processing.

- Storage: Solid State Drives (SSDs) are preferred over Hard Disk Drives (HDDs) for faster data read/write operations. The storage capacity will depend on the size of the dataset and the number of models being stored but should start at a minimum of 512GB SSD.

- Network: A stable and fast internet connection is essential for online deployment, especially if the system interacts with cloud-based data storage or APIs for real-time data ingestion.

Software Configuration

- Operating System: The tool should be compatible with major operating systems such as Linux (Ubuntu, CentOS), Windows, or macOS, depending on the deployment environment's requirements. It was developed initially and in current version in Windows.
- Python Environment: A Python 3.11 environment with package management tools like pip or conda to manage dependencies.
- Dependencies: Key Python libraries including pandas for data manipulation, numpy for numerical operations, matplotlib for plotting, tensorflow or keras for building and training LSTM models, and any other library required by the specific predictive model or data preprocessing steps.
- Web Server (for online deployment): For deploying the model online, a web server like Apache, Nginx, or a Python-based web framework such as Flask or Django can be used to handle HTTP requests and serve the model's predictions (as the input of Numerical Weather Predictions)
- Security: Proper security measures, including firewalls, SSL encryption, and secure API endpoints, are crucial to protect the online system from unauthorized access and data breaches.

Scalability and Reliability Considerations

For online deployment, it's also important to consider scalability and reliability. Containerization tools like Docker can encapsulate the application and its environment, making it easier to deploy across different systems. Kubernetes or other orchestration tools can manage containerized applications to ensure high availability, load balancing, and automatic scaling based on demand.

Monitoring and Maintenance

Finally, ongoing monitoring and maintenance are essential to ensure the system performs optimally. Logging, performance monitoring, and alerting tools should be in place to detect and respond to issues promptly.

Tool structure

Concerning the data input function, which deals with the processing of energy data, its flow is summarized in Figure 50. The data_input function serves as a preprocessing step in a predictive modeling workflow, focusing on the preparation of energy-related time series data. This function operates by first setting up a structured path to access the input CSV files, which contain time-stamped data on photovoltaic (PV) power output, solar irradiation, and synthetic temperature measurements. These CSV files are then loaded into separate pandas DataFrames.

Each DataFrame undergoes a conversion process where the first column, assumed to be datetime strings, is transformed into pandas datetime objects, enabling time-based indexing. The original datetime columns are then discarded, and the data is resampled to a uniform timestep specified by the timestep_minutes parameter. This resampling utilizes linear interpolation to fill in any missing values, ensuring a continuous and consistent time series.

Post-resampling, the function combines the individual DataFrames into a single DataFrame, aligning them by their datetime indices. This combined DataFrame undergoes further refinement, rounding the data to three decimal places for numerical stability and precision. An additional feature, 'Hour of the Day', is calculated and appended to the DataFrame to provide an extra dimension of temporal context, potentially useful for predictive modeling.

The final step involves saving this processed and combined DataFrame to a new CSV file, designated by the output_filename parameter. This file is intended to be used in subsequent steps of the predictive modeling workflow, such as feature engineering with lagged variables and the construction of a predictive model using an LSTM neural network. The function concludes by returning the path to the newly created CSV file, providing a direct link for the next stages of analysis.

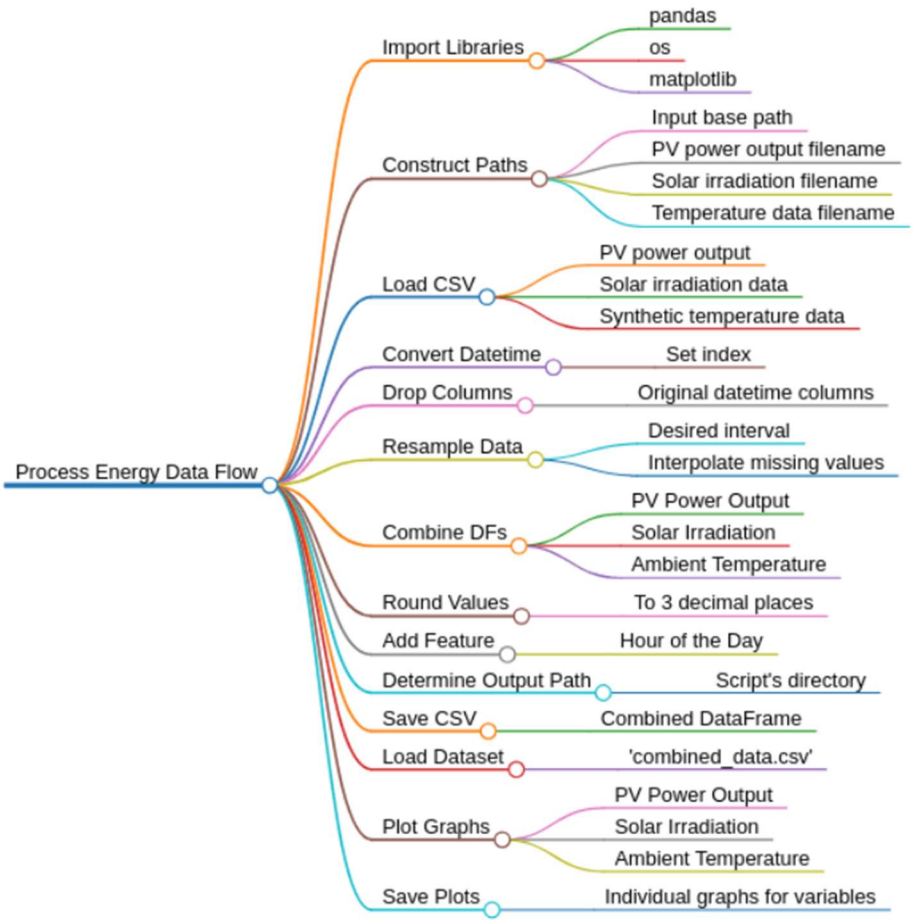


Figure 50. Data input and process function diagram.

Following, the model performs feature engineering, namely the lagging variable to capture temporal patterns. Its overall flux is summarized in Figure 51. The lagging variable function is a critical component in the feature engineering process of time series forecasting. This function enhances the dataset by introducing historical context, enabling the model to recognize and learn from patterns over time. It begins by leveraging libraries such as pandas for data manipulation and os for file path management.

The function's first step is to ascertain the directory of the script currently running. It then constructs the full path to the 'combined_data.csv' file, which contains the time series data that will be used for lagging. The data is then loaded into a pandas DataFrame, with the first column set as the index and converted into datetime format to facilitate time-based operations.

Once the DataFrame is correctly indexed by time, the function proceeds to create lagged variables. For each variable specified in the 'lags' list, the function shifts the data by the number of periods defined, creating new columns in the DataFrame that represent the historical values. This is done for each variable, enriching the dataset with past observations which serve as new input features.

However, the lagging process inherently introduces NaN values for the initial entries where historical data is not available. To maintain data integrity, the function removes these rows from the DataFrame.

The final step in the function's flow is to save the augmented dataset, now containing the lagged variables, to a new CSV file named 'lag_combined_data.csv'. This file is then ready to be used in the subsequent phases of model building, where the LSTM neural network or other forecasting algorithms can utilize the constructed features to predict future trends based on historical patterns.

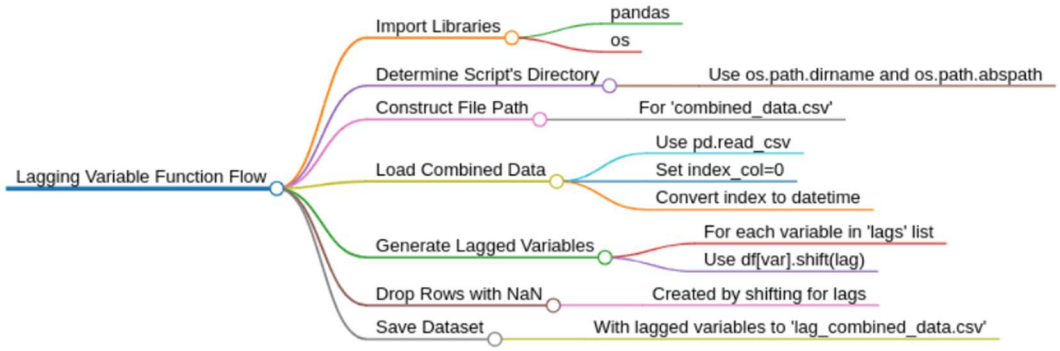


Figure 51. Feature engineering process function diagram.

The LSTM prediction function encapsulates a methodical approach to time series forecasting, employing a specialized neural network architecture known as Long Short-Term Memory (LSTM) (as per the overall flow is described in detail in Figure 52). The process begins by importing essential libraries for numerical computations, data manipulation, model training, and visualization. Parameters defining the structure of the LSTM model—such as the number of input and output steps, epochs for training, batch size, and the desired datetime interval for detailed results—are specified upfront.

The dataset is loaded from a CSV file using Pandas, ensuring that the datetime index is correctly parsed. A preparation phase follows, where the data is segmented into input features and target variables, with the target being the first column of the dataset, typically representing the time series to predict. The model leverages the LSTM's ability to remember long-term dependencies and patterns within the data, making it adept at capturing complex temporal behaviors. This attribute is particularly beneficial for time series data where the chronological order and past values significantly influence future predictions.

After defining the LSTM model, which comprises LSTM layers and dense layers to learn from the sequence data, the model is compiled using an optimizer and loss function suitable for regression tasks. The training process utilizes an early-stopping mechanism to prevent overfitting by halting the training when the validation loss ceases to decrease.

After training, the model predicts future values based on the test dataset. These predictions are then evaluated against the true values using metrics such as mean absolute error and root mean squared error to quantify the model's performance. The results are visually represented through plots that juxtapose the predicted values with the actual data, offering an intuitive understanding of the model's accuracy and its behaviour over time. With its modular design, this function enables a detailed analysis of prediction performance and can be adjusted for various LSTM configurations to suit different forecasting scenarios.

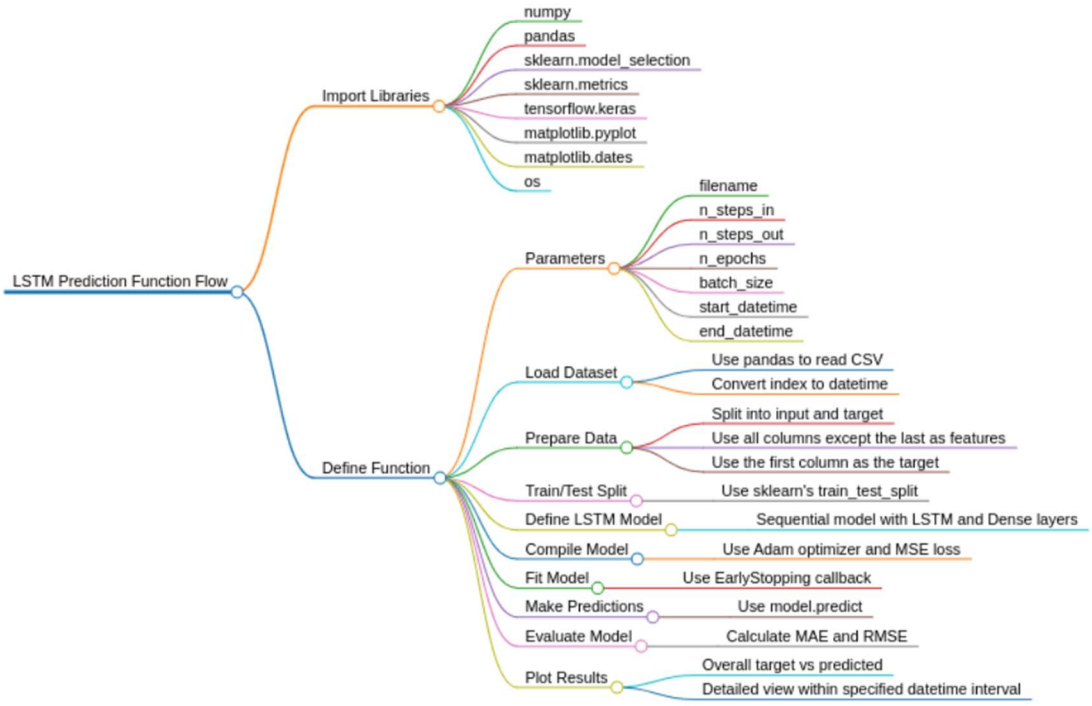


Figure 52. LSTM prediction process function diagram.

Parameters

The numerical tool developed through the integration of multiple Python functions facilitates the construction, execution, and analysis of a predictive model focusing on time series data. The main function, serving as the orchestrator, allows users to interact with and control various parameters influencing each stage of the data processing and prediction workflow.

Current user-controlled parameters include (as exemplified in the diagram flow of the main function in Figure 53):

input_base_path: The directory path where all input CSV files are located.

file_names: A list containing the filenames of the PV power output, solar irradiation, and temperature data, allowing for the customization of input data sources.

timestep_minutes: The interval in minutes for resampling the data, enabling users to define the granularity of the time series analysis.

lags: A list detailing the number of lagged observations to create for each variable, which directly impacts the features used in the LSTM model.

lstm_params: A dictionary that includes parameters for the LSTM model such as the number of input and output steps (n_steps_in, n_steps_out), the number of epochs (n_epochs), and the batch size (batch_size).

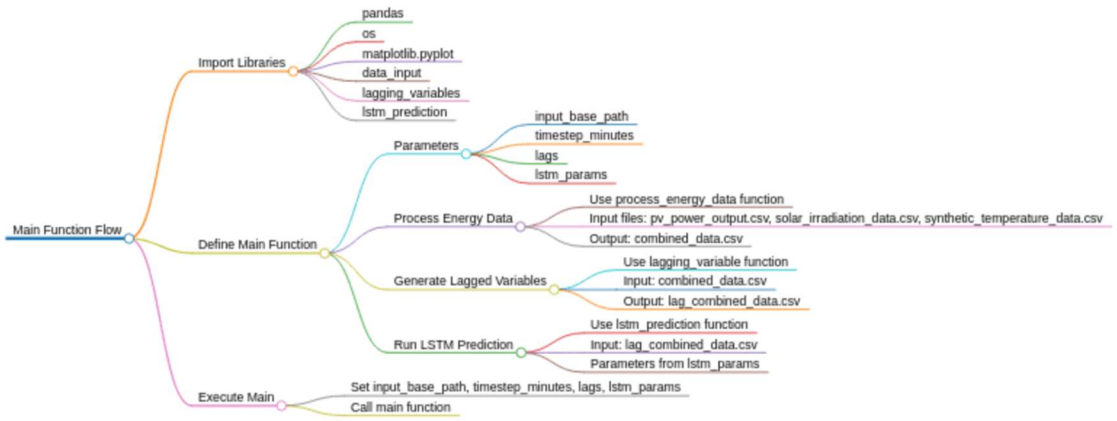


Figure 53. Main process function diagram.

In future iterations, the model will be enhanced to allow users to adjust additional parameters, including but not limited to:

Learning rate: Providing control over the rate at which the model learns, enabling fine-tuning for convergence and performance.

Activation functions: Allowing the selection of different activation functions to control the non-linearity introduced in the model.

Dropout rates: Offering the option to specify dropout rates to prevent overfitting and improve model generalization.

Number and size of LSTM layers: Enabling users to design the depth and complexity of the LSTM network to capture more complex patterns.

Custom date ranges for training and testing splits: Empowering users to define specific periods for model training and evaluation, which is crucial for temporal data with seasonal or cyclic patterns.

Hyperparameter tuning interface: Introducing a user-friendly interface for automated hyperparameter optimization, further refining model performance.

By equipping the model with a broader range of user-controlled parameters, the predictive power and adaptability of the tool will be significantly enhanced, promoting a more nuanced understanding of time series data and facilitating more accurate forecasting.

End-to-End use example

This example shows how to set and end-to-end run of the tool. The input data should be as shown in following figure, in .csv format (this is a sample for one day, and how the heading should look like, but in this .csv the whole historic data under analysis should be present, e.g. months or years). The input files should ideally be in the same folder as the python code.

```
,Solar Irradiation
2019-01-01 00:00:00,0.0
2019-01-01 01:00:00,0.0
2019-01-01 02:00:00,0.0
2019-01-01 03:00:00,0.0
2019-01-01 04:00:00,0.0
2019-01-01 05:00:00,0.0
2019-01-01 06:00:00,0.0
2019-01-01 07:00:00,0.0
2019-01-01 08:00:00,0.8887732273492923
2019-01-01 09:00:00,68.73746976826881
2019-01-01 10:00:00,147.43491059975392
2019-01-01 11:00:00,209.84521743945797
2019-01-01 12:00:00,214.8631832926877
2019-01-01 13:00:00,192.19800677133662
2019-01-01 14:00:00,120.08743233730253
2019-01-01 15:00:00,24.104566847849874
2019-01-01 16:00:00,0.0
2019-01-01 17:00:00,0.0
2019-01-01 18:00:00,0.0
2019-01-01 19:00:00,0.0
2019-01-01 20:00:00,0.0
2019-01-01 21:00:00,0.0
2019-01-01 22:00:00,0.0
2019-01-01 23:00:00,0.0
```

Figure 54. .csv input format example.

The parameters could be set, currently, in the main function. An example is given bellow in Figure 55, with parameters set as follows.

```python
if __name__ == '__main__':
    # Define your base path and file names
    input_base_path = r'C:\Users\karol\PycharmProjects\deliverable_fore2\deliverable_fore'
    file_names = [
        'pv_power_output_aug.csv',
        'solar_irradiation_data.csv',
        'synthetic_temperature_data.csv'
    ]

    # Define the timestep for resampling and lags for each variable
    timestep_minutes = 15
    lags = [2, 3, 3, 1]  # Example: PV Power Output, Solar Irradiation, Temperature, Hour of the Day

    # Define LSTM parameters w
    lstm_params = {
        'n_steps_in': 3,
        'n_steps_out': 1,
        'n_epochs': 10,
        'batch_size': 32,
        'start_datetime': '2021-10-01 00:00:00',
        'end_datetime': '2021-10-10 00:00:00'
    }

    # Call the main function
    main(input_base_path, file_names, timestep_minutes, lags, lstm_params)
```

Figure 55. Example of parameters input in the main function.

Following running the function, the results are as follows. It confirms the variable lagging with a comparison showing power output in time t and in timet-1, as an example, while the code itself is regressing the lags as requested in the function call. It will display as follows:

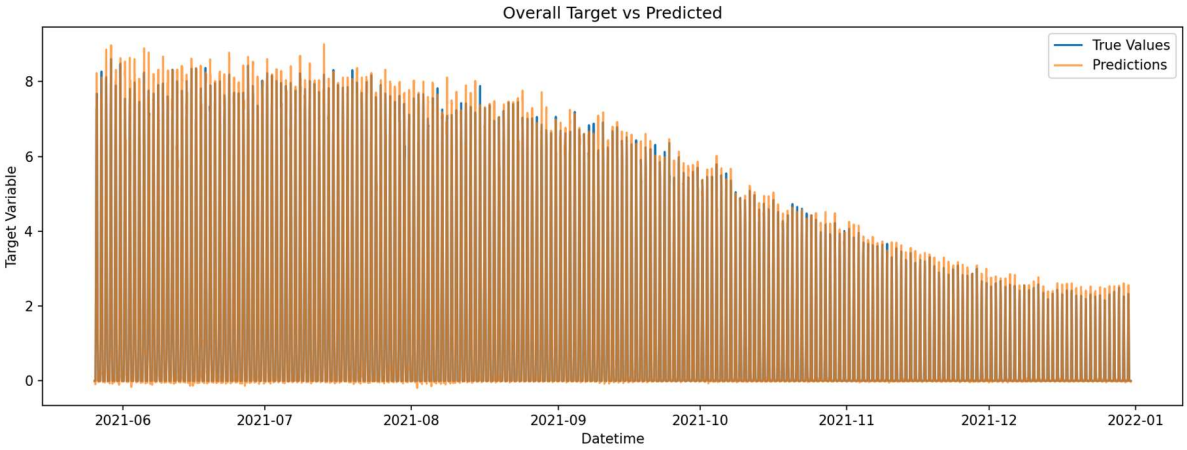
```
Number of samples in raw data (PV Power Output): 26281
Number of samples in resampled data (PV Power Output): 105121
Number of samples in raw data (Solar Irradiation): 26281
Number of samples in resampled data (Solar Irradiation): 105121
Number of samples in raw data (Ambient Temperature): 26281
Number of samples in resampled data (Ambient Temperature): 105121
Combined data saved to: C:\Users\karol\PycharmProjects\deliverable_fore2\deliverable_fore\combined_data.csv
DataFrame structure after adding lagged variables:
                     PV Power Output  ...  Hour of the Day Lag 1
Unnamed: 0                            ...
2019-01-01 00:00:00              0.0  ...                   NaN
2019-01-01 00:15:00              0.0  ...                   0.0
2019-01-01 00:30:00              0.0  ...                   0.0
2019-01-01 00:45:00              0.0  ...                   0.0
2019-01-01 01:00:00              0.0  ...                   0.0

[5 rows x 13 columns]
Number of rows after dropping NaNs: 105118
```

Figure 56. Sampling confirmation example.

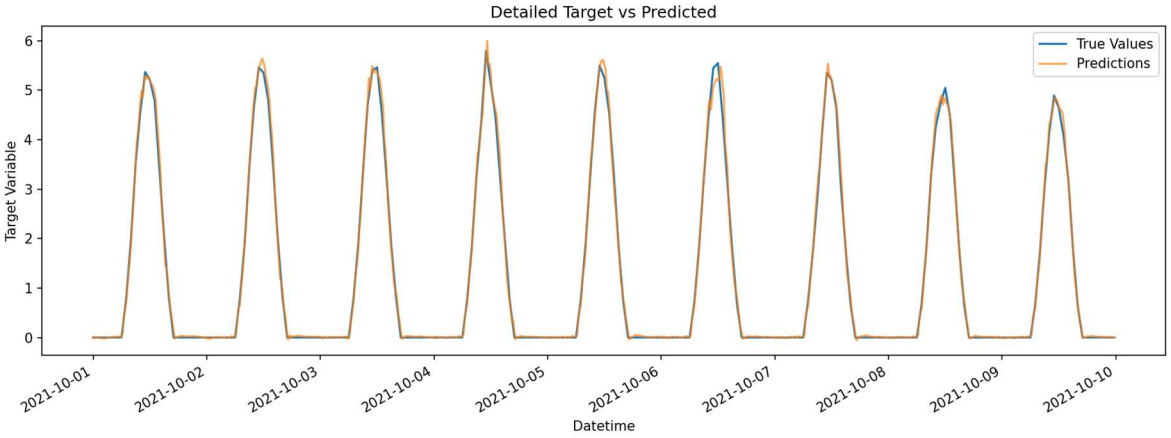
```
Epoch 7/10
2103/2103 [==============================] - 3s 2ms/step - loss: 0.4398 - val_loss: 0.0324
Epoch 8/10
2103/2103 [==============================] - 3s 2ms/step - loss: 0.0624 - val_loss: 0.0367
Epoch 9/10
2103/2103 [==============================] - 3s 2ms/step - loss: 0.0470 - val_loss: 0.0211
Epoch 10/10
2103/2103 [==============================] - 3s 2ms/step - loss: 0.0409 - val_loss: 0.0255
657/657 [==============================] - 1s 783us/step
Mean Absolute Error: 0.10559377028202344
Root Mean Squared Error: 0.20135454477423287
```

Figure 57. Display of epoch training, final MAE and RMSE.

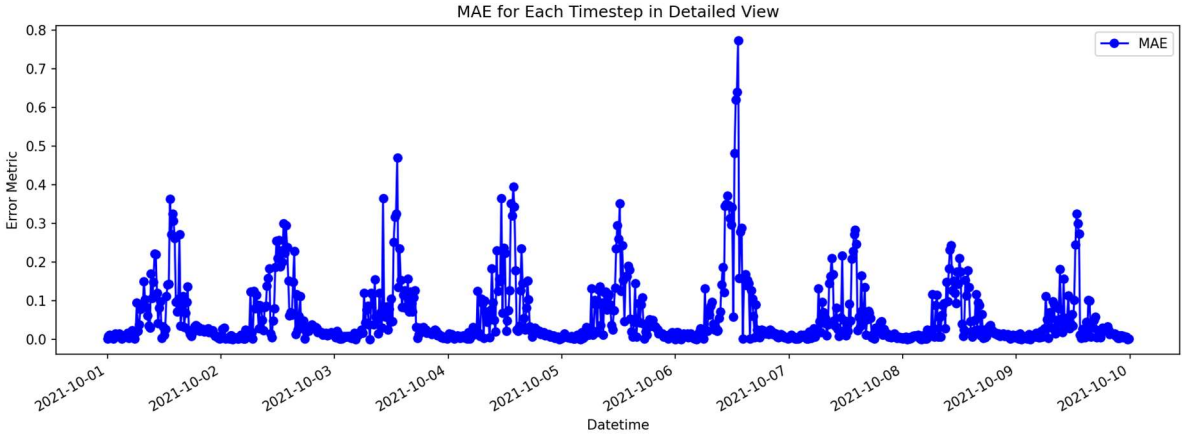
Then, it plots the predicted vs. the target values (example is shown below).



i. Total testing period.



ii. Detailed period requested in the main function.



iii. Mean absolute error for the detailed period.

Figure 58. Plotting of the results - demonstration example.

There is a complementary function to run after the main function, called json_convert, to save the output in Json format. Figure 59 shows an example the procedure to call the function (a), and an example of the results (b).

```
# The name of the CSV file with the prediction results
csv_file_name = 'prediction_results.csv'

# The desired name for the output JSON file
json_file_name = 'output.json'

# The resource name as shown in the example image
res_name = 'pv_1'

# Call the function to perform the conversion
convert_csv_to_json(csv_file_name, json_file_name, res_name)
```

(a) Json convertion function call example.

```
{
    "res": [
        {
            "name": "pv_1",
            "profile_MWh": [
                {
                    "datetime": "2020-10-19 02:15:00",
                    "value": 0.025017679
                },
                {
                    "datetime": "2020-10-19 02:30:00",
                    "value": -0.0026100222
                },
                {
                    "datetime": "2020-10-19 02:45:00",
                    "value": 0.018060237
                },
                {
                    "datetime": "2020-10-19 03:00:00",
                    "value": -0.0045471955
                },
                {
                    "datetime": "2020-10-19 03:15:00",
                    "value": 0.020483494
                },
                {
                    "datetime": "2020-10-19 03:30:00",
                    "value": -0.0040414818
                },
                {
                    "datetime": "2020-10-19 03:45:00",
                    "value": 0.019519806
                },
```


(b) Example of the piece of the results.

Figure 59. Json convertion.

Access to the PV forecast model implementation

The solar PV forecast model implementation can be downloaded via a link that can be provided upon request. The final version of the model will be available open-source via a public repository, such as GitHub.

Development timeline

The current version of the tool will be further improved according to the timeline presented in Table 53.

Table 53 - Expected timeline of model development

Task	2024		2025	
	S1	S2	S3	S4
Definition of data architecture				

Selection of data source				
Integration of API				
Implementation of feature engineering				
Model validation				
Data gathering				
Definition of methodology				
Extraction of results				
Implementation of adjustments				
Integration with MAGPIE digital tools				
Definition of IT architecture				
Implementation of adjustments				

5 Conclusion

This document described the current level of implementation of the digital tools, backend models, digital services, and the data-sharing infrastructure at the basis of the port digital twin. This is intended as a companion to the delivery of the codes, platforms and apps resulting from the implementation of the different components of the port digital infrastructure as outlined in deliverables 4.2 and 4.3.

The three tools under development in 4.5, i.e., the GHG tooling, the Energy Matching Tool (EMT) and the Smart and Green logistics tool are either still developing the underlying models (GHG tool) or have implemented a "mock" version of the tool (EMT and Smart and Green Logistics tool). This document has also outlined the expected advances within the next 18 months of development to deliver the final versions of the tools in M48.

Several backend models – i.e., electricity demand estimation, flexibility modelling for terminal assets and buildings, and renewable power sizing and forecast (for wind and solar PV) – have been implemented, albeit using simplified modelling or mock and synthetic data. Further development will allow for modelling more complex relationships between parameters and the connection to real data. The first version of the linkages between the backend models and the EMT is underway and should be tested with a use case in M30-31. The development of the backend models will continue in parallel with the integration with the EMT tool and the digital data-sharing infrastructure to deliver a final version in M48.

Finally, the document also describes the current level of implementation of the data-sharing infrastructure and ontology outlined in deliverables D4.2 and D4.3. At least 3 use cases are envisioned, with the possibility of expanding to other MAGPIE demos.

The next deliverable (4.5.2) is scheduled for M48, which will deliver the final version of all the digital tools, backend models and the digital infrastructure supporting the use cases and respective digital services.

6 References

[1] FEDeRATED https://www.federatedplatforms.eu/ (accessed Nov. 23, 2023).
[2] SAREF4AUTO https://saref.etsi.org/saref4auto/v1.1.1/ (accessed Nov. 23, 2023).
[3] European Union Agency for Railways (2023). "ERA vocabulary." 2023. Accessed: Nov. 23, 2023. [Online]. Available: http://publications.europa.eu/resource/dataset/era-vocabulary
[4] SAREF4ENER https://saref.etsi.org/saref4ener/v1.1.2/ (accessed Nov. 23, 2023).
[5] Kadaster (2023). 2023. https://www.kadaster.nl/ (accessed Nov. 23, 2023).
[6] UNE "Methodology for calculation and declaration of energy consumption and GHG emissions of transport services (freight and passengers). UNE-EN 16258:2013."
[7] ISO (2023). "Greenhouse gases. Quantification and reporting of greenhouse gas emissions arising from transport chain operations. ISO 14083:2023(E)." 2023.
[8] TNO "Decamod: toolbox voor rekenen aan CO2-reductie in transport en logistiek," TNO 2020 R11938. Accessed: Nov. 23, 2023. [Online]. Available: http://resolver.tudelft.nl/uuid:d14b3031-cab3-4a13-a0db-a970ef188143
[9] GLEC Framework (2023). 2023. https://www.smartfreightcentre.org/en/our-programs/global-logistics-emissions-council/ (accessed Nov. 23, 2023).
[10] BigMile (2023). 2023. https://bigmile.eu (accessed Nov. 23, 2023).
[11] EmissionInsider: Pave the way to a zero-emissions port (2023). *port-xchange.com*, 2023. https://port-xchange.com/emissioninsider/ (accessed Nov. 23, 2023).
[12] Routescanner (2023). 2023. https://www.routescanner.com/ (accessed Nov. 23, 2023).
[13] BasGoed https://www.basgoed.nl/basgoed/ (accessed Nov. 23, 2023).
[14] Institute of Energy Economics at the University of Cologne (2021). 2021. Accessed: Jan. 18, 2024. [Online]. Available: https://www.ewi.uni-koeln.de/en/publikationen/globales-ptx-produktions-und-importkostentool
[15] World Port Sustainability Program–WPSP (2019). 2019. https://sustainableworldports.org/project/port-of-amsterdam-fritzy-and-friends (accessed Nov. 23, 2023).
[16] TNO (2023). "Ondersteuning Haven Emissie Service Platform (HESP) model," 2023.
[17] NEXUS (2023). 2023. https://nexuslab.pt (accessed Nov. 23, 2023).
[18] J. N. P. van Stralen, F. Dalla Longa, B. W. Daniëls, K. E. L. Smekens, and B. van der Zwaan (Dec. 2021). "OPERA: a New High-Resolution Energy System Model for Sector Integration Research," *Environ Model Assess*, vol. 26, no. 6, pp. 873–889, Dec. 2021, doi: 10.1007/s10666-020-09741-7.
[19] World Port Sustainability Program–WPSP (2018). 2018. https://sustainableworldports.org/project/jadeweserport-port-energy-consumption-management-tool (accessed Nov. 23, 2023).
[20] TNO (2021). "POSEIDON gebruikershandleiding Prognosis Of Shipping Emissions by Improved enDuring Observation of Navigation," TNO 2020 R12350, 2021. Accessed: Nov. 23, 2023. [Online]. Available: https://www.pbl.nl/sites/default/files/downloads/handleiding_poseidon_tno-2020-r12350.pdf
[21] Fraunhofer (2023). 2023. https://reff.iml.fraunhofer.de (accessed Nov. 23, 2023).
[22] CE Delft (2023). "Shipping GHG emissions 2030. Analysis of the maximum technical abatement potential," 2023. Accessed: Nov. 23, 2023. [Online]. Available: https://cedelft.eu/publications/shipping-ghg-emissions-2030/
[23] Eurocontrol (2023). 2023. https://www.eurocontrol.int/tool/small-emitters-tool (accessed Jan. 18, 2024).
[24] European Commission (2023). "Vehicle Energy Consumption calculation TOol - VECTO." 2023. Accessed: Nov. 23, 2023. [Online]. Available: https://climate.ec.europa.eu/eu-action/transport/road-transport-reducing-co2-emissions-vehicles/vehicle-energy-consumption-calculation-tool-vecto_en
[25] European Institute on Economics and the Environment (2019). 2019. https://www.eiee.org (accessed Jan. 18, 2024).

[26] Eurostat (2007). "Standard goods classification for transport statistics (NST 2007)." 2007. Accessed: Nov. 23, 2023. [Online]. Available: http://publications.europa.eu/resource/dataset/nst2007
[27] G. Geilenkirchen *et al.* (2023). "Methods for calculating the emissions of transport in the Netherlands," PBL Netherlands Environmental Assessment Agency, PBL publication number: 4923, 2023.
[28] European Commission *et al.* (2020). *JEC well-to-tank report V5 – JEC well-to-wheels analysis – Well-to-wheels analysis of future automotive fuels and powertrains in the European context*. Publications Office, 2020. doi: 10.2760/959137.
[29] PBL – Netherlands Environmental Assessment Agency. (2023). "Climate and energy outlook 2023. Estimates of greenhouse gas emissions, renewable energy, and energy savings in outline.," PBL publication number: 5243, 2023.
[30] Port of Rotterdam Authority (2023). "Scenarios for 2050," 2023. https://www.portofrotterdam.com/sites/default/files/2022-12/white-paper-future-scenarios-2050_0.pdf (accessed Jan. 18, 2024).
[31] T. Brown, J. Hörsch, and D. Schlachtberger (Mar. 27, 2018). "Python for Power System Analysis (PyPSA) Version 0.13.1." Zenodo, Mar. 27, 2018. doi: 10.5281/zenodo.1208706.
[32] J. Heilmann, M. Wensaas, P. Crespo del Granado, and N. Hashemipour (Nov. 2022). "Trading algorithms to represent the wholesale market of energy communities in Norway and England," *Renewable Energy*, vol. 200, pp. 1426–1437, Nov. 2022, doi: 10.1016/j.renene.2022.10.028.
[33] N. Liu, X. Yu, C. Wang, C. Li, L. Ma, and J. Lei (Sep. 2017). "Energy-Sharing Model With Price-Based Demand Response for Microgrids of Peer-to-Peer Prosumers," *IEEE Transactions on Power Systems*, vol. 32, no. 5, pp. 3569–3583, Sep. 2017, doi: 10.1109/TPWRS.2017.2649558.
[34] D. Kanakadhurga and N. Prabaharan (Dec. 2022). "Peer-to-Peer trading with Demand Response using proposed smart bidding strategy," *Applied Energy*, vol. 327, p. 120061, Dec. 2022, doi: 10.1016/j.apenergy.2022.120061.
[35] Buildings - Energy System *IEA*. https://www.iea.org/energy-system/buildings (accessed Apr. 18, 2024).
[36] ISO (2008). "Energy performance of buildings. Calculation of energy use for space heating and cooling. ISO 13790:2008." 2008. [Online]. Available: https://www.iso.org/standard/41974.html
[37]ISO (2017). "Energy performance of buildings. Energy needs for heating and cooling, internal temperatures and sensible and latent heat loads. Part 1: Calculation procedures. ISO 52016-1:2017." 2017. [Online]. Available: https://www.iso.org/standard/65696.html

Annex 1

Task T4.5 description as per the Grant Agreement

T4.5: Implementation and Integration of Digital Twin Platforms and Services (M18-M48) [INESC (35 PM), CEA (6 PM), CIRCOÉ (7 PM), IFPEN (15 PM), TNO (25 PM)] This task will implement several open platforms that will compose the digital twins of the different ports. It will use of the outputs generated by previous tasks to consolidate the implementation which will be carried out in 2 phases: i) the initial deployment of digital tools (until M24); ii) iterative refinements based on the early results of the platform's operation (until M48). These will be strongly articulated with energy supply chains set in WP3 and the demonstration activities of WP5, and 6.

Subtask 4.5.1: GHG Tooling (TNO lead): Based on real-world data (e.g. CO2eq per (tonne / TEU) per transport chain, CO2eq per tonne-km emission intensity per service category), this task will implement modelling and prediction capabilities to facilitate emission reduction related to efficiency of operations, fuel - and modal shift at operational and strategic levels: 1) At operational level and in compliance with international standards (EN 16258, future ISO 14083), emission information will be used in situational decision-making allowing evidence-based selection of the least polluting transport options (e.g. synchro-modality, carrier choice, routing); 2) At strategic level, it will assess the impact of decarbonisation measures and provide evidence-based assessment functionality for analysis of trends, innovations and policy measures. This task delivers: 1) real world data on transport chain emissions related to ports; 2) emission-relevant data exchange mechanism for 3) Decision Support System (DSS) for decarbonisation at operational level and 4) and intelligence for autonomous and strategic decisions.

Subtask 4.5.2: Energy Matching (EDP lead): This task will deliver an energy matching platform to balance the need of supplying loads with existing (or soon to be installed/considered) renewable energy sources in the different segments of ports (maritime, in-port, hinterland). Monitoring and controlling of energy match will be ensured to provide insights regarding the achieved improvements in the operation of ports. P2P market-based systems will be employed to facilitate the interaction between agents considering high standards of transparency and market negotiation, while ensuring energy provision. This platform will use data from these systems to establish optimal energy usage patterns, combined with enhanced storage elements (e.g., batteries, ZES containers) to induce flexibility in the demand side that can influence the reduction of GHG emissions. To support an optimal energy matching, the learning algorithms previously developed on buildings/batteries/H2 system will provide forecast on consumptions and updated predictive models of each energy object.

Subtask 4.5.3 Ports Smart and Green Logistics (lead: EUR): This task will deliver an integrated DSS, embedding a process DT like functionality, enabling the port to deploy and manage more efficient, reliable, environmentally sustainable, and less energy consuming operations. Based on a synchro-modality concept, event discrete simulation will be used at the tactical and operational levels, and system dynamics models will support strategic and long- term decisions. Optimisation, based on multi-objective metaheuristics, will be explored concerning design decisions and operations planning and scheduling. These models will perform over the digital twin (T4.2) but will also interoperate with the data-driven services developed in WP3 and T4.4, and with the set of legacy systems and real-time sensors (T4.1), to provide on-time guidance and orientations for managing the system. Sustainability, energy efficiency, CO2 emissions, as well as other environmental impacts (noise and atmospheric pollution), and renewable and green energies usage will be the main indicators to be fulfilled.