BUILDCHAIN

Funded by the European Union

TRACE Structures

API for

Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

API Version Vβ-1.2
Version of documentation: V1.3

Documentation of the BUILDCHAIN Project Deliverable D4.5

Digital Twin Generating APIs for Improved Structural Resilience

Noémi Friedman, Bence Popovics, Áron Friedman, Emese Vastag, András Urbanics,
Filippo Landi, Constanza Guarducci, Giulia Pierotti, Giada Bartolini,
Blaž Kurent, Boštjan Brank, Juan Chiachio Ruano, Emilio Daroch,
Uroš Bohinc, Stevan Nesovic
editor and coordinator: Noémi Friedman

European HORIZON Programme - BUILDCHAIN GA #101092052

D4.5 Digital Twin Generating APIs for Improved Structural Resilience

Deliverable number	**D4.5**
Deliverable title	Digital Twin Generating APIs for Improved Structural Resilience
Nature[1]	OTHER
Dissemination Level[2]	PU
Author (email) Institution	Noémi Friedman (n.friedman@ilab.sztaki.hu), Bence Popovics (popbence@gmail.com), Áron Friedman (friedrron@gmail.com), Emese Vastag (vastag.mse@gmail.com), András Urbanics(urbanicsandras@sztaki.hu) **SZTAKI**, Filippo Landi(filippo.landi@ing.unipi.it), Constanza Guarducci(costanza.guarducci@unifi.it), Giulia Pierotti(giulia.pierotti@phd.unipi.it), Giada Bartolini (giada.bartolini@phd.unipi.it) **UNIPI**, Blaž Kurent (blaz.kurent@fgg.uni-lj.si), Boštjan Brank (Bostjan.Brank@ikpir.fgg.uni-lj.si) UL, Juan Chiachio Ruano (jchiachio@ugr.es), Emilio Daroch (edaroch@go.ugr.es) **UGR**, Uroš Bohinc (uros.bohinc@zag.si) **ZAG**, Stevan Nesovic (stevan.nesovic@origin-trail.com) **TRACE**
Editor (email) Institution	Noémi Friedman (n.friedman@ilab.sztaki.hu) SZTAKI
Leading partner	SZTAKI
Participating partners	SZTAKI, UNIPI, UL, UGR, ZAG, TRACE
Official submission date	31st of July, 2025
Actual submission date	31st of July, 2025

[1]**R**=Document, report; **DEM**=Demonstrator, pilot, prototype; **DEC**=website, patent fillings, videos, etc.; **OTHER**=other

[2]**PU**=Public, **CO**=Confidential, only for members of the consortium (including the Commission Services), **CI**=Classified, as referred to in Commission Decision 2001/844/EC

Modifications Index

Date	Version
30/06/2025	1.1
18/07/2025	1.2
31/07/2025	**1.3**

This work is a part of the **BUILDCHAIN** project. **BUILDCHAIN** has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement no **101092052**.

ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
API	Application Programming Interface
BIM	Building Information Model
CLT	Cross-Laminated Timber
DNN	Deap Neural Network
DBL	Digital Building Logbook
EC	Eurocode
EC8	Eurocode 8 (Earthquake-resistant design)
FE Model	Finite Element Model
GBT	Gradient Boosted decision Tree
GPC	Generalized Polynomial Chaos
IFC	Industry Foundation Classes
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NMAE	Normalized MAE
NRMSE	Normalized RMSE
PCE	Polynomial Chaos Expansion
R^2	Coefficient of Determination
RMSE	Root Mean Square Error
SHAP	SHapley Additive exPlanations
SHM	Structural Health Monitoring
UI	User Interface
UQ	Uncertainty Quantification

Executive Summary

This document presents a comprehensive overview of a multi-module software framework designed to support advanced modeling, analysis, and decision-making processes within the BUILDCHAIN project. The framework integrates data-driven modeling, hybrid digital twinning, multi-building design updating, and resilience assessment modules addressing earthquake and climate risks. Each module is detailed through problem statements, tool overviews, core functionalities, methodologies, user workflows, and alignment with BUILDCHAIN goals. Demonstrations on toy models and real pilot cases illustrate the practical applications and effectiveness of the tools. The document further provides step-by-step user guides for library packages and the `TRACE-Structures` user interface to facilitate adoption by diverse stakeholders. Strengths, limitations, and technical architectures are discussed to guide future development and deployment.

Contributions

The development of the herein documented python library package and the TRACE-STRUCTURES API and user interface as well as this document is a result of a highly multidisciplinary, collaborative work of the BUILDCHAIN consortium (SZTAKI, UNIPI, UL, UGR, ZAG, RINA-C).

Contributions and developers:

- **Leed, coordinating project and development**: Noémi Friedman (SZTAKI) - main leed, Bence Popovics (SZTAKI), Filippo Landi (UNIPI) - assistant leeds, Jairis Arleen Alvarez Trujillo (RINA-C)- BUILDCHAIN WP4 coordination
- **Front-end development**: Áron Friedman (SZTAKI) -developer, Bence Popovics - testing
- **Conceptualization**: Noémi Friedman (SZTAKI), Blaž Kurent (UL), Boštjan Brank (UL), Juan Chiachio Ruano (UGR), Filippo Landi (UNIPI), Uroš Bohinc (ZAG)
- **Hybrid Digital Twinning back-end**: Bence Popovics (SZTAKI), András Urbanics (SZTAKI), Emese Vastag (SZTAKI), Elmar Zander (external), Noémi Friedman (SZTAKI), Blaž Kurent (UL)
- **Data-driven Modelling backend**: András Urbanics (SZTAKI), Emese Vastag (SZTAKI), Bence Popovics (SZTAKI)
- **Multi-Building Design Updating backend**: Bence Popovics (SZTAKI), András Urbanics (SZTAKI), Noémi Friedman (SZTAKI), Blaž Kurent (UL), Boštjan Brank (UL)
- **Assessment of Earthquake Resilience backend**: Filippo Landi (UNIPI) – push-over analysis algorithmic development, MATLAB code development, Bence Popovics (SZTAKI) – translation of code from MATLAB to Python, code structing, efficiency, Noémi Friedman (SZTAKI), Filippo Landi, Giada Bertolini – streamlining building properties from IFC
- **Analyzing Climate Resilience**: Constanza Guarducci(UNIPI), Giulia Pierotti (UNIPI) - main developers, Filippo Landi (UNIPI) - coordination, Bence Popovics (SZTAKI), Áron Friedman (SZTAKI) - translation from MATLAB to Python, refactoring
- **Ontology design, knowledge assets, integration with BUILDCHAIN API**: Stevan Nesovic (TRACE), Áron Friedman (SZTAKI), Bence Popovics (SZTAKI), Juan Chiahio Ruano (UGR), Emilio Daroch (UGR), Filippo Landi (UNIPI)
- **Piloting SHM based on optimized 'self-attentive' sensing IoT systems**: Juan Chiahio Ruano (UGR), Emilio Daroch (UGR)
- **Logo Design:** Hanna Schuchmann

The digital twinning Python functionalities used in Modules 1–3 (Data-Driven Modelling, Hybrid Modelling, and Multi-Building Updating), such as the GPC codes and other foundational structures for defining parameter sets, were developed using the MATLAB library package SGLIB (https://github.com/ezander/sglib) as a basis. The earlier development of this package was coordinated by Elmaz Zander at TU Braunschweig. We would like to acknowledge this valuable contribution.

Contents

Chapter 1

Introduction

Digital twin technology has recently emerged as a transformative approach to building management, enhancing transparency, sustainability, safety, health, and occupant comfort. This document outlines how the European HORIZON project **BUILDCHAIN** leverages this technology through an open-source library package and a web-based user interface that enable advanced digital twinning capabilities. The platform is also built on **Digital Building Logbook (DBL)** technology, allowing it to retrieve trusted building data—such as sensor measurements, simulation models, and metadata—from the DBL system. These data inputs are processed using advanced computational methods and machine learning algorithms to generate new knowledge assets, including digital twin models, automated alerts, and resilience or sustainability metrics. These outputs can then be written back into the DBL, ensuring that building data remains enriched, current, and actionable throughout the building's lifecycle.

The BUILDCHAIN platform integrates diverse data sources—including Building Information Models (BIM), Finite Element (FE) models, and real-time sensor data—through an API documented herein, the `TRACE-Structures` API that leverages these inputs to construct dynamic, real-time virtual replicas of physical buildings. By combining physics-based simulation models with measured data, these digital twins offer a continuously updated and comprehensive view of building performance. This integration enables real-world data to calibrate and refine FE models and assess resilience metrics more accurately. Additionally, the platform supports continuous improvement of design procedures and helps bridge the gap between as-designed and as-built performance by utilizing measurements from multiple, similar buildings.

We demonstrate the practical application of this technology through several BUILDCHAIN Use Cases and real-world pilots, as part of the Digital Logbook platform. This tool is capable of isolating environmental influences, updating FE simulation models based on measured responses, and producing accurate digital replicas of buildings. Using updated material properties of the building, earthquake resilience of the building structure can be evaluated and climate effects can be analyzed.

Furthermore, we also demonstrate on real-life problem how integrating data from multiple buildings of similar structure can support the refinement and re-evaluation of existing design standards, offering a new paradigm for performance-based design validation. As a core component of the BUILDCHAIN platform, the `TRACE-Structures` API enhances structural transparency and resilience by enabling advanced monitoring and data-driven decision-making. It also contributes to innovation in design practices through continual feedback from in-use structures.

Altogether, the BUILDCHAIN project lays the groundwork for more resilient, efficient, and sustainable building methodologies through the integration of cutting-edge digital twin technologies.

The structure of this document is as follows. It begins with introductory sections covering the background, objectives (Chapter 2), scope, intended users (Chapter 3), and development context aligned with the BUILDCHAIN goals (Chapter 4). It then details the general functionalities (Chapter 5), the system architecture and interface (Chapter 6) and the system requirements, setup and configuration (Chapter 7) before presenting five major modules. (Chapter 8) describes Module 1, which focuses on ***Data-Driven Modeling***, describing the problem addressed, tool overview, methodology, workflows, demos, and user guides, followed by testing results and an assessment of strengths and limitations. Modules 2 to 5 follow a similar format, covering ***Hybrid Digital Twinning*** (Chapter 9), ***Multi-Building Design Updating*** (Chapter 10), ***Earthquake Resilience Assessment*** (Chapter 11), and ***Climate Resilience Analysis*** (Chapter 12) respectively. Each module section includes problem statements, tool functionalities, conceptual approaches, user procedures, demos aligned with BUILDCHAIN goals, step-by-step guides for library and API usage, and detailed results of testing. A dedicated chapter covers Troubleshooting and FAQs(Chapter 13), Updates and Maintenance (Chapter 14), Licensing, and the potential for Replication and Scaling (Chapter 15), along with a section discussing Security, Privacy, and Performance Considerations (Chapter 16). The document concludes with a discussion on strengths and limitations, key lessons learned, and a summary aimed at informing users and guiding future development (Chapter 17).

Chapter 2

Background and Objectives

This section outlines the background and objectives of developing the `TRACE-Structures` open-source library packages and the accompanying API and user interface created under Task T4.5 of the BUILDCHAIN project. The primary goal was to develop a digital twinning framework capable of modeling and tracking the behavior of building structures under varying environmental conditions and aging effects—enabling quasi real-time structural health monitoring (SHM), follow-up procedures of design methods, and computations for improved resilience of building structures.

The core of this framework is an library package that ingests SHM data and generates real-time digital twins of buildings. Initially, the API was designed to support the development of two distinct types of digital twin models using Machine Learning (ML):

- ***Data-Driven Modelling:*** This module implements a purely data-driven approach using Explainable Artificial Intelligence (XAI). It analyzes SHM data to identify how variations in environmental or loading conditions (e.g., humidity, temperature) affect structural behavior. The system is capable of subtracting these environmental influences from the measured data to isolate genuine structural changes—such as shifts in dynamic properties (e.g., eigenfrequencies, modal shapes). It also generates alerts in the presence of anomalies or potential damage for early warning, enhanced seismic resilience, and facilitates SHM data interpretability. This module is in detail described in Chapter 8.

- ***Hybrid Digital Twinning:*** This module combines ML with physics-based FE modeling. It employs a non-intrusive approach to create a surrogate model that approximates the FE simulation. By comparing real structural responses with simulation outputs, it updates uncertain input parameters—such as material properties or boundary conditions—using Bayesian inference. This hybrid model can trace evolving structural behavior throughout the building's lifecycle and improve the accuracy of simulation-based predictions. It also supports re-evaluation of design assumptions based on actual performance data. This module is in detail described in Chapter 9.

Together, these two modules form the backbone of the BUILDCHAIN `TRACE-Structures` API. Over the course of the project, the API was expanded to include three additional modules:

- ***Multi-Building Design Updating:*** This module extends the hybrid digital twin approach to a network of models and buildings. It introduces a joint Bayesian updating framework that identifies shared parameters across multiple digital twin models—enabling parameter estimation using combined datasets. By linking surrogate models of different buildings (or different experiments), it refines common input parameters (e.g., timber material behavior) and improves consistency in design calibration based on as-built performance. This module of the TRACE-STRUCTURES API is described in detail in Chapter 10 and, together with the first two base modules, is integrated into a unified, open-source python library package: digital-twinning.

- ***Assessment of Earthquake Resilience:*** This module analyzes the earthquake resilience of a building by computing the vulnerability factors of both individual walls and the entire structure based on the EPUSH routines. It supports 2D and 3D visualization of structural components—such as load-bearing walls. Both global and local vulnerability metrics are computed through linear and nonlinear static analyses. Results are visualized via color-coded layouts and Acceleration Displacement Response Spectra (ADRS) diagrams, providing actionable insights for seismic design and retrofit strategies. This module is described in detail in Chapter 11, and its backend is built on the open-source library package earthquake-resilience.

- ***Analyzing Climate Resilience*** This module assesses a building's current and future exposure to climate hazards. It integrates high-resolution climate projections from the Copernicus Climate Data Store, performs extreme value analyses, and quantifies potential risks under different future scenarios. This capability supports the development of Climate Vulnerability and Risk Assessments (CVRA), informing long-term adaptation strategies for building designs. This module is described in detail Chapter 12, and its backend is build on the open-source library package climate-resilience.

Chapter 3

Scope and Intended Users

3.1 Scope

The `TRACE-Structures` library packages, along with their associated user interface developed within the BUILDCHAIN project, are designed to support advanced modeling, analysis, and monitoring of building structures. The platform enables performance-based design validation, structural health monitoring, and resilience assessment through data-driven and hybrid simulation techniques. It also supports both simplified earthquake resilience analysis, as well as the evaluation of future climate impacts on buildings.

This toolset supports a broad range of use cases—from capturing real-time structural behavior by subtracting environmental influences, to updating Finite Element (FE) models with measured data forming a digital twin of the building structures. It enables the evaluation of seismic resilience and future climate effects and allows for the refinement of design procedures through multi-building calibration of design modeling parameters based on observed structural performance. The models developed with the toolset can be embedded as modular elements into Structural Health Monitoring (SHM) data streams, and is compatible with an external DBL framework, the BUILDCHAIN platform. This flexibility supports both single-building applications and large-scale, fleet-level assessments, promoting scalable and data-informed decision-making in structural engineering.

By bridging the gap between domain-specific engineering knowledge and data-driven insights, the BUILDCHAIN platform empowers users to develop more resilient, sustainable, and transparent building practices. The system is designed to be extensible, allowing for future enhancements and integration with broader digital twin ecosystems in the construction and infrastructure sectors.

3.2 Intended Users

The modules and functionalities of the BUILDCHAIN platform have been developed to meet the needs of the following user groups:

- **Structural Engineers:** To support the design, monitoring, and ongoing assessment of building using real-world performance data to validate and refine design assumptions, to evaluate earthquake resilience and to analyse future climate effects.

- **Standardization Authorities and Regulatory Bodies:** To evaluate and revise building codes, design standards, and safety protocols based on performance insights derived from digital twins and real-use data across multiple structures.

- **Researchers in Civil and Structural Engineering:** To explore new methodologies for digital twinning, hybrid modeling, uncertainty quantification, and data-driven structural analysis.
- **Data Scientists and Machine Learning Practitioners:** To apply advanced analytics and machine learning models to structural health monitoring data, contribute to the development of explainable AI models, and support predictive maintenance or anomaly detection frameworks in the built environment.

Chapter 4

BUILDCHAIN Goals Alignment, Development Context

4.1 BUILDCHAIN Goals Alignment

The web-based BUILDCHAIN `TRACE-Structures` API hosted on the SZTAKI server (https://buildchain.ilab.sztaki.hu/) and the accompanying open source Python library packages were developed within the framework of the European HORIZON BUILDCHAIN project (https://buildchain-project.eu/), and as a core element of the BUILDCHAIN platform. The API and python library package gives the BUILDCHAIN users access to a set of tools which facilitates the services of the following BUILDCHAIN use cases.

- **Structural Health Monitoring, reliability assessment and anomaly detection:** The use case (BUILDCHAIN *Use Case U3*) focuses on leveraging continuous sensor data on structural behavior – such as measurements of strains, displacements, accelerations – to detect anomalies and assess structural reliability effectively. It is primary supported by Module 1 - ***Data-Driven Modeling***, which enables the isolation of changes in building behavior from environmental effects and flags measurements that exceed defined thresholds. This use case is tested on both a toy model based on synthetic data and on a pilot building (BUILDCHAIN *Pilot 1* from *Taks T5.1*) – the Granada Hospital Real building in Granada. This module can serve as a preparatory step for Module 2 and subsequent use cases by producing corrected data that is more suitable for the digital twinning process than the original sensor data, which may be influenced not only by structural changes but also by environmental effects.

- **Earthquake and climate proof of building based on digital twinning:** The objective of the use case (BUILDCHAIN *Use Case U4*) is to improve the earthquake and climate resilience of buildings, which can be achieved by assessing the vulnerability and risk related to natural hazards, facing the assets and then by planning intervention strategies based on available financial resources. This objective is supoprted by two functionalities of the `TRACE-Structures` API i) the evaluation of the impact of changing climate conditions on structures, with projections extending to the year 2100, supported by Module 5 – ***Climate Resilience***; ii) facilitating streamlined seismic vulnerability assessment based on DBL data, and enhancement of transparency of building stock resilience through comparable results derived from homogenized and simplified methods, enabled by Module 4 – ***Earthquake Resilience*** and the BUILDCHAIN DBL system. These methods are designed for rapid resilience evaluation, assisting municipalities in prioritizing renovation works.

The approach is validated on a benchmark toy model for earthquake resilience, as well as on real-life pilot buildings (BUILDCHAIN Pilots 3 and 4 under *Tasks T5.3–4*)—a strategic building and several school buildings owned by the Municipality of Florence.

Optionally, this use case can be combined with the preceding and subsequent use cases: if structural monitoring data is available, it can first be corrected using Module 1 – ***Data-Driven Modeling***, and the updated data can then be used to refine material properties using Module 2 - ***Hybrid Digital Twinning*** before computing resilience, and reliability metrics.

- **SHM tools for cultural heritage:** The use case (BUILDCHAIN *Use Case U5*) aims to sequentially update the structural properties of buildings using sensor data to enhance the assessment of structural integrity —- particularly for cultural heritage buildings —- ensuring their preservation and long-term sustainability. This goal is supported by Module 2 – ***Hybrid Digital Twinning***, which facilitates the creation of hybrid models by approximating physics-based representations of the structure and updating them with data from structural health monitoring systems. Through this process, key building properties (e.g., material characteristics) can be refined, enabling more accurate reliability assessments.

  The use case is demonstrated through both a simplified toy model and a real-life scenario, the Hospital Real cultural heritage building in Granada (BUILDCHAIN Pilot 1).

  Ideally, this use case is preceded by the first use case described above, which isolates environmental effects from structural behavior to provide corrected data for digital twinning.

  The digital-twinning library package provides modular tool components for assembling a complete smart structural health monitoring system. It leverages an IoT sensor network that captures both weather and building behavior data, and utilizes hybrid digital twinning algorithms to assess the state of the building.

- **Follow-up procedures of design processes by Bayesian updating:** The use case (BUILDCHAIN *Use Case U6*) focuses on following-up design procedures by integrating design models with performance measurements of the as-built structures via Bayesian inference. Structural design parameters are refined using measured performance data from completed buildings, enabling improved calibration and future-proofing of design assumptions. This is enabled by Module 2 – ***Hybrid Digital Twinning*** and Module 3 – ***Multi-Building Updating***, which together allow for cross-referenced parameter estimation across multiple buildings. The approach is tested on updating of design parameters of tall timber buildings made of cross laminated timber(CLT) panels using measured data of modal properties of two such buildings (BUILDCHAIN *Pilot 5*).

- **Precipitation monitoring for active control of local flooding:** The use case (BUILDCHAIN *Use Case U7*) explores the integration of real-time precipitation monitoring into resilience workflows for climate adaptation. The goal is to assess and anticipate local flood risks using climate-resilient digital twin models that incorporate environmental sensor data. Supported by Module 5 – ***Climate Resilience***, this use case is demonstrated on *Pilot 4*, showing how future climate scenarios can inform proactive adaptation strategies.

- **Optimized 'self-attentive' sensing:** This use case (BUILDCHAIN *Use Case U8*) aims to improve sensing efficiency through model-informed sensor placement and prioritization. By leveraging Module 2 – ***Hybrid Digital Twinning***, the system dynamically identifies the most informative measurements for updating digital twin models, reducing data redundancy while maintaining model accuracy. The method is tested on *Pilot 1*, the Hospital Real cultural heritage building in Granada.

4.2 Development Context

The development of the `TRACE-Structures` platform is motivated by the growing demand for more resilient, sustainable, and intelligent management of building infrastructures in the face of evolving environmental, structural, and regulatory challenges. The BUILDCHAIN project responds to this need by integrating cutting-edge methodologies in data-driven modeling, hybrid digital twinning, and uncertainty quantification into a unified software framework and making it interoperable with the highly structured BUILDCHAIN DBL data-base.

The `TRACE-Structures` platform is designed to be modular and extensible, supporting diverse use cases across different stages of a building's lifecycle—from design validation and real-time monitoring to risk assessment and long-term resilience planning. This is achieved by incorporating:

- **Physics-based models** that represent the structural behavior of buildings through finite element (FE) simulations and Eurocode-based design principles.
- **Data-driven methods** that extract patterns from sensor data and environmental measurements, enabling anomaly detection and predictive analysis.
- **Hybrid modeling techniques** that combine physics-based and data-driven approaches to continuously update digital twins with real-world measurements.
- **Bayesian inference frameworks** that allow for probabilistic updating of model parameters, supporting robust decision-making under uncertainty.
- **Streamlined computation of earthquake resilience** that enhances transparency in assessing the seismic vulnerability of the building stock through harmonized and simplified evaluation methods, suitable for large-scale screening and prioritization.
- **Climate projections and impact modeling** that enable long-term assessment of structural performance under changing climate conditions—such as temperature, humidity, and precipitation—supporting future-proof design and retrofitting strategies.
- **Ontology-based data interoperability** that enables the import and export of structured building-related data, ensuring semantic consistency and seamless integration with the BUILDCHAIN Digital Building Logbook (DBL), and other standardized data repositories.

To meet the objectives outlined above and aligned with the goals of the BUILDCHAIN project, the development focused on creating a scalable and flexible platform architecture. A key aim was to make advanced structural assessment functionalities accessible both for expert users and also to non-coding engineers, thereby broadening the platform's usability and impact.

The resulting platform is implemented as a suite of open-source Python packages, grouped into three methodological units (digital-twinning, earthquake-resilience, climate-resilience). These packages are exposed via a modular `TRACE-Structures` API, documented in this deliverable, which enables seamless integration with external services -— such as the BUILDCHAIN Digital Building Logbook (DBL) platform.

Development has been driven by real-world applications across multiple BUILDCHAIN pilots, including cultural heritage buildings, schools, hospitals, and municipal infrastructure. These pilots have played a central role in shaping the platform's functionality, generalizability, and interoperability, ensuring its applicability in both research and real-world deployment contexts.

The development process followed a highly multidisciplinary approach, involving close collaboration among structural modeling experts, structural health monitoring (SHM) researchers, and machine learning practitioners from across the BUILDCHAIN consortium (SZTAKI, UGR, UNIPI, UL, ZAG, RINA-C). Agile development practices—supported by version control (Git)—enabled iterative validation, integration, and refinement of functionalities, informed by early testing on actual pilot buildings.

The resulting tool and its documentation are products of this collaborative, cross-disciplinary effort. Ultimately, `TRACE-Structures` serves as a core technological enabler within the BUILDCHAIN ecosystem, providing tools for resilient digital building management through intelligent, adaptive, and evidence-based modeling.

Chapter 5

General functionalities

The `TRACE-Structures` API offers to address the previously outlined objectives in the given context by the the below listed key functionalities.

- **Exploratory Data Analysis (EDA):**
  - Computation of data statistics;
  - Correlation analysis;
  - Different visualization techniques for data analysis.
- **Generation of purely data-driven models, evaluation of what if scenarios, prediction**:
  - Predictive modeling of building structure behavior from external effects using long-term weather and structural health monitoring data (e.g. predicting measurable frequencies from humidity and temperature data).
  - Analysis of predictive models, explainability, transparency, robustness:
    - Uncertainty Quantification (UQ);
    - Global sensitivity analysis by Sobol and SHAP analyses;
    - Local sensitivity analysis by SHAPley additive values;
    - Analysis of what if scenarios.
  - Subtracting effects from input feature variations.
- **Generation of hybrid models**
  - Probabilistic description of uncertain modeling parameters, building structure properties;
  - Sampling from probabilistic description;
  - Description of measurement error model;
  - Surrogate modeling (with the help of the data-driven methods and generation of input-output data);
  - Bayesian inference/inversion/parameter updating.
- **Multi-model updating for joined hybrid models**
  - Joint parameter description linking input parameters of different surrogate models;

  - Integration of individual parametric models;
  - joint updating using the linked models and measurements of their model outputs.

- **Earthquake resilience analysis:**
  - Creation of structural models for masonry buildings;
  - Computation of wall safety factors for different failure criteria (linear structural analysis under seismic loads);
  - Evaluation of the global seismic vulnerability index for the building.

- **Climate resilience analysis:**
  - Automated extraction of climate data from Copernicus Data Store or BUILD-CHAIN DBL;
  - Extreme value analysis of climate data series and computation of characteristic values for structural design;
  - Evaluation of factors of change for different climate statistics depending on future scenarios.

- **Interoperability with the BUILDCHAIN DBL:**
  - Import of standardized building-structure-related data and ontologies;
  - Export of enriched knowledge assets—such as predictive or surrogate models, statistical summaries, corrected SHM data, resilience metrics, and alarms—using a standardized, semantic format compatible with the BUILDCHAIN DBL system based on the Decentralized Knowledge Graph (DKG).

Chapter 6

System Architecture

The `TRACE-Structures` system is a comprehensive, modular solution designed to facilitate advanced digital twinning and structural health monitoring within the BUILDCHAIN ecosystem. It combines flexible software components, user-friendly interfaces, and seamless data integration to empower engineers and researchers at various expertise levels. The system supports the full workflow—from data acquisition and preprocessing to modeling, uncertainty quantification, and decision support—while ensuring interoperability with standardized formats and external data sources. The key components of the system include:

- A set of **adaptable open-source Python library packages** designed to interface with the `TRACE-Structures` API, providing a flexible, scalable system, that can be integrated into a modular structural health monitoring system, and tailored to user's specific needs.

- A **user-friendly**, web-based `TRACE-Structures` UI that assists engineers without coding experience or deep theoretical knowledge by providing **intuitive access to advanced digital twinning functionalities**. The `TRACE-Structures` API is seamlessly integrated with the BUILDCHAIN API, enabling effortless import and export of data to and from the Digital Building Logbook (DBL).

- The API was built on the Tornado framework, which was implemented to support a **secure, device-independent, web-based user interface**. It operates using temporary storage only, without retaining any persistent data, thereby minimizing the risks associated with handling and securing sensitive information.

- Support for **data pre-processing**, wrapping FE models for **uncertainty quantification and Bayesian inference** in hybrid digital twinning workflows, integration of **explainable AI** pipelines for structural health monitoring (SHM), as well as tools for **seismic analysis conforming to Eurocode standards** and **efficient computation of climate effects**.

- Integration of **standardized data formats and BUILDCHAIN ontologies** for easy integration with external data sources and the BUILDCHAIN platform (e.g., BIM files in the format of IFC, climate data from Copernicus, weather data from different providers via BUILDCHAIN API).

The web API architecture was selected to ensure interoperability with the BUILDCHAIN Digital Logbook and to enable modular deployment across pilots and testbeds. The separation of backend logic (Python library) from the API layer facilitates reusability, scalability, and integration in both cloud and on-premise environments.

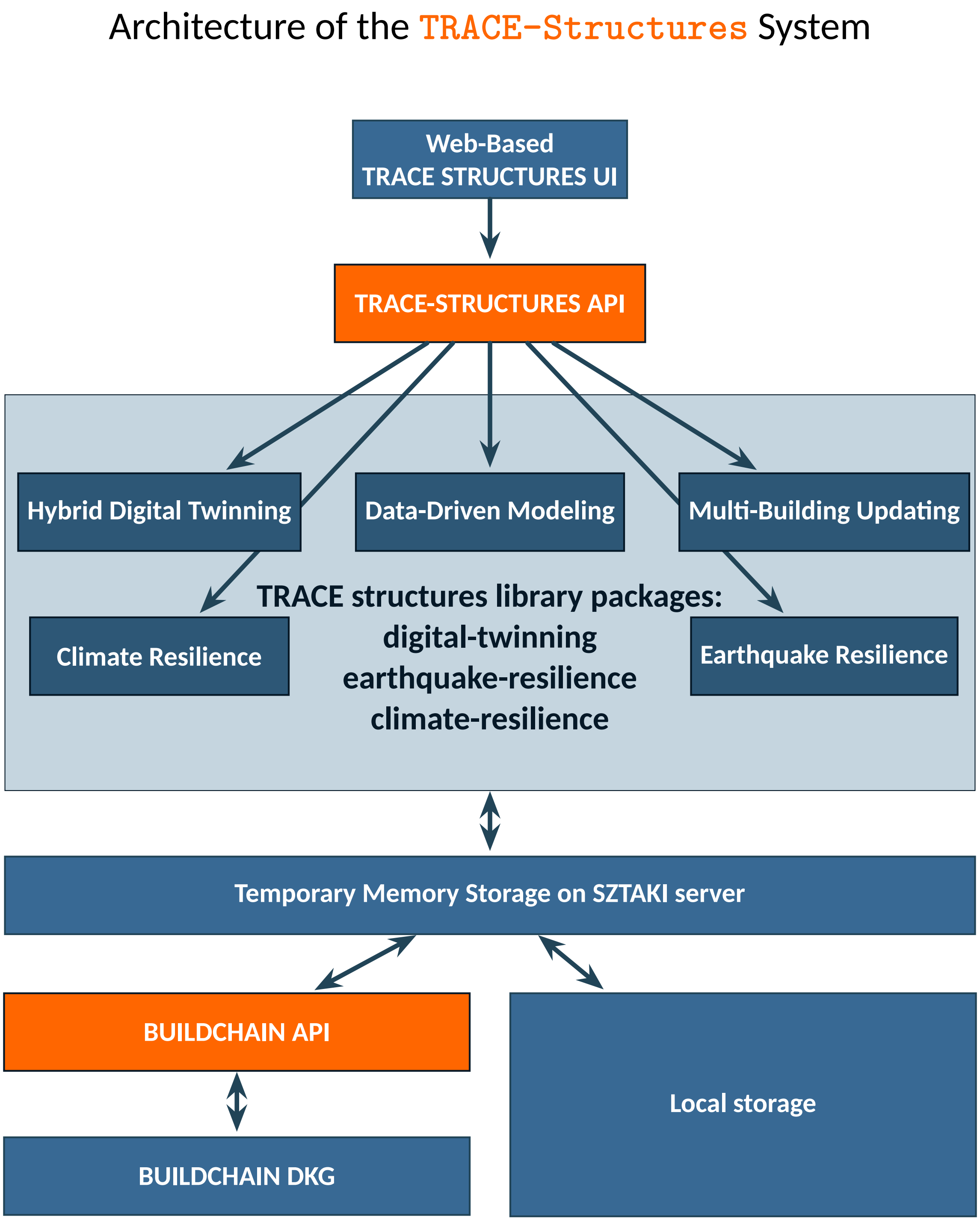


Figure 6.1: System architecture showing the Python modules and their connection with the REST API and BUILDCHAIN user interface.

As illustrated on Fig. 6.1, the resulting system architecture is divided into three conceptual layers:

- **Top Layer (User Interface)**: A web-based front-end linked directly from the BUILDCHAIN Digital Logbook. It provides users with access to analysis tools and visualization for structural and environmental data.
- **Middle Layer (Web API)**: Acts as the communication bridge between the front-end and the backend. It enables interaction with the Python classes that implement the digital twinning workflows.
- **Bottom Layer (Python Library):** Implements the core functionalities accessible through the API. The Python codebase is organized into three open-source library packages (digital-twinning, earthquake-resilience, climate-resilience), which are exposed via five main modules. These modules correspond to the key use cases supported by the tool.

The primary modules tailored to specific use cases include:

1. ***Data-Driven Modeling*** Module:
   - Learns from SHM data to characterize how environmental conditions (e.g., temperature, humidity) influence structural response.
   - Subtracts predictable environmental effects to isolate deviations from expected behavior, enabling anomaly detection and early warning.
2. ***Hybrid Digital Twinning*** Module:
   - Utilizes a parameterized physics-based simulation model to compute quantities of interest (QoIs), such as modal properties.
   - Assumes a priori uncertainty distributions over the input parameters (e.g., material, geometric, or modeling parameters).
   - Generates input-output samples which are processed through external simulation tools. These samples are used to train a surrogate model, replacing expensive simulations.
   - Uses Bayesian inference to update the uncertain input distributions based on the observed QoIs and the surrogate model, resulting in posterior parameter estimates and enhanced predictive capacity.
3. ***Multi-Building Design Updating*** Module:
   - Allows for joint parameter identification across multiple buildings sharing common design parameters.
   - Implements a Bayesian formulation to update design models using measurements from different structures simultaneously.
4. ***Climate Resilience Analysis*** Module:
   - Assesses current and future climate-related risks for a given location.
   - Uses climate projections and extreme value analysis to evaluate exposure and derive Climate Vulnerability Risk Analyses (CVRA).

5. ***Earthquake Resilience Analysis*** Module:

    - Computes seismic vulnerability of structures based on wall and floor geometry, material properties, and EPUSH routines.
    - Outputs local and global safety indicators, and visualizes performance on 2D/3D layouts and Acceleration-Displacement Response Spectra (ADRS) planes.

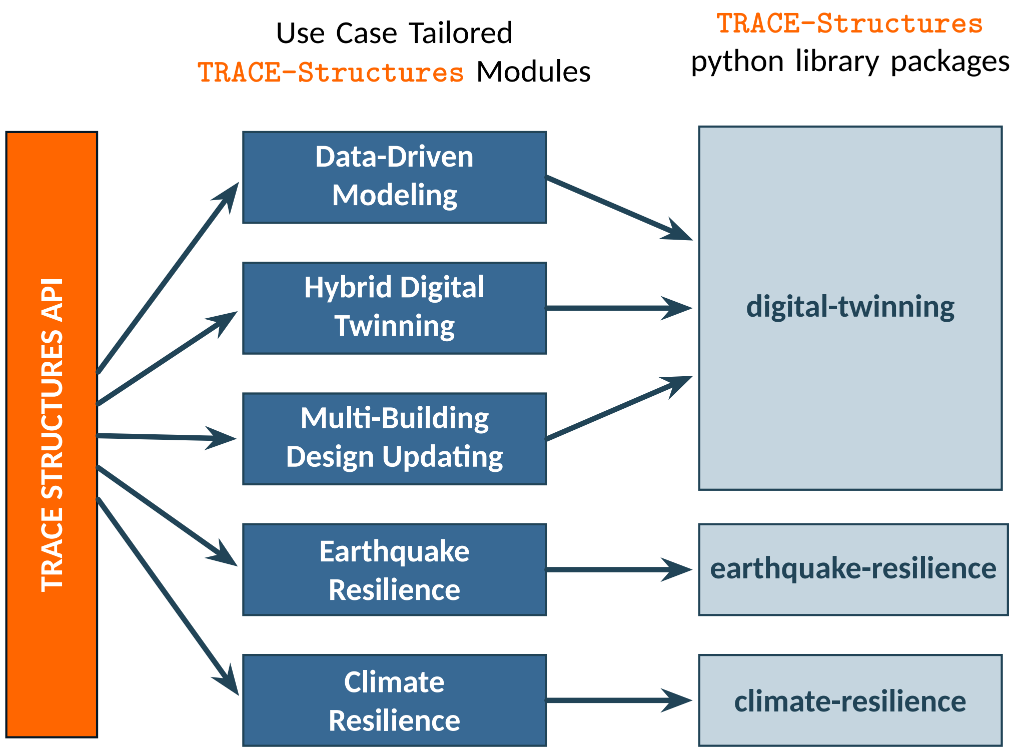


Figure 6.2: The functionalities of the `TRACE-Structures` API can be accessed via user case tailored modules (illustrated in the left hand side boxes) that are supported by three backend Python library packages (illustrated in the right hand side boxes).

The first three modules, ***Data-Driven Modeling***, ***Hybrid Digital Twinning***, and ***Multi-Building Updating*** are built on the digital-twinning python library package, while the ***Climate Resilience*** and ***Earthquake Resilience*** modules rely on two independent separate python library packages earthquake-resilience and climate-resilience. These mappings between the modules and their core functionalities are illustrated in Fig. 6.2. A more detailed description of the architecture of the supporting Python library packages is provided in the corresponding module sections and in Annex B, where each functionality—as well as the relevant classes—is presented and illustrated in detail.

The modules are designed to be used independently or in combination, enabling stacked workflows for more advanced use cases. For example, when updating design procedures, the process may begin by using the Data-Driven Modeling module to subtract environmental effects from the measured structural behavior. Subsequently, a surrogate model can be trained using the Hybrid Digital Twinning module. Finally, the Multi-Building Updating module can link multiple surrogate models from different structures to jointly update shared design parameters based on real-world measurements.

Similarly, in the case of earthquake resilience assessment, the analysis can begin by updating the material properties of load-bearing walls using the hybrid digital twinning procedure, thereby improving the accuracy of subsequent vulnerability and risk evaluations.

Chapter 7

System Requirements, Setup

7.1 Requirements

7.1.1 Hardware Requirements

The TRACE STRUCTURES platform is primarily accessed through a web-based user interface connected to backend services hosted on the SZTAKI server infrastructure. As such, general users do not require any specific local hardware to run computations or access functionalities.

For advanced users or developers who wish to work directly with the underlying Python libraries, computations can be performed either on a personal machine or using cloud-based environments such as Google Colab, JupyterHub, or other remote servers. In these cases, hardware requirements depend on the scale and complexity of the analysis. No dedicated hardware is required to use the system in its standard configuration.

7.1.2 Software Dependencies, Interoperability

The solution consists of three dedicated Python libraries supporting five modules, and an integrated API and web-based User Interface (UI) that builds on top of them. The Python libraries can be cloned or downloaded from GitHub (see Section 7.2) and are compatible with Python version 3.9 or later. Furthermore, the libraries operate with additional core python packages, such as `numpy`, `pandas`, `scikit-learn`, `matplotlib`, `plotly`, `dash`. The exact dependencies are detailed in Section B.1 of the Appendix and in the respective `requirements.txt` files included with each library package (see also the installation instructions in Section 7.2).

The system ensures interoperability with external services, such as the DKG, by supporting standard data exchange formats including `json`, `jsonld`, `unv`, `uff` and `csv`.

7.2 Installation Instructions

7.2.1 Using the Web-Based User Interface

The `TRACE-Structures` web-based interface is hosted on the SZTAKI server and requires no local installation. Users can access the full functionality of the platform directly through their web browser, without the need to download or configure any software locally.

7.2.2 Using the Python Library Packages

For users who prefer programmatic control or want to develop custom workflows, the `TRACE-Structures` functionalities are also available through open-source Python libraries. These can be cloned or downloaded from the official BUILDCHAIN GitHub repositories. Instructions for downloading and setting up the underlying libraries are provided in the following subsections.

1. Clone or Download the Repositories

The TRACE library packages are available at:

```
https://github.com/trace-structures/digital-twinning
https://github.com/trace-structures/climate-resilience
https://github.com/trace-structures/earthquake-resilience
```

You can clone e.g. the digital-twinning repository using

```
git clone https://github.com/trace-structures/digital-twinning
```

command in a terminal.

2. Set Up a Python Environment (Optional but Recommended)

It is recommended to use a virtual environment to manage dependencies:

```
python -m venv venv
source venv/bin/activate    # On Windows: venv\Scripts\activate
```

3. Install Dependencies

Each library contains a `requirements.txt` file specifying the required packages. Install them using:

```
pip install -r requirements.txt
```

4. Import the Library in Your Python Project

Once installed, you can import TRACE library components in your Python code like this:

```
from digital-twinning.surrogate_model import SurrogateModel
from digital-twinning.digital_twin import DigitalTwin
```

5. Run in Cloud Environments (Optional)

Alternatively, you can use the libraries in cloud-based environments such as Google Colab or JupyterHub by cloning the repositories by issuing the command

```
!git clone https://github.com/trace-structures/digital-twinning
```

and installing dependencies in the notebook environment.

7.3 Connecting to External Services

The `TRACE-Structures` system is designed to interact with the external service of the BUILDCHAIN Digital Building Logbook (DBL), to enable secure access to building-related data and metadata and streamlined computations.

7.3.1 Accessing DBL Data via the Web Interface

Users accessing the `TRACE-Structures` web-based interface can connect to the DBL system to pull or register building data, provided they have valid credentials (authentication token). Once authenticated, users can:

- Import trusted building data, such as sensor measurements, simulation models, IFC files.
- Register new knowledge assets generated with the `TRACE-Structures` modules through analysis, including digital twin models, performance metrics, and report on resilience indicators.

These interactions are handled seamlessly within the user interface, with no additional setup required beyond credential input (see later in the Module descriptions of this document).

7.3.2 Programmatic Access via the Python Library

For developers or researchers using only the Python library packages outside of the UI, direct integration with the DBL system is not provided out-of-the-box. In these cases, users must refer to the **BUILDCHAIN API User Manual**, which provides comprehensive guidance on how to:

- Authenticate against the BUILDCHAIN API.
- Access or register DBL-compliant data through RESTful endpoints.
- Programmatically interface with external services using appropriate credentials and API keys.

This approach enables advanced users to develop customized workflows and automation pipelines while ensuring secure and standardized data exchange with external platforms.

Chapter 8

Module 1: Data-Driven Modeling

8.1 Problem Statement and Needs Addressed

Dynamic properties of a building –such as its eigenfrequencies and modal shapes are constantly affected by environmental factors such as temperature and humidity. Structural Health Monitoring (SHM) systems such as acceleration sensors processed by Operation Modal Analysis (OMA) can capture such changes of building behavior, but SHM systems are meant to capture changes of the structural properties which can be hindered due to the environmental noise that can mask early warning signs of degradation or damage. There is a critical need for tools that can intelligently separate environmental effects from structural changes to enhance the interpretability of SHM data. This module addresses the need to detect and explain changes in building behavior by isolating environmental influences, enabling proactive maintenance, more accurate assessments of structural health, and improved safety.

8.2 Tool Overview, Purpose, Core Functionality

The ***Data-Driven Modeling*** module is designed to provide insight into the dynamic behavior of buildings by analyzing SHM data and subtracting the effects of environmental variability. Its core purpose is to detect abnormal structural behavior by learning normal patterns associated with environmental changes and flagging deviations. The module uses machine learning models to map environmental parameters to modal properties such as eigenfrequencies of the building. It integrates with the BUILDCHAIN Decentralized Knowledge Graph (DKG) based DBL, allowing weather data and building measurements to be pulled from the BUILDCHAIN DBL. The resulting average expected building behavior—after filtering out environmental effects— with its confidence regions can be registered in the DKG, along with the trained model, enabling future monitoring data to be analyzed consistently. Additionally, the module supports the generation of an alert knowledge asset, which can be registered in the DKG to signal deviations from expected behavior, along with their severity.

8.3 Key Features and Capabilities

The module uses machine learning models to map environmental parameters to modal properties (e.g., eigenfrequencies, mode shapes), enabling the user to:

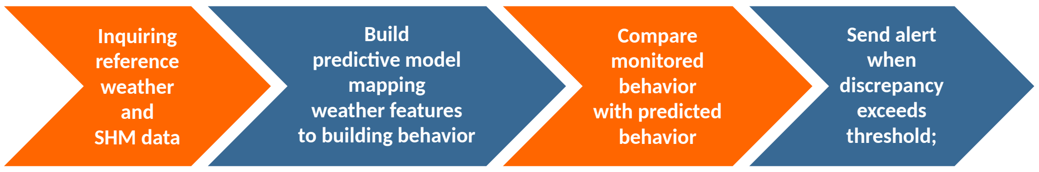

Inquiring reference weather and SHM data
Build predictive model mapping weather features to building behavior
Compare monitored behavior with predicted behavior
Send alert when discrepancy exceeds threshold;

Figure 8.1: Methodology, conceptual approach of using data-driven model in structural health monitoring

- Explain observed variations in modal properties and attribute it to the different environmental features (e.g. temperature, humidity) with the help of SHAP values (local feature attributions) and Sobol sensitivities (global feature importances).
- Subtract predicted environmental influences to reveal underlying structural changes.
- Provide alerts when deviations exceed expected patterns with some given threshold.
- Integrate with the BUILDCHAIN Decentralized Knowledge Graph (DKG) to import measurement data and to register trained models, expected structural behavior, and alert knowledge assets—enabling traceable, consistent analysis and automated monitoring over time.

8.4 Methodology, Conceptual Approach

This module is based on a supervised machine learning approach. Environmental parameters such as temperature, humidity are used as input features, while the dynamic properties of the structure (e.g., eigenfrequencies) serve as output targets. A regression model is trained to predict these dynamic properties under normal environmental variations.

8.4.1 Algorithms, Models, and Assumptions

The ML models implemented that can be used to relate input data (environmental conditions) to output data (modal properties) include:

- Linear Regression (LR) [1]
- Gradient Boosted Decision Trees (GBDT) [2]
- Generalized Polynomial Chaos Expansion (GPC) [3, 4]
- Deep Neural Networks (DNN) [5]

Explainability is achieved through:

- SHAP (SHapley Additive exPlanations) values for model-agnostic local and global interpretation (see [6] for detailed description).
- Sobol sensitivity indices (computed via sampling [7, 8] or GPC based techniques [9]) for quantifying the influence of input features on outputs.

Assumptions:

- Sufficient historical SHM data is available.

- Environmental factors are the primary source of variability in the monitored quantities.
- Structural changes are infrequent compared to environmental variations.

8.4.2 Calculations and Formulas

Data-Driven Models

Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ denote some specific realization of input features, e.g. some environmental inputs and $\mathbf{y} = [y_1, y_2, \ldots, y_d]^T$ of some quantity of interest (QoI), e.g. the modal response, a vector collecting all the frequencies and mode shapes. The regression models establish a functional relationship between the vector of input features $\mathbf{x}$ and output labels $\mathbf{y}$. The regression model estimates y by a function

$$\mathbf{y} = f(\mathbf{x}) + \varepsilon, \tag{8.1}$$

where f is an unknown function approximated by the model, $\hat{\mathbf{y}} = f(\mathbf{x})$, and ε is the observation noise. The API supports the following approximating models:

- **Linear Regression:** Assumes a linear relationship, i.e., $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$, where $\mathbf{w}$ is the weight vector and $\mathbf{b}$ is the intercept. For a more detailed description of the algorithms used, interested readers are referred to [1].

- **Gradient Boosted Decision Trees (GBDT):** An ensemble of decision trees optimized sequentially to minimize prediction error expressed by some chosen metric (e.g. mean squared error (MSE)), suitable for capturing nonlinear relationships. The approximation takes the form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \gamma_m h_m(\mathbf{x}), \tag{8.2}$$

  where h_m are regression trees and γ_m are weights or the learning rate. The tree model divides the space of input features $\mathbf{x}$ by hyperplanes and approximates the solution by a piecewise constant function where the pieces are bounded by these hyperplanes. The position of the hyperplanes are defined by minimizing the approximation error. While a single deep decision tree is prone to overfitting and may fail to generalize well, an ensemble of shallow trees—such as in Gradient Boosted Decision Trees—often achieves superior performance compared to many other regression models. For a more detailed discussion of the algorithms used, interested readers are referred to [2].

- **Generalized Polynomial Chaos (gPC) Expansion:** Represents $f(\mathbf{x})$ as a series of polynomials $\Psi_\alpha(\mathbf{x})$ that are orthogonal with respect to the underlying a-priori probability space of input features $\mathbf{X}$ (whose one realization is denoted by $\mathbf{x}$), expressed as random variables. This orthogonality of the polynomials are expressed by

$$\int_{I_{\mathbf{X}}} \Psi_\alpha(\mathbf{x}) \Psi_\beta(\mathbf{x}) \pi_{\mathbf{X}}(\mathbf{x}) = \delta_{\alpha,\beta}, \tag{8.3}$$

  where $I_{\mathbf{X}}$ is the support of the random variable $\mathbf{X}$, whose realizations are the input parameters $\mathbf{x}$, and $\pi_{\mathbf{X}}(\mathbf{x})$ is its a-priori distribution, estimated by the samples $\mathbf{x}_i$ of input features and finally $\delta_{\alpha,\beta}$ is the Kronecker delta (that takes value one when $\alpha = \beta$ otherwise its zero). The approximation then takes the form

$$f(\mathbf{x}) = \sum_{\alpha \in \mathcal{A}} c_\alpha \Psi_\alpha(\mathbf{x}), \tag{8.4}$$

where Ψ_α are the multivariate orthogonal polynomials indexed by multi-indices α, and c_α are coefficients to be trained using the input-output data. Further algorithmic details can be found in e.g. [3, 4].

- **Deep Neural Networks (DNN):** Multi-layer networks capable of learning complex, high-dimensional nonlinear functions through backpropagation and gradient-based optimization. The approximation takes the form

$$f(\mathbf{x}) = \sigma_L(\mathbf{W}_L \ldots \sigma_i(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \cdots + \mathbf{b}_L), \tag{8.5}$$

where $\mathbf{W}_i$, $\mathbf{b}_i$ are the matrix of layer weights and vector of biases, and σ_i are activation functions. Further algorithmic details can be found in [5].

Evaluation of the Predictive Models

The generated regression surrogate models are evaluated using different metrics to validate model accuracy. These metrics were the following:

- **Mean Squared Error (MSE)**

  The MSE measures the average of the squared differences between $i = 1, 2, \ldots, N$ samples of predicted values $\hat{\mathbf{y}}_i$ and ground truth values $\mathbf{y}_i$ across all samples and dimensions. The Mean Squared Error is defined as:

$$\text{MSE} = \frac{1}{N \cdot d} \sum_{i=1}^{N} \sum_{j=1}^{d} (y_{i,j} - \hat{y}_{i,j})^2 ,$$

  where i is the index of the sample and j is the index of the component of the response vector $\mathbf{y}$. Alternatively, using vector notation:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \frac{\|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2}{d}$$

- **Mean Absolute Error (MAE)** The MAE computes the average of absolute differences between predictions and true values, that is

$$\text{MAE} = \frac{1}{N \cdot d} \sum_{i=1}^{N} \sum_{j=1}^{d} |y_{i,j} - \hat{y}_{i,j}|$$

  . Alternatively, using vector notation:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} \frac{\|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_1}{d}$$

- **Root Mean Squared Error (RMSE)** The square root of MSE, often used to retain interpretability in the original units. For one component y of the QoI $\mathbf{y}$ the root mean squared error is expressed by

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

- **Normalized RMSE (NRMSE)**: The NRMSE is the RMSE normalized by the range of the target variable:

$$\text{NRMSE} = \frac{\text{RMSE}}{y_{\text{max}} - y_{\text{min}}},$$

where y_{min} and y_{max} are the minimum and maximum values of the samples of the y component.

- **Normalized MAE (NMAE)** The NMAE is the MAE normalized by the range of the target variable:

$$\text{NMAE} = \frac{\text{MAE}}{y_{\text{max}} - y_{\text{min}}}.$$

- **Relative RMSE (rel_RMSE)** The RMSE can be scaled by the standard deviation of the training labels

$$\text{rel\_RMSE} = \frac{\text{RMSE}}{\sigma_y} \quad \text{where} \quad \sigma_y = \text{std}(y_{\text{train}}),$$

which helps expressing how big part of the predictive model captures the variance. The ratio of captured variance can be computed from

$$R^2 = 1 - \text{rel\_RMSE}^2,$$

where 1 represents a perfect model and 0 or smaller values a model that does not learn anything.

- **Pearson Correlation Coefficient**: Evaluates the strength of the linear relationship between predictions and true values:

$$r = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}}$$

- **Spearman's Rank Correlation Coefficient** This correlation coefficient measures how well the relationship between variables can be described by a monotonic function

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \quad \text{where } d_i \text{ is the rank difference}$$

.

- **Kendall's Tau**: This metric measures ordinal association between variables based on concordant and discordant pairs

$$\tau = \frac{C - D}{\frac{1}{2}n(n-1)} \quad \text{where } C, D \text{ are counts of concordant and discordant pairs.}$$

- **Composite Metric (summed_metric)**: A custom aggregate score combining rank and linear correlations, penalized by relative RMSE:

$$\text{summed\_metric} = \frac{\tau + r + \rho - \text{rel\_RMSE}}{3}$$

Local and Global Sensitivity Analysis, Explainability

Explainability is achieved through:

- **SHAP (SHapley Additive exPlanations):** Provides local and global feature importance scores by quantifying each feature's contribution to individual predictions. For a more detailed discussion of the algorithms used, interested readers are referred to [6].
- **Sobol Sensitivity Indices:** Quantify the effect of each input variable and their interactions on the output variance, enabling global sensitivity analysis. Further algorithmic details can be found in [7, 9].

Computing feature distribution to achieve desired label

In this optional feature of the UI the user can select a desired value of labels with a given confidence level (given by the standard deviations the predicted value can deviate from the desired ones), and the UI will compute with Bayesian method the resulting distribution, of whose the samples gives a matching prediction to the desired one within the defined confidence interval. This functionality was in principal developed for the next, *Hybrid Digital Twinning* module for calibrating simulation models to give the measured outputs, but can become handy for this module as well. The reader is refered to Section 9.4.2 for the theoretical background and equations used for the Bayesian posterior. The only difference is that the input features are there called input parameters and the desired values of QoIs are there refered as measurements.

8.5 Procedures, Data Workflows

8.5.1 Procedures, User Workflow

This ***Data-Driven Modelling*** module of `TRACE-Structures` is designed to model the relationship between a building's modal properties and environmental conditions, with the goal of distinguishing structural changes from environmental effects. The approach assumes that, within a defined reference period (e.g., the current year), the structural properties of the building remain constant.

Using time series data of modal characteristics (frequencies and mode shapes) and synchronized weather data, a predictive model is trained to capture the influence of environmental factors on the structure's dynamic behavior. This model serves as a baseline that allows compensation for environmental variability in future measurements.

Once deployed within an IoT-based monitoring system, the model is used to continuously adjust new modal data by removing predicted environmental effects. This enables the detection of anomalies that are more likely to indicate actual structural changes rather than benign environmental variations.

The workflow of the process is illustrated in Figure 8.2, which shows the types of data that need to be ingested and the outputs produced by the various components of the process.

8.5.2 Data Flow, Data Formats

A more detailed overview of the data flow is provided in Fig. 8.3. The figure distinguishes between data that must be directly provided by the user and data that can be imported

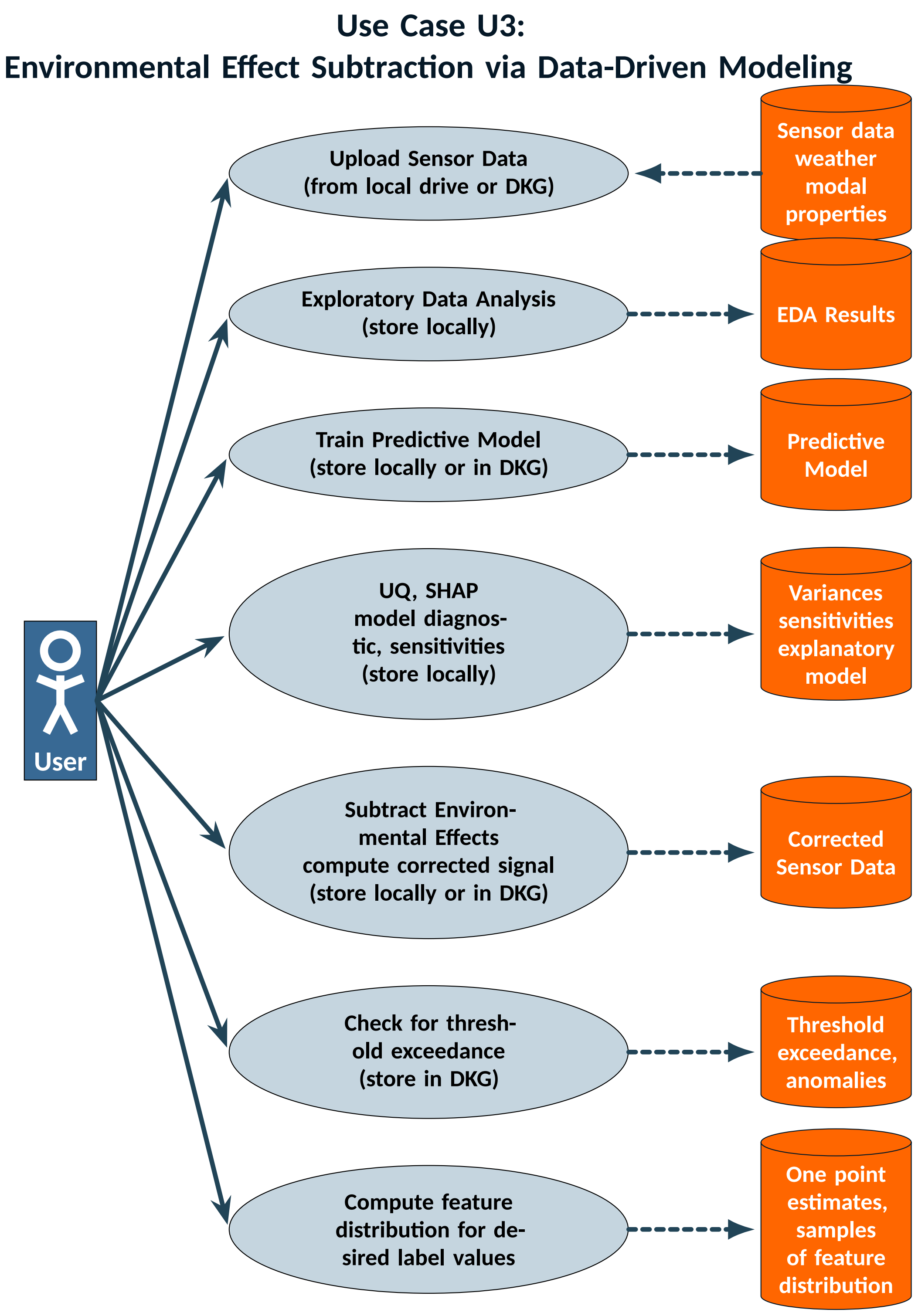

Use Case U3:
Environmental Effect Subtraction via Data-Driven Modeling
User
Upload Sensor Data (from local drive or DKG)
Exploratory Data Analysis (store locally)
Train Predictive Model (store locally or in DKG)
UQ, SHAP model diagnostic, sensitivities (store locally)
Subtract Environmental Effects compute corrected signal (store locally or in DKG)
Check for threshold exceedance (store in DKG)
Compute feature distribution for desired label values
Sensor data weather modal properties
EDA Results
Predictive Model
Variances sensitivities explanatory model
Corrected Sensor Data
Threshold exceedance, anomalies
One point estimates, samples of feature distribution

Figure 8.2: User interactions and data flows for environmental effect subtraction.

into the user interface from either the BUILDCHAIN DKG or other external platforms. It also separates the results of the analysis into two categories: outputs that assist the user in understanding the phenomena, and those that can be reused by other users—or by the same user at a later time—and are therefore worth registering in the DKG. Each of these components is described in detail below.

Input retrieved

- Reference period (training) and actual period (monitoring) time series of input data (e.g. weather data) imported from one of the following sources
  - local hardware (from `.csv` file)
  - DKG, Oracles via BUILDCHAIN API (`.json` or `.csv` file)
- Reference period (training) and actual period (monitoring) time series of output data (e.g. building response such as mode shapes and frequencies), imported from one of the following sources:
  - local harware (from `.csv`, or `.unv` file formats)
  - BUILDCHAIN DKG via BUILDCHAIN API (from `.json` or `.unv` file formats)

Other (internal) user inputs based on engineering/data science expertise:

- Selection of input features the model should use from the uploaded reference period input data for the prediction, selection of output target labels from the imported reference period output data (features such as modal properties that the model should predict, e.g. selection of modes, frequencies or mode shapes).
- Hyperparameters of the model, train-test split specifications.

Intermediate output:

- CSV files containing merged input-output data

Output, optionally stored in the DKG:

- Data-driven predictive model: Python pickle saved with the custom ".ddpm" extension and can be attached as a document to the building knowledge asset. In the future, a new type of knowledge asset of the metadata of the model could be created by generating an `.jsonld` file from the model main properties.
- The reference mode shapes and frequencies are averaged modal properties computed after subtracting environmental effects. These values are first exported to the universal `.unv` file format, and then by the BUILDCHAIN API to a `.json` file which becomes a "measurement knowledge asset".
- The same type of modal data can be computed from the monitored period modal data, which can be attached to the building knowledge asset the same way as the reference modal data.
- Alert information, signals of anomalies are stored in a json including threshold setting and severity level. The alert info can be also registered in the DKG.

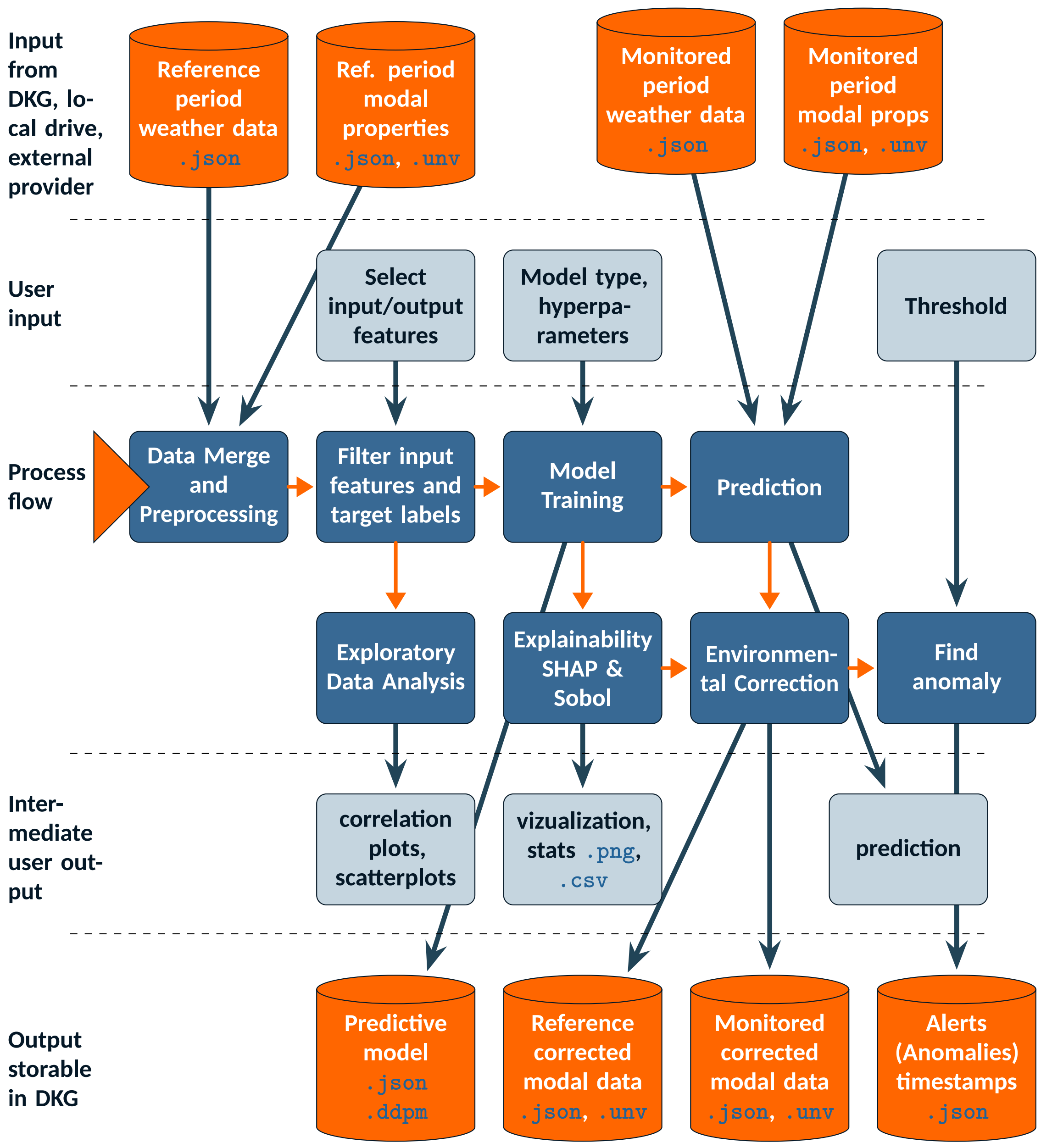

Input from DKG, local drive, external provider
Reference period weather data .json
Ref. period modal properties .json, .unv
Monitored period weather data .json
Monitored period modal props .json, .unv
User input
Select input/output features
Model type, hyperparameters
Threshold
Process flow
Data Merge and Preprocessing
Filter input features and target labels
Model Training
Prediction
Exploratory Data Analysis
Explainability SHAP & Sobol
Environmental Correction
Find anomaly
Intermediate user output
correlation plots, scatterplots
vizualization, stats .png, .csv
prediction
Output storable in DKG
Predictive model .json .ddpm
Reference corrected modal data .json, .unv
Monitored corrected modal data .json, .unv
Alerts (Anomalies) timestamps .json

Figure 8.3: Data flow of the Data-Driven module used for structural health monitoring. Modal and weather data can be imported from local drive or from DKG. Trained models can be stored locally or in the DKG. Averaged corrected data with subtracted environmental effects and alert info can also be uploaded to the DKG. All other result can be stored in local drive.

The module semantically aligns environmental and modal property terms with the BUILDCHAIN DKG ontology. This alignment supports structured querying and interaction through the DKG chat module, ensuring interoperability and traceable data integration. All data types and their integration to the BUILDCHAIN DKG are listed in Table 8.1.

8.6 Technical Architecture, Python classes

Dedicated Python classes with various functionalities were developed to support the digital twinning workflows of Modules 1–3. The architecture of the `TRACE` Python library package includes classes that cover the entire workflow—from data-driven modeling to analyzing feature sensitivities and attributions, as well as subtracting selected effects. These classes are illustrated through UML diagrams presented in Annex B, which highlight the key classes and their associated functionalities.

8.7 Demo, Alignment with BUILDCHAIN

This module was developed specifically for the BUILDCHAIN project's Use Cases U3 titled *Structural Health Monitoring (SHM), reliability assessment and anomaly detection with AI sensor fusion* and U5 titled *SHM tools for cultural heritage buildings*. However, both the library package and user interface are designed to be flexible and can be used to train any type of purely data-driven model—- not limited to predicting structural properties from weather data. The module has been tested and demonstrated using a toy model as well as a real-world application within the BUILDCHAIN pilot project: the *Use of DBL for proactive maintenance of cultural heritage* pilot, Pilot 1, which aims at the structural health monitoring the Hospital Real building in Granada.

8.7.1 Toy Model

The toy model is based on a simple analytical function

$$y = f(t, a, b, c) = a \sin(bt + c), \tag{8.6}$$

where y is the model output is the model output, and the input features are represented as $\mathbf{x} = [a, b, c]^T$. The objective is to approximate this analytical relationship by generating synthetic data and training a data-driven model that captures the dependency of $\mathbf{y}$ on the input features $\mathbf{x}$.

The time dependency was discretized by evaluating the function at a sequence of time steps $\mathbf{t} = [t_1, t_2, ..t_N]$ with $N = 9$. The task, therefore, becomes approximating the vector-valued function:

$$\mathbf{y} = \begin{bmatrix} y(t_1) \\ y(t_2) \\ \vdots \\ y(t_N) \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} a \sin(bt_1 + c) \\ a \sin(bt_2 + c) \\ \vdots \\ a \sin(bt_N + c) \end{bmatrix} = \begin{bmatrix} x_1 \sin(x_2 t_1 + x_3) \\ x_1 \sin(x_2 t_2 + x_3) \\ \vdots \\ x_1 \sin(x_2 t_N + x_3) \end{bmatrix} = f(\mathbf{x}). \tag{8.7}$$

To generate data for this experiment, predefined probability distributions were assigned to the input parameters a, b, and c. A set of samples $\{\mathbf{x}_i\}_{i=1}^{M}$ was drawn from these distributions, and for each sample, the corresponding function response $f(\mathbf{x}_i)$ was

Data type	Description	csv	unv	json	pickle: ddpm, xm	Integration	User Input	Stored in DKG	Indexed in DKG	Oracle	Additional Info
Input											
Reference weather data	Time series with timestamps	x		x		from DKG/local hardware			json	x	
Reference modal properties	Time series with timestamps	x	x	x		from DKG/local hardware			unv		
Monitored weather data	Time series with timestamps	x		x		from DKG/local hardware			json	x	
Monitored modal properties	Time series with timestamps	x	x	x		from DKG/local hardware			unv		
Threshold	Scalar value					no, given by direct user input	x				included in alert
Hyper parameter, model types	Scalar values, string					given by direct user input	x				included in data-driven model
Output											
Data-driven predictive model	Pickle file with own extension (`ddpm`)			x	x	to DKG/local hardware		json	ddpm		json with main modeling info
Explanatory model	Pickle file with own extension (`xm`)			x	x	to DKG/local hardware		json	xm		json linking explanatory model to predictive model
Reference modal properties	Reference period, one value of mode shapes and frequencies for selected modes	x	x			to DKG/local hardware		json	unv		modal properties of reference period averaging time series with subtracted environmental effects
Monitored modal properties	Time period, one value of mode shapes and frequencies for selected modes	x	x			to DKG/local hardware		json	unv		modal properties of monitored period averaging time series with subtracted environmental effects
Alert	With set threshold, statistics of severity and frequency of anomaly signs			x		to DKG/local hardware		json			

Table 8.1: Data types, formats, and integration paths for input/output in the DKG-based system used in the data-driven modeling module.

computed. To simulate observational noise, zero-mean Gaussian errors ε_i were added to each computed response, resulting in: $\mathbf{y}_i = f(\mathbf{x}_i) + \varepsilon_i$. The resulting input-output pairs $\mathbf{x}_i$ and $\mathbf{y}_i$ were saved in two separate texttttcsv files, which were then imported into the `TRACE-Structures` user interface.

After training a machine learning model (e.g., a gradient boosted decision tree regression model) with selected hyperparameters to approximate the sine function, a new set of input samples $\mathbf{x}_j$ and predictions $\mathbf{y}_j$ was generated. The effect of varying input parameters on the model output $\mathbf{y}$ is demonstrated both in the example Python notebooks (which use the underlying library package) and in the `TRACE-Structures` UI (shown in the accompanying video).

Furthermore, SHAP (SHapley Additive exPlanations) analysis is used to interpret the model by attributing output variation to individual input features. This enables, for example, isolating and subtracting the effect of the phase shift parameter c from the response $\mathbf{y}$.

8.7.2 SHM of the Hospital Real in Granada

The module was tested and demonstrated on the IoT-based Structural Health Monitoring (SHM) data —- specifically, on weather data as input features

$$\mathbf{x} = [T, H]^T, \tag{8.8}$$

with T the temperature and H the humidity, and time series of frequencies and mode shapes as quantity of interests

$$\mathbf{y} = [f_1, f_2, f_3, f_4, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \boldsymbol{\psi}_3, \boldsymbol{\psi}_4] \tag{8.9}$$

from the Hospital Real building. The two datasets, the one of the input features and the one of the output features were linked based on the timestamp indices of the data. Where dependencies on environmental conditions (e.g., temperature and humidity) were suspected, a data-driven model was developed to capture these effects. SHAP (SHapley Additive exPlanations) analysis was then used to attribute portions of the model's predictions to individual environmental variables, allowing their influence to be quantified and subtracted from the original data.

Testing on the real-world dataset showed that the building's natural frequencies were more significantly affected by environmental changes than the mode shapes. Once the model was trained, these environmental effects could be removed from the time series, enabling the calculation of a corrected reference signal by averaging the adjusted data over a selected time period.

8.8 Step by Step Library Package User Guide for the Module

A dedicated Python library package has been developed to support the full data-driven modeling workflow described in this report. The package provides functionality for importing and preprocessing time-series data, training predictive models, performing explainability analyses (e.g., SHAP, Sobol), and subtracting selected environmental effects to compute reference values. Its modular architecture is designed for flexibility, enabling both expert users and engineers to define inputs, select features, tune model hyperparameters, and analyze results.

To demonstrate the library's capabilities, two example Jupyter notebooks are provided for a step-by-step user guide which is annexed in Annex C.1:

Notebook 1 (see Annex C.1.1): A toy model example (explained in Section 8.7.1) that illustrates the complete functionality of the library using synthetic data generated from a known analytical function. This example helps to understand each step of the pipeline, including data generation, model training, explainability, and effect subtraction.

Notebook 2 (see Annex C.1.2): An application to real-world data from the Hospital Real building in Granada (see Section 8.7.2), part of the BUILDCHAIN Pilot 1. This use case showcases how the library can be applied to IoT-based structural health monitoring data, demonstrating how environmental influences such as temperature and humidity affect modal properties, and how these can be corrected using data-driven techniques. These notebooks serve as practical guides for both validating the approach and facilitating adoption in other use cases.

8.9 Step by Step API User Guide for the Module

This section presents in a step-by step manner how the data-driven module of the TRACE-Structures BUILDCHAIN UI can be applied. To carry out the full workflow of subtracting environmental effects using the API, users should follow the steps below:

1. **Access the BUILDCHAIN web interface:** Visit the TRACE-Structures interface at: `https://buildchain.ilab.sztaki.hu` and navigate to the TRACE-Structures API

2. **Module Selection:** Navigate to the ”Data-Driven Modeling” section by either clicking the blue `Data-Driven Modeling` button or selecting the corresponding item from the left-hand navigation menu.

3. **Getting Started:** If no model has been trained yet, select `I don't have a model`. If a model has already been trained and the goal is to evaluate sensitivities, view explainability results, make predictions, or subtract the effects of new input feature changes, select `I already have a model`. This skips the training and data upload steps.

4. **Upload an Existing Model (optional):** If you selected the `I already have a model` option, upload a previously trained predictive model in `.ddpm` format. You can then proceed directly to the analysis steps.

5. **Upload Features and Labels:** Upload the input data (e.g., weather parameters) and corresponding output data (e.g., modal properties) to be used for model training.

6. **Exploratory Data Analysis (EDA):** Explore the uploaded data to understand its structure, identify correlations, and detect possible dependencies between features.

7. **Data-Driven Modeling:** Select features to be used as input of the model from the uploaded input features and target labels from the uploaded target labels. Choose the model type and set the relevant hyperparameters to train the predictive model.

8. **Uncertainty Quantification and Sensitivity Analysis:** Evaluate predictions, variance of the output, residuals, and generated alerts using the visual dashboard.

9. **Remove External Effects, Detect Threshold Exceedance:** Investigate the influence of input features using SHAP and Sobol explainability analyses, and subtract their effects to derive corrected outputs either directly from the reference period output

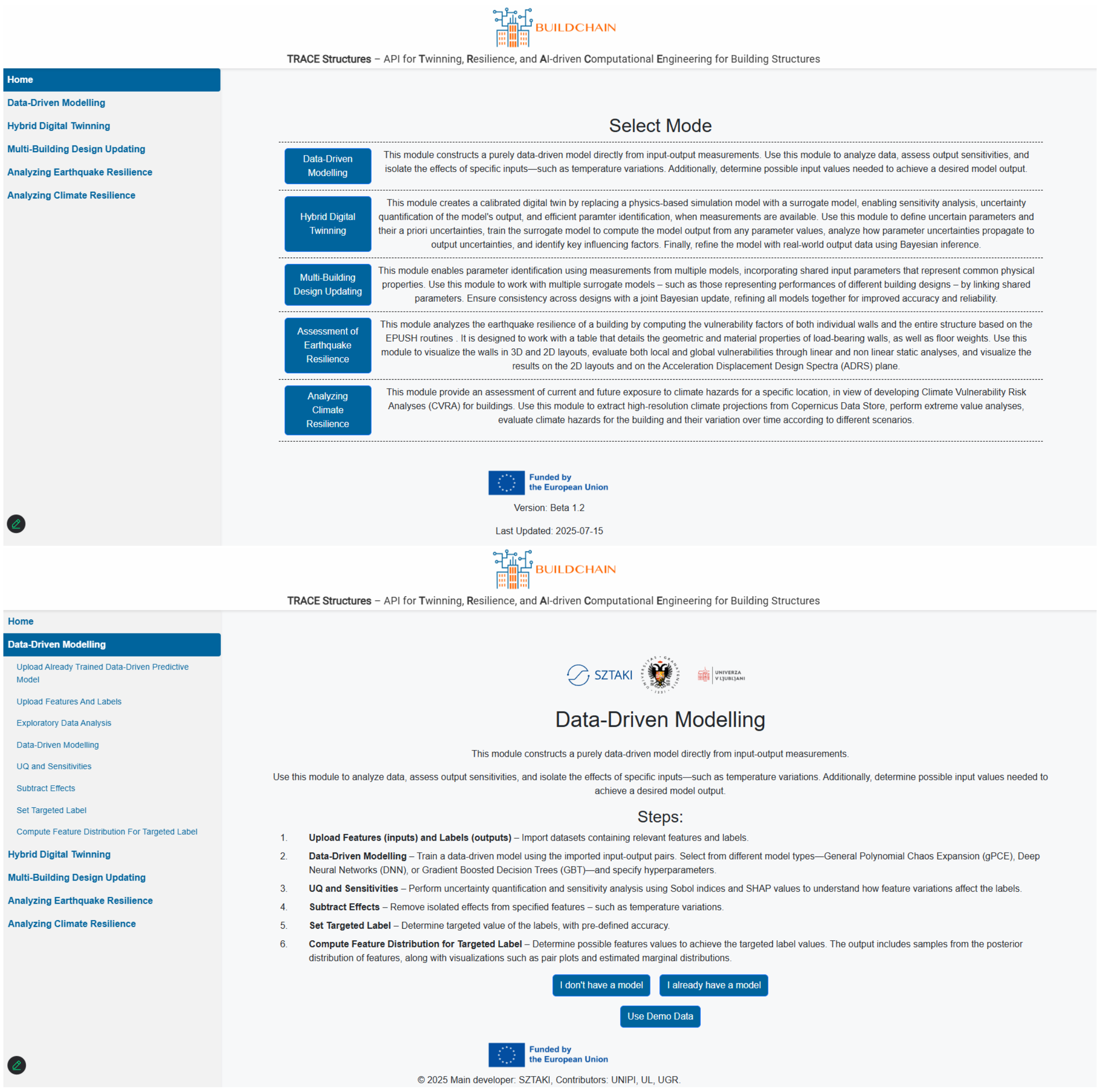
data or from newly uploaded input-output data taken from a monitoring period. Here the user can detect threshold exceedence indicating anomalies of the system.

10. **Set Target Label:** Optionally, you can analyze which environmental conditions would lead to a desired target label value. To do this, define the desired value of the target label along with the acceptable standard deviation around it.

11. **Compute Feature Distribution for Target Label:** Based on the specified target value and tolerance, compute the input feature distribution given as samples of the input features that would result in the target label falling within the defined range.

8.9.1 Step 1: Choosing the Module, Getting Started

Visit the TRACE-Structures interface at: `https://buildchain.ilab.sztaki.hu` and click on `Start app`. Navigate to the ”Data-Driven Modeling” section by either clicking the blue `Data-Driven Modeling` button or selecting the corresponding item from the left-hand navigation menu (see upper panel of Fig 8.4. On the first page of the module (see lower panel of Fig 8.4), if you are just getting started, and no model has been trained yet, click on the `I don't have a model` button and continue with Step 3. If a model has already been trained and saved, and the goal is only to evaluate sensitivities, view explainability results, make predictions from the trained model, or subtract the effects of new input feature changes, click on the `I already have a model` button. This way, you can import the already trained model and skip the training and data import steps (step 3 and 5). By clicking on `Use Demo Data`, you can load the built-in input–output dataset generated for the toy model described in Section 8.7.1. The screenshots provided in the following steps of the UI correspond to this toy model example.


BUILDCHAIN
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Funded by the European Union

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Select Mode

Data-Driven Modelling — This module constructs a purely data-driven model directly from input-output measurements. Use this module to analyze data, assess output sensitivities, and isolate the effects of specific inputs—such as temperature variations. Additionally, determine possible input values needed to achieve a desired model output.

Hybrid Digital Twinning — This module creates a calibrated digital twin by replacing a physics-based simulation model with a surrogate model, enabling sensitivity analysis, uncertainty quantification of the model's output, and efficient paramter identification, when measurements are available. Use this module to define uncertain parameters and their a priori uncertainties, train the surrogate model to compute the model output from any parameter values, analyze how parameter uncertainties propagate to output uncertainties, and identify key influencing factors. Finally, refine the model with real-world output data using Bayesian inference.

Multi-Building Design Updating — This module enables parameter identification using measurements from multiple models, incorporating shared input parameters that represent common physical properties. Use this module to work with multiple surrogate models – such as those representing performances of different building designs – by linking shared parameters. Ensure consistency across designs with a joint Bayesian update, refining all models together for improved accuracy and reliability.

Assessment of Earthquake Resilience — This module analyzes the earthquake resilience of a building by computing the vulnerability factors of both individual walls and the entire structure based on the EPUSH routines . It is designed to work with a table that details the geometric and material properties of load-bearing walls, as well as floor weights. Use this module to visualize the walls in 3D and 2D layouts, evaluate both local and global vulnerabilities through linear and non linear static analyses, and visualize the results on the 2D layouts and on the Acceleration Displacement Design Spectra (ADRS) plane.

Analyzing Climate Resilience — This module provide an assessment of current and future exposure to climate hazards for a specific location, in view of developing Climate Vulnerability Risk Analyses (CVRA) for buildings. Use this module to extract high-resolution climate projections from Copernicus Data Store, perform extreme value analyses, evaluate climate hazards for the building and their variation over time according to different scenarios.

Version: Beta 1.2

Last Updated: 2025-07-15

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Home
Data-Driven Modelling
- Upload Already Trained Data-Driven Predictive Model
- Upload Features And Labels
- Exploratory Data Analysis
- Data-Driven Modelling
- UQ and Sensitivities
- Subtract Effects
- Set Targeted Label
- Compute Feature Distribution For Targeted Label

Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience

SZTAKI UNIVERZA V LJUBLJANI

Data-Driven Modelling

This module constructs a purely data-driven model directly from input-output measurements.

Use this module to analyze data, assess output sensitivities, and isolate the effects of specific inputs—such as temperature variations. Additionally, determine possible input values needed to achieve a desired model output.

Steps:

1. **Upload Features (inputs) and Labels (outputs)** – Import datasets containing relevant features and labels.
2. **Data-Driven Modelling** – Train a data-driven model using the imported input-output pairs. Select from different model types—General Polynomial Chaos Expansion (gPCE), Deep Neural Networks (DNN), or Gradient Boosted Decision Trees (GBT)—and specify hyperparameters.
3. **UQ and Sensitivities** – Perform uncertainty quantification and sensitivity analysis using Sobol indices and SHAP values to understand how feature variations affect the labels.
4. **Subtract Effects** – Remove isolated effects from specified features – such as temperature variations.
5. **Set Targeted Label** – Determine targeted value of the labels, with pre-defined accuracy.
6. **Compute Feature Distribution for Targeted Label** – Determine possible features values to achieve the targeted label values. The output includes samples from the posterior distribution of features, along with visualizations such as pair plots and estimated marginal distributions.

I don't have a model | I already have a model

Use Demo Data

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Figure 8.4: TRACE-Structures UI - Opening page (upper panel), and Step 1 of module 1 – Getting started (lower panel)

8.9.2 Step 2: Upload Already Trained Model

If a model has already been trained in a previous session or is available via the BUILDCHAIN DKG through the BUILDCHAIN API, it can be imported using the `Choose File` button under the `Upload Already Trained Data-Driven Predictive Model` submodule (see Fig. 8.5). In this case, the next step—importing features and labels—can be skipped, as well as the setup of the data-driven model in Step 5. Once the upload is successful, simply click the `Next` button to proceed. If you choose to begin by uploading an existing model, navigate to the `Upload Already Trained Surrogate Predictive Model` section in the left navigation bar (if it is not already selected), and click the `Input Loader` button to upload the surrogate model (see Fig. 8.5). Clicking this button will open a new data importer window, where you can choose to upload the model either from your local storage by clicking the `Choose File` button, or via the BUILDCHAIN API gateway by entering your credentials—specifically, by copying your authentication token and clicking the `Login` button (these steps are also explained at Step 2 of Module 2 with snapshots of the UI). Once logged in, you can search for available data associated with a specific building by locating its building knowledge asset using various criteria—such as name, code, type, presence of measurements, city, DID, or zone. After entering your search criteria, click the `Get Data` button to retrieve matching building knowledge assets. Select the desired asset and click `Load Selected Building` to load the corresponding building data. Once the building is loaded, you can choose which linked knowledge asset you wish to import. To proceed, select the model knowledge asset and click the `Load` button to import it into the `TRACE-Structures` system.

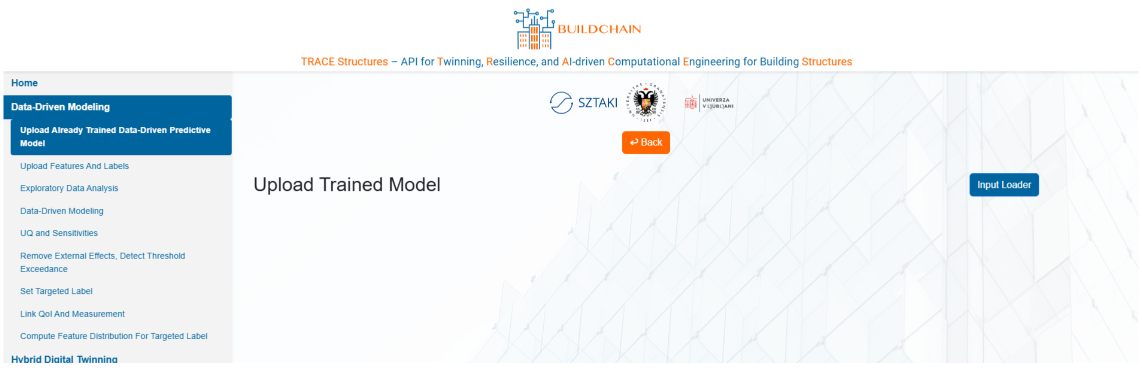
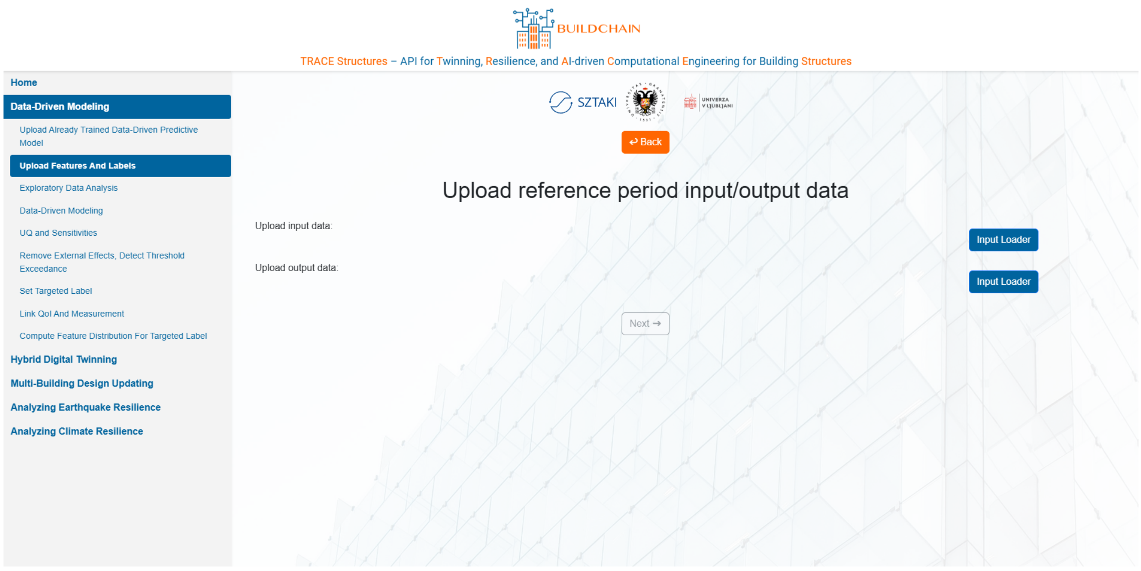

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels
Exploratory Data Analysis
Data-Driven Modeling
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
SZTAKI
Univerza v Ljubljani
Back
Upload Trained Model
Input Loader

Figure 8.5: Step 2 of module 1 – importing already trained model from local storage or the DKG.

8.9.3 Step 3: Import Features and Labels

Click the `Input Loader` buttons (see Fig. 8.6) to upload input features (e.g., weather time series) and output labels (e.g., modal time series) from either local storage or via the BUILDCHAIN API. The procedure for importing data follows the same steps outlined in Step 2 for importing an existing model. Supported formats include `.csv` and `.json` for input features, and `.csv` and `.unv` for output labels. After successful upload, the interface displays input feature names and estimated distributions (Fig. 8.7, which are primarily used for General Polynomial Chaos (GPC) models, where they are required for defining orthogonal polynomials. For all other model types, this information is provided for user reference only. Summary statistics of the input and output labels are also provided, including percentiles (25th, 50th, 75th), sample count, minimum, maximum, mean, and standard deviation. Click `Next` or use the left-hand menu to proceed.

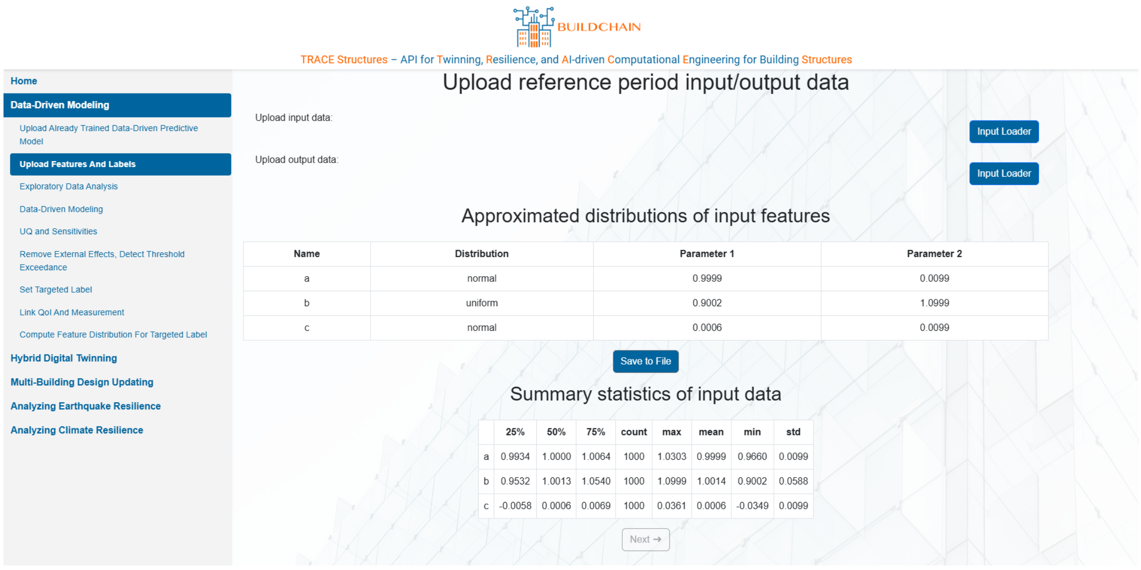

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels
Exploratory Data Analysis
Data-Driven Modeling
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
Back
Input Loader
Input Loader
Next

Upload reference period input/output data

Upload input data:

Upload output data:

Figure 8.6: Module 1 – Data-driven modeling: Step 2: Upload features (inputs) and labels (outputs, quantity of interests)


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels
Exploratory Data Analysis
Data-Driven Modeling
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Input Loader
Input Loader
Save to File
Next

Upload reference period input/output data

Upload input data:

Upload output data:

Approximated distributions of input features

Name	Distribution	Parameter 1	Parameter 2
a	normal	0.9999	0.0099
b	uniform	0.9002	1.0999
c	normal	0.0006	0.0099

Summary statistics of input data

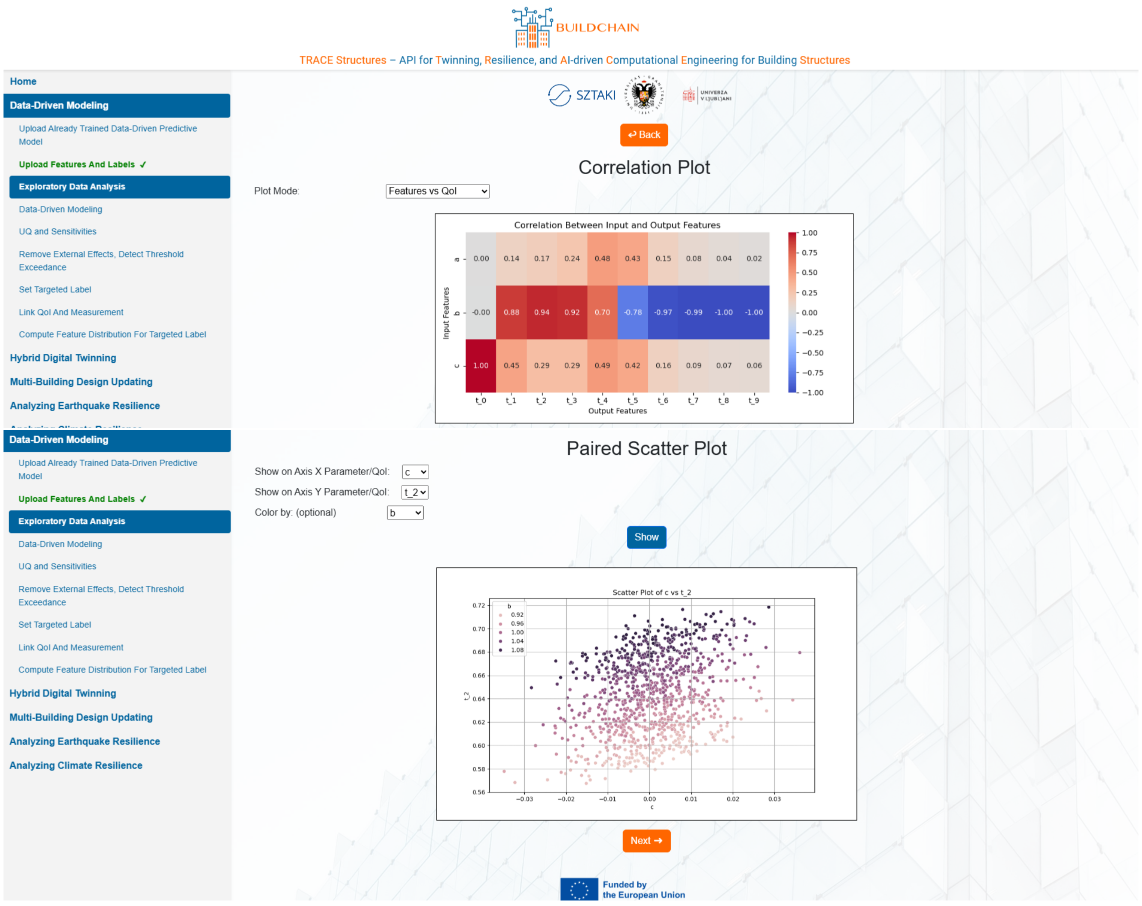
	25%	50%	75%	count	max	mean	min	std
a	0.9934	1.0000	1.0064	1000	1.0303	0.9999	0.9660	0.0099
b	0.9532	1.0013	1.0540	1000	1.0999	1.0014	0.9002	0.0588
c	-0.0058	0.0006	0.0069	1000	0.0361	0.0006	-0.0349	0.0099

Figure 8.7: Module 1 – Data-driven modeling: Step 3 show approximated distribution of input features, summary statistics of target labels.

8.9.4 Step 4: Exploratory Data Analysis

As a first step in understanding the relationship between input features and output labels, the *Exploratory Data Analysis* sub-module offers an *Exploratory Modal Analysis* view (see Fig. 8.8). This includes a color-coded correlation matrix that visualizes relationships such as parameter-to-parameter, parameter-to-QoI (Quantity of Interest), or QoI-to-QoI. The user can select which type of relationship to display, helping to identify potential linear dependencies and guiding feature selection for subsequent modeling tasks.

In addition to the correlation matrix, users can generate scatterplots by selecting specific variables for the x- and y-axes (either input features or QoIs), and optionally enhance interpretability by coloring points based on a third variable. Click `Next` or use the left-hand menu to proceed to the next step.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modeling
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Show
Next ➔
Funded by the European Union

Correlation Plot

Plot Mode: Features vs QoI

Correlation Between Input and Output Features

Input Features	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
a	0.00	0.14	0.17	0.24	0.48	0.43	0.15	0.08	0.04	0.02
b	-0.00	0.88	0.94	0.92	0.70	-0.78	-0.97	-0.99	-1.00	-1.00
c	1.00	0.45	0.29	0.29	0.49	0.42	0.16	0.09	0.07	0.06

Output Features

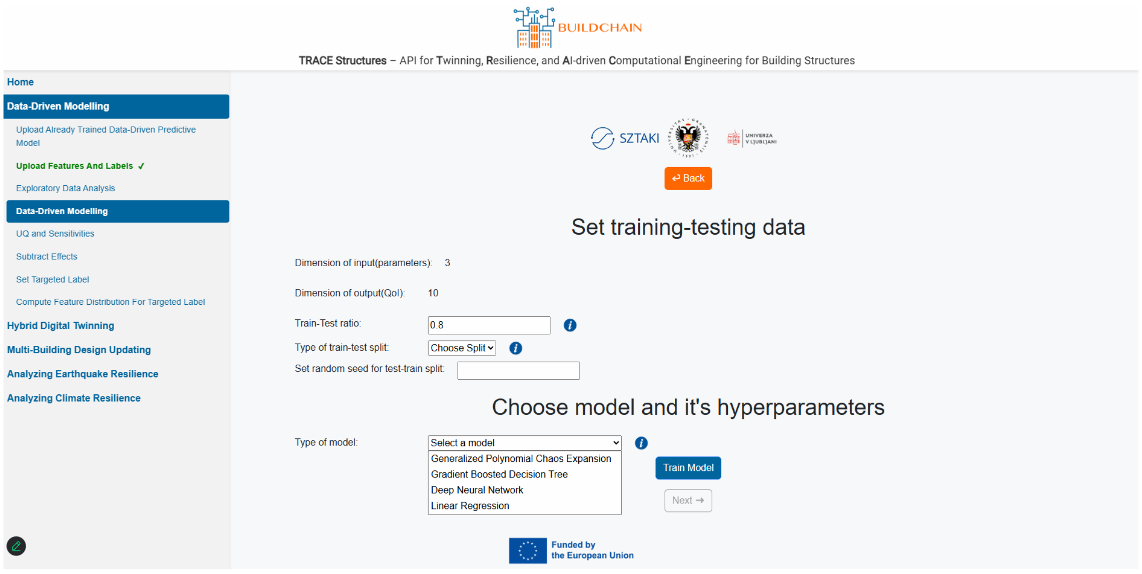
Paired Scatter Plot

Show on Axis X Parameter/QoI: c
Show on Axis Y Parameter/QoI: t_2
Color by: (optional) b

Scatter Plot of c vs t_2
b
0.92
0.96
1.00
1.04
1.08
t_2
c

Figure 8.8: Step 4 of module 1 – exploratory data analysis: the first image shows the correlation between the features and the labels while the second plot shows a scatter plot of c values against the observed output at t_2, colored by the value of the b parameter.

8.9.5 Step 5: Data-Driven Modeling


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Train Model
Next ➔
Funded by the European Union

Set training-testing data

Dimension of input(parameters): 3

Dimension of output(QoI): 10

Train-Test ratio: 0.8

Type of train-test split: Choose Split

Set random seed for test-train split:

Choose model and it's hyperparameters

Type of model: Select a model
- Generalized Polynomial Chaos Expansion
- Gradient Boosted Decision Tree
- Deep Neural Network
- Linear Regression

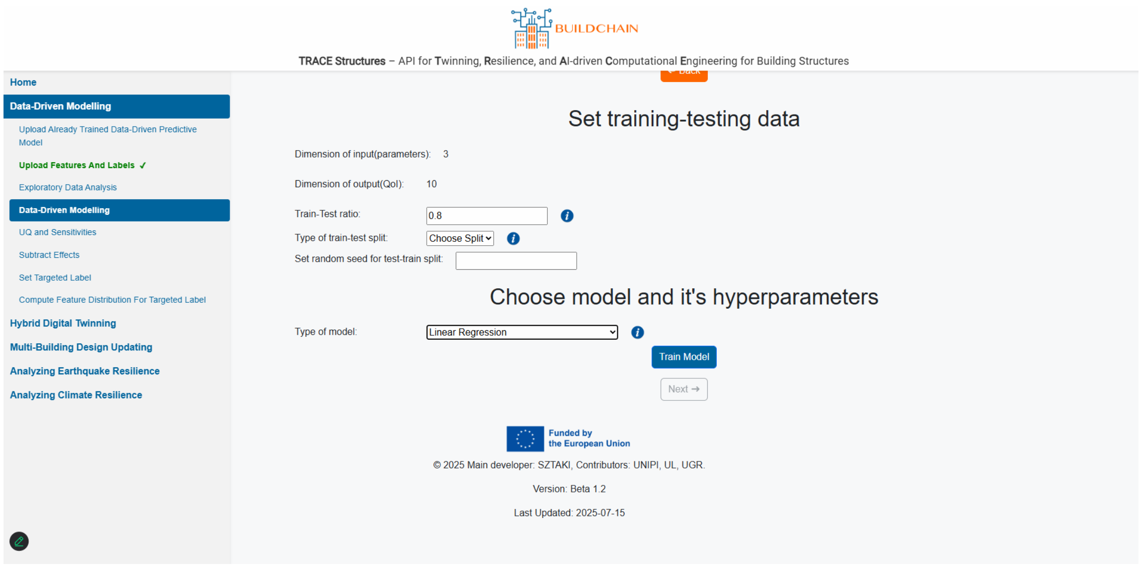
Figure 8.9: Step 5 of module 1 – configuring model training

To begin training a data-driven model, the user must configure on the `Data-Driven Modeling` page (see Fig. 8.9), three key components: (1) the data split for training and testing, (2) the type of model to be used (Linear Regression/GPC/GBT/DNN, and (3) the associated hyperparameters. The TRACE-Structures UI provides an intuitive interface for each of these steps, as illustrated in Figures 8.10–8.13.

For the train-test split confituration, users can define how the dataset is split into training and testing subsets by specifying: i) the ratio of data used for training (train-test ratio); ii) whether the dataset should be shuffled before splitting; iii) Optionally, a random seed can be specified to ensure the process is repeatable.

For the `Type of model specification` the following model types are available, along with their required settings:

- **Linear Regression (LR)** *Hyperparameters:* None required (see Fig. 8.10). This model assumes a linear relationship between inputs and outputs.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Train Model
Next →
Funded by the European Union

Set training-testing data

Dimension of input(parameters): 3

Dimension of output(QoI): 10

Train-Test ratio: 0.8

Type of train-test split: Choose Split

Set random seed for test-train split:

Choose model and it's hyperparameters

Type of model: Linear Regression

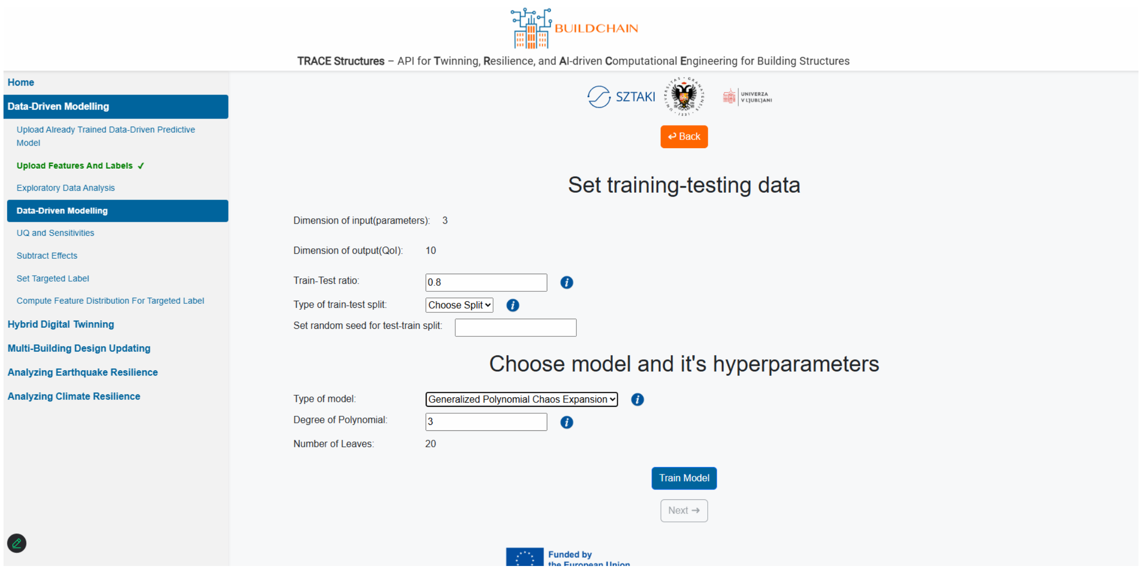
© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Figure 8.10: Step 5 of module 1 – inear regression does not require any setting.

- **Generalized Polynomial Chaos Expansion (GPC)**


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
SZTAKI
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
↩ Back
Train Model
Next →
Funded by the European Union

Set training-testing data

Dimension of input(parameters): 3

Dimension of output(QoI): 10

Train-Test ratio: 0.8

Type of train-test split: Choose Split

Set random seed for test-train split:

Choose model and it's hyperparameters

Type of model: Generalized Polynomial Chaos Expansion

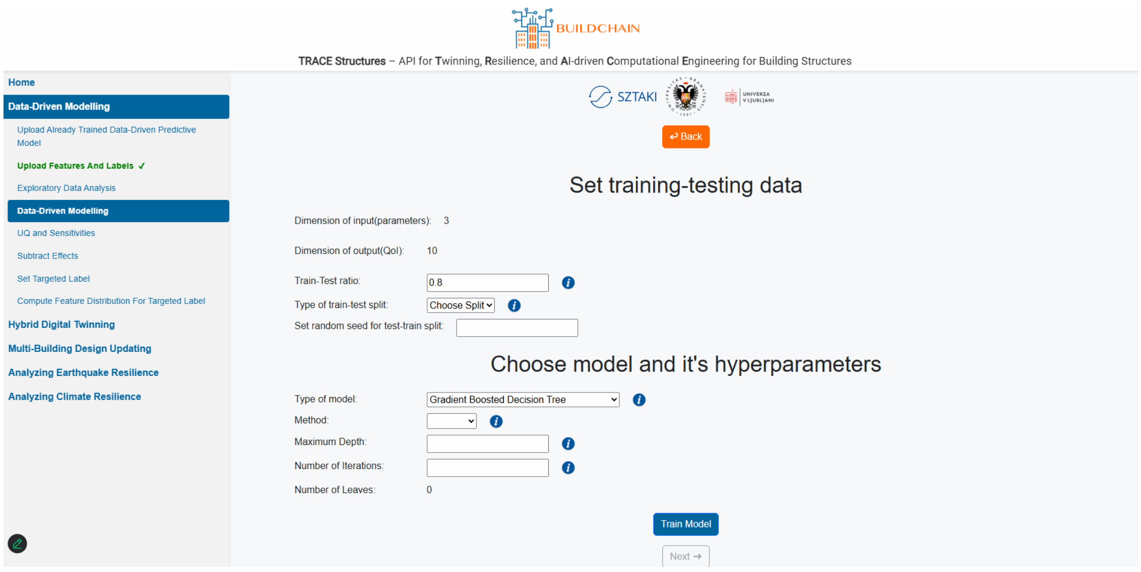
Degree of Polynomial: 3

Number of Leaves: 20

Figure 8.11: Step 5 of module 1 – configuring generalized polynomial chaos model.

Hyperparameters to be set: maximum degree of the univariate polynomial (if a single output label is selected), or the total degree of the multivariate polynomial (for multiple output labels). See setting in Fig. 8.11.

- **Gradient Boosted Decision Tree (GBT)** *Hyperparameters* to be defined by the user: method (xgboost/lightgbm/catboost), maximum depth of the tree, number of iterations, that is number of ensembles. See Fig. 8.12.


BUILDCHAIN

TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience

SZTAKI

↩ Back

Set training-testing data

Dimension of input(parameters): 3
Dimension of output(QoI): 10
Train-Test ratio: 0.8
Type of train-test split: Choose Split
Set random seed for test-train split:

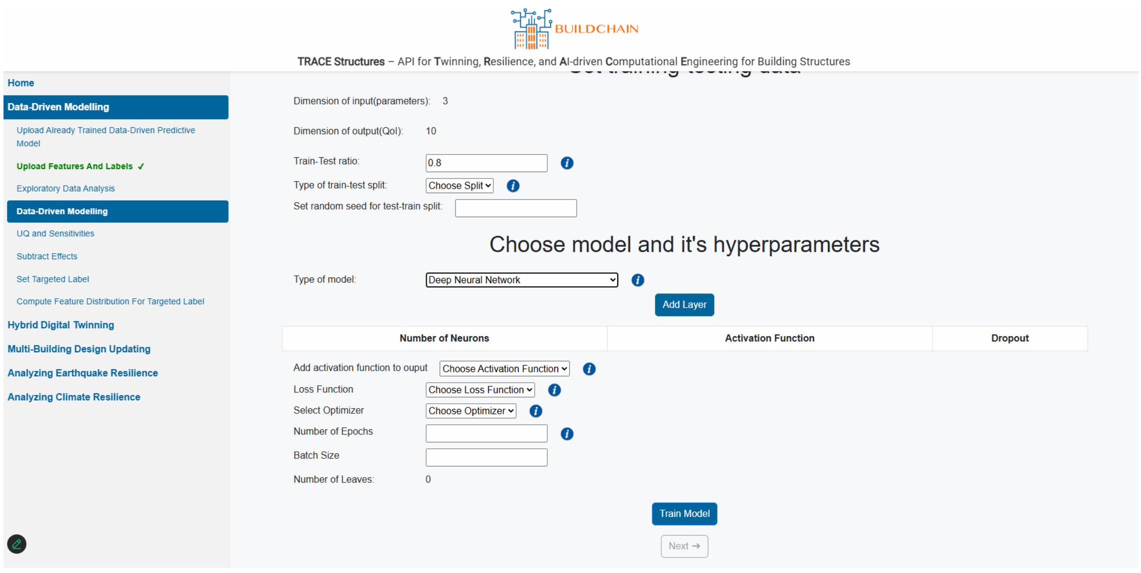
Choose model and it's hyperparameters

Type of model: Gradient Boosted Decision Tree
Method:
Maximum Depth:
Number of Iterations:
Number of Leaves: 0

Train Model
Next →

Figure 8.12: Step 5 of module 1 – configuring gradient boosted decision tree model.

- **Deep Neural Network (DNN)** The user should define the network architecture by adding layers by clicking on the `Add Layer` button and specifying the following properties for each layer: number of neurons, activation function, dropout rate (if applicable). Furthermore, the user can define the loss function to be minimized in the training, the optimizer, the number of epochs (iterations) and the batch size. See DNN setting on the UI in Fig. 8.13.


BUILDCHAIN

TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience

Dimension of input(parameters): 3
Dimension of output(QoI): 10
Train-Test ratio: 0.8
Type of train-test split: Choose Split
Set random seed for test-train split:

Choose model and it's hyperparameters

Type of model: Deep Neural Network

Add Layer

Number of Neurons	Activation Function	Dropout

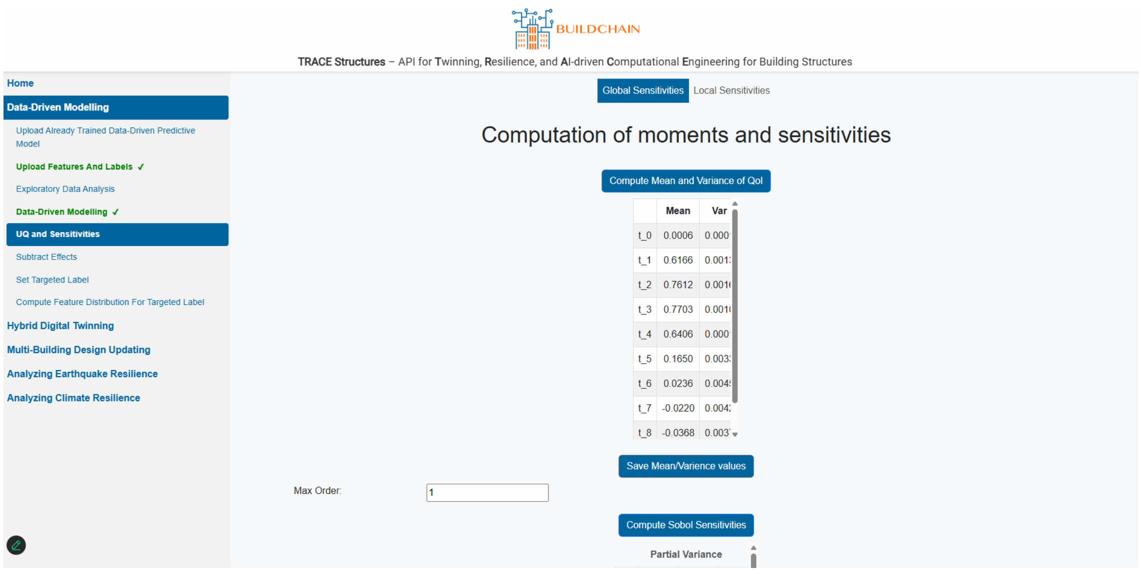
Add activation function to ouput: Choose Activation Function
Loss Function: Choose Loss Function
Select Optimizer: Choose Optimizer
Number of Epochs:
Batch Size:
Number of Leaves: 0

Train Model
Next →

Figure 8.13: Step 5 of module 1 – configuring deep neural network model

8.9.6 Step 6: UQ, Sensitivities, Feature Attributions

Global Sensitivities To perform a global sensitivity analysis, click on the *Global Sensitivities* tab. This section allows the user to evaluate the mean and variance of each element in the Quantity of Interest (QoI) vector by clicking on `Compute Mean and Variance of QoIs`. The resulting table can be saved using the `Save Mean and Variance Values` button (see Fig. 8.14).


BUILDCHAIN
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling ✓
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Global Sensitivities
Local Sensitivities
Compute Mean and Variance of QoI
Save Mean/Varience values
Compute Sobol Sensitivities

TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

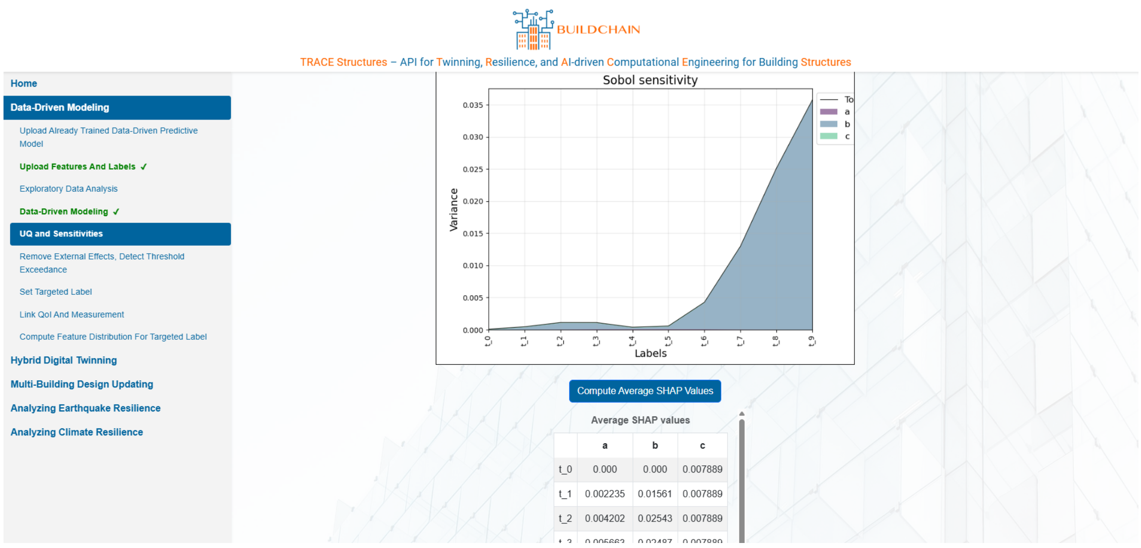
Computation of moments and sensitivities

	Mean	Var
t_0	0.0006	0.000
t_1	0.6166	0.001
t_2	0.7612	0.001
t_3	0.7703	0.001
t_4	0.6406	0.000
t_5	0.1650	0.003
t_6	0.0236	0.004
t_7	-0.0220	0.004
t_8	-0.0368	0.003

Max Order: 1

Partial Variance

Figure 8.14: Step 5 of module 1 – global sensitivity analysis, computing mean and variance of QoIs, as well as Sobol partial variances


BUILDCHAIN
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modeling ✓
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Sobol sensitivity
Variance
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
t_0
t_1
t_2
t_3
t_4
t_5
t_6
t_7
t_8
t_9
Labels
To
a
b
c
Compute Average SHAP Values

TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

Average SHAP values

	a	b	c
t_0	0.000	0.000	0.007889
t_1	0.002235	0.01561	0.007889
t_2	0.004202	0.02543	0.007889
t_3	0.005663	0.02487	0.007889

Figure 8.15: Step 5 of module 1 – global sensitivity analysis, visualization of partial variances, computation of average SHAP values

For Sobol sensitivity computation, first select the *maximum order* of interaction. The Sobol index order determines which combinations of input features are considered:

- Order 1: Measures the variance contribution from each input feature individually (i.e., main effects).

- Order 2: Besides linear effects, captures interaction effects between pairs of input features.
- Order 3 and above: Quantifies higher-order interactions involving three or more inputs.

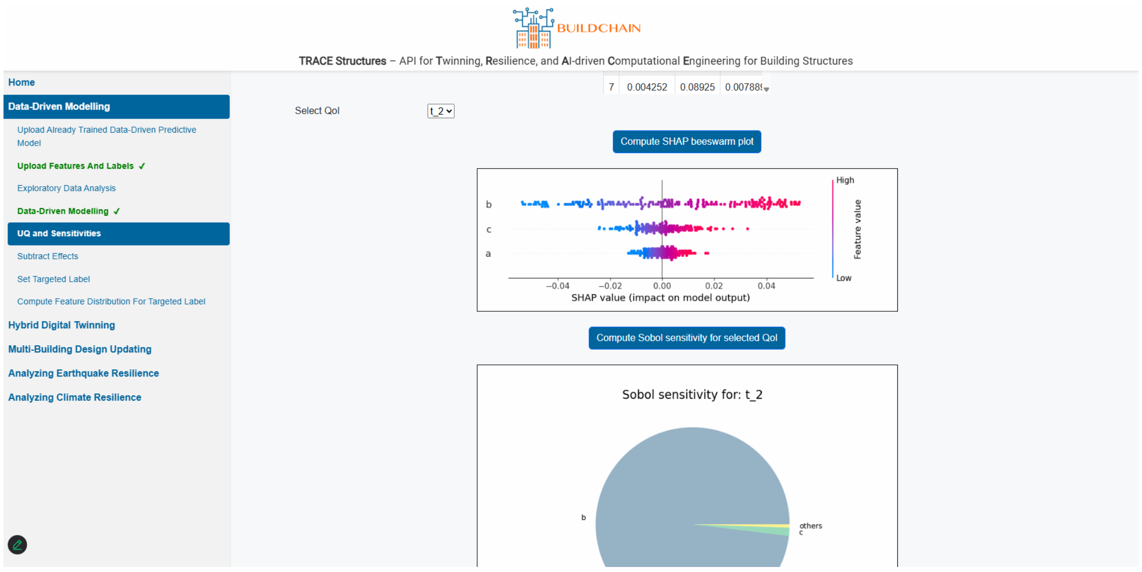

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling ✓
UQ and Sensitivities
Subtract Effects
Set Targeted Label
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
7 0.004252 0.08925 0.00788
Select QoI
t_2
Compute SHAP beeswarm plot
b
c
a
High
Feature value
Low
−0.04
−0.02
0.00
0.02
0.04
SHAP value (impact on model output)
Compute Sobol sensitivity for selected QoI
Sobol sensitivity for: t_2
b
others
c

Figure 8.16: Step 5 of module 1 – global sensitivity analysis, exploring feature attributions in the prediction (SHAP beewwarm plot) and in the variance (Sobol pie plot) of a specific QoI.

After selecting the desired order, click on `Compute Sobol Sensitivities` to display the computed partial variances and Sobol indices up to the specified order. Figure 8.15 illustrates how the UI presents the total variance and corresponding partial variances.

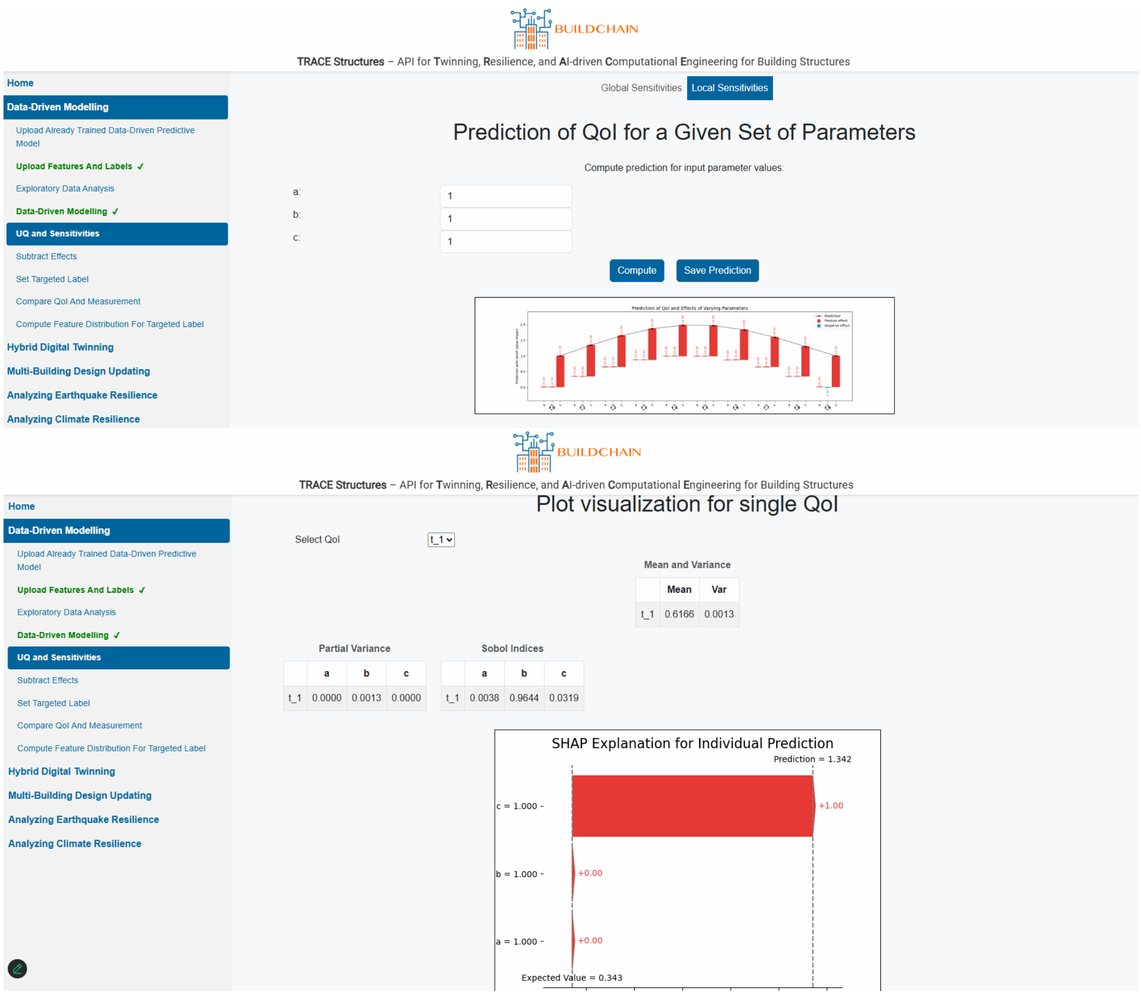
To compute global feature importances using SHAP (SHapley Additive exPlanations), click on `Compute Average SHAP Values`. This generates a table showing feature importances based on the average of local attributions across the dataset.

In the next step, the user can analyze the influence of input features on a specific component of the QoI. To do this, select the desired QoI component and click on `Compute SHAP Feature Attributions for Selected QoI` (see Fig. 8.16). The resulting beeswarm plot visualizes how input features shift the prediction away from the expected value. In the plot each row represents an input feature, and each point a realization of that feature from the test data set. The point colors indicate the value of the specific feature (e.g., low to high). Point positions reflect the magnitude and direction of the SHAP contribution, and shows weather the specific input features shifted the prediction to the right or left of the expected value and with what magnitude. Finally, by clicking on `Compute Sobol Sensitivity for Selected QoI`, the interface displays a pie chart showing how much of the total variance of the selected QoI component is attributed to each input feature, based on Sobol indices.

By clicking on the `Local Sensitivities` tab at the top of the submodule, users can explore local feature attributions for individual predictions (see upper panel of Fig.8.17 for SHAP attributions across all QoIs, and lower panel for a selected QoI component) by specifying values of the input parameters. This functionality allows the computation of SHAP values that attribute the contribution of each input feature to the model's prediction—either for all QoIs or for a selected component.

The attributions are visualized using SHAP waterfall plots, which show how individual input features shift the prediction away from a baseline (expected) value.

By selecting a specific input instance, users can see how that set of input values influenced the output. The plot clearly indicates both positive and negative contributions, enabling transparent, case-by-case interpretation of the model's behavior.


BUILDCHAIN

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

- Home
- Data-Driven Modelling
  - Upload Already Trained Data-Driven Predictive Model
  - Upload Features And Labels ✓
  - Exploratory Data Analysis
  - Data-Driven Modelling ✓
  - UQ and Sensitivities
  - Subtract Effects
  - Set Targeted Label
  - Compare QoI And Measurement
  - Compute Feature Distribution For Targeted Label
- Hybrid Digital Twinning
- Multi-Building Design Updating
- Analyzing Earthquake Resilience
- Analyzing Climate Resilience

Global Sensitivities | Local Sensitivities

Prediction of QoI for a Given Set of Parameters

Compute prediction for input parameter values:

a:	1
b:	1
c:	1

Compute | Save Prediction

BUILDCHAIN

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Plot visualization for single QoI

Select QoI t_1

Mean and Variance

	Mean	Var
t_1	0.6166	0.0013

Partial Variance

	a	b	c
t_1	0.0000	0.0013	0.0000

Sobol Indices

	a	b	c
t_1	0.0038	0.9644	0.0319

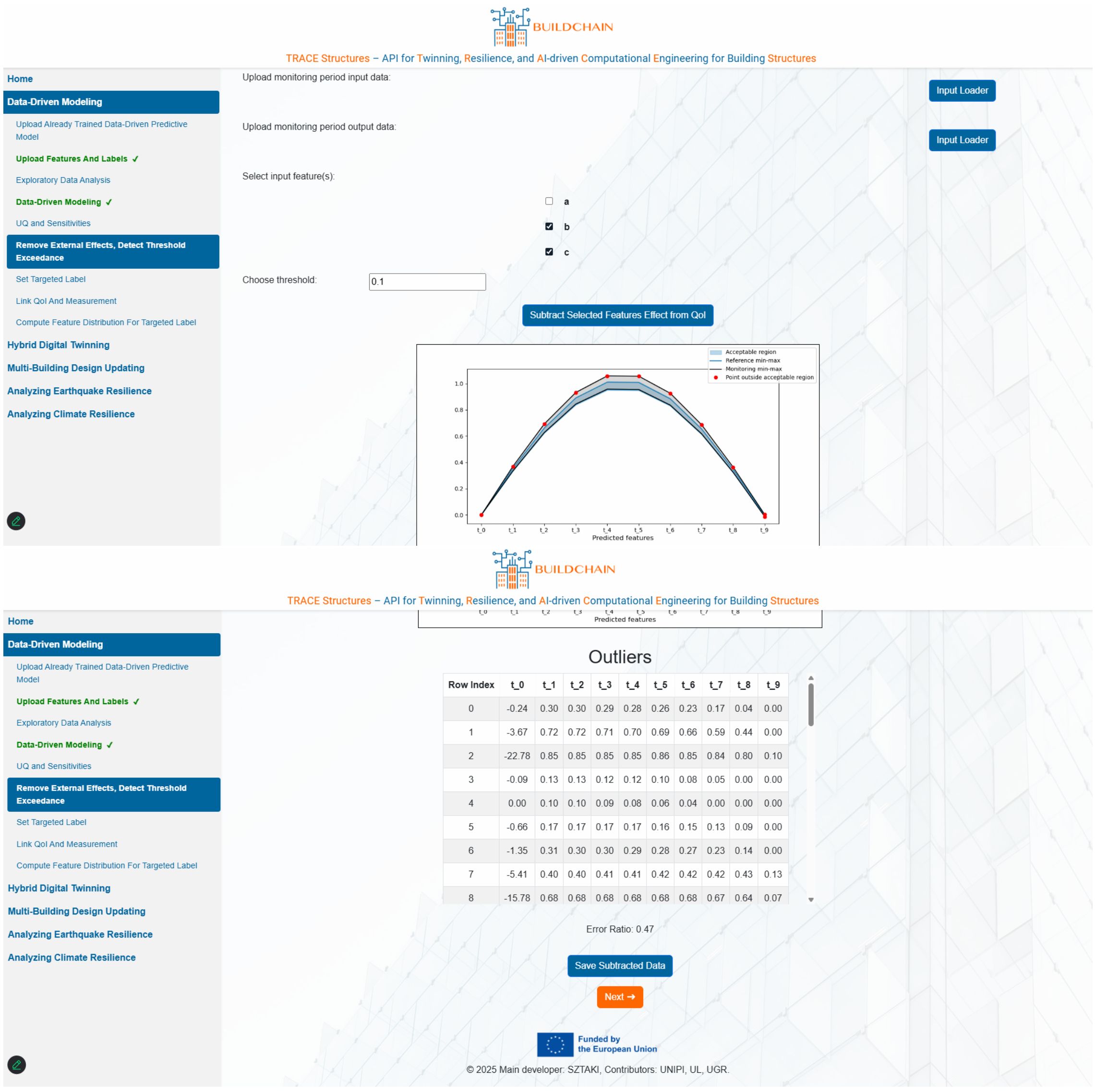
Figure 8.17: Step 5 of module 1 – local sensitivity analysis, attribution of input features in the prediction of QoIs for specific combination of input features (upper panel for all QoIs, lower panel for selected QoI).

Local Sensitivity Analysis

8.9.7 Step 7: Remove External Effects, Detect Threshold Exceedance

This step is a key component of the module, enabling users to subtract the influence of selected input features from a new set of input–output data collected during a monitoring period. To begin, the user uploads the monitoring dataset using the `Input Loader` buttons. In this example, synthetic monitoring data has been generated for the toy case by modifying the amplitude parameter a—specifically, by slightly shifting its mean value and increasing its variance.

In the demo example, we demonstrate how the effects of selected parameters—specifically frequency variation (feature b) and phase shift (feature c)—are subtracted, resulting in a sinusoidal signal that varies only in amplitude (see Fig. 8.18). The computation begins by subtracting environmental effects from the reference period observation data, with the resulting min-max envelope shown as a blue line in the figure. The same subtraction is then applied to the uploaded monitoring data, and its corresponding min-max values after effect removal are also visualized in the figure.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modeling ✓
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Upload monitoring period input data:
Input Loader
Upload monitoring period output data:
Input Loader
Select input feature(s):
☐ a
☑ b
☑ c
Choose threshold: 0.1
Subtract Selected Features Effect from QoI
Acceptable region
Reference min-max
Monitoring min-max
Point outside acceptable region
Predicted features
Outliers
Save Subtracted Data
Next ➜
Funded by the European Union
© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Outliers

Row Index	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
0	-0.24	0.30	0.30	0.29	0.28	0.26	0.23	0.17	0.04	0.00
1	-3.67	0.72	0.72	0.71	0.70	0.69	0.66	0.59	0.44	0.00
2	-22.78	0.85	0.85	0.85	0.85	0.86	0.85	0.84	0.80	0.10
3	-0.09	0.13	0.13	0.12	0.12	0.10	0.08	0.05	0.00	0.00
4	0.00	0.10	0.10	0.09	0.08	0.06	0.04	0.00	0.00	0.00
5	-0.66	0.17	0.17	0.17	0.17	0.16	0.15	0.13	0.09	0.00
6	-1.35	0.31	0.30	0.30	0.29	0.28	0.27	0.23	0.14	0.00
7	-5.41	0.40	0.40	0.41	0.41	0.42	0.42	0.42	0.43	0.13
8	-15.78	0.68	0.68	0.68	0.68	0.68	0.68	0.67	0.64	0.07

Error Ratio: 0.47

Figure 8.18: Step 7 of module 1 – removing external effects, detect threshold exceedance.

After the selected effects are subtracted, the user can define a threshold to determine the acceptable range for monitored observations. Any values falling outside this range are flagged as potential anomalies. The threshold is applied symmetrically around the min–max range of the reference (training) period data. Specifically:

$$\begin{aligned} \text{max}_{\text{accepted}} &= \text{max}_{\text{train}} + \theta \cdot (\text{max}_{\text{train}} - \text{min}_{\text{train}}), \\ \text{min}_{\text{accepted}} &= \text{min}_{\text{train}} - \theta \cdot (\text{max}_{\text{train}} - \text{min}_{\text{train}}), \end{aligned}$$

where θ is the user-defined threshold. The monitored values are then compared against this acceptable region, and any exceedances are automatically highlighted.

The user interface visualizes the acceptable range alongside the min–max bounds of both the reference and monitoring period data, allowing users to quickly assess deviations and identify anomalies. It also provides a table listing the detected outliers, along with the magnitude of their exceedance relative to the reference min–max range. This table can be stored or registered in the DKG. Additionally, the UI displays the error ratio, calculated as the number of outliers divided by the total number of samples. The user can store the displayed statistics—such as minimum, maximum, and the 25th, 50th, and 75th percentiles—for both the reference period and the monitoring period (after removing the selected effects), either locally or on the BUILDCHAIN DKG.

8.9.8 Step 8: Set Target Label

This sub-module allows the user to determine the values (or distributions) of input features that would produce a specific target value of the output label, within a specified error margin defined by an unbiased Gaussian distribution (characterized by its standard deviation). The module employs Bayesian inference (see Module 2: Calibration of Model Inputs for more details, e.g. in Section 9.4.2) to identify the distribution of the input features, using samples generated through an MCMC procedure.

To proceed, the user must provide two `.csv` files by clicking on the `Choose File` button: one specifying the desired value of the target labels, and another specifying its associated standard deviations (see Fig. 8.19), which values will be presented in the UI.

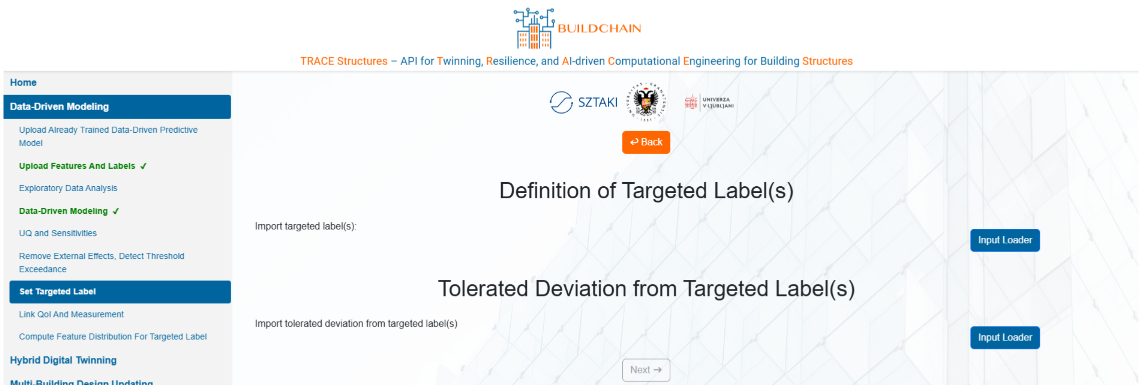

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modeling ✓
UQ and Sensitivities
Remove External Effects, Detect Threshold Exceedance
Set Targeted Label
Link QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
SZTAKI
↩ Back
Definition of Targeted Label(s)
Import targeted label(s):
Input Loader
Tolerated Deviation from Targeted Label(s)
Import tolerated deviation from targeted label(s)
Input Loader
Next →

Figure 8.19: Step 8 of module 1 – set target label.

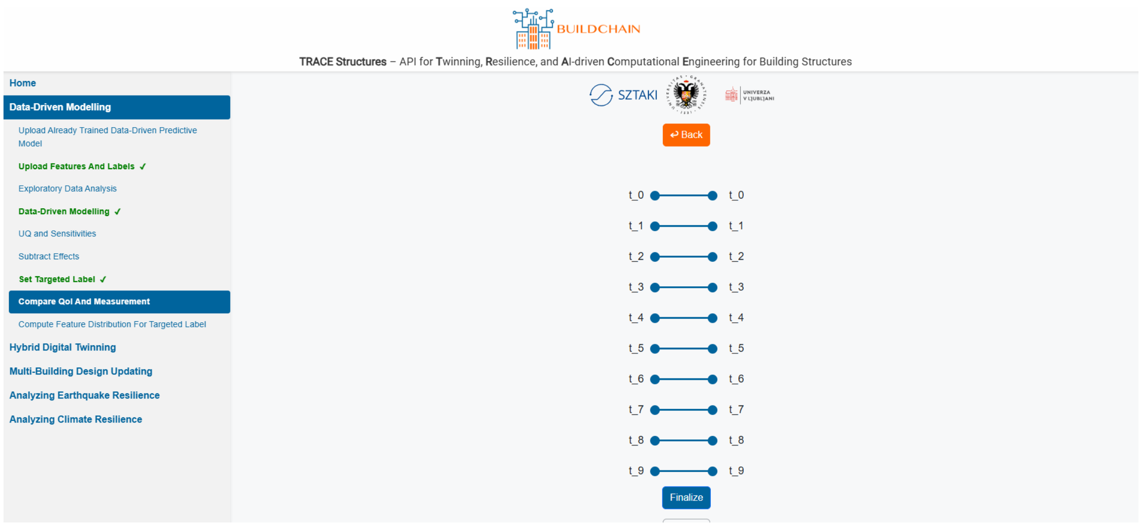

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling ✓
UQ and Sensitivities
Subtract Effects
Set Targeted Label ✓
Compare QoI And Measurement
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
↩ Back
t_0 t_0
t_1 t_1
t_2 t_2
t_3 t_3
t_4 t_4
t_5 t_5
t_6 t_6
t_7 t_7
t_8 t_8
t_9 t_9
Finalize

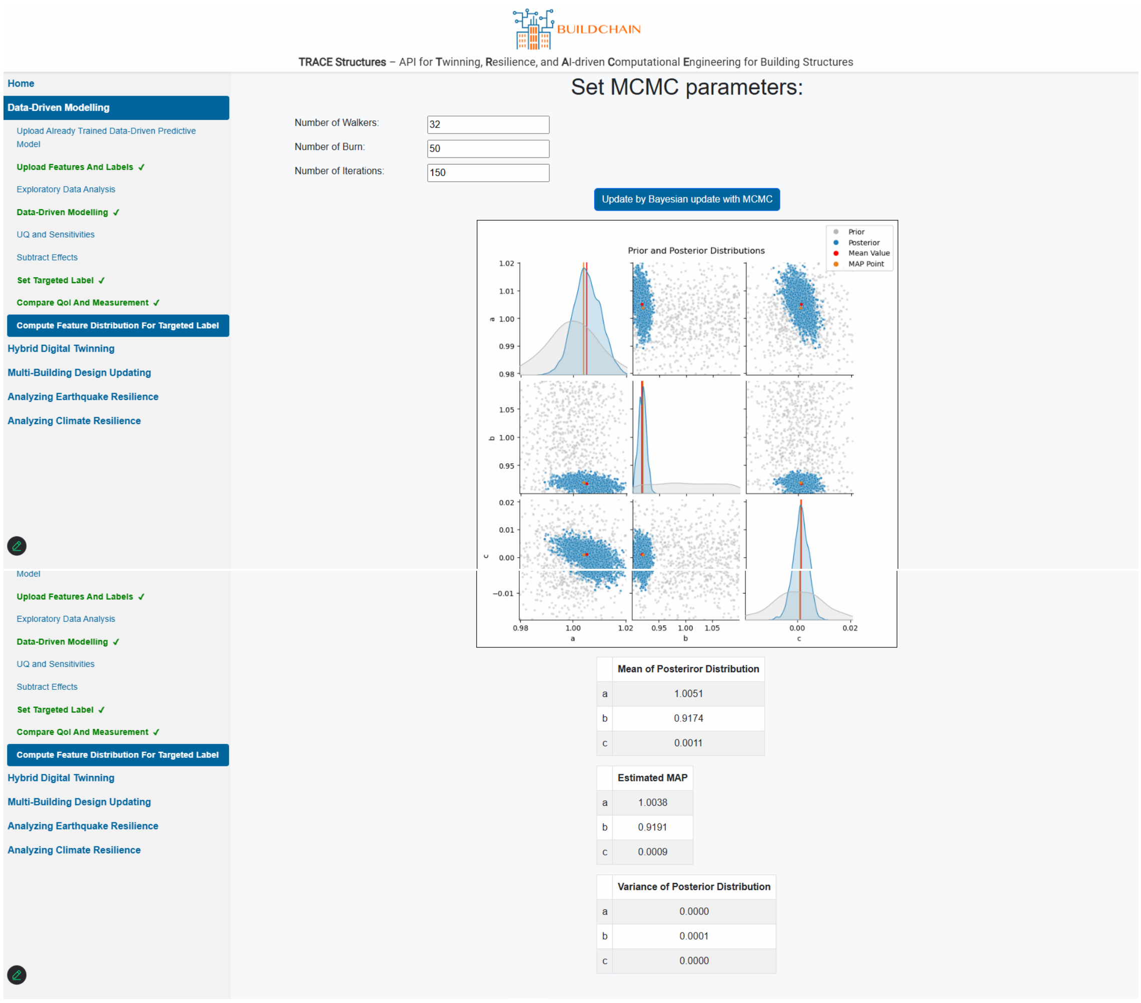
Figure 8.20: Step 8 of module 1 – link QoI names with targeted label names.

After clicking the `Next` button, the user is prompted to link the QoI (Quantity of Interest) names from the reference period output data (e.g., column names from an imported `.csv` file) to the corresponding target label names. This step is particularly useful when the source of the target labels differs from the original dataset and the QoI naming conventions do not match.

The UI automatically matches names that are identical in both datasets. If no matches are found, or if matching is not correct the user must manually pair each QoI by clicking first on the name from the reference data and then on the corresponding target label (see Fig. 8.20).

8.9.9 Step 9: Compute Feature Distribution for Targeted Label

The specified target label values are used to initiate an MCMC random walk, which samples from the resulting posterior distribution of the input features. In this section, the user must configure the MCMC procedure by selecting key parameters, including the number of walkers, the number of steps that are burned, and the number of steps (iterations) after the burn period. The resulting sample number will be the product of this number and the number of walkers.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Upload Already Trained Data-Driven Predictive Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling ✓
UQ and Sensitivities
Subtract Effects
Set Targeted Label ✓
Compare QoI And Measurement ✓
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Model
Upload Features And Labels ✓
Exploratory Data Analysis
Data-Driven Modelling ✓
UQ and Sensitivities
Subtract Effects
Set Targeted Label ✓
Compare QoI And Measurement ✓
Compute Feature Distribution For Targeted Label
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Update by Bayesian update with MCMC
Prior and Posterior Distributions
Prior
Posterior
Mean Value
MAP Point
a
b
c

Set MCMC parameters:

Number of Walkers: 32
Number of Burn: 50
Number of Iterations: 150

	Mean of Posteriror Distribution
a	1.0051
b	0.9174
c	0.0011

	Estimated MAP
a	1.0038
b	0.9191
c	0.0009

	Variance of Posterior Distribution
a	0.0000
b	0.0001
c	0.0000

Figure 8.21: Step 9 of module 1 – MCMC configuration and resulting samples.

The UI displays a paired matrix plot comparing the training data samples (”prior”) with those from the resulting posterior distribution. It also shows point estimates such as the mean and the maximum a posteriori (MAP) estimate. These values, along with summary statistics of the resulting distribution, are presented alongside the visualizations (see Fig. 8.21.

8.10 Results of Testing, Interpretation of Outputs

8.10.1 Toy Model

The results for the toy model were already presented in the step-by-step user guide in Section 8.9. These plots demonstrated that the primary input feature influencing the output **y** is parameter b, which controls the signal frequency (see Figs. 8.15–8.17). It is important to note that this is not only due to the stronger influence of frequency in the signal, but also to the relatively small variations in amplitude and phase shift.

That section also validated the functionality for removing external effects. After subtracting the influence of phase shift and frequency variations, the resulting data—shown in Fig. 8.18—exhibited a more stable min–max range, consistent with data in which phase and frequency remain constant.

Finally, the Bayesian updating functionality was validated using the analytical toy model. Starting from the estimated a priori distribution (learned from training data) and applying the desired target label with a specified confidence interval, the resulting posterior distribution was sampled. These samples were then used to run the analytical model, confirming that the predicted outputs matched the desired label values. This process, detailed further in the description of the ***Hybrid Digital Twinning*** module, demonstrates the effectiveness of the Bayesian inversion approach implemented here.

8.10.2 Hospital Real Pilot Model

Following the toy model, the module was tested and applied to the first BUILDCHAIN pilot project, which focuses on the structural health monitoring of the Hospital Real—a cultural heritage building in Granada (see Section 8.7.2 for details). The results are available through the linked Jupyter Notebook in the Annex Section C.1.2.

The exploratory data analysis revealed a notable dependence of natural frequencies (see scatter plots of frequencies as a function of the temperature) on environmental factors, particularly temperature and humidity. The analysis showed that higher modes are especially sensitive to these weather conditions. It was shown, that about 20-30% of variability of frequencies can be predicted from temperature and humidity. Using the developed API, we successfully removed these environmental effects from the frequency data.

This capability is particularly valuable for subsequent Bayesian updating of the building's material properties, as it isolates the structural response from external influences. The resulting weather-independent data—affected only by measurement noise and actual structural changes—can significantly enhance the reliability of model calibration and structural assessment.

8.11 Strengths and Limitations

The *Exploratory Data Analysis* submodule of the TRACE-STRUCTURES User Interface currently only displays the correlation matrix between input and output features as well as scatter plots of chosen pairs of variables. However, several additional visualization techniques are available for exploring the data, which are, for now, accessible only through the library package.

The current implementation supports Generalized Polynomial Chaos (GPC) expansions through a regression-based approach. While the underlying Python library is capable of supporting additional techniques such as spectral projection, these are not yet available in the user interface.

For GPC basis construction, the module currently allows the use of all multivariate polynomials whose total degree does not exceed a user-defined threshold. While this approach is general, it may lead to unnecessarily large basis sets. More efficient and sparse basis selection strategies—such as those based on explained variance or elliptic index sets—could be incorporated in future versions.

In the case of deep neural networks (DNNs), the current UI supports only basic feed-forward architectures. Advanced neural network structures, such as convolutional or recurrent layers, are not yet available but could be added to support more complex modeling tasks.

For modal shapes, we observed that challenges such as mode crossing or veering may require specialized surrogate modeling techniques. One such method is the MOSAIC model, that we developed within the framework of the BUILDCHAIN project, but which is currently not integrated into the user interface.

Chapter 9

Module 2: Hybrid Digital Twinning

9.1 Problem Statement and Needs Addressed

In the field of structural and civil engineering, Finite Element Models (FEMs) are widely used to simulate the physical behavior of buildings, either as a part of the design procedure or for computing performance metrics of an aging building. However, these models often rely on input parameters (e.g., material properties, boundary conditions, damping ratios) that may be uncertain or difficult to define/measure precisely. Ensuring the accuracy of such models is critical.

The main challenge lies in calibrating FEMs to reflect the true behavior of the physical system based on real-world measurements. Traditional calibration techniques are computationally expensive and time-consuming, particularly for large-scale models. Moreover, users require methods to interpret the influence of different parameters and confidently assess uncertainties associated with model predictions.

9.2 Tool Overview, purpose, core functionality

This module provides a powerful framework for data-informed digital twinning through surrogate-based Bayesian model calibration. By replacing expensive FEM evaluations with a data-driven surrogate model, the module enables efficient and robust parameter estimation using Bayesian inference. This results in a posterior distribution of input parameters that best explain observed output data (e.g., from sensors or experiments).

The tool is especially relevant for digital twin applications, where ongoing calibration and synchronization between the physical asset and its virtual counterpart are necessary. Additionally, the module offers comprehensive analysis of model sensitivities and interpretability via Sobol indices and SHAP (SHapley Additive exPlanations), making it possible to evaluate which input parameters have the most influence and how predictions are formed.

The Python functionalities and the surrogate model developed within the BUILDCHAIN platform can be seamlessly integrated into an autonomous SHM workflow driven by IoT sensor data. The surrogate model, trained to approximate the computationally expensive simulations, enables real-time or near-real-time decision-making by rapidly predicting structural responses based on sensor inputs. IoT devices deployed on structures continuously stream measurement data (e.g., accelerations, strains, displacements) to a central processing unit or cloud infrastructure. This live data feed can automatically trigger the surrogate model via Python-based API calls, enabling timely parameter updates, anomaly detection, or damage localization. Through this integration, the surrogate model not

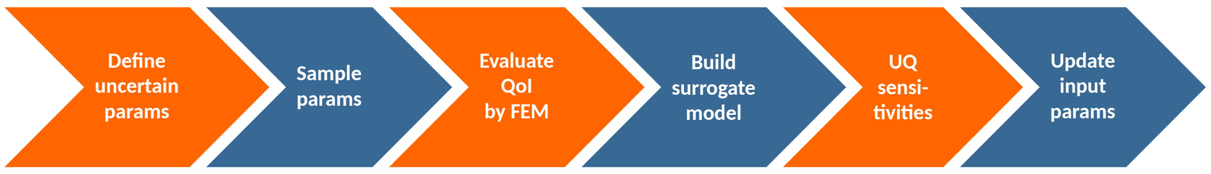


Figure 9.1: Methodology, conceptual approach of using surrogate model for digital twinning with the aim of structural health monitoring or design process updating

only reduces computational costs but also acts as a core component in a self-updating digital twin framework, making SHM systems more adaptive, scalable, and capable of autonomous decision-making without manual intervention.

In addition, this module integrates with the BUILDCHAIN DKG-based DBL system by enabling direct import of measurement data and registration of the resulting surrogate model. It also supports storing updated material properties and modeling parameters inferred from the measured building behavior.

9.3 Key Features and Capabilities

This module, integrated into the `TRACE-Structures` platform, provides the following core functionalities:

- **Supports Digital Twinning:** Enables real-time synchronization between a simulated model and a monitored physical system by updating model parameters with live or historical sensor data.
- **Surrogate Modeling:** Efficiently replaces high-fidelity FEMs with data-driven approximations for rapid evaluations.
- **Bayesian Calibration:** Provides probabilistic estimates of input parameters with uncertainty quantification using MCMC techniques, when measurements of the output of the FEM model are available.
- **Interactive Visualization:** Displays posterior parameter distributions, sensitivites and SHAP feature attributions.
- **Sensitivity Analysis:** Identifies key input variables affecting outputs via Sobol indices.
- **Explainability with SHAP:** Interprets how each input feature contributes to the model's prediction.
- **Reusability and Portability:** Allows saving, reloading, and reusing surrogate models across sessions or use cases from/to local drive or the DKG of the BUILDCHAIN DBL.

9.4 Methodology, Conceptual Approach

This section outlines the theoretical foundations and computational methodology underpinning the surrogate-based calibration module. The approach integrates surrogate modeling, Bayesian inference, and sensitivity analysis to enable efficient and interpretable parameter estimation for high-fidelity numerical models such as FEMs. The main steps of the module is presented in Fig 9.1.

9.4.1 Algorithms, Models, and Assumptions

Let **X** be a random vector representing a set of uncertain input parameters in a simulation model,

$$\mathbf{X} = [X_1, X_2, \ldots, X_n]^T. \tag{9.1}$$

These parameters may include physical or modeling properties of a building structure. In the context of structural health monitoring, **X** may consist of material properties, geometric characteristics, or connection parameters that reflect the current condition of the building. Alternatively, when the aim is to support the design process, the uncertain parameters may represent design variables or modeling assumptions.

Uncertainty in the exact values of these parameters is described by their joint probability distribution $\pi_{\mathbf{X}}$. A specific realization of these random variables is denoted by **x**.

Let $\mathbf{y} = f_{\text{FEM}}(\mathbf{x})$ denote the output of the simulation model—i.e., a quantity of interest (QoI) such as modal properties, including natural frequencies and mode shapes, computed from a given input realization **x**.

The goal of this module is to reduce the initial uncertainty in **X** by incorporating measurements of the QoIs. These observations are represented as:

$$\mathbf{z}_m = \mathbf{y} + \varepsilon, \tag{9.2}$$

where ε accounts for both measurement noise and model discrepancy.

Surrogate Modeling

To reduce the computational cost associated with repeated evaluations of the full finite element model (FEM), a surrogate model $\tilde{f}(\mathbf{x})$ is constructed to approximate the system's behavior:

$$\tilde{f}(\mathbf{x}) \approx f_{\text{FEM}}(\mathbf{x}). \tag{9.3}$$

The surrogate model is built using the same machine learning techniques available in the data-driven modeling module (see a more detailed description in 8.4.2), and may take one of the following forms:

- Linear Regression (LR)
- Gradient Boosted Decision Trees (GBDT)
- Generalized Polynomial Chaos Expansion (GPC)
- Deep Neural Networks (DNN)

The key difference from the environmental-effect subtraction use case of the previous, *Data-Driven Modeling* module is the source of the input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)$: here, they are generated by a physics-based simulation model (the FEM), rather than being acquired from sensor data.

The modeling workflow is as follows:

1. The user defines the prior uncertainties of the input parameters **x** by specifying their probability distributions.
2. Samples are drawn from these distributions using the sampling functionality of the ***Hybrid Modeling*** module.

3. The user locally runs the FEM code with each sample $\mathbf{x}_i$ as input, generating corresponding QoIs, $\mathbf{y}_i$.

4. The resulting input-output pairs $\mathbf{x}_i - \mathbf{y}_i$ are stored in table format and used to train the surrogate model.

Interested readers are directed to e.g. [4, 10] for more detailed information on surrogate based probabilistic model calibration techniques.

Uncertainty quantification (UQ), sensitivity analysis, explainability

The explainability of the model as well as the evaluation of statistics of model output follows the same framework as in the ***Data-Driven Predictive Modeling*** module, though here the results have different meanings:

- Once the model is trained, the user can assess how uncertainties in the input parameters propagate to the model response $\mathbf{y}$. This is achieved by computing the mean and variance of the predicted quantities of interest (QoIs).

- To understand the influence of individual input parameters on the model output, global sensitivity analysis can be performed using Sobol indices [9, 7]. These provide a quantitative measure of the contribution of each input parameter to the overall output variance.

- In addition, local explainability is available through SHAP (SHapley Additive exPlanations) values. This allows users to interpret individual predictions by attributing changes in the predicted value—relative to the global mean—to specific input features, indicating both the direction and magnitude of their influence [6].

Bayesian Inference

Bayesian inference is employed to update prior knowledge about model parameters in light of observational data. It provides a probabilistic framework to quantify uncertainty in parameter estimation.

MCMC Sampling

Markov Chain Monte Carlo (MCMC) methods are used to sample from the posterior distribution. The module supports ensemble samplers (e.g., `emcee`), which utilize multiple walkers to efficiently explore the parameter space in parallel.

Assumptions in the hybrid modeling procedure

The hybrid modeling framework operates under several simplifying assumptions to ensure tractability and facilitate implementation within the platform. First, it is assumed that the input parameters—whether physical, geometric, or material—are statistically independent, which simplifies the prior modeling and surrogate construction process. This assumption may not hold in all practical cases but allows for a more efficient sampling and training pipeline. Second, the measurement noise affecting the observed quantities of interest (QoIs) is modeled as additive, and normally distributed, with independent errors across observations. The current user interface supports only Gaussian likelihood models and does not allow specification of correlated noise, assuming that there is no systematic model discrepancy between the physical (e.g., FEM-based) simulations and the true underlying system behavior. In other words, the physical model is considered sufficiently accurate for generating training data and is treated as a surrogate for the real-world process without explicit modeling of bias or structural error. These assumptions are common in surrogate-assisted UQ workflows, but users should be aware of their limitations when interpreting results or extending the framework to more complex scenarios.

9.4.2 Calculations and Formulas

Bayes' Theorem

The posterior distribution of model parameters**x** given observed (or targeted) data $\mathbf{z}_m$ is computed using Bayes' theorem:

$$\pi_{\mathbf{X}|\mathbf{z}_m}(\mathbf{x}) = \frac{\pi_{\mathbf{z}_m|\mathbf{x}}(\mathbf{x})\pi_{\mathbf{X}}(\mathbf{x})}{\underbrace{\int_{I_{\mathbf{X}}} \pi(\mathbf{z}_m|\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x}}_{e}}, \tag{9.4}$$

where:

- $\pi_{\mathbf{X}}(\mathbf{x})$: Prior distribution of parameters,
- $\pi_{\mathbf{z}_m|\mathbf{x}}(\mathbf{x})$: Likelihood of the observed data given the parameters,
- e: Evidence (normalizing constant, so that the resulting posterior distribution integrates to one),
- $\pi_{\mathbf{X}|\mathbf{z}_m}(\mathbf{x})$: Updated, posterior distribution of the input parameters after learning measurement $\mathbf{z}_m$.

For a more detailed discussion of the algorithms used and a description of the MCMC sampling sampling, interested readers are referred to e.g. [4].

Likelihood Function

Assuming Gaussian observation noise with zero mean and a diagonal covariance of the error terms with known, diagonal elements of variances σ_i^2, the likelihood function is:

$$\pi_{\mathbf{z}_m|\mathbf{x}}(\mathbf{x}) \propto \prod_i \exp\left(-\frac{1}{2\sigma_i^2}\|z_{m,i} - \tilde{f}(\mathbf{x})_i\|^2\right), \tag{9.5}$$

where $z_{m,i}$ and $\tilde{f}(\mathbf{x})_i$ddenote the i-th components of the measured quantity and the predicted quantity of interest (QoI), respectively.

9.5 Procedures, Data Workflows

9.5.1 Procedures, User Workflow

The main goal of the ***Hybrid Digital Twinning*** module is to calibrate uncertain input parameters of a simulation model using observed measurements of its output, enabling the training of a digital twin of the building structure. The module accomplishes this in two stages: (1) generating a surrogate model of Finite Element (FE) simulations to support sensitivity analysis, and (2) calibrating input parameters through Bayesian inversion using observed output data.

The workflow begins by specifying uncertain input parameters, modeled as independent random variables with initial distributions based on best engineering judgment taking into account available measurements, manufacturers data. These distributions can be defined interactively via the `TRACE-Structures` UI or programmatically using the SimParamSet class and associated distribution classes in the Python library. Once defined, parameter samples are drawn using various sampling techniques (e.g., Latin Hypercube, Sobol sequences).

These sampled inputs are then propagated through a local execution of the simulation model—typically parametric FE analyses of the structure—to generate input–output pairs. This dataset is used to train a computationally efficient surrogate model that approximates the mapping from inputs (e.g., material stiffness, boundary conditions) to structural responses (e.g., natural frequencies, displacements).

With the surrogate model in place, global sensitivity analysis is performed to quantify the influence of each input parameter on the output variance. Techniques such as Sobol indices and SHAP (SHapley Additive exPlanations) values are supported, enabling both global and local interpretability of model behavior. This information helps prioritize parameters for calibration and improves transparency in decision-making.

Subsequently, Bayesian calibration is applied using observed modal data from measurements or laboratory testing. The surrogate model acts as a fast forward model during the calibration process, where MCMC (Markov Chain Monte Carlo) sampling is used to infer posterior distributions of uncertain input parameters. These updated distributions reflect the uncertainty reduction achieved through data assimilation and can be used for downstream predictive tasks or refinement of future simulation design spaces.

This module complements the data-driven pipeline by incorporating physics-based simulation and prior expert knowledge, while retaining explainability and uncertainty quantification through surrogate modeling and Bayesian inference. The complete data workflow is illustrated in Fig. 9.2.

9.5.2 Data Flow, Data Formats

A more detailed overview of the data flow is provided in Fig. 9.3. The figure distinguishes between directly user-provided input data, and data imported into the user interface from either the BUILDCHAIN DKG or local storage, and the resulting outputs. Each of these components is described in detail below.

Input retrieved from the DKG or from local drive

- **Measured QoIs** (e.g. Mode Shapes and Frequencies):
  - **Format:** Universal File Format (`.unv`), or `.json` or `.csv`
  - **Content:** Experimental data, in the BUILDCHAIN project typically modal data ideally corrected one, from which environmental effects were removed (created with Module 1)
- **Variance of measurement (and modeling) error**:
  - **Format:** `.csv`

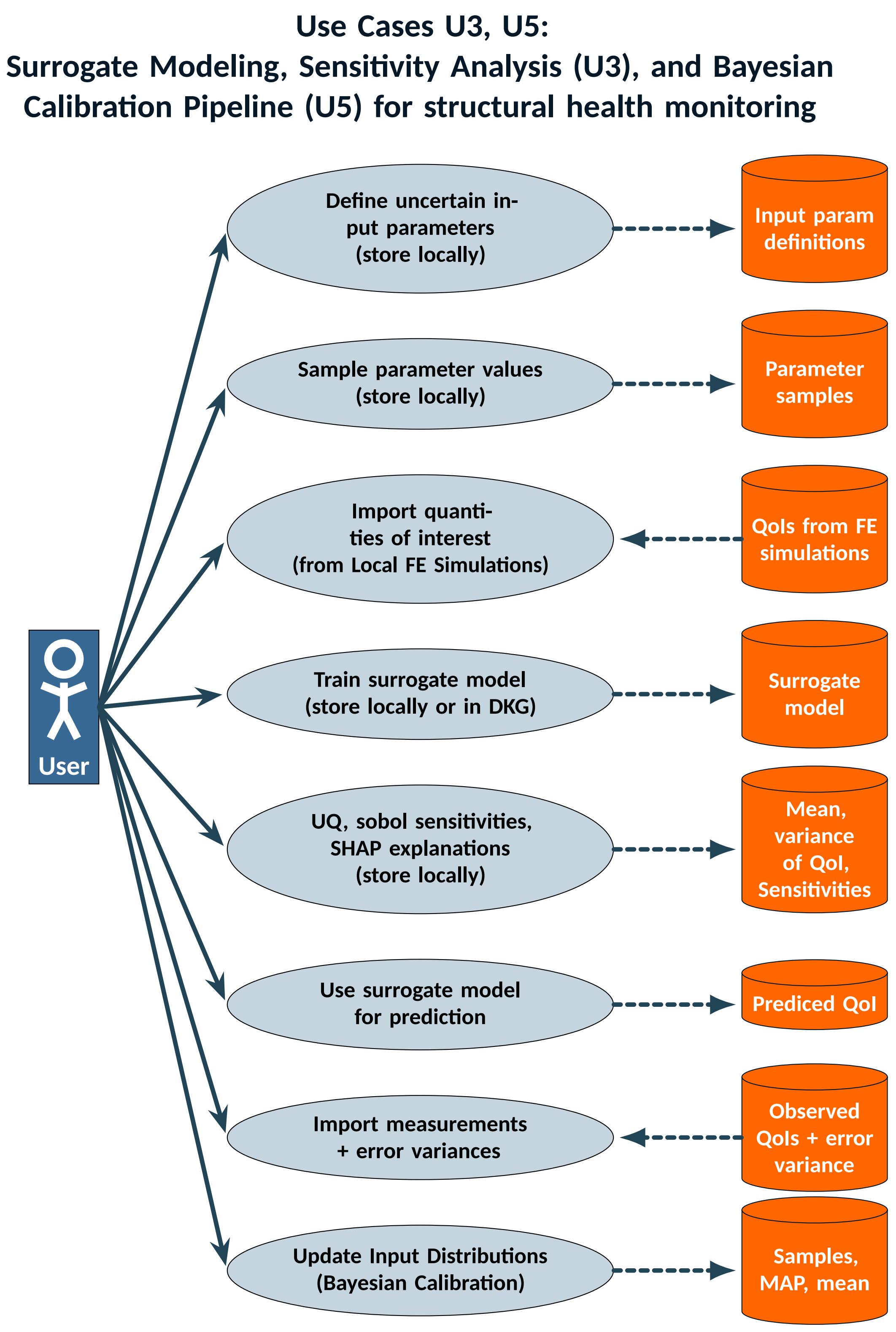

Use Cases U3, U5:
Surrogate Modeling, Sensitivity Analysis (U3), and Bayesian Calibration Pipeline (U5) for structural health monitoring
User
Define uncertain input parameters (store locally)
Input param definitions
Sample parameter values (store locally)
Parameter samples
Import quantities of interest (from Local FE Simulations)
QoIs from FE simulations
Train surrogate model (store locally or in DKG)
Surrogate model
UQ, sobol sensitivities, SHAP explanations (store locally)
Mean, variance of QoI, Sensitivities
Use surrogate model for prediction
Prediced QoI
Import measurements + error variances
Observed QoIs + error variance
Update Input Distributions (Bayesian Calibration)
Samples, MAP, mean

Figure 9.2: User workflow of module 5: Hybrid digital twinning.

- **Content:** Variances of the components of measurement error ε

Other internal/direct user inputs

- **Uncertain Input Parameters:**
  - **Format:** Created with the `SimParamSet` python class, or by direct inputs on the `TRACE-Structures` UI (see Sec. 9.9.3). The created description of parameters is exportable to and later importable again from a `.json` file.
  - **Content:** List of parameter names and associated distributions (possible distributions are: normal, uniform, lognormal, beta distributions).

```
{
  "params": [
    { "name": "density",
    "distribution": "uniform",
    "distParam1": "0", "distParam2": "5"},
    { "name": "E_modulus",
    "distribution": "normal",
    "distParam1": "2", "distParam2": "3"},]
}
```

Listing 9.1: Example JSON input for parameter definition

- **Configuration of sampling procedure**
  - **Format:** distinct inputs of the `SimParamSet.sample` function or direct inputs in the UI;
  - **Content:** Number of samples (depends on the smoothness of the $f_{\text{FEM}}(\mathbf{x})$ function and on the dimension of the parameter space n), model of sampling (Monte Carlo (MC), Quasi Monte Carlo (QMC) - Halton sequence, QMC-Latin hypercubic sampling. optionally the random seed used for the random sampling techniques (by fixing that the results are repeatable).
- **QoI Values from Simulation Model:**
  - **Importable formats:** Multiple `.unv` files with hashcodes encoding parameter values, or one single `.unv` file with multiple loading cases, or simply a `.csv` file.
  - **Content:** Simulation results (e.g., modal frequencies and shapes) for all samples of parameter configurations (data should have as many rows of QoIs, as many samples were generated, the number of columns is the number of QoIs).
- **Model Training Configuration:**
  - **Format:** input setting can be saved to a `.json` file, but the hyperparaemters are available as a part of the predictive model. `PredictiveModel`
  - **Content:** Type of model, hyperparameters, train-test split ratio, and hyperparameters of the selected model.

9.5.3 Intermediate Results (Not Stored as KAs)

- **Sampled Parameter Sets:**
  - **Format:** `.csv`
  - **Content:** Matrix of sampled input values; filenames or hashcodes can encode individual samples
- **Statistics and sensitivity outputs:**
  - **Format:** data in `csv`, vizualization of results in `.png`
  - **Content:**
    - ∗ Mean value and variance of outputs: table with mean and variance for each component of the QoI
    - ∗ Global Sensitivities (Sobol): sobol indices and partial variances of each input parameters and for each element of the QoI;
    - ∗ Local Explanations (SHAP): averaged, global SHAP contributions and individual attributions of the different input parameters for one specific prediction.

9.5.4 Outputs, that may be stored in the DKG

- **Surrogate Model:**
  - **Format:** Python pickle format (`.pkl`) with a custom extension `.spm`
  - **Content:** regression model trained to approximate FEM simulations;
  - **DKG Integration:** can be stored optionally as a ”document” knowledge asset attached to the building knolwedge asset or later to a separate knowledge asset of the metadata of the model stored in `jsonld`.
- **Updated Material Properties:**
  - **Format:** samples in `.csv`, one point estimates and statistics in `json`;
  - **Content:** Posterior samples or summary statistics (mean, std, percentiles, maximum a-posteriori estimate (MAP) and mean posterior) of calibrated parameters
  - **DKG Integration:** the `json` file can be stored optionally as an attached document in the DKG.

All data types and their integration to the BUILDCHAIN DKG are listed in Table 9.1.

9.6 Technical Architecture, Python classes

Dedicated Python classes with various functionalities were developed to support the digital twinning workflows of Modules 1–3. The architecture of the digital-twinning Python library package includes classes that cover the entire workflow —- from surrogating physics based models to global and local sensitivity analysis, as well as Bayesian updating of the input parameters of the simulation model. These classes are illustrated through UML diagrams presented in Annex B, which highlight the key classes and their associated functionalities.

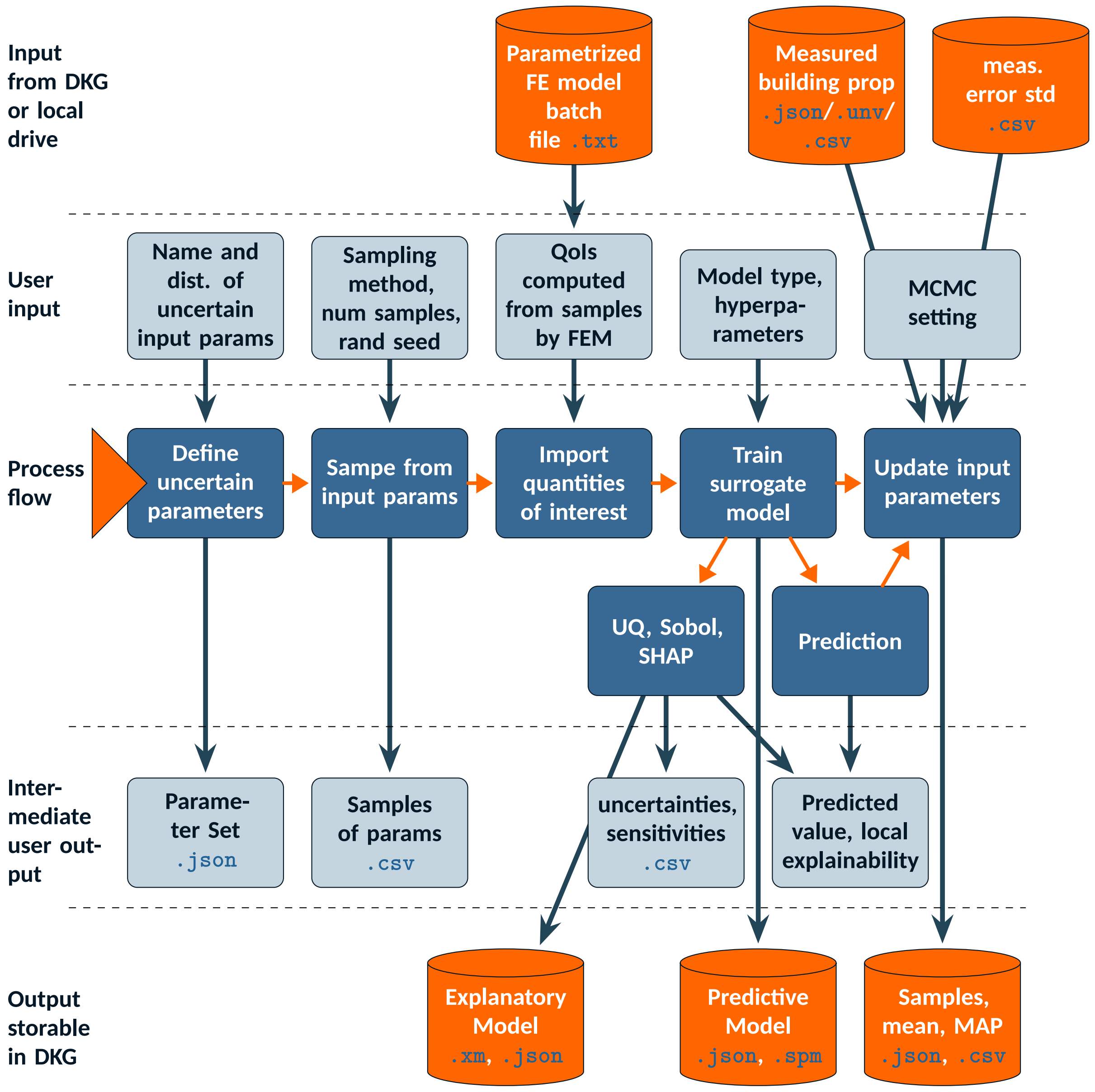

Input from DKG or local drive
Parametrized FE model batch file .txt
Measured building prop .json/.unv/.csv
meas. error std .csv
User input
Name and dist. of uncertain input params
Sampling method, num samples, rand seed
QoIs computed from samples by FEM
Model type, hyperparameters
MCMC setting
Process flow
Define uncertain parameters
Sampe from input params
Import quantities of interest
Train surrogate model
Update input parameters
UQ, Sobol, SHAP
Prediction
Intermediate user output
Parameter Set .json
Samples of params .csv
uncertainties, sensitivities .csv
Predicted value, local explainability
Output storable in DKG
Explanatory Model .xm, .json
Predictive Model .json, .spm
Samples, mean, MAP .json, .csv

Figure 9.3: Data flow of hybrid digital twinning module – defining uncertain input parameters, surrogate modeling, sensitivity analysis, and Bayesian calibration. Input data can be uploaded or defined manually. Outputs such as trained models, sensitivity indices, and posterior distributions can be saved locally or in the BUILDCHAIN DKG.

Data type	Description	csv	json	txt	pickle: .xm/ .spm	Integration	User In-put	Stored in DKG	Additional Info
Input									
Uncertain input parameters	Name and distribution of input parameters		x			none, given by direct user input	x		stored info in surrogate model can be stored directly in `json`
Sampling config	Sampling method, number of samples, seed		x			none, given by direct user input	x		can be stored directly in `json`
Computed QoIs	Computed quantities of interest from FEM	x				none, uploaded from local drive	x		stored as part of the surrogate model
Measured structural response	Experimental output	x	x			input from DKG/local drive			
Model type and hyperpa-rameters	String or scalar values		x			none, given by direct user input	x		
Parametrized FE model	physics based model			x		can be stored on DKG/local hardware			not directly needed as an input, used by the user, batch file in .txt
Measurement error std	Standard deviation of each measurement	x				input from DKG/local hardware			Describes how trustful the measurements are
MCMC setting	For updating parameters		x			none, direct user input	x		used in Bayesian update
Output									
Parameter samples	Sampled parameter values		x			not stored in DKG			
Explanatory model	SHAP explanatory model		x		x (.xm)	to DKG/local hardware		x	`xm` model can be linked, meta data in `json`
Predictive model	Surrogate regression model		x		x (.spm)	to DKG/local hardware		x	`spm` model can be linked, meta data in `json`
Updated parameters	samples of the posterio distribution of parameters, one point estimates	x	x			to DKG/local hardware		x	includes one-point estimates as MAP or mean posterior
Resulting images	Vizualization of results (variance, sensitivites, SHAP explanations								vizualized results can be stored on local drive

Table 9.1: Input and output data types, formats, and integration in the hybrid modeling digital twinning module.

9.7 Demo, Alignment with BUILDCHAIN Goals

This module was developed specifically for the BUILDCHAIN project's Use Cases U5, titled *Structural Health Monitoring (SHM) tools for cultural heritage buildings*, U6, titled *Follow-up procedures of design processes by Bayesian updating* and U8, titled *Optimized 'self-attentive' sensing*. However, both the underlying library package and the user interface have been designed to be flexible and broadly applicable. They can be used to train any type of hybrid model based on input-output data from a physics-based simulation model—- not limited to replacing finite element models (FEM) of structural behavior —- and to calibrate uncertain input parameters of such models using measurements of quantities of interest derived from simulation outputs.

The module has been tested and validated using both a simplified example —-calibrating the stiffness of a spring and the mass of an object by measuring the displacement in a spring–mass system —- and a real-world application within the BUILDCHAIN pilot project: the digital twinning of a timber building as part of Pilot 5.

9.7.1 Toy Model: calibrating oscillator model

To demonstrate and test the functionality of the module, a simplified toy model was developed based on classical dynamics: a mass–spring system. The physics-based simulation model is represented analytically using the equations of motion for a single degree-of-freedom system, where a mass is attached to a linear spring. The system is displaced from its equilibrium position by a unit distance and then released.

In this setup, the uncertain input parameters are the spring stiffness k and the mass m of the object

$$\mathbf{x} = [k, m]^T. \tag{9.6}$$

The displacement of the mass over time, $u(t)$, serves as the quantity of interest (QoI) at some specific snapshots

$$\mathbf{y} = \begin{bmatrix} u(t_1) \\ u(t_2) \\ \vdots \\ u(t_m) \end{bmatrix}. \tag{9.7}$$

This displacement response can be computed analytically as a function of k and m, assuming negligible damping and initial conditions $u(0) = 1$ and $\dot{u}(0) = v_0$:

$$y_i = u(t_i) = \cos(\omega t_i) + \frac{v_0}{\omega} \sin(\omega t_i), \tag{9.8}$$

Bayesian inversion is used to calibrate the uncertain parameters by incorporating observed displacement values at the selected time snapshots. These synthetic measurements

$$\mathbf{z}_m = \mathbf{y} + \varepsilon \tag{9.9}$$

(with or without added noise ε) are compared against the analytical model predictions, and the posterior distributions of k and m are inferred using Markov Chain Monte Carlo (MCMC) sampling and some uniform a-priori distribution of these parameters. This example serves as a controlled benchmark for validating the surrogate modeling, sensitivity analysis, and calibration workflows.

9.7.2 Design-procedure update - the Yoker Tall Timber Building

As a real-world demonstration, the module was applied to a pilot case within the BUILD-CHAIN project involving a tall timber building constructed using cross-laminated timber (CLT) panels. In this pilot, the uncertain input parameters correspond to design-level material properties, which implicitly account for the modified behavior at the connections between CLT panels. These properties are modeled in a homogenized (smeared) manner, acknowledging that joint behavior significantly affects the global structural dynamics.

A deterministic finite element (FE) model of the seven-storey Yoker building was used to compute its modal properties. The uncertain model parameters are defined as

$$\mathbf{x}_1 = [e_1, e_2, e_3, g_1, g_2, q], \tag{9.10}$$

where e_1, e_2, and e_3 are the elastic moduli in the major material directions, g_1 and g_2 are the in-plane shear moduli, and q represents the distributed mass. The observed outputs include natural frequencies and mode shapes obtained from forced vibration tests, denoted by

$$\mathbf{y}_1 = \left[f_1, f_2, f_3, f_4, f_5, \psi_1^T, \psi_2^T, \psi_3^T, \psi_4^T, \psi_5^T\right]^T = f_{1,\text{FEM}}(\mathbf{x}_1). \tag{9.11}$$

Measurement noise was modeled as Gaussian, with variances estimated from operational modal analysis.

To avoid the high computational cost of repeated FE simulations during parameter calibration, a surrogate model was developed to approximate the mapping from uncertain input parameters to the building's dynamic response. This surrogate enabled rapid evaluation of the structural model's modal properties based on input samples. Once measured data from vibration testing or structural monitoring became available, Bayesian updating was applied to refine the distributions of the uncertain input parameters, thereby improving the accuracy and predictive capability of the building's digital twin.

In addition, Module 3—developed specifically for this pilot case—extended the approach to a multi-building framework by coupling the surrogate model of the Yoker building with that of another similar CLT structure. This facilitated a joint Bayesian update of shared uncertain parameters across the two buildings, promoting generalisability and enabling transferable calibration. By leveraging structural similarities, this multi-building inference framework enhances robustness and consistency in parameter identification for CLT systems.

9.8 Step by Step Library Package User Guide for the Module

The TRACE-Structure Python library package supports the full digital twinning workflow described in this chapter. The package provides functionality for the definition of the uncertain input parameters, sampling from the defined parameter set, import of quantity of interests as well as the measurements and its error variances and finally the Bayesian updating of the parameters.

To demonstrate the library's capabilities, two example Jupyter notebooks are provided for a step-by-step user guide that are annexed in Annex C.2:

Notebook 1 (see Annex C.2.1): This notebook present the toy model example explained in Section 9.7.1, the digital twining of a one degree freedom oscillator. This example illustrates the complete functionality of the library using synthetic measurement data and helps to understand each step of the pipeline, including data generation, model training, uncertainty quantification, sensitivities, explainability, and model parameter updating.

Notebook 2 (see Annex C.2.2): An application to real-world data for the digital twinning of the Yoker building ((see Section 9.7.2), part of the BUILDCHAIN Pilot 5. This notebook showcases how the library can be applied to surrogate the FE model computing the modal properties of a timber building, and how measurements of the modal properties can be used to calibrate the input parameters of the model.

These notebooks serve as practical guides for both validating the approach and facilitating adoption in other use cases.

9.9 Step by Step API User Guide for the Module

This section presents in a step-by step manner how the hybrid-modeling module of the TRACE-Structures BUILDCHAIN UI can be applied. To carry out the full workflow of digital twining of a building structure using the API, users should follow the steps below:

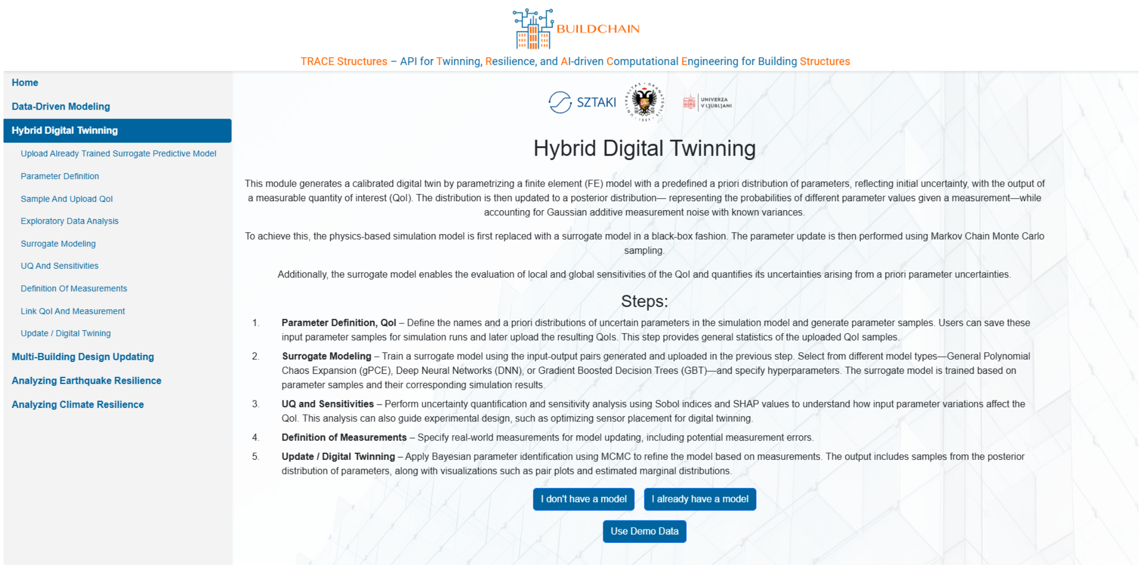
1. **Accessing the BUILDCHAIN web interface** by visiting the `TRACE-Structures` web-interface;
2. **Module Selection** to start the ***Hybrid Digital Twinning*** module;
3. **Definition of Uncertain Input Parameters** specifying the names and probability distributions of uncertain model parameters;
4. **Configuration of Sampling Settings** by selecting a sampling method and specifying the number of samples;
5. **Generation of Samples** to generate and store samples of uncertain parameters;
6. **Off-line computation of Quantities of Interest (QoIs)** by run simulations (e.g., FEM) using the sampled parameters and a parameterized FE model. **Upload of resulting data and its exploratory analysis** to discover patterns correlations and dependencies, data structure.
7. **Training of the Surrogate Model** with Specified model type and hyperparameters;
8. **Uncertainty Quantification (UQ) and Explainability** to estimate mean, variance, and confidence intervals of predictions, and get local explainability of the prediction;
9. **Import Measured Structural Responses and Measurement Variances**;
10. **Bayesian Updating of Parameters** using MCMC sampling to update input parameters based on measured responses and store them on the DKG.

9.9.1 Step 1: Access the BUILDCHAIN Web Interface

Visit the TRACE-Structures interface at: `https://buildchain.ilab.sztaki.hu` and click on `Start app`.

9.9.2 Step 2: Module Selection

Navigate to the ***Hybrid Digital Twinning*** section by either clicking the blue `Hybrid Digital Twinning` button or selecting the corresponding item from the left-hand navigation menu (see upper panel of Fig 8.4). On the first page of the module (see Fig. 9.4), if you are just getting started, and no model has been trained yet, click on the `I don't have a model` button and continue with Step 3. If a surrogate model of your simulation has already been trained and saved, and the goal is only to evaluate sensitivities, view explainability results, make predictions from the trained model, or to calibrate the surrogate model, click on the `I already have a model` button and proceed to the first submodule `Upload Already Trained Surrogate Predictive Model`. This way, you can import the already trained model and skip the parameter definition, sampling and data import steps (steps 3-7) and proceed to the submodule `UQ and Sensitivities`. By clicking on `Use Demo Data`, you can load the built-in input–output dataset generated for the toy model described in Section 9.7.1. The screenshots provided in the following steps of the UI correspond to this toy model example.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition
Sample And Upload QoI
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI

Hybrid Digital Twinning

This module generates a calibrated digital twin by parametrizing a finite element (FE) model with a predefined a priori distribution of parameters, reflecting initial uncertainty, with the output of a measurable quantity of interest (QoI). The distribution is then updated to a posterior distribution— representing the probabilities of different parameter values given a measurement—while accounting for Gaussian additive measurement noise with known variances.

To achieve this, the physics-based simulation model is first replaced with a surrogate model in a black-box fashion. The parameter update is then performed using Markov Chain Monte Carlo sampling.

Additionally, the surrogate model enables the evaluation of local and global sensitivities of the QoI and quantifies its uncertainties arising from a priori parameter uncertainties.

Steps:

1. **Parameter Definition, QoI** – Define the names and a priori distributions of uncertain parameters in the simulation model and generate parameter samples. Users can save these input parameter samples for simulation runs and later upload the resulting QoIs. This step provides general statistics of the uploaded QoI samples.
2. **Surrogate Modeling** – Train a surrogate model using the input-output pairs generated and uploaded in the previous step. Select from different model types—General Polynomial Chaos Expansion (gPCE), Deep Neural Networks (DNN), or Gradient Boosted Decision Trees (GBT)—and specify hyperparameters. The surrogate model is trained based on parameter samples and their corresponding simulation results.
3. **UQ and Sensitivities** – Perform uncertainty quantification and sensitivity analysis using Sobol indices and SHAP values to understand how input parameter variations affect the QoI. This analysis can also guide experimental design, such as optimizing sensor placement for digital twinning.
4. **Definition of Measurements** – Specify real-world measurements for model updating, including potential measurement errors.
5. **Update / Digital Twinning** – Apply Bayesian parameter identification using MCMC to refine the model based on measurements. The output includes samples from the posterior distribution of parameters, along with visualizations such as pair plots and estimated marginal distributions.

Figure 9.4: Step 2 of module 2 – Getting started (lower panel)

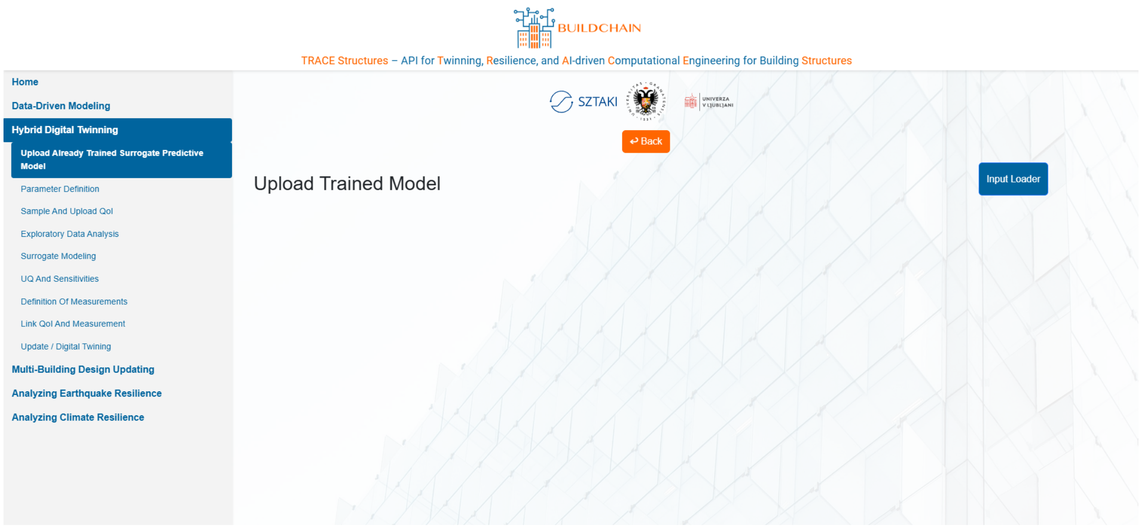

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition
Sample And Upload QoI
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Upload Trained Model
Input Loader

Figure 9.5: Step 2 of module 2 – choosing already existing surrogate model.

If you choose to begin by uploading an existing model, navigate to the `Upload Already Trained Surrogate Predictive Model` section in the left navigation bar (if it is not already selected), and click the `Input Loader` button to upload the surrogate model (see Fig. 9.5). Clicking this button will open a new data importer window, where you can choose to upload the model either from your local storage by clicking the `Choose File` button, or via the BUILDCHAIN API gateway by entering your credentials—specifically, by copying your authentication token and clicking the `Login` button (see upper panel of Fig 9.6). Once logged in, you can search for available data associated with a specific building by locating its building knowledge asset using various criteria—such as name, code, type, presence of measurements, city, DID, or zone. After entering your search criteria, click the `Get Data` button to retrieve matching building knowledge assets. Select the desired asset and click `Load Selected Building` to load the corresponding building data. Once the building is loaded, you can choose which linked knowledge asset you wish to import. To proceed, select the model knowledge asset with `.sm` extension and click the `Load` button to import it into the `TRACE-Structures` system (see lower panels of Fig 9.6).

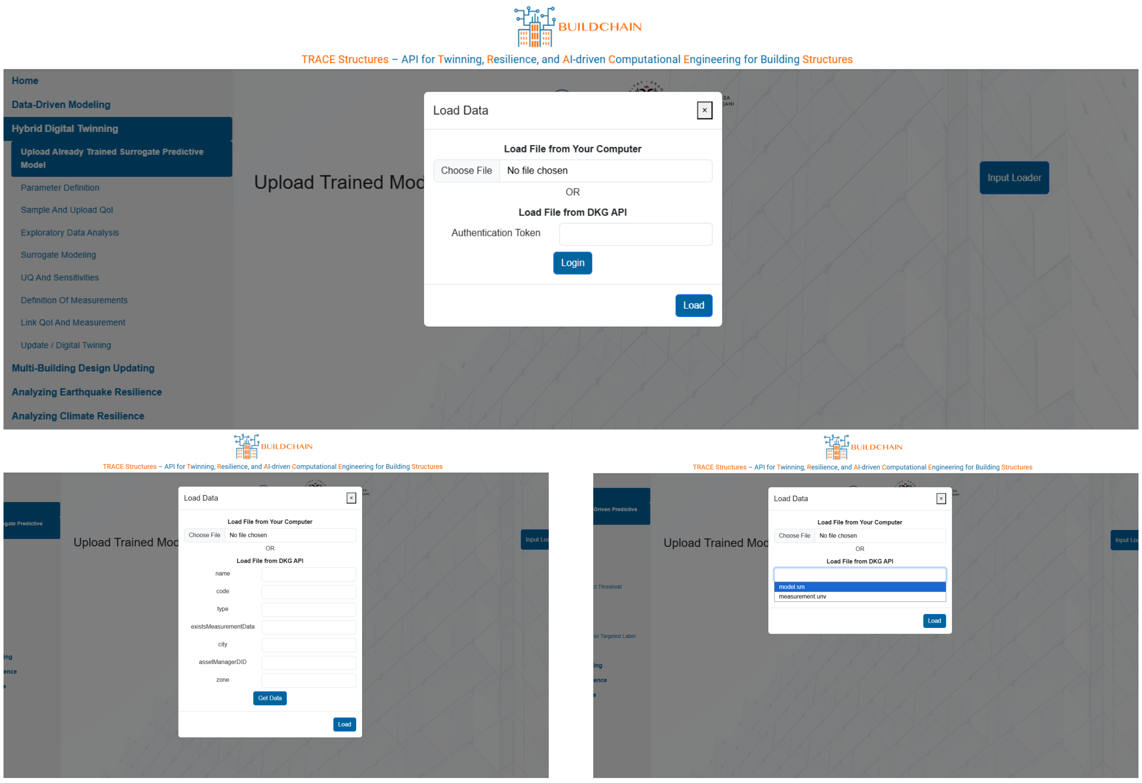
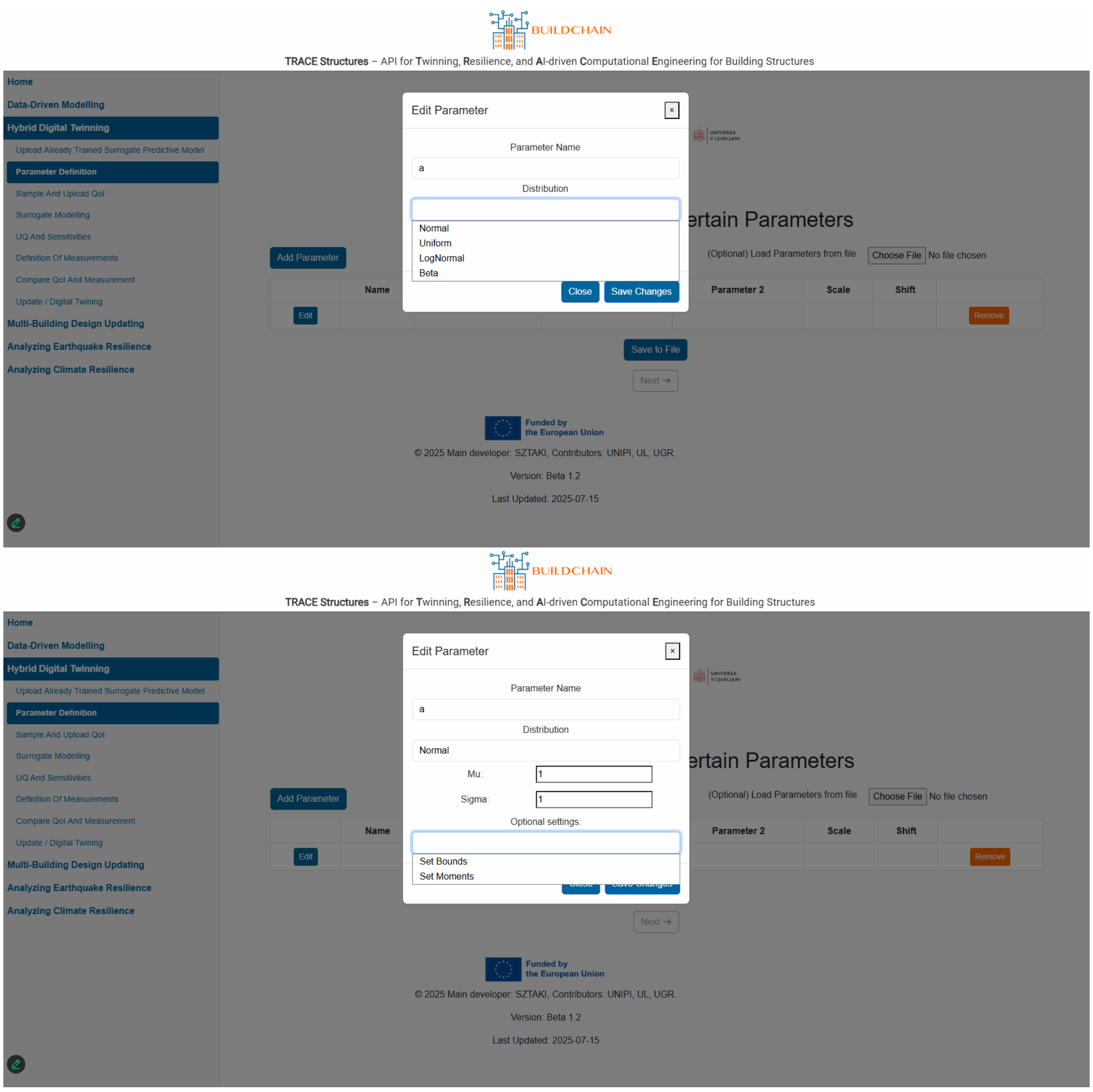


Figure 9.6: The input loader: uploading already existing knowledge asset from local storage or from the DKG via the BUILDCHAIN API

9.9.3 Step 3: Define Uncertain Input Parameters

If you do not yet have a stored surrogate model, start here. Navigate to the `Parameter Definition` sub-module using the left-hand navigation bar, if you're not already there. To begin defining parameters, click on`Add Parameter`, then click the `Edit` button on the left side of the newly added row. In the pop-up *Edit Parameter* window, specify the name and choose the probability distribution for the uncertain input parameter (see upper panel of Fig. 9.7). Available distribution types include *uniform*, *beta*, *normal*, and *lognormal*.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition
Sample And Upload QoI
Surrogate Modelling
UQ And Sensitivities
Definition Of Measurements
Compare QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Edit Parameter
Normal
Uniform
LogNormal
Beta
Close
Save Changes
Optional settings:
Set Bounds
Set Moments
Add Parameter
(Optional) Load Parameters from file
Choose File No file chosen
Save to File
Next →
Funded by the European Union
© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.
Version: Beta 1.2
Last Updated: 2025-07-15

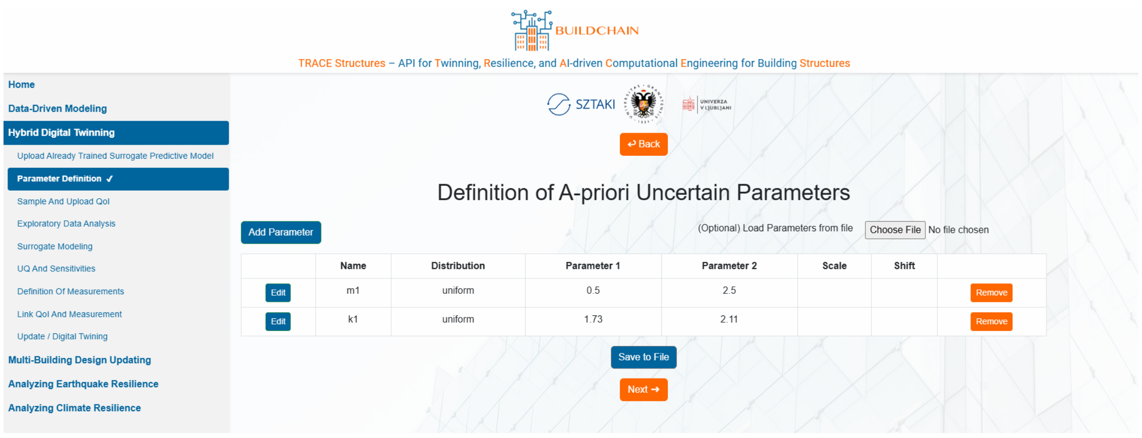
Parameter Name: a

Distribution: Normal

Mu: 1

Sigma: 1

Figure 9.7: Step 3 of module 2 – definition of name and distribution of the added parameter. The upper panel shows the definition of parameter name and the selection of distribution, and the lower panel the optional translation (scaling and shifting) of the distribution to match its bounds or moments to the desired, user-defined values.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
↩ Back
Add Parameter
(Optional) Load Parameters from file
Choose File No file chosen
Save to File
Next →

Definition of A-priori Uncertain Parameters

	Name	Distribution	Parameter 1	Parameter 2	Scale	Shift	
Edit	m1	uniform	0.5	2.5			Remove
Edit	k1	uniform	1.73	2.11			Remove

Figure 9.8: Step 3 of module 2 – definition of uncertain parameters.

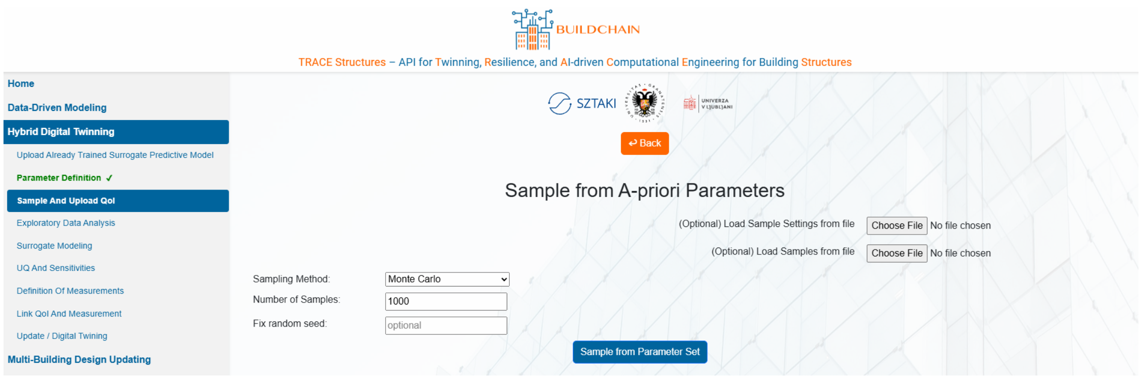
Some distributions—such as the Beta distribution—have fixed bounds, and in certain cases, additional computations are required to match specific statistical characteristics, such as desired bounds, quantiles, or moments (such as mean and variance). You can enable this by clicking into the box under *Optional Settings*(see lower panel of Fig. 9.7), where you can define these constraints. When setting bounds for an unbounded or semi-bounded distribution, the backend algorithm adjusts the distribution so that the 2nd and/or 98th percentile aligns with the specified bound value

Repeat this process to add and configure as many parameters as needed (see defined parameters for the toy model in Fig. 9.8.

The user can save the defined parameter set to a `.json` file by clicking the `Save to File` button. Alternatively, if a parameter set has been previously defined and saved, it can be reloaded by clicking the `Choose File` button in the upper right corner and selecting the corresponding `.json` file. Once finished defining the parameters, continue by clicking on the `Next` button.

9.9.4 Step 4: Configure Sampling Settings

Select a sampling method (see Fig. 9.9) to generate realizations from the defined input parameter distributions. The available options include: i) **Monte Carlo (MC)**, that draws random samples, providing a straightforward approach to approximate the underlying uncertainty; ii) **Quasi-Monte Carlo (QMC)** methods, that use deterministic low-discrepancy sequences, or quasi random sampling techniques that cover the sampling space more uniformly than purely random sampling. These include: a) *Latin Hypercube Sampling (LHS)*, ensuring that the entire range of each input distribution is sampled by stratifying the space into intervals of equal probability; b) *Halton and Sobol sequences*: widely used low-discrepancy sequences for multidimensional sampling, providing better convergence rates than standard Monte Carlo in many cases; c) *Saltelli Sampling*: a variance-based sampling method particularly suited for global sensitivity analysis, enabling efficient estimation of Sobol indices.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Choose File
Choose File
Sample from Parameter Set

Sample from A-priori Parameters

(Optional) Load Sample Settings from file — No file chosen

(Optional) Load Samples from file — No file chosen

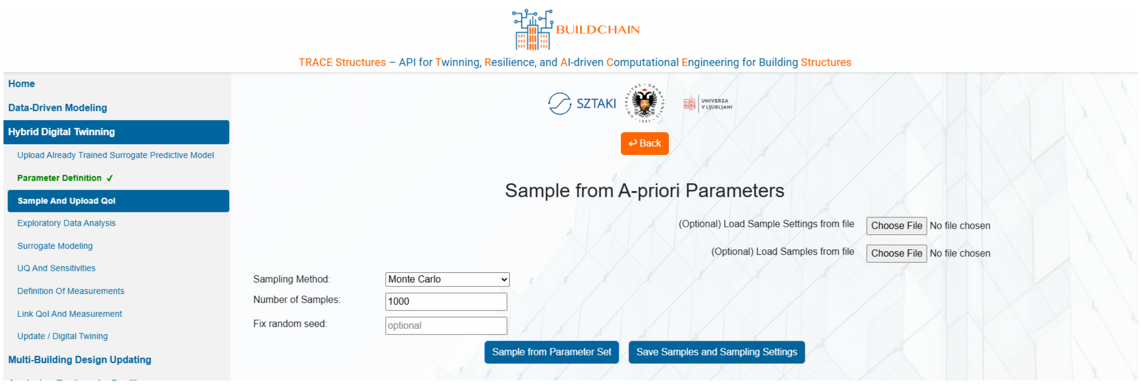
Sampling Method:	Monte Carlo
Number of Samples:	1000
Fix random seed:	optional

Figure 9.9: Step 4 of module 2 – Configure Sampling Settings

In addition to selecting the sampling method, specify the number of samples to be generated and set a random seed to ensure the reproducibility of the sampling process. Alternatively, if samples have already been generated and saved previously, the user can reload them by clicking the `Choose File` buttons in the upper-right corner of the window and selecting the appropriate files (see Fig. 9.9).

9.9.5 Step 5: Generate Samples

Using the defined input parameter distributions and sampling configuration, generate the parameter samples by clicking on the `Sample from Parameter Set` button (see Fig. 9.9).


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
SZTAKI
Back
Choose File
Sample from Parameter Set
Save Samples and Sampling Settings

Sample from A-priori Parameters

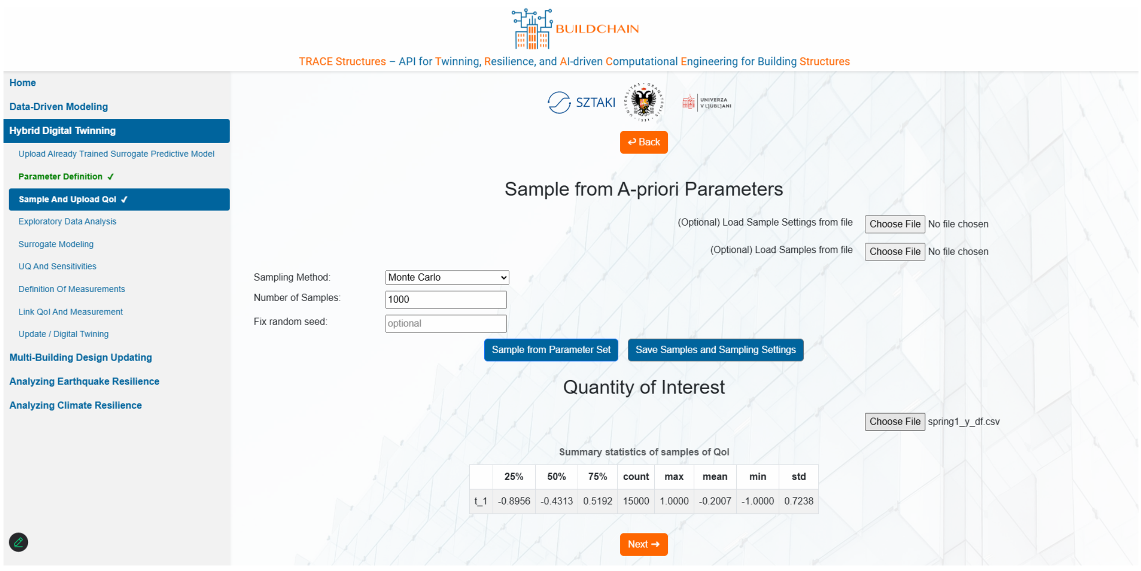
(Optional) Load Sample Settings from file: No file chosen
(Optional) Load Samples from file: No file chosen
Sampling Method: Monte Carlo
Number of Samples: 1000
Fix random seed: optional

Figure 9.10: Step 5 of module 2 – Sample from the uncertain parameter set using the defined sampling configuration

After sampling is complete, the user can save the results by clicking the `Save Samples and Sampling Setting` button (see Fig. 9.10). This action will export the generated samples to a `.csv` file and the corresponding sampling configuration to a `.json` file.

9.9.6 Step 6: Compute Quantities of Interest, Explore Data

The following substep is performed offline: the user must evaluate the quantity of interest (QoI) at each sample point using an external simulation model (e.g., a finite element (FE) model).


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
Back
Choose File
Sample from Parameter Set
Save Samples and Sampling Settings
Next ➜

Sample from A-priori Parameters

(Optional) Load Sample Settings from file: No file chosen
(Optional) Load Samples from file: No file chosen
Sampling Method: Monte Carlo
Number of Samples: 1000
Fix random seed: optional

Quantity of Interest

Choose File: spring1_y_df.csv

Summary statistics of samples of QoI

	25%	50%	75%	count	max	mean	min	std
t_1	-0.8956	-0.4313	0.5192	15000	1.0000	-0.2007	-1.0000	0.7238

Figure 9.11: Step 6 of module 2 – Uploading the samples of the Quantities of Interest computed from the generated input samples by a simulation model, and evaluation of its statistics

Once the simulations have been completed and the QoI values corresponding to each parameter sample (in the same order as in the parameter sample file) have been computed, the user can upload the results by clicking the `Choose File` button under the *Quantity of Interest* section (see Fig. 9.11).

Importantly, it is not a problem to interrupt the session before completing this offline sub-step. As long as the parameter set, the generated samples, and the sampling configuration have been properly saved, the process can be resumed later in a new session without any loss of information. These saved files can be reloaded via the corresponding `Choose File` buttons.

After the QoI sample file is uploaded, the interface will display basic statistics for each QoI, including the 25th, 50th, and 75th percentiles, the number of samples, as well as the minimum, maximum, mean, and standard deviation (see Fig. 9.11).

Click the `Next` button at the bottom of the page to proceed to the next step, which allows for further exploration of the generated data (see Fig. 9.12) under the *Exploratory Data Analysis* sub-module. Here, users can choose which correlation values to visualize—such as parameter-to-parameter, parameter-to-QoI, or QoI-to-QoI relationships. They can also configure the scatterplot display by selecting which variable (parameter or QoI) to place on the x-axis and which on the y-axis. Optionally, users can color the scatterplot based on a third variable to enhance interpretability.

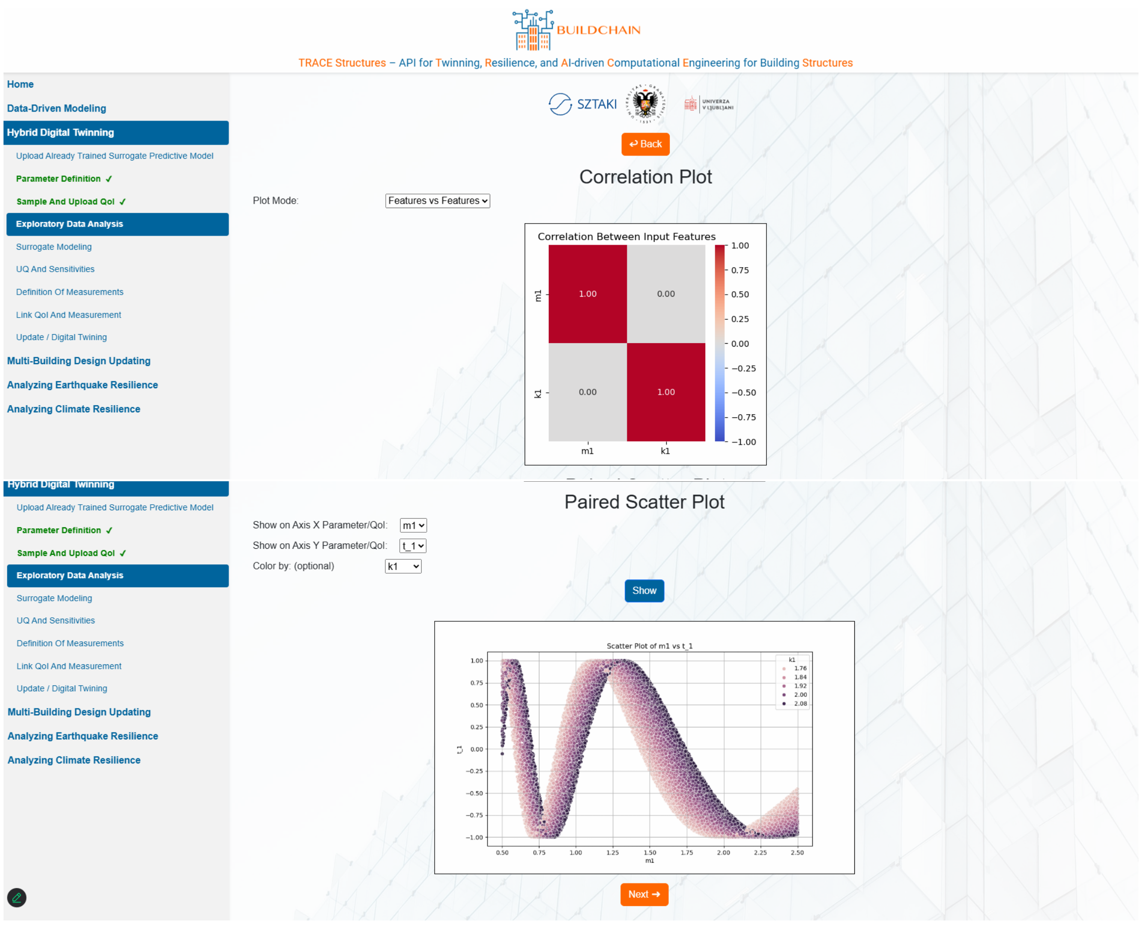
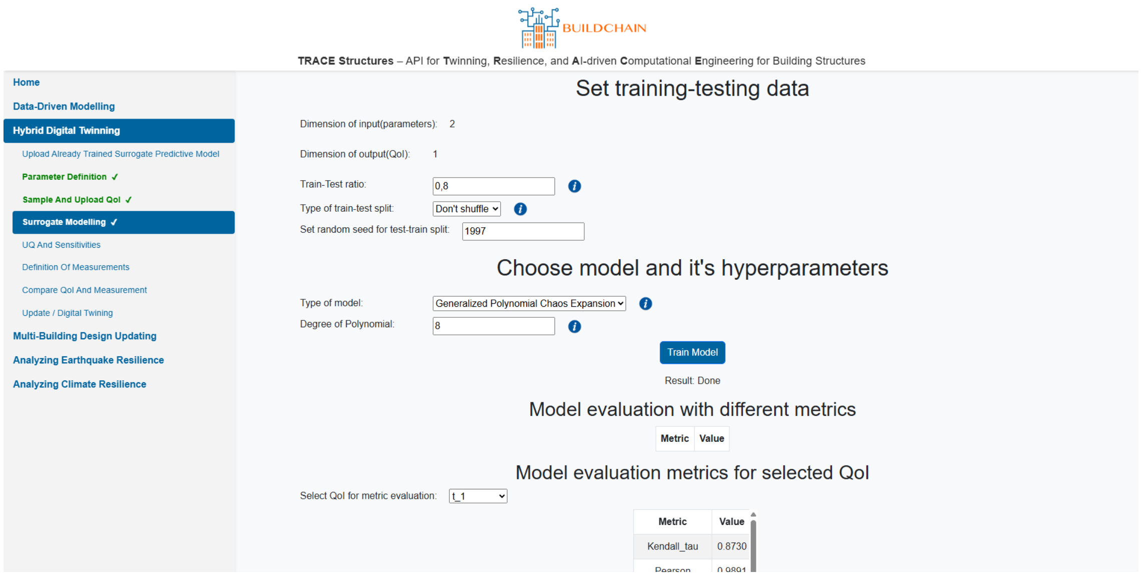

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Exploratory Data Analysis
Surrogate Modeling
UQ And Sensitivities
Definition Of Measurements
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
↩ Back
Correlation Plot
Plot Mode: Features vs Features
Correlation Between Input Features
m1
k1
1.00
0.00
0.00
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Paired Scatter Plot
Show on Axis X Parameter/QoI: m1
Show on Axis Y Parameter/QoI: t_1
Color by: (optional) k1
Show
Scatter Plot of m1 vs t_1
k1
1.76
1.84
1.92
2.00
2.08
t_1
m1
Next ➔

Figure 9.12: Step 6 of module 2 – exploratory data analysis.

9.9.7 Step 7: Train Surrogate Model

To configure the surrogate model in the moduleHybrid Digital Twinning module, the user must complete three main setup steps on the `Surrogate Modelling` page: i) define the train–test data split; ii) choose the model type (e.g., Linear Regression, GPC, GBT, or DNN); iii) set the corresponding hyperparameters.

These steps are functionally identical to the configuration described in the ***Data-Driven Module*** (see Step 5 in the step-by-step guide of Section 8.9), where the input parameters are referred to as *input features* and the quantities of interest (QoIs) as *target labels*.


BUILDCHAIN
Home
Data-Driven Modelling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Surrogate Modelling ✓
UQ And Sensitivities
Definition Of Measurements
Compare QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Train Model

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Set training-testing data

Dimension of input(parameters): 2

Dimension of output(QoI): 1

Train-Test ratio: 0,8

Type of train-test split: Don't shuffle

Set random seed for test-train split: 1997

Choose model and it's hyperparameters

Type of model: Generalized Polynomial Chaos Expansion

Degree of Polynomial: 8

Result: Done

Model evaluation with different metrics

Metric	Value

Model evaluation metrics for selected QoI

Select QoI for metric evaluation: t_1

Metric	Value
Kendall_tau	0.8730
Pearson	0.9891

Figure 9.13: Step 7 of module 2 – configuration of the training process and the surrogate model.

The TRACE-Structures UI offers an intuitive interface to guide the user through this process. (see Fig. 9.13).

9.9.8 Step 8: Perform Uncertainty Quantification and Explainability

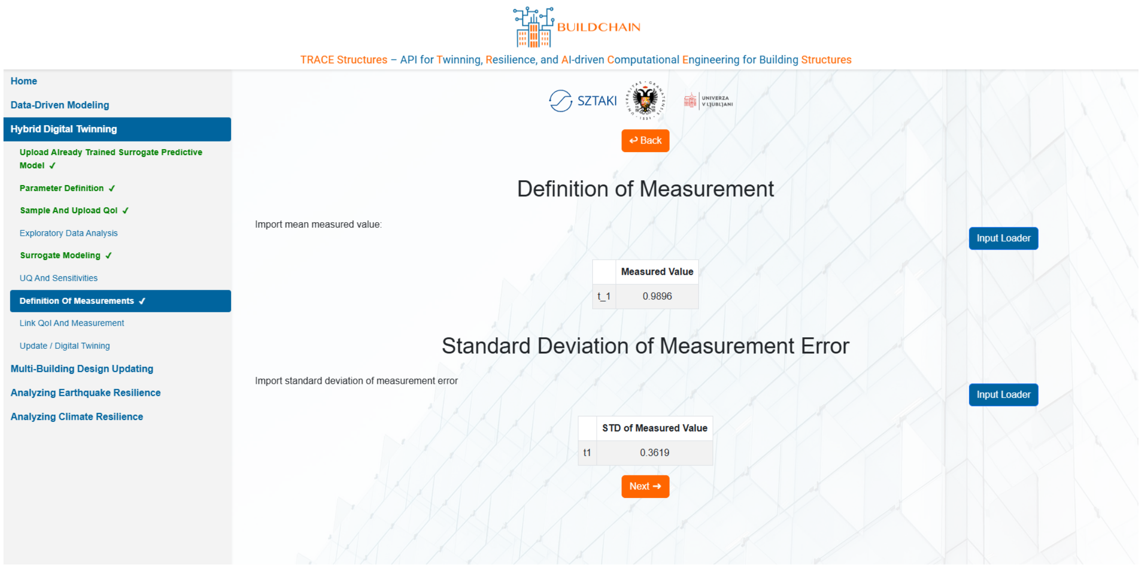
This step allows users to analyze the influence of input parameters on the variability of the model's outputs using global and local sensitivity analysis techniques. The configuration and available functionality are identical to those described in Step 6 of the *Data-Driven Module* (see Section 8.9). Users can:

- Compute statistical summaries (mean, variance, percentiles) of each Quantity of Interest (QoI);
- Evaluate global sensitivity using Sobol indices up to a chosen interaction order;
- Analyze global feature importance with SHAP values;
- Investigate local feature attributions for individual input configurations using SHAP waterfall and beeswarm plots.

These tools enable both global and instance-specific insight into how input parameters affect model outputs, supporting deeper understanding and decision-making.

For a detailed step-by-step guide with illustrations, refer to step 6 of Section 8.9 of the *Data-Driven Module*.

9.9.9 Step 9: Import Measured Structural Responses and Measurement Variances


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model ✓
Parameter Definition ✓
Sample And Upload QoI ✓
Exploratory Data Analysis
Surrogate Modeling ✓
UQ And Sensitivities
Definition Of Measurements ✓
Link QoI And Measurement
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
Back
Input Loader
Input Loader
Next

Definition of Measurement

Import mean measured value:

	Measured Value
t_1	0.9896

Standard Deviation of Measurement Error

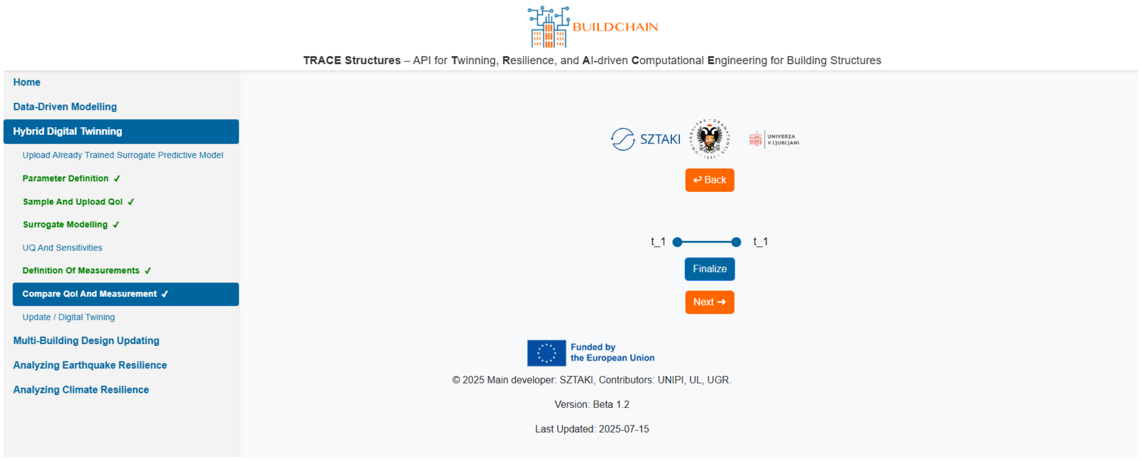
Import standard deviation of measurement error

	STD of Measured Value
t1	0.3619

Figure 9.14: Step 9 of module 2 – uploading measured values of the QoIs and the standard deviations of the measurement error model.

Load measured system responses from experimental data stored in `json` or `unv` or simple `csv` format by clicking on the `Input Loader` button on the top right side of the window (see Fig 9.14). This opens the same `Input Data Loader` window described in Step 2, used for importing existing models. Users can import data either from local storage using the `Choose File` button or by logging into the BUILDCHAIN portal. To enable probabilistic updates via Bayesian inference, the standard deviation of measurement error can be included from a separate `.csv` file.

After uploading the measurements, for a detailed step-by-step guide with illustrations how to link quantities of interest with measured variables (see Figs 9.14 9.15), refer to step 8 of Section 8.9 of the ***Data-Driven Module***.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Surrogate Modelling ✓
UQ And Sensitivities
Definition Of Measurements ✓
Compare QoI And Measurement ✓
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
Back
t_1
t_1
Finalize
Next
Funded by the European Union

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

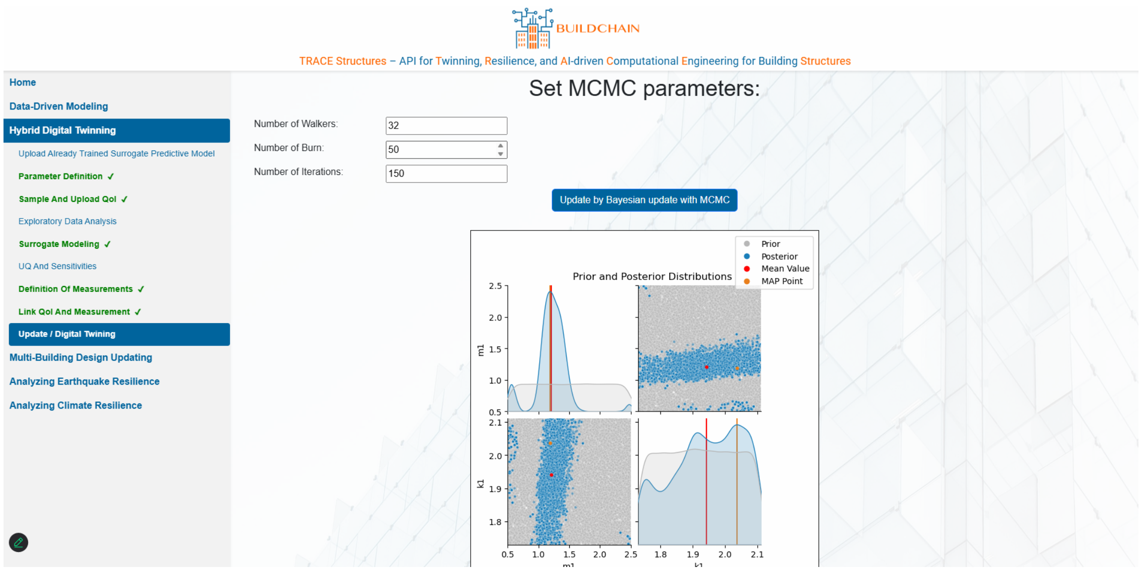
Version: Beta 1.2

Last Updated: 2025-07-15

Figure 9.15: Step 9 of module 2 – linking QoIs with measurement variables.

9.9.10 Step 10: Bayesian Updating of Parameters

The specified Quantity of Interest (QoI) values are used to initiate a Markov Chain Monte Carlo (MCMC) procedure, which samples from the resulting posterior distribution of the input parameters.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Exploratory Data Analysis
Surrogate Modeling ✓
UQ And Sensitivities
Definition Of Measurements ✓
Link QoI And Measurement ✓
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Update by Bayesian update with MCMC
Prior and Posterior Distributions
Prior
Posterior
Mean Value
MAP Point
m1
k1

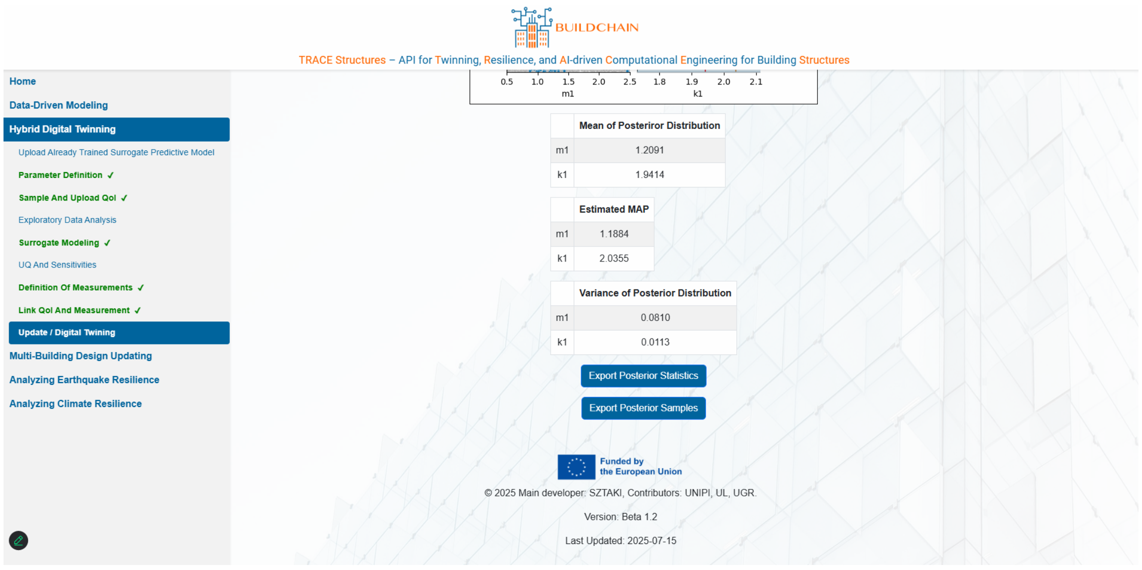
Set MCMC parameters:

Number of Walkers: 32
Number of Burn: 50
Number of Iterations: 150

Figure 9.16: Step 10 of Module 2 – MCMC configuration and posterior sampling results

This process allows for probabilistic calibration of the model by incorporating observed or desired QoI values. In this step, the user configures the MCMC sampling algorithm by selecting several key settings:

- **Number of walkers**: the number of parallel chains exploring the parameter space.
- **Burn-in steps**: the initial portion of the chain discarded to allow convergence.
- **Post-burn-in steps**: the number of iterations retained after burn-in.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Upload Already Trained Surrogate Predictive Model
Parameter Definition ✓
Sample And Upload QoI ✓
Exploratory Data Analysis
Surrogate Modeling ✓
UQ And Sensitivities
Definition Of Measurements ✓
Link QoI And Measurement ✓
Update / Digital Twining
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
m1
k1
Export Posterior Statistics
Export Posterior Samples
Funded by the European Union

	Mean of Posterior Distribution
m1	1.2091
k1	1.9414

	Estimated MAP
m1	1.1884
k1	2.0355

	Variance of Posterior Distribution
m1	0.0810
k1	0.0113

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

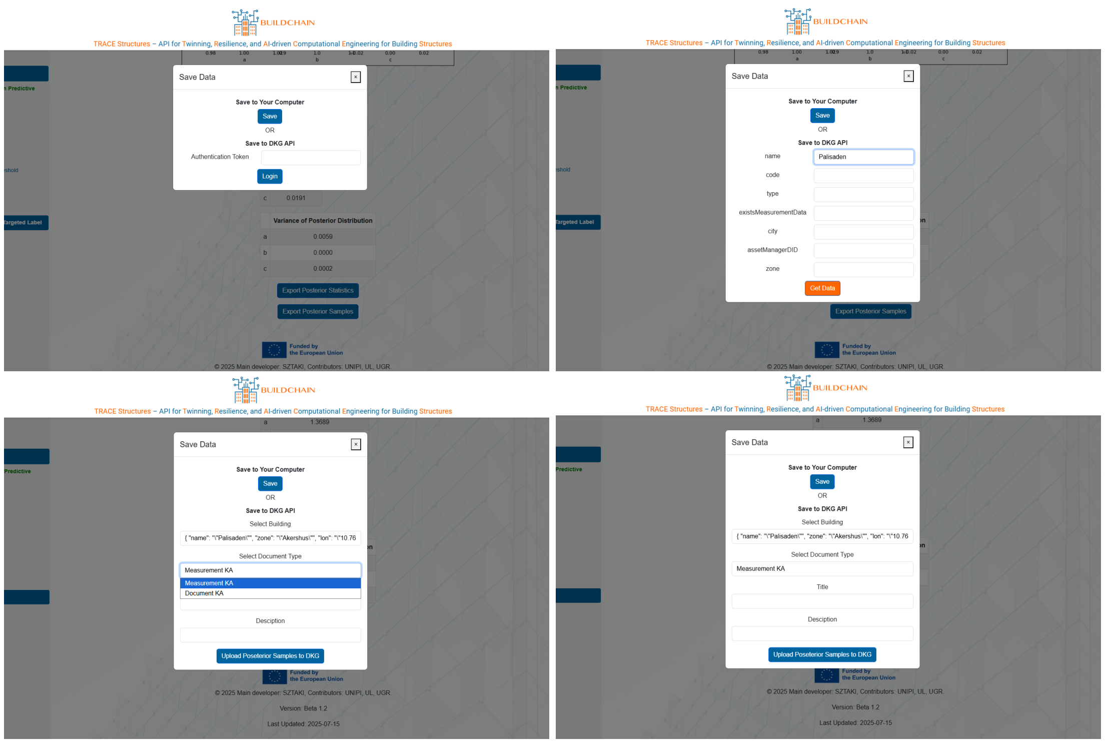
Figure 9.17: Step 10 of Module 2 – Statistics and one point estimates of the posterior distribution.

The total number of posterior samples will be the product of the number of walkers and the number of post-burn-in steps. For a more detailed step-by-step guide refer to step 9 of Section 8.9 of the *Data-Driven Module*. Once sampling is completed, the `TRACE-Structures` UI displays a paired matrix plot comparing prior input parameter samples (from the training dataset) with those drawn from the posterior distribution. It also presents key point estimates such as the posterior mean and the maximum a posteriori (MAP) estimate. These values, along with relevant summary statistics of the posterior distribution, are displayed alongside the visualizations to support interpretation (see Fig. 9.16).

The posterior samples and summary statistics can be stored by clicking the `Export Posterior Statistics` and `Export Posterior Samples` buttons. The former creates a `json` file with the mean, MAP and variance values, the later stores the samples in a `csv` file. By clicking on these buttons the UI opens the *Save Data* window, where users can choose to either save the data locally by clicking `Save`, or export it to the BUILDCHAIN DKG. To upload to the DKG, users must first authenticate by copying their BUILDCHAIN API token and logging in through the BUILDCHAIN gateway.

Once logged in—following a process similar to data import—the user selects the building or knowledge asset to which the newly created data should be linked. Knowledge assets can be found using various search criteria such as name, code, type, existence of linked measurement data, city, DID, or zone. After entering the desired criteria and clicking `Get Data`, the user can choose from the matching results.

Next, the user selects the type of Knowledge Asset (KA) for storing the exported data. Posterior samples, for example, can be saved as a *Document KA*. Upon selecting this option, the user can provide metadata including the document type, title, and description to be associated with the KA. Clicking `Save` registers a new knowledge asset and links it to the selected existing asset in the DKG.


Save Data
Save to Your Computer
Save
OR
Save to DKG API
Authentication Token
Login

	Variance of Posterior Distribution
a	0.0059
b	0.0000
c	0.0002

Save Data
Save to Your Computer
Save
OR
Save to DKG API

name	Palisaden
code	
type	
existsMeasurementData	
city	
assetManagerDID	
zone	

Get Data

Save Data
Save to Your Computer
Save
OR
Save to DKG API
Select Building
{ "name": "\"Palisaden\"", "zone": "\"Akershus\"", "lon": "\"10.76
Select Document Type
Measurement KA
Measurement KA
Document KA
Desciption
Upload Poseterior Samples to DKG

Save Data
Save to Your Computer
Save
OR
Save to DKG API
Select Building
{ "name": "\"Palisaden\"", "zone": "\"Akershus\"", "lon": "\"10.76
Select Document Type
Measurement KA
Title
Desciption
Upload Poseterior Samples to DKG

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.
Version: Beta 1.2
Last Updated: 2025-07-15
BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Export Posterior Statistics
Export Posterior Samples
Funded by the European Union

Figure 9.18: Step 10 of Module 2 – Exporting results to the BUILDCHAIN DKG

9.10 Results of Testing, Interpretation of Outputs

9.10.1 Toy Model: calibrating oscillator model

To validate the Bayesian updating functionality of the hybrid digital twinning module, a toy problem involving a single-degree-of-freedom oscillator was tested. The objective was to calibrate the stiffness (k) and mass (m) parameters of the oscillator to match a set of measured displacement values.

From the analytical expression of the oscillator's response, it is known that the displacement is governed by the ratio k/m. This means that different combinations of k and m yielding the same ratio will produce identical observable displacements. Consequently, the Bayesian updating process is expected to identify a posterior distribution concentrated along lines of constant k/m.

The results confirmed this behavior. The posterior samples of the (k, m) parameter space formed a narrow region aligned with lines of constant k/m, as theoretically expected. This validates both the correctness of the implemented MCMC algorithm and the ability of the hybrid model to learn the underlying parameter relationships from data.

The observed posterior shape also highlights an important property of the inverse problem: identifiability may be limited when only the ratio of parameters influences the outputs. This confirms the model's internal consistency and the appropriateness of the uncertainty quantification strategy employed.

These findings, visualized through pair plots of the sampled posterior (Fig. 9.15 and comparisons to the prior, demonstrate the reliability and accuracy of the hybrid modeling and calibration approach.

9.10.2 Design-procedure update - the Yoker Building

As part of the pilot testing, the hybrid digital twinning module was applied to the tall timber *Yoker* building. This test served as a preparatory step toward the broader goal of Pilot 5, which involves multi-building design parameter updating using a joint Bayesian inference procedure (as described in Module 3).

The first objective was to develop a surrogate model capable of replicating the finite element (FE) response of the Yoker building. This model acts as a computationally efficient replacement for the full-scale simulations, making the updating process tractable. Once the surrogate was trained, we proceeded to test the Bayesian updating procedure using only this single building. This intermediate step allowed us to assess the performance of the updating method in isolation before scaling it to the full multi-building framework.

The posterior distributions of selected design parameters, inferred from the observed building responses, are presented in Figure 9.19. These results offer valuable insight into the identifiability and uncertainty associated with the physical parameters of the structure. The figure displays a matrix plot, which visualizes the marginal and joint distributions of the parameters. Along the diagonal, marginal distributions are shown, while the off-diagonal elements illustrate pairwise joint distributions.

In this plot, blue points represent samples from the posterior distribution, obtained after updating the model with the observed building response data. For comparison, samples from the initial (prior) distribution, before any updating, are also included. The plot additionally highlights the posterior mean and the maximum a posteriori (MAP) estimate, providing point estimates alongside the distributional insight.

From the figure, it is evident that some parameters—particularly the material properties of the CLT panels (e_1, g_1, g_2)—are well identified, meaning that the updating signifi-

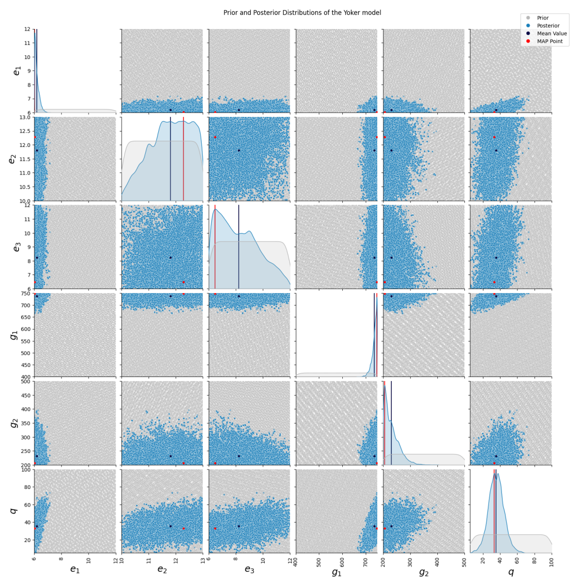


Figure 9.19: Yoker building prior and posterior distributions of design parameters

cantly reduced their uncertainty. The measurements also contain valuable information about the variable loading q acting on the structure. In contrast, the parameters e_2 and e_3 remain poorly identified, as the available measurements did not provide sufficient information to significantly reduce their posterior uncertainty.

While a single-building update does not fully exploit the potential of joint inference, it demonstrates the module's ability to integrate simulation data and measurements in a transparent and statistically consistent manner.

9.11 Strengths and Limitations

The Hybrid Modeling module presents a powerful integration of physics-based simulations with data-driven surrogate modeling to enable accurate and explainable digital twins. It leverages parameterized finite element models and combines them with machine learning techniques to predict quantities of interest under uncertainty. The inclusion of uncertainty quantification (UQ) allows the module to analyze the propagation of uncertainties in the input parameters of the model. Unlike black-box models, the Hybrid Modeling module also offers explainability through SHAP values and Sobol sensitivity indices, highlighting the influence of each input feature on the outputs. Integrated within the TRACE-Structures platform, the module benefits from a user-friendly interface suitable for usage by non-coding engineers, and seamless linkage with the Digital Knowledge Graph (DKG) for importing and registering or indexing new knowledge assets, generated by the API process.

Nonetheless, the module inherits several limitations similar to those described for Module 1, since it employs the same underlying procedure for training regression models. One key limitation is the sensitivity of surrogate model performance to the quality and informativeness of the input data. When training data is sparse, biased, or poorly sampled, the resulting models may provide unreliable predictions and inaccurate uncertainty estimates. Furthermore, the computational cost of generating simulation data—especially through FEM runs—can be high, posing challenges for iterative procedures such as active learning or MCMC-based Bayesian updating. Currently, only Gaussian likelihood is implemented for handling measurement error in the user interface, and it is not possible to specify correlated measurement noise, which restricts the fidelity of the probabilistic inference in realistic settings. Additionally, while MCMC-based posterior estimation is supported, the UI lacks functionality to visualize the MCMC random walk, which is often crucial for diagnosing convergence.

Chapter 10

Module 3: Multi-Building Design Updating

10.1 Problem Statement and Needs Addressed

In Module 2, ***Hybrid Digital Twinning***, we demonstrated how selected design parameters can be updated using measurement data from the as-built structure. However, a key limitation identified was the assumption that all model discrepancies stem solely from uncertainties in these selected parameters. In reality, simulation models inevitably involve simplifications—such as neglecting the stiffening effects of non-structural components, idealized joint behavior, or incorrect boundary conditions—which introduce systematic modelling errors not captured by parameter uncertainty alone.

A central problem arises when attempting to calibrate a limited set of uncertain input parameters to match measurement data, while ignoring underlying model-form errors. The result is often an unphysical set of updated parameters. This occurs because the updating procedure attempts to compensate for these hidden discrepancies by forcing the selected parameters to adjust in such a way that the simulated responses align more closely with observations.

For example, consider a cantilever beam that is assumed to be fully constrained at the wall, while in reality it has a slight rotational freedom due to imperfect fixing. A more accurate model would use a rotational spring to represent this partial constraint. However, if the model uses a full constraint and attempts to calibrate only the Young's modulus of the cantilever material using end deflection measurements, the optimizer may converge on an unrealistically low stiffness value. This artificially softens the cantilever to produce larger deflections, thereby reducing the discrepancy between simulated and measured responses—despite the material property having little to do with the actual source of the error.

Although this leads to a model that numerically fits the measurement data, the updated parameter values are not physically meaningful. As a result, the model cannot be generalized to other buildings that exhibit different simplifications or structural characteristics. Such non-generalizable updates limit the broader utility of the calibration process.

To overcome these limitations, the BUILDCHAIN approach proposes a multi-building model updating strategy that leverages data from multiple structures. Rather than calibrating parameters on a per-building basis—risking overfitting to local discrepancies—this method aims to identify consistent trends across ensembles of buildings. By learning from a wider variety of measurement conditions and structural configurations, the resulting updated parameters reflect more generalizable design corrections.

Multi-building model updating is therefore a key enabler for robust and scalable structural health monitoring (SHM) in urban environments. Traditional methods, focused on individual buildings, often result in fragmented insights and underutilized shared information. As discussed in [11], coordinated updating across building ensembles leads to improved parameter estimates and more reliable digital twins. To operationalize this concept, an accessible, API-based tool has been developed and deployed on the BUILDCHAIN platform (`https://buildchain.ilab.sztaki.hu/`), offering practitioners a scalable and reproducible framework for joint model updating.

10.2 Tool Overview, purpose, core functionality

This module provides a powerful framework for updating numerical models of multiple buildings simultaneously using shared data and structural similarities. It enables users to calibrate uncertain design model parameters (e.g., stiffness, damping) against observed structural response (such as modal data) of the as-built building, taking advantage of the statistical coupling across similar structures. Hosted as part of the `TRACE-Structures` system, users have access to both the underlying Python code—available through the digital-twinning library package—and a user-friendly interface that supports both automated digital twin pipelines and user-guided exploratory workflows.

In addition to combining information across multiple buildings, the module also supports the merging of surrogate models developed independently for different subsets of quantities of interest (QoIs). This flexibility allows users to modularize model training, for example, when separate surrogate models are needed for different physical components or output types due to differing complexity or computational cost. These sub-models can then be recombined in a joint inference scheme to produce consistent updates across the full parameter space.

Like the other modules, this one also integrates with the DKG-based BUILDCHAIN DBL through the import and export functionalities of the `TRACE-Structures` API.

10.3 Key Features and Capabilities

This module, integrated into the `TRACE-Structures` platform, provides the following core functionalities:

- Bayesian parameter estimation using shared priors across building ensembles or model ensembles (but not all input parameters of the different models have to be necessarily linked).
- Integration with observed modal data (frequencies, mode shapes) from structural monitoring.
- Joint posterior sampling via MCMC for uncertainty quantification.
- Import of models from, and export of updated parameters to, the Digital Knowledge Graph (DKG).

Additional key features include:

- **Cross-building knowledge reuse:** Enables sharing of prior information across multiple buildings with similar typology by a joint parameter updating procedure.
- **Modular model integration:** Supports combination of separately trained surrogate models for different QoIs or system components, allowing for efficient hybrid modeling workflows.

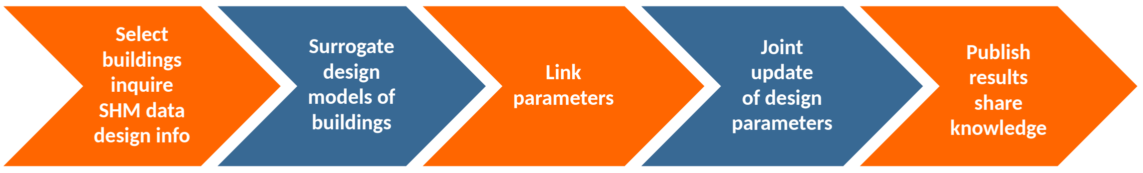

Select buildings inquire SHM data design info
Surrogate design models of buildings
Link parameters
Joint update of design parameters
Publish results share knowledge

Figure 10.1: Methodology, conceptual approach of the multi-building updating module

10.4 Methodology, Conceptual Approach

This section outlines the theoretical foundations and computational methodology underpinning the multi-building updating module. The approach integrates Bayesian inference with FEM-based surrogate modeling. Shared hyperpriors are used to statistically couple the parameter distributions of multiple structures. The main steps of the module are presented in Fig. 10.1.

10.4.1 Algorithms, Models, and Assumptions

Linking input parameters

When unconventional materials or connections are used in multiple buildings, measurements from each structure (or experimental setup) can be integrated for joint parameter updating. Instead of updating parameters separately for each building, a unified approach introduces a joint parameter set $\boldsymbol{\xi}$ mapped to individual building parameters $\mathbf{x}_i$ via transformation functions. Shared parameters ensure consistency across models, while independent parameters remain unconstrained. Without the loss of generality, let's suppose we have two similar building structures. One with uncertain parameters $\mathbf{X}_1$ and the other one with parameters $\mathbf{X}_2$. By defining maps

$$\mathbf{x}_1 = \mathcal{G}_1(\boldsymbol{\xi}) \quad \mathbf{x}_2 = \mathcal{G}_2(\boldsymbol{\xi}) \tag{10.1}$$

allows that for the shared parameters, the random walk can be done for the two models in a joined manner.

Merging measurable building properties and measurements

The measurable quantities, or quantities of interest $\mathbf{y}_1 = f_{1,\text{FEM}}(\mathbf{x}_1)$ of the first building and the one of the second building $\mathbf{y}_2 = f_{2,\text{FEM}}(\mathbf{x}_1)$ from both buildings form a joint vector,

$$\mathbf{y} = f(\boldsymbol{\xi}) = [\mathbf{y}_1, \mathbf{y}_2]^T = [f_{1,\text{FEM}}(\mathbf{x}_1), f_{2,\text{FEM}}(\mathbf{x}_2)]^T = [f_{1,\text{FEM}}(\mathcal{G}_1(\boldsymbol{\xi})), f_{2,\text{FEM}}(\mathcal{G}_2(\boldsymbol{\xi}))]^T \tag{10.2}$$

where $f_{1,\text{FEM}}$ and $f_{2,\text{FEM}}$ are the two forward operators, mapping from the parameters $\mathbf{x}_1$ and $\mathbf{x}_2$ to the measurable properties of the two buildings $\mathbf{y}_1$ and $\mathbf{y}_2$. The measured value of the joined quantity of interest can be this way described as the combined vector of the two measurements, that is

$$\mathbf{z}_m = f(\boldsymbol{\xi}) + \varepsilon = [\mathbf{z}_{m,1}, \mathbf{z}_{m,1}]^T = [f_{1,\text{FEM}}(\mathcal{G}_1(\boldsymbol{\xi})), f_{2,\text{FEM}}(\mathcal{G}_2(\boldsymbol{\xi}))]^T + [\varepsilon_1, \varepsilon_2]^T, \tag{10.3}$$

where ε is the combined Gaussian measurement noise ε_1 and ε_2 of the two buildings

$$\varepsilon = [\varepsilon_1, \varepsilon_2]^T, \tag{10.4}$$

with Gaussian components that are assumed to be independent.

Surrogate modeling

The joint forward operator f is defined by combining the two individual forward operators of the building structures, $f_{1,\text{FEM}}$ and $f_{2,\text{FEM}}$. To enable efficient evaluation of this operator during the updating process—while avoiding repeated, computationally expensive calls to licensed finite element (FE) solvers—we adopt the same surrogate modeling approach introduced in the previous module. Specifically, Module 2 is responsible for constructing separate machine learning surrogate models that approximate $f_{1,\text{FEM}}$ and $f_{2,\text{FEM}}$. This module then focuses on merging their evaluations for use in the likelihood computations, thus facilitating a streamlined and computationally efficient Bayesian updating process.

Joint updating

With this joint formulation, the updating process identifying the posterior distribution $\pi_{\boldsymbol{\xi}|\mathbf{z}_m}(\boldsymbol{\xi})$ of the joint parameter vector $\boldsymbol{\xi}$ can be performed with the help of the MCMC sampling process in a unified manner, ensuring that shared parameters lead to a consistent posterior distribution.

Assumptions

The algorithm assumes statistical independence between input parameters unless shared priors are defined. Measurement errors are modeled as independent Gaussian noise with known variance. No systematic model discrepancy is assumed between simulation output and real structural response. The MCMC algorithm samples posterior distributions under these assumptions.

10.4.2 Calculations and Formulas

The Bayesian posterior of the joined parameter set is then expressed as

$$\pi_{\boldsymbol{\Xi}|\mathbf{z}_m}(\boldsymbol{\xi}) = \frac{\pi_{\mathbf{z}_m|\boldsymbol{\xi}}(\boldsymbol{\xi})\pi_{\boldsymbol{\Xi}}(\boldsymbol{\xi})}{\int_{I_{\boldsymbol{\Xi}}} \pi_{\mathbf{z}_m|\boldsymbol{\xi}}(\boldsymbol{\xi})\pi_{\boldsymbol{\Xi}}(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi}}, \tag{10.5}$$

where $\boldsymbol{\Xi}$ is the random variable of the joint parameter vector whose realization is noted with $\boldsymbol{\xi}$.

Using MCMC sampling, the posterior estimates of $\boldsymbol{\xi}$ are obtained and mapped back to each model via transformation functions $\mathcal{G}_1$ and $\mathcal{G}_2$, improving parameter estimation across buildings.

10.5 Procedures, Data Workflows

10.5.1 Procedures, User Workflow

The main objective of the multi-model updating module is to calibrate uncertain parameters of simulation models across multiple buildings or experimental setups using shared measurement data. This ensemble-based calibration improves generalizability and robustness compared to single-structure updates. The procedure follows a two-stage process: (1) generation of local surrogate models for each building or experiment, and (2) joint calibration of shared design parameters using a multi-output Bayesian inversion approach.

The multi-model updating module builds upon the foundation established in module ***Hybrid Digital Twinning***, where as part of the first process, surrogate models are trained for individual buildings using parametric FE simulations. These surrogates approximate the relationship between uncertain input parameters (e.g., material properties, boundary conditions) and measurable structural responses (e.g., frequencies, mode shapes). The sensitivity analysis step, although supported, is not required in this module.

In this module, the focus shifts to the second process, the coordination and calibration of shared model parameters across multiple buildings. This ensemble-based inference allows for generalizable updating of design parameters that are expected to have common physical meaning (e.g., timber stiffness, joint rigidity) across structurally similar assets.

The first step in the workflow is to upload surrogate models previously trained in the ***Hybrid Digital Twinning*** module. These models are lightweight and contain sufficient metadata to reconstruct the input–output mapping for each building.

Next, the user establishes links between the input parameters of different building models by assigning them to shared latent variables. This mapping defines which parameters are assumed to have a common physical meaning across the models. For example, the "timber stiffness" parameter in three different buildings may be assigned to the same shared design variable, while other parameters remain building-specific.

Finally, the user initiates the multi-model calibration procedure. Bayesian inference is performed using MCMC sampling, where the surrogate models serve as forward models and measurement data from all buildings are used collectively to inform the posterior distributions of shared and individual parameters. Each building contributes to reducing uncertainty in the global parameter space while accounting for local discrepancies.

This module enables structurally consistent calibration across an ensemble of assets, improves robustness to individual model errors, and lays the foundation for generalizable digital twin construction.

The procedural workflow of the multi-model updating module is visualized in Figure 10.2. The process begins with uploading previously trained surrogate models for individual buildings. These models are then linked via shared design parameters, after which joint Bayesian calibration is executed using MCMC sampling across all included buildings.

10.5.2 Data Flow, Data Formats

A more detailed overview of the data flow is provided in Fig. 10.3. The figure distinguishes between user-provided input data, data imported from either the BUILDCHAIN DKG or local storage, and the outputs resulting from the multi-model calibration pipeline. Each of these components is described in detail below.

Input retrieved from the DKG or from local drive

- **Surrogate Models (for each building)**:
  - **Format:** `.spm` (BUILDCHAIN surrogate model format, based on Python `.pikle`);
  - **Content:** Predictive model approximating FE simulations of individual buildings, previously trained using Module 2.
- **Measured Quantities of Interest (QoIs)**:
  - **Format:** `.unv`, `.json`, or `.csv`;
  - **Content:** Experimental measurements for each building, typically modal data; imported from local file or retrieved via the DKG.

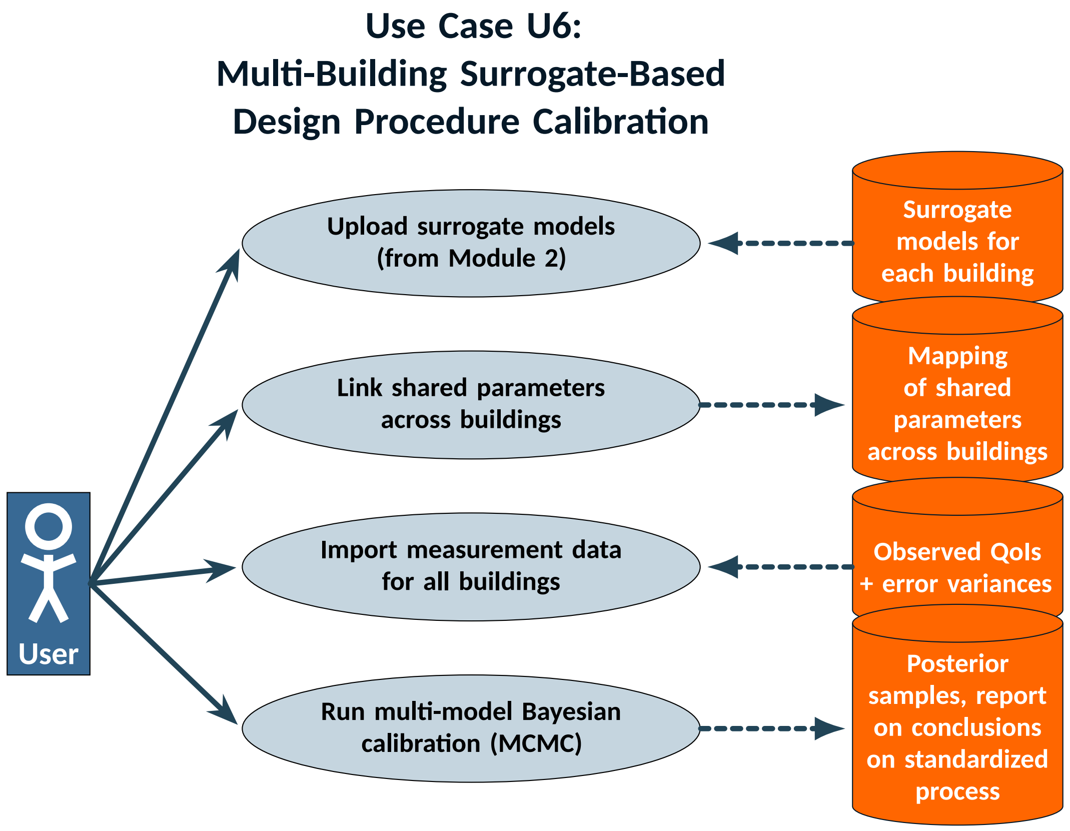

Use Case U6:
Multi-Building Surrogate-Based
Design Procedure Calibration
Upload surrogate models (from Module 2)
Link shared parameters across buildings
Import measurement data for all buildings
Run multi-model Bayesian calibration (MCMC)
User
Surrogate models for each building
Mapping of shared parameters across buildings
Observed QoIs + error variances
Posterior samples, report on conclusions on standardized process

Figure 10.2: User interactions and data flows for the multi-model updating module. After uploading surrogate models from prior simulations, shared parameters are mapped across buildings and measurement data is imported. The module then performs joint Bayesian calibration using MCMC sampling to infer shared input parameter distributions.

- **Variance of Measurement Error:**
  - **Format:** `.csv`
  - **Content:** Diagonal entries of the covariance matrix for each building's measurement error model

Other direct user inputs

- **Shared Parameter Mapping:**
  - **Exportable format:** `.json`
  - **Content:** List of linked parameters in building-specific surrogates to global/shared parameters for multi-model calibration.
- **MCMC Configuration:**
  - **Format:** Directly provided in UI or stored as `.json`
  - **Content:** Number of MCMC samples, number of chains, random seed (optional)

Intermediate Results (not stored as Knowledge Assets)

- **Posterior Sample Chains:**
  - **Format:** `.csv` or `.json`
  - **Content:** Raw posterior samples across MCMC chains; used for statistical diagnostics and downstream inference
- **Visualizations:**
  - **Format:** `.png`
  - **Content:** Matrix plots, marginal density estimates

Outputs, that may be stored in the DKG

- **Updated Shared Parameters:**
  - **Format:** `.csv`, `.json`
  - **Content:** Posterior summary statistics (mean, standard deviation, percentiles, MAP estimate) for shared parameters
  - **DKG Integration:** Output can be stored as a digital knowledge asset with metadata describing building linkage, calibration time, and QoI type
- **Report on design procedure:**
  - **Format:** `.pdf`
  - **Content:** Report linked to a design standard on lessons learned from multi-model updating.

The module semantically aligns shared parameter and QoI concepts with the BUILDCHAIN ontology. This alignment enables structured interaction with the DKG and ensures that outputs from joint calibration are interoperable with other modules. All data types and their potential for DKG integration are summarized in Table 10.1.

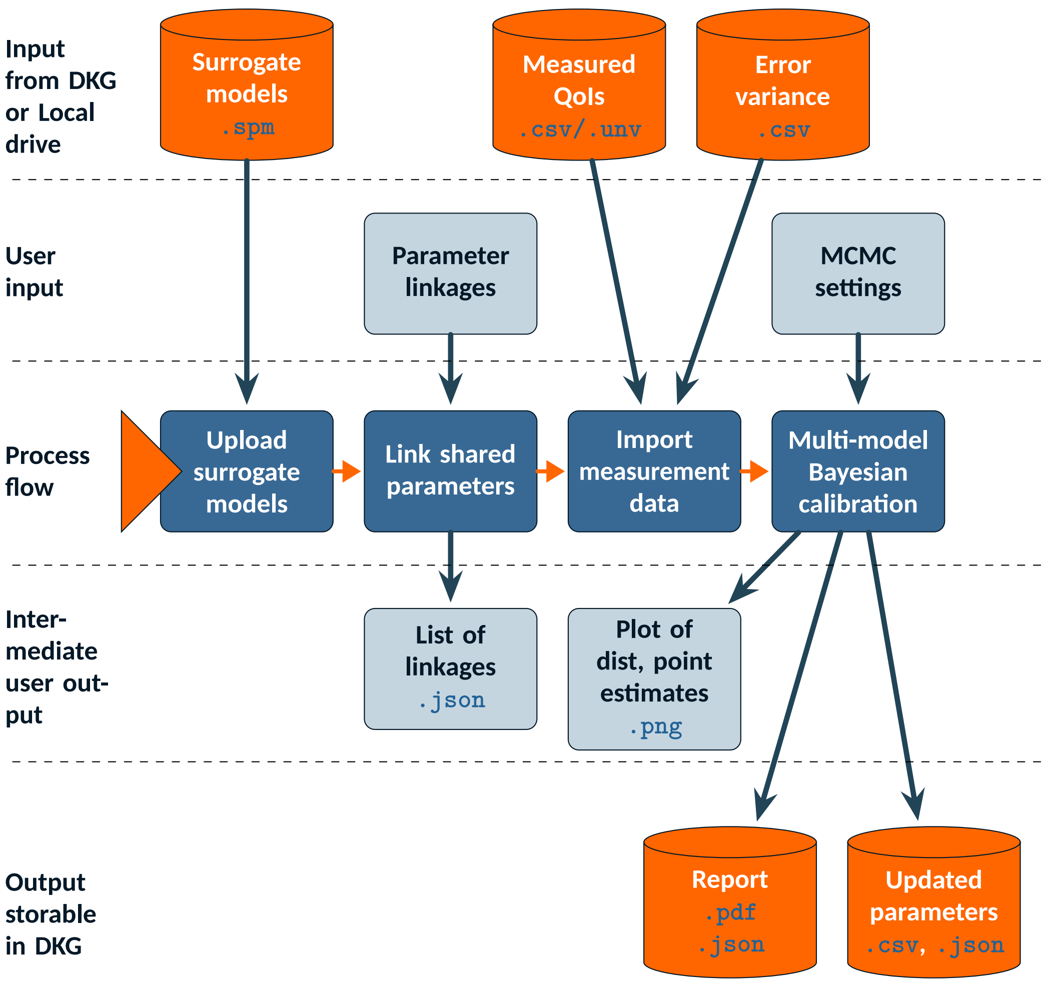


Figure 10.3: Data flow in the multi-model updating module.

Data type	Description	csv	json	unv	pickle: .xm/ .spm	Integration	User In-put	Stored in DKG	Additional Info
Input									
Surrogate model	Trained model of individual building				x (.spm)	Input from DKG or local drive		x	Includes prior parameter set
Measured QoIs	Observed modal frequen-cies/shapes	x	x	x		Input from DKG or local drive		x	Uncertain structural response
Error variance	standard deviation of measured values	x				Input from DKG/local drive		x	Used in likelihood computation
Shared parameter config	Mapping between global/local parameters		x			User input	x		Defines priors and shared structure
MCMC settings	Chains, steps, seeds, etc.		x			User input	x		Used for sampling strategy
Output									
Posterior samples	Sampled parameter chains, one point estimates	x	x			Storable in DKG		x	
Diagnostics	Matrix plots, MAP, mean posterior								Visual inspection of posterior
Technical report	report on lessons learned		x			To DKG/local hardware		x	linked to standard, design procedure

Table 10.1: Input and output data types, formats, and integration in the multi-model updating module.

10.6 Technical Architecture, Python classes

Dedicated Python classes with various functionalities were developed to support the digital twinning workflows of Modules 1–3. The architecture of the `TRACE` Python library package includes classes that cover the entire workflow—from linking simulation parameters and merging models and measurements. These classes are illustrated through UML diagrams presented in Annex B, which highlight the key classes and their associated functionalities.

10.7 Demo, Alignment with BUILDCHAIN Goals

This module was developed as part of the BUILDCHAIN project's Use Case U6, titled *Follow-up procedures of design processes by Bayesian updating*. The module addresses scenarios where multiple physics-based models are available for the same or related systems, and where some parameters are shared across these models while others are model-specific.

The underlying library and user interface have been designed with flexibility in mind. They enable multi-model Bayesian inference by integrating data from several sources or models, facilitating the joint calibration of shared parameters and individual updating of local parameters. The module can be applied to a wide range of engineering and scientific problems where multi-fidelity, multi-source, or multi-physics modeling is used.

The module has been tested and validated using both a controlled toy example—updating the mass and spring stiffnesses in two coupled spring–mass systems with a shared mass—and a real-world application from the BUILDCHAIN Pilot 5: the joint calibration of two finite element models of similar timber buildings using experimental vibration data. These demonstration cases showcase the module's capabilities in both synthetic and realistic digital twinning contexts.

10.7.1 Toy Model: Coupled 1D Oscillators

The first test case considers two independent single-degree-of-freedom (SDOF) oscillators with identical mass but different spring stiffnesses. Each oscillator is treated as a separate model with its own uncertain input parameters: the mass m and the spring stiffness k_i for oscillator $i = 1, 2$. Synthetic observations of the system frequencies are generated for both models. The key idea in this demonstration is that the mass parameter m is shared between the two models, while the stiffness parameters are model-specific. This leads to the following integrated system. The joint parameter vector reads

$$\boldsymbol{\xi} = [k_1, k_2, m]^T, \tag{10.6}$$

with the local parameter vectors of the two distinct oscillators

$$\mathbf{x}_1 = [k_1, m]^T = \mathcal{G}_1(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ m \end{bmatrix}, \tag{10.7}$$

$$\mathbf{x}_2 = [k_2, m]^T = \mathcal{G}_2(\mathbf{x}) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ m \end{bmatrix} \tag{10.8}$$

Supposing $u_0 = 1$ initial displacement from the equilibrium position and v_0 initial velocity in both oscillatory, the measurable displacements fo the mass at time t_i can be given for

a) b)

Figure 10.4: The two CLT buildings analysed in Pilot 5: a) Yoker (UK), b) Palisaden (Norway).

the two systems by

$$y_{1,i} = u_1(t_i) = \cos(\omega_1 t_i) + \frac{v_0}{\omega_1}\sin(\omega_1 t_i), \quad \mathbf{y}_1 = [u_1(t_1), u_1(t_2), u_1(t_3), u_1(t_4), u_1(t_5)]^T, \tag{10.9}$$

$$y_{2,i} = u_2(t_i) = \cos(\omega_2 t_i) + \frac{v_0}{\omega_2}\sin(\omega_2 t_i), \quad \mathbf{y}_2 = [u_2(t_1), u_2(t_2), u_2(t_3), u_2(t_4), u_2(t_5)]^T \tag{10.10}$$

with

$$\omega_1 = \sqrt{\frac{k_1}{m}}, \quad \text{and} \quad \omega_2 = \sqrt{\frac{k_2}{m}}. \tag{10.11}$$

The synthetic measurements $\mathbf{z}_{m,1}$ and $\mathbf{z}_{m,2}$ were generated by adding random samples of an unbiased gaussian random noise to the computed displacements, with the resulting joint measurement vector

$$\mathbf{z}_m = [\mathbf{z}_{m,1}, \mathbf{z}_{m,2}]^T. \tag{10.12}$$

This setup allows testing the ability of the module to correctly infer the shared and individual parameters using multi-model Bayesian updating. The likelihoods from both models are combined, and parameter inference is performed jointly using Markov Chain Monte Carlo (MCMC).

10.7.2 Design-procedure update - using two mid-rise timber buildings made of CLT panels

A second, more realistic demonstration case is based on BUILDCHAIN Pilot 5, which involves the digital twinning of two CLT buildings: one in Glasgow, UK (the “Yoker building”), already detailed in Section 9.7.2, and the other in Ås, Norway (the “Palisaden building”), as shown in Figure 10.4. Traditional Bayesian updating has been effective for each building individually [12, 13], but independent updates require manual interpretation to generalize across structures. The present module streamlines this by enabling a unified joint update based on both models.

Yoker Building. As already described in Section 9.7.2 deterministic finite element (FE) model of the seven-storey Yoker building was used to compute modal properties [13]. The uncertain model parameters are:

$$\mathbf{x}_1 = [e_1, e_2, e_3, g_1, g_2, q], \tag{10.13}$$

where e_1, e_2, e_3 are elastic moduli in major material directions, g_1, g_2 are in-plane shear moduli, and q is the distributed mass. Natural frequencies f_i and mode shapes ψ_i were obtained from forced vibration tests:

$$\mathbf{y}_1 = \left[f_1, f_2, f_3, f_4, f_5, \psi_1^T, \psi_2^T, \psi_3^T, \psi_4^T, \psi_5^T\right]^T = f_{1,\text{FEM}}(\mathbf{x}_1). \tag{10.14}$$

Measurement noise was assumed Gaussian, with variances estimated from operational modal analysis.

Palisaden Building. The Palisaden building is an eight-storey CLT structure with exposed and covered wall elements. The uncertain model parameters are:

$$\mathbf{x}_2 = [\gamma_{e1}, \gamma_{G_{12}}, \gamma_{G_{12},\text{ext}}, k_{\text{spring}}, \gamma_\rho], \tag{10.15}$$

representing scale factors for elastic and shear moduli, foundation stiffness, and mass. Ambient vibration data were used to extract four natural frequencies:

$$\mathbf{y}_2 = [f_1, f_2, f_3, f_4]^T = f_{2,\text{FEM}}(\mathbf{x}_2). \tag{10.16}$$

Shared Parameter and Joint Update. While the two buildings use different parametrizations, they both model the vertical elastic modulus of CLT walls. In the Yoker model, this is e_1; in the Palisaden model, it is expressed as a scaled ETA value, $e_1 = \gamma_{e1} \cdot E_1$. To link the models, a set of dimensionless reference parameters with standard distributions (shift and scaled) is used to drive both e_1 values, with model-specific transformations for the remaining parameters. Each of the ten latent parameters follows a standard uniform distribution, $\xi_i \sim \mathcal{U}(0, 1)$.

10.8 Step by Step Library Package User Guide

The Python library developed for the digital twinning modules (Modules 1-3) enables the creation and calibration of hybrid models based on physics-informed surrogate modeling. It supports both single-model and multi-model parameter updating using measurements, and is designed to integrate easily with the data and simulation infrastructure of the BUILDCHAIN project. The functionality covers the definition of uncertain input parameters, the import and linking of data from simulations and experiments, the training of predictive and explanatory surrogate models, and Bayesian updating using either standard or joint parameterizations.

To facilitate the adoption of the package and provide a clear, reproducible workflow, two example Jupyter notebooks are included as a step-by-step user guide that are annexed in Annex C.3:

- **Notebook 1** (see Sec C.3.1): A toy model example that demonstrates the complete hybrid modeling and multi-model updating pipeline using synthetic data from two coupled one-degree-of-freedom spring–mass systems explained in Section 10.7.1.
- **Notebook 2** (see Sec C.3.2): The real-world application to the BUILDCHAIN Pilot 5 described in Section 10.7.2, involving two CLT buildings: the Yoker building in Glasgow and the Palisaden building in Ås. This notebook showcases the joint Bayesian inference to update material and structural parameters. It illustrates how shared parameters (e.g., vertical elastic modulus) can be jointly inferred to identify generalized corrections across different structures.

These notebooks serve both as validation of the methodology and as practical templates for applying the library to other digital twinning scenarios involving surrogate modeling and Bayesian updating.

10.9 Step by Step API User Guide for the Module

This section presents, in a step-by-step manner, how the Multi-building Updating module of the TRACE-Structures BUILDCHAIN UI can be applied.

To carry out the full workflow for joint updating of multiple digital twin models using the API, users should follow the steps below:

1. **Access the BUILDCHAIN Web Interface**: Available via `https://buildchain.ilab.sztaki.hu`;
2. **Select Module**: Navigate to the ***Multi-Building Design Updating*** module.;
3. **Import Surrogate Models for Each System**: Upload surrogate models for each structure or component trained separately using the Hybrid Digital Twinning module;
4. **Link Uncertain Parameters**: Specify the shared and specific parameters of the different models;
5. **Import Measured Structural Responses and Error Standard Deviation for Each System**: Upload observation data (e.g., modal frequencies, mode shapes) for each system, and the corresponding standard deviations of errors;
6. **Configure MCMC Settings for Joint Bayesian Updating**: Define the number of chains, number of iterations, and burn-in period;
7. **Run Joint Bayesian Updating**: Execute the MCMC inference to update the joint latent parameters based on all measurement datasets;
8. **Export Data and Reporting**: Finalize the workflow when generalizable conclusions can be drawn.

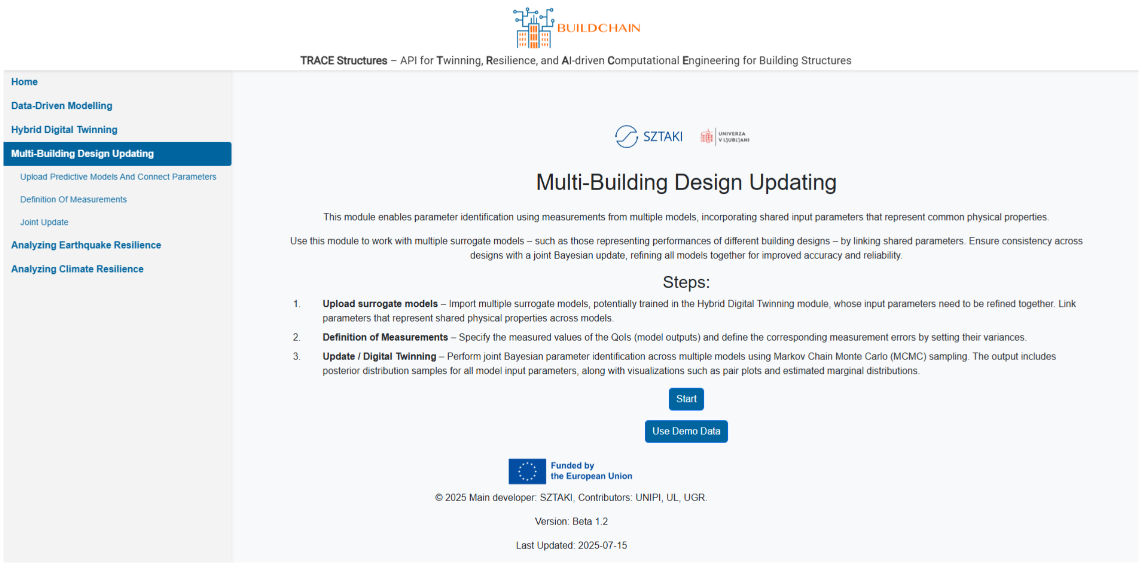
Note: While originally designed to link parameters across different buildings, the same workflow can be applied to merge separately trained surrogate models for different subsets of QoIs within a single structure. This is useful when model components vary in complexity or are developed modularly to optimize training costs.

10.9.1 Step 1: Access the BUILDCHAIN web interface

Visit the TRACE-Structures UI at `https://buildchain.ilab.sztaki.hu` and click on `Start app`.

10.9.2 Step 2: Module Selection

Navigate to the *"Multi-building Updating"* section by either clicking the blue `Multi-building Updating` button or selecting the corresponding item from the left-hand navigation menu (see upper panel of Fig 8.4. On the first page of the module (see Fig 10.5), click on the `Start` button to proceed. By clicking here instead, on `Use Demo Data`, you can load the built-in input–output dataset generated for the toy model described in Section 10.7.1. The screenshots provided in the following steps of the UI correspond to this toy model example.


BUILDCHAIN
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models And Connect Parameters
Definition Of Measurements
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
Start
Use Demo Data
Funded by the European Union

TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Multi-Building Design Updating

This module enables parameter identification using measurements from multiple models, incorporating shared input parameters that represent common physical properties.

Use this module to work with multiple surrogate models – such as those representing performances of different building designs – by linking shared parameters. Ensure consistency across designs with a joint Bayesian update, refining all models together for improved accuracy and reliability.

Steps:

1. **Upload surrogate models** – Import multiple surrogate models, potentially trained in the Hybrid Digital Twinning module, whose input parameters need to be refined together. Link parameters that represent shared physical properties across models.
2. **Definition of Measurements** – Specify the measured values of the QoIs (model outputs) and define the corresponding measurement errors by setting their variances.
3. **Update / Digital Twinning** – Perform joint Bayesian parameter identification across multiple models using Markov Chain Monte Carlo (MCMC) sampling. The output includes posterior distribution samples for all model input parameters, along with visualizations such as pair plots and estimated marginal distributions.

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

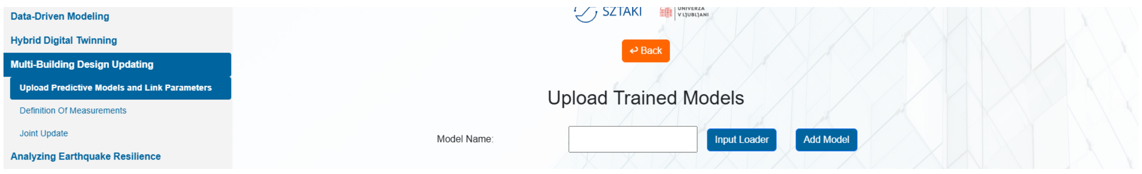
Version: Beta 1.2

Last Updated: 2025-07-15

Figure 10.5: Step 2 of module 3 – Getting started.

10.9.3 Step 3: Define Surrogate Models for Each System

Train surrogate models for each structure or component separately using the Hybrid Modeling module following the step by step guide of Module 2. Save the models, register in in the DKG or save locally, and import it in this module. First start adding an already defined surrogate model by first naming it, then uploading the surrogate model by clicking on `Input Loader` and choosing a file from local storage or importing the model by logging in the BUILDCHAIN platform. After importing the file, click on `Add Model` (see Fig. 10.6). Repeat this procedure until all models to be linked are added. The user interface shows all uploaded models, with their names and their uncertain input parameters (see Fig. 10.7).

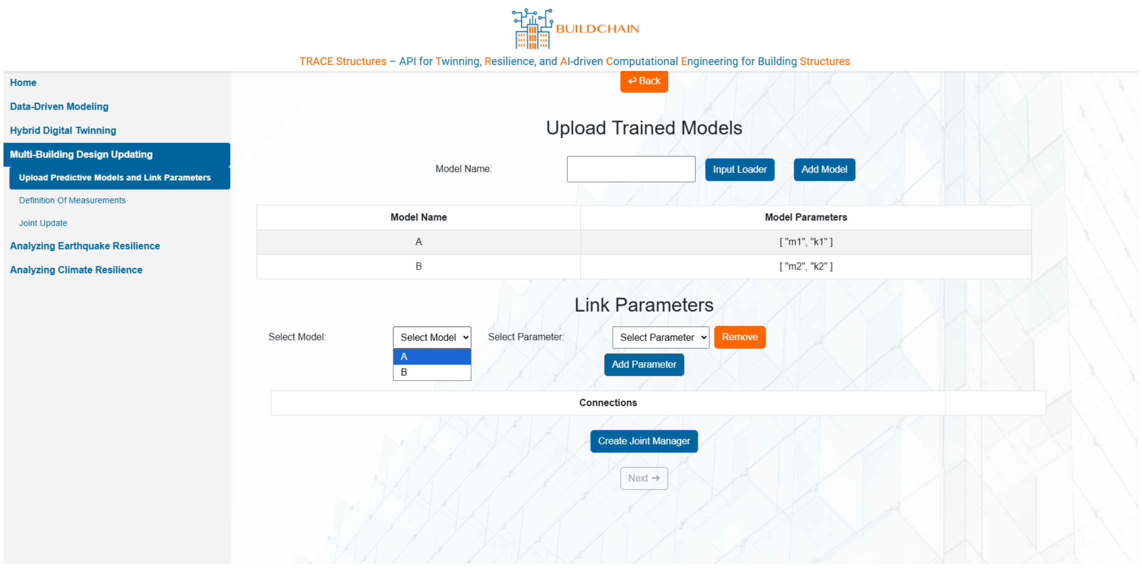

Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters
Definition Of Measurements
Joint Update
Analyzing Earthquake Resilience
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Input Loader
Add Model

Upload Trained Models

Model Name:

Figure 10.6: Step 3 of module 3 – Uploading models to be linked.

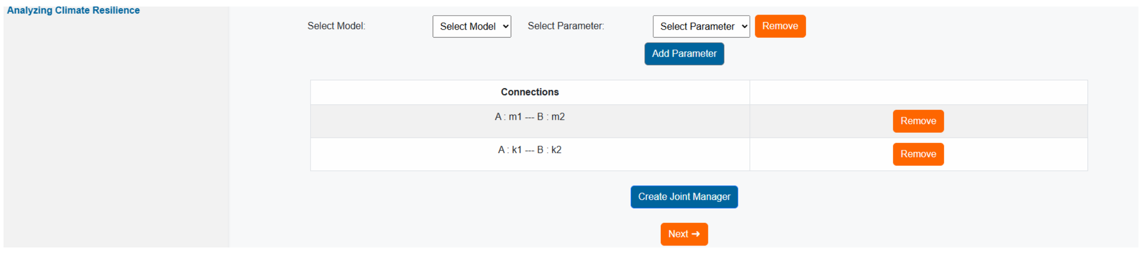
10.9.4 Step 4: Linking Uncertain Parameters


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters
Definition Of Measurements
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
↩ Back
Upload Trained Models
Model Name:
Input Loader
Add Model
Link Parameters
Select Model:
Select Model
A
B
Select Parameter:
Select Parameter
Remove
Add Parameter
Create Joint Manager
Next →

Model Name	Model Parameters
A	["m1", "k1"]
B	["m2", "k2"]

Connections	

Figure 10.7: Step 4 of Module 3 – Linking uncertain parameters across uploaded models.



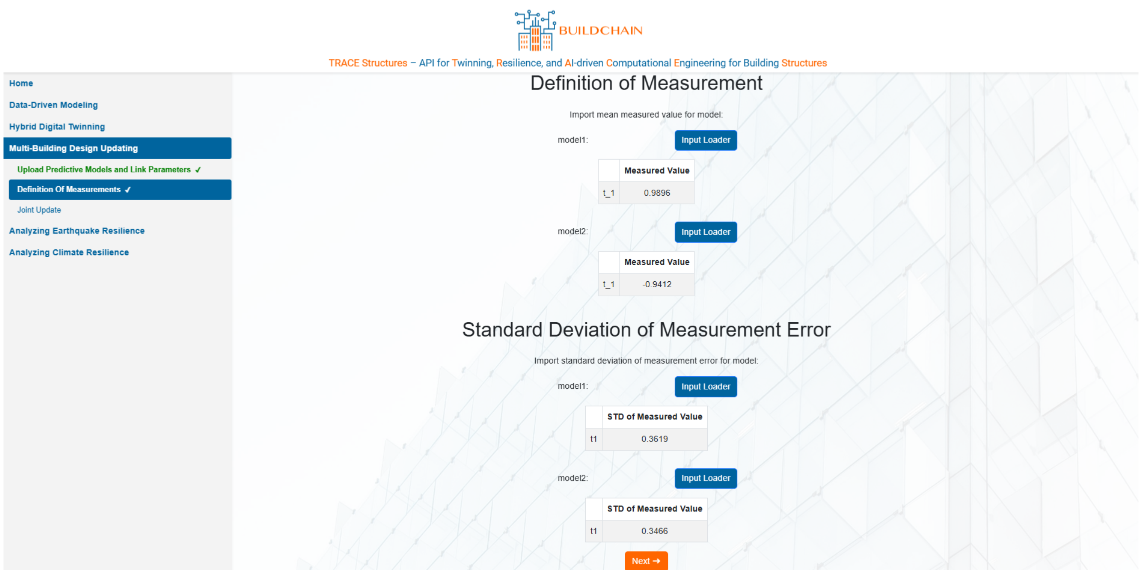
Connections	
A : m1 — B : m2	Remove
A : k1 — B : k2	Remove

Figure 10.8: Step 4 of Module 3 – list of linked parameters.

If the surrogate models were created using Module 2, their associated uncertain parameter definitions are automatically imported into the interface. To link shared parameters across different models, users must assign them to a common latent parameter. This is achieved by selecting one parameter from each model that represents the same physical or design quantity. This is done by first selecting a model and its parameter, and then clicking the `Add Parameter` button to select new parameters of new models (see Fig. 10.7). Once all linked parameters are added, click on the `Connect` button. The linking of these parameters will be shown in the table of *Connections* (see table in Fig. 10.8). Repeat this procedure for each group of parameters that should be treated as shared across models. When all linkages are defined, click on the `Create Joint Manager` button. Once the *Joint Manager* created, these parameters can be jointly inferred during the multi-model design updating process.

10.9.5 Step 5: Import Measured Structural Responses for Each System

Upload structural response data (e.g., natural frequencies, mode shapes) for each system in `csv`, `json`, or `unv` format by clicking on the `Choose File` button beside the imported models (see upper part of Fig. 10.9). Then upload the list of standard deviations of the measurement errors given for each component for each quantity of interest to enable a consistent likelihood model during Bayesian inference. This step has to be done for each model by clicking on the `Choose File` button beside the imported models (see lower part of Fig. 10.9).


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters ✓
Definition Of Measurements ✓
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Input Loader
Next ➔

Definition of Measurement

Import mean measured value for model:

model1:

	Measured Value
t_1	0.9896

model2:

	Measured Value
t_1	-0.9412

Standard Deviation of Measurement Error

Import standard deviation of measurement error for model:

model1:

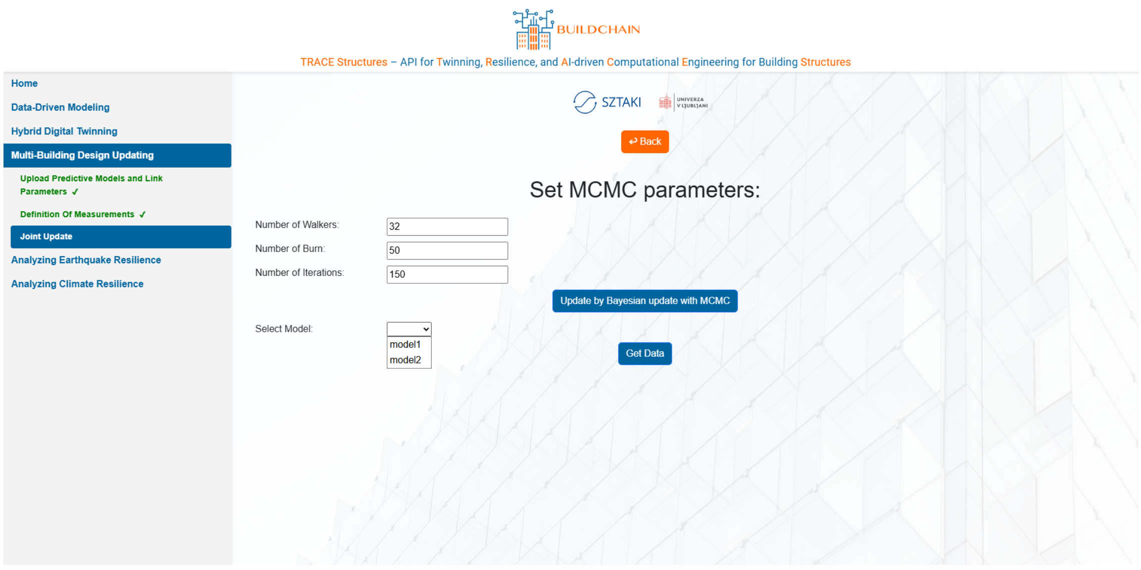
	STD of Measured Value
t1	0.3619

model2:

	STD of Measured Value
t1	0.3466

Figure 10.9: Step 5 of Module 3 – import measurements and standard deviations.

10.9.6 Step 6: Configure MCMC Settings for Joint Bayesian Updating


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters ✓
Definition Of Measurements ✓
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
SZTAKI
UNIVERZA V LJUBLJANI
↩ Back
Update by Bayesian update with MCMC
Get Data
model1
model2

Set MCMC parameters:

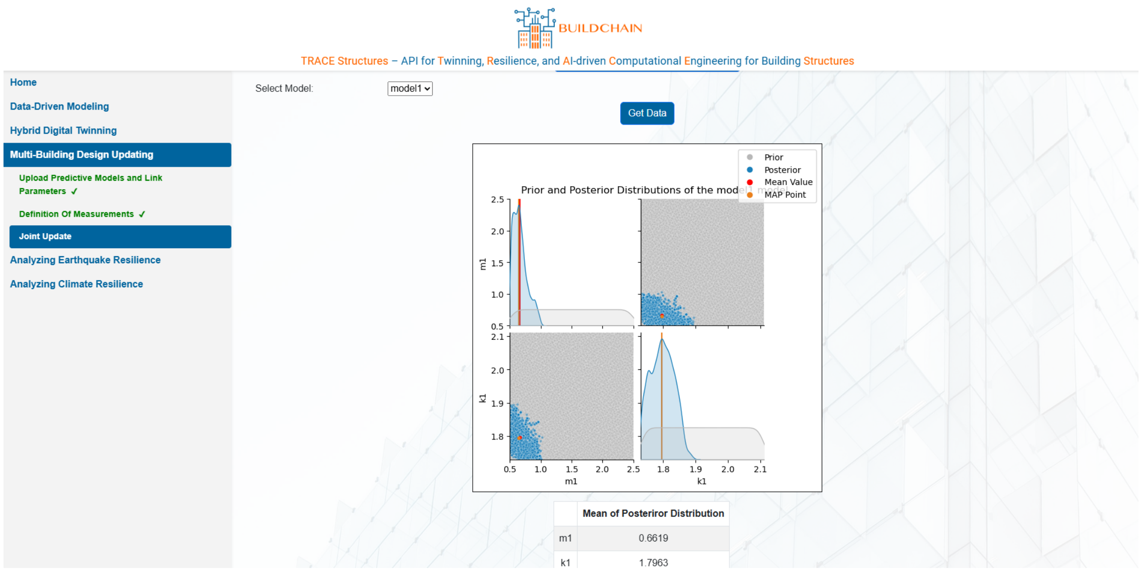
Number of Walkers:	32
Number of Burn:	50
Number of Iterations:	150
Select Model:	

Figure 10.10: Step 6 of Module 3 – Setting MCMC sampling procedure.

Define the number of MCMC chains, number of burned steps, and the number of iterations (Fig. 10.11). For a more detailed explanation of the interface and outputs, refer to the full description under Step 9 of Section 8.9 of Module 1, where identical MCMC functionality is described in the context of data-driven modeling.

10.9.7 Step 7: Run Joint Bayesian Updating

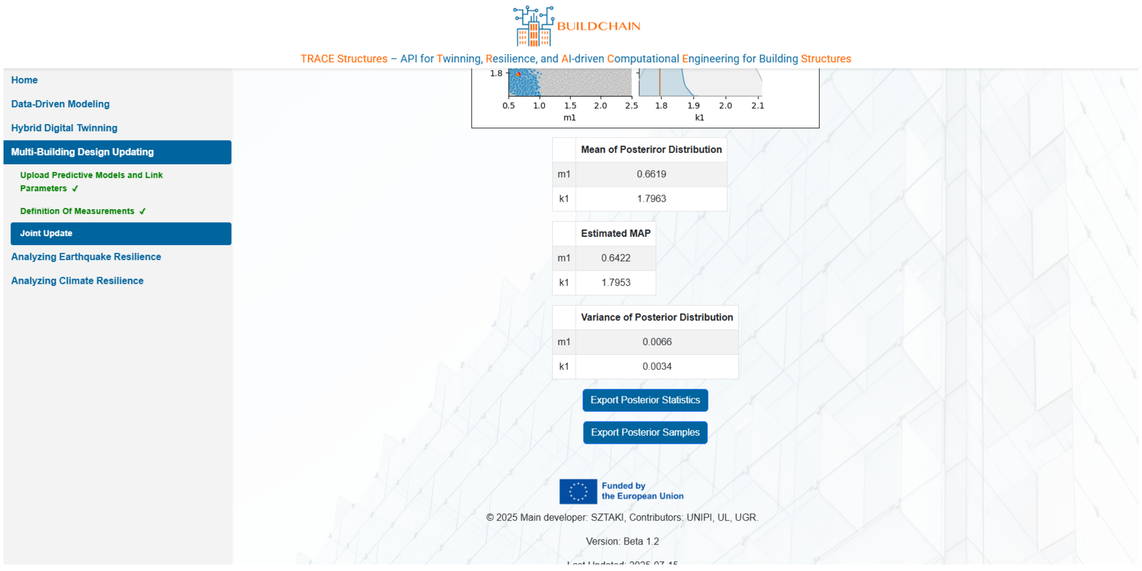
Execute the MCMC inference to jointly update the latent parameters across all available measurement datasets by clicking the `Update by Bayesian Update with MCMC` button. This process simultaneously calibrates both shared and model-specific parameters while preserving independence among unrelated quantities. The user must select the model for which the posterior samples will be visualized in a paired scatterplot. After clicking the `Get Data` button, the results of the joint update will be available for the selected model. An example output for the first oscillators is shown in Fig. 10.11.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters ✓
Definition Of Measurements ✓
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Select Model:
model1
Get Data
Prior and Posterior Distributions of the mod
Prior
Posterior
Mean Value
MAP Point
m1
k1

	Mean of Posteriror Distribution
m1	0.6619
k1	1.7963

Figure 10.11: Step 7 of Module 3 – Results of Joint Bayesian Updating for the first model.

10.9.8 Step 8: Export Results and Reporting


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modeling
Hybrid Digital Twinning
Multi-Building Design Updating
Upload Predictive Models and Link Parameters ✓
Definition Of Measurements ✓
Joint Update
Analyzing Earthquake Resilience
Analyzing Climate Resilience
m1
k1
Export Posterior Statistics
Export Posterior Samples
Funded by the European Union

	Mean of Posteriror Distribution
m1	0.6619
k1	1.7963

	Estimated MAP
m1	0.6422
k1	1.7953

	Variance of Posterior Distribution
m1	0.0066
k1	0.0034

© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Version: Beta 1.2

Figure 10.12: Step 8 of Module 3 – Results of Joint Bayesian Updating for the first model, exporting results.

- Export posterior samples for each model in `csv` format, and store it locally or register it on the DKG by clicking on `Export Posterior Samples`.
- Compute one-point estimates such as MAP (maximum a posteriori) or posterior means in `json` format, and store it locally or register it on the DKG by clicking on `Export Posterior Statistics`.

- Visualize marginal posterior distributions of shared and individual parameters.
- Evaluate uncertainty reduction and correlations between parameter estimates.
- Retrain or validate surrogate models using the updated parameter distributions if needed.
- When generalizable conclusions (e.g., correction of manufacturer input values) can be drawn, synthesize and summarize key findings in a technical report. Publish and link this new knowledge to the DKG for broader dissemination across projects or future updates.

10.10 Results of Testing, Interpretation of Outputs

10.10.1 Toy Model, Coupled 1D Oscillators

To validate the multi-model design updating methodology, we first applied the module to the toy example described in Section 10.7.1 involving two coupled one-degree-of-freedom oscillator systems. Each oscillator had slightly different physical properties, but they shared a subset of design parameters such as stiffness or mass. The goal was to jointly infer the shared latent parameters using synthetic displacement data generated from each oscillator. This example allowed us to test the multi-building update procedure in a fully controlled setting, with analytically known behavior. The posterior distribution of the shared parameters confirmed that the module successfully integrates information from multiple sources, enabling robust and consistent parameter inference across structurally similar systems.

10.10.2 Design-procedure multi-building update of design parameters of tall timber buildings

As a real-world application, the module was tested on two tall timber buildings: the Yoker building in Glasgow and the Palisaden building in Ås, both part of the BUILDCHAIN Pilot 5 (see description of the buildings in Section 10.7.2. Each structure was first individually modeled and equipped with surrogate models replacing their high-fidelity FE counterparts. Using the multi-model updating framework, we jointly inferred shared material properties—such as the elastic modulus of CLT panels—across the two buildings. This process supports a design-based parameter calibration workflow that can generalize findings across similar building typologies. The resulting posterior distributions improved consistency between observed behavior and model predictions, demonstrating the module's capability to support scalable, cross-asset digital twinning for resilience analysis. See the joint posterior distribution of all the input parameters of the two buildings in Fig 10.13.

10.11 Strengths and Limitations

The Multi-building Updating module offers several notable strengths that make it a powerful tool for uncertainty quantification and model calibration across multiple systems. A key advantage lies in its ability to perform cross-model knowledge transfer by linking shared physical parameters—such as material properties—between different buildings or components. This approach enables a more informed and efficient updating process compared to treating each model in isolation.

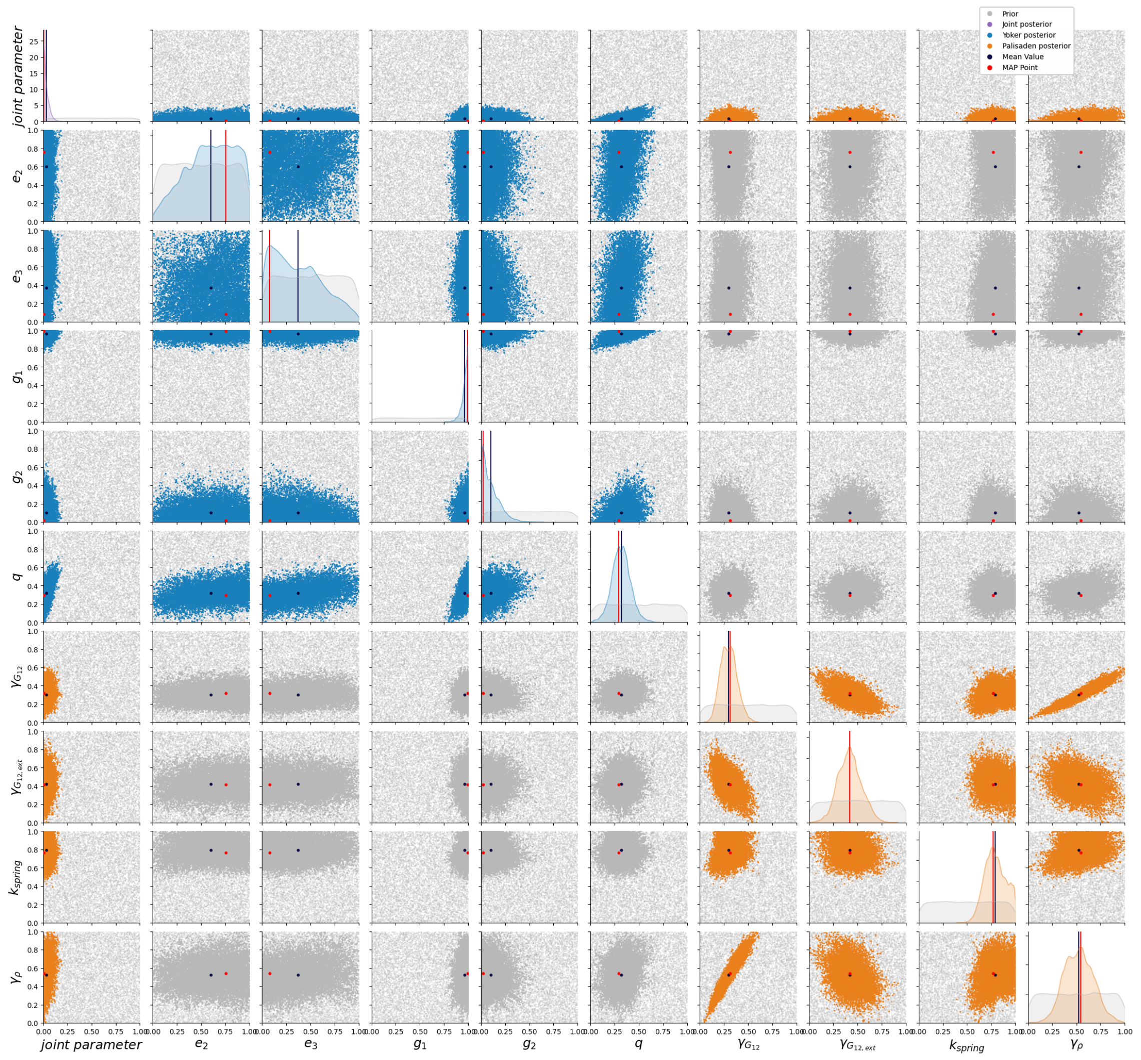


Figure 10.13: Resulted joint design parameter update of two tall CLT buildings, with linked elastic module.

Another major strength is the flexibility of its design. The module can be used not only to join numerical models of different buildings but also to merge separate surrogate models that were trained for different subsets of quantities of interest (QoIs). This is particularly useful when different parts of a structure require different modeling complexities, or when computational efficiency demands decomposition into smaller models.

The module supports flexible parameter linking, allowing users to specify which parameters are globally shared and which are model-specific. This enables partial coupling between systems while preserving independence for unrelated aspects. The built-in Bayesian inference engine is based on Markov Chain Monte Carlo (MCMC) sampling and allows for realistic uncertainty quantification by incorporating both measurement noise and prior knowledge.

Operationally, the module can be accessed both via a graphical user interface and programmatically through an API. This supports both exploratory and automated workflows, making it suitable for integration into digital twin pipelines or for research and development. It also integrates seamlessly with surrogate models trained using the Hybrid Modeling module, thereby reducing computational costs during the updating phase. Final results, including posterior distributions and point estimates, can be exported in standard formats and optionally linked to the Digital Knowledge Graph (DKG), enhancing transparency and interoperability.

Despite these advantages, the module also has certain limitations. Its effectiveness depends strongly on the quality of the surrogate models provided. Poorly trained or non-generalizable surrogates can lead to inaccurate inference results. Additionally, linking of shared parameters across models must currently be done manually, requiring prior engineering judgment. There is no automated support yet for detecting or suggesting parameter couplings based on structural similarities.

Furthermore, while the system supports modular modeling, users must ensure consistency in the complexity and dimensionality of the surrogate models. Combining very different models without proper coordination may result in identifiability issues or slow or even no MCMC convergence. The current implementation only shows a real-world example focusing on updating parameters based on measured modal properties.

Finally, while the module is capable of scaling to handle multiple structures, performance tuning of the MCMC sampler may be necessary when working with large ensembles or highly coupled models.

Chapter 11

Module 4: Assessment of Earthquake Resilience

11.1 Problem Statement and Needs Addressed

The vulnerability of existing buildings to earthquakes is one of the major threat for the EU built environment in seismic prone areas. As the European Union accelerates its renovation wave, it is essential that this structural vulnerability is addressed to achieve a safe and resilient built environment. To identify effective strategies and prioritize retrofitting and strengthening interventions, there is a need to perform large scale seismic vulnerability assessments for existing constructions located in earthquake- prone areas.

Current procedures for the assessment require significant efforts to first retrieve relevant information about the building, its geometry, structural scheme, building materials, history of interventions, and so on. Moreover, the evaluation of seismic performance is often subjected to several assumptions regarding modeling techniques, analysis software, and adopted material properties that can lead to inconsistent classification of buildings. Digital Building Logbooks (DBLs), making accessible trustable data and knowledge related to the properties of the building, can provide useful input for structural analysis tools. However, there is still a need to provide an open tool to perform the assessment and create vulnerability reports in accordance with the EU regulations , which can be logged in the DBL as well and thus accessible to relvant authorities. Module 4 of the TRACE-STRUCTURES platform, provides this tool focusing on masonry buildings, one of the most common structural typology for historical constructions.

11.2 Tool Overview, purpose, core functionality

This module provides a powerful framework for the assessment of earthquake resilience of masonry buildings. The evaluation of structural performance is carried out based on EPUSH routines [14]. E-PUSH is a software program developed by the research group at the University of Pisa for the structural analysis of masonry structures under seismic and non-seismic actions. Input data for the analysis include geometric and mechanical properties of masonry walls, as well as stresses due to vertical loads, which are collected in a excel table, which can be compiled manually by an engineer. In the BUILDCHAIN project, the possibility to automatically extract this table from BIM (.ifc file) has been developed. Requiring a limited number of input data and based on clear and simple structural models, the program is nearly independent of the users, avoiding inconsistencies in the seismic classification of different buildings.

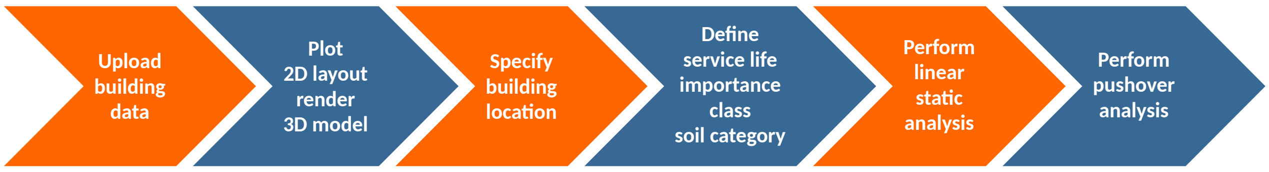

Upload building data
Plot 2D layout render 3D model
Specify building location
Define service life importance class soil category
Perform linear static analysis
Perform pushover analysis

Figure 11.1: Main steps of the earthquake resilience module.

The module is designed for strong interoperability with the DKG-based BUILDCHAIN DBL. In the future, geometric and material properties of structures will be directly retrievable from IFC files imported via the DBL, enabling automated reporting on the seismic vulnerability of buildings.

11.3 Key Features and Capabilities

This module, integrated into the TRACE-STRUCTURES platform, provides the following core functionalities:

- creation of an equivalent frame model of the masonry building;
- computation of wall safety factors for different failure criteria, shear, bending in plane and out of plane, based on the results of linear static analysis
- evaluation of global vulnerability index through non-linear static analyses and verification on the Acceleration Displacement Response Spectra (ADRS) plane.

The key features and functionalities are:

- **Building structural model creation and visualization:** Enables the creation of the structural model of the building and its visualization in the web-based API, based on a tabular input (.xls or .xlsx file) or a BIM model (.IFC).
- **Analyzing earthquake resilience through linear and non-linear static analyses:** Enables the evaluation of wall safety factors and global vulnerability index of the building to be included in a safety report.
- **Visualization of seismic analysis results:** Provides layouts showing the safety coefficients of the walls for the different failure criteria highlighting the vulnerabilities.

11.4 Methodology, Conceptual Approach

This section outlines the theoretical foundations and computational methodology underpinning the assessment of earthquake resilience module. The approach integrates linear and non-linear static analyses to provide a safety report of the structural performance of the building against seismic hazards in accordance with the EU regulations (Structural Eurocodes and National Building Standards). The main steps of the module are presented in Fig 11.1.

11.4.1 Algorithms, Models, and Assumptions

The earthquake resilience module is based on the E-PUSH routines, an original method for linear and non-linear static analysis of masonry buildings, developed by the research group of the University of Pisa and presented in [14]. The program relies on a simple structural model of the building, which is obtained based on a limited number of input data collected in an excel table: namely, the geometry and the location of the walls, the material properties (the elastic modulus, E, the shear modulus, G, the shear strength, τ_k, and the compressive strength, f_c) and the compressive stress σ_0 induced on them by the quasi-permanent load combination as defined in Eurocode EN1990.

The basic assumptions of the program are the following:

- only wall panels extending from a given floor to the foundations are taken into account;
- each shear wall is assumed to be effective only in its longitudinal direction; therefore, only the lateral stiffness of the wall is considered, disregarding the transverse (out-of-plane) stiffness;
- for non-linear calculations, the capacity curve of each wall is approximated by a bi-linear elastic-plastic curve, where the plastic plateau, defined by the ultimate shear resistance given by the diagonal or sliding shear failure, is bounded by the elastic drift δ_e and the ultimate drift δ_u, for which two different formulations can be set, in terms of a ductility factor or considering and inter-story drift limitation.

The pushover analysis is implemented in Python code and consists of an iterative procedure which starts from the highest floor to the foundation. The lateral forces are increased, and at each step the inter-story drift of each shear resistant wall is compared with the elastic drift, δ_e, and the ultimate drift, δ_u, considering the three possible conditions:

- the wall is still in elastic phase since its displacement δ is lower than δ_e: the stiffness of the wall is the elastic one, k_e, and the shear force H is proportional to the displacement ($H = k_e\delta$);
- the wall is in the plastic phase, $\delta_e < \delta \leq \delta_u$: the shear force is equal to the wall resistance and an apparent stiffness k can be assumed, given by k=($\delta_e k_e$)/δ;
- the wall is collapsed, since its displacement δ is higher than δ_u: its shear resistance and its stiffness are set to zero and the wall is assumed to sustain only vertical loads.

The pushover analysis terminates when the base shear resistance reduces to 80 % of the relative maximum base shear resistance, or the walls belonging to the same floor collapse. Then, the obtained force-displacement capacity curve of the whole structure is first transformed in an equivalent bilinear curve and then converted into an acceleration-displacement capacity diagram for the equivalent single degree-of-freedom (SDOF). The capacity and demand spectrum are finally compared in the Acceleration Displacement Response Spectra (ADRS) plane according to the N2 method ([15]) and safety indexes are evaluated as the ratio between the demand Peak Ground Acceleration (PGA) given by the the hazard map and affected by the site amplification factor (PGA$_D$) and the capacity PGA (PGA$_C$) ([16]).

11.4.2 Calculations and Formulas

The relevant values defining the capacity curves of each wall, namely the elastic stiffness k_e, the shear strength H_Rd, the elastic displacement δ_e and the ultimate displacement δ_u, are computed as

$$k_e = \frac{GLt}{1.2H}\left(1 + \frac{G}{1.2E}\left(\frac{H}{L}\right)^2\right)^{-1} \tag{11.1}$$

$$H_{Rd} = \frac{1.5Lt\tau_k}{b}\sqrt{1 + \frac{\sigma_0}{1.5\tau_k}} \tag{11.2}$$

$$\delta_e = \frac{H_{Rd}}{k_e} \tag{11.3}$$

$$\delta_u = \mu\delta_e \text{ for the ductility check mode} \tag{11.4}$$

$$\delta_u = \gamma_u H \text{ for the drift check mode} \tag{11.5}$$

where L, t and H represent the length, the thickness and the height of the wall, respectively, E and G are the elastic modulus and the shear modulus of the masonry, respectively, τ_k is the masonry shear strength, b is a corrective coefficient, depending on the ratio H/L (1 $\leq$ b = H/L $\leq$ 1.5), σ_0 is the compressive stress on the wall induced by the quasi-permanent load combination, μ is the ductility coefficient and γ_u is the drift ratio.

Verifications of structural resistances are carried out according to the Italian Building Code considering the following formulation for diagonal shear failure,

$$V_{Rd} = \frac{1.5Lt\tau_k}{b}\sqrt{1 + \frac{\sigma_0}{1.5\tau_k}} \tag{11.6}$$

in plane bending moment,

$$M_{Rd,in} = L^2 t\frac{\sigma_0}{2}(1 - \frac{\sigma_0}{0.85f_c}) \tag{11.7}$$

and out-of-plane bending moment

$$M_{Rd,out} = t^2 L\frac{\sigma_0}{2}(1 - \frac{\sigma_0}{0.85f_c}) \tag{11.8}$$

The values of the safety coefficients for each wall are thus calculated as the ratio between resistances and load effects:

$$CS_V = \frac{V_{Rd}}{V_{Ed}} \tag{11.9}$$

$$CS_{M,in} = \frac{M_{Rd,in}}{M_{Ed,in}} \tag{11.10}$$

$$CS_{M,out} = \frac{M_{Rd,out}}{M_{Ed,out}} \tag{11.11}$$

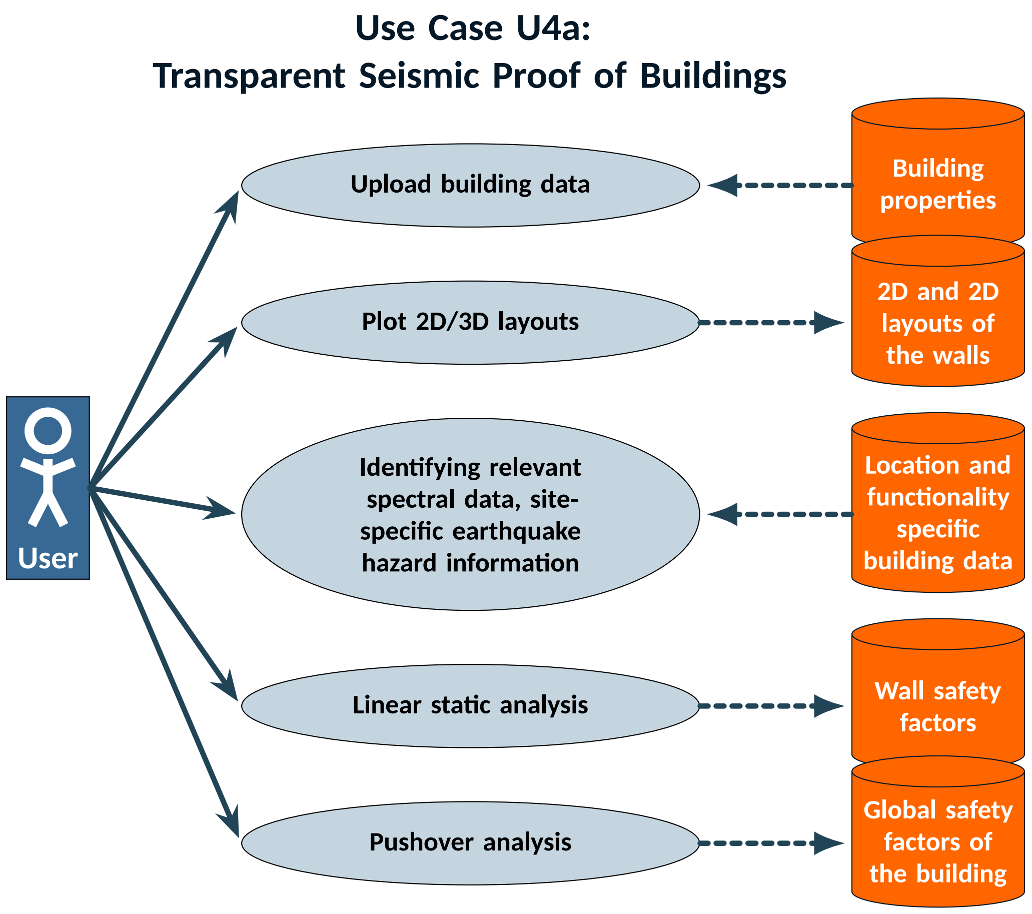

Use Case U4a:
Transparent Seismic Proof of Buildings
User
Upload building data
Plot 2D/3D layouts
Identifying relevant spectral data, site-specific earthquake hazard information
Linear static analysis
Pushover analysis
Building properties
2D and 2D layouts of the walls
Location and functionality specific building data
Wall safety factors
Global safety factors of the building

Figure 11.2: User workflow and data flows for the Earthquake Resilience module. The figure illustrates how users interact with the system to upload building data, visualize layouts, define site-specific seismic hazard parameters, and perform both linear static and nonlinear pushover analyses. The outputs include wall-level safety factors and global vulnerability indices of the building.

11.5 Procedures, Data Workflows

11.5.1 Procedures, User Workflow

This module provides a structured workflow to assess the earthquake resilience of masonry buildings using a combination of linear and nonlinear static analyses. The following UML diagram of Figure 11.2 describe how users can interact with the TRACE-STRUCTURES interface to carry out this evaluation.

11.5.2 Data Flow, Data Formats

A more detailed overview of the data flow is provided in Fig. 11.3. The figure distinguishes between user-provided input data, data imported from either the BUILDCHAIN DKG or local storage, and the outputs resulting from the earthquake resilience compuation pipeline. Each of these components is described in detail below.

Input imported from DKG or other local storage

- Building data retrieved from local hardware (from `.xls` or `.xlsx` file)

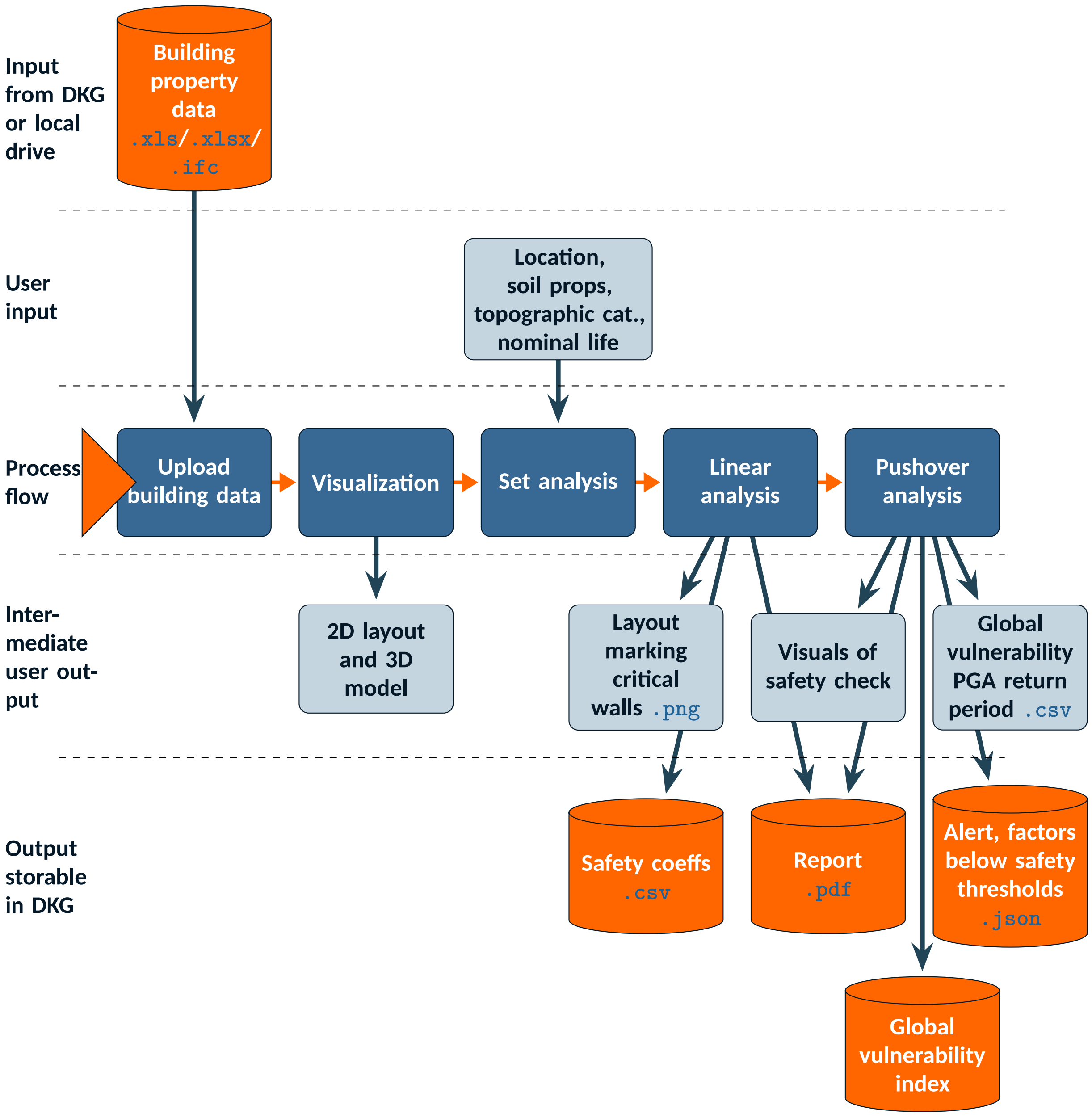

Input from DKG or local drive
Building property data .xls/.xlsx/ .ifc
User input
Location, soil props, topographic cat., nominal life
Process flow
Upload building data
Visualization
Set analysis
Linear analysis
Pushover analysis
Inter-mediate user out-put
2D layout and 3D model
Layout marking critical walls .png
Visuals of safety check
Global vulnerability PGA return period .csv
Output storable in DKG
Safety coeffs .csv
Report .pdf
Alert, factors below safety thresholds .json
Global vulnerability index

Figure 11.3: Data flow of the earthquake resilience module. The user uploads building material and geometric data, and gives other functionality and location specific data for the structural model. This model undergoes linear and nonlinear analyses to assess wall-level and global seismic safety factors.

Other (internal) user inputs:

- Building location;
- Soil Category;
- Topographic category;
- Building nominal life.

Intermediate output:

- Pictures of the layout and 3D structural model;
- Pictures highlighting walls with safety coefficients lower than one;
- CSV files containing safety coefficients for different failure criteria;
- Pictures of the safety verification on the ADRS plane;
- CSV files containing global vulnerability index, PGA and return period corresponding to building capacity.

Output, optionally stored in the DKG:

- Safety factors;
- Global vulnerability index;
- Alert, signals of anomalies, factors and indexes below predefined safety thresholds: stored in json with threshold setting in full in the DKG, linked with BUILDCHAIN API alerting functionality.

11.6 Demo, Alignment with BUILDCHAIN Goals

This module was developed specifically for the BUILDCHAIN project's Use Case U4, titled "*Earthquake and Climate Proof of Buildings Based on Digital Twinning*." It has been tested on both a benchmark toy model and a real-world applications within the BUILDCHAIN Pilots 3 and 4: the Palazzo Poniatowski Guadagni building of Pilot *"Use of DBL for Management of Heritage and Strategic Buildings"*, and one school of the Pilot buildings *"Improved earthquake resilience and carbon footprint of school buildings based on DBL"*. These pilots focus on strategic buildings managed by the Municipality of Florence, with the goal of enhancing the transparency of the structural resilience of these building by providing easily accessible and trustworthy information for the municipality. Here only the former Pilot will be demonstrated.

11.6.1 Toy Model, a Low Complexity Benchmark Building

This module was initially tested on a toy example—a simplified building model featuring only orthogonal walls and three stories. While simple enough to allow easy hand calculations for rapid validation of computational results, the model was also sufficiently complex to include multiple floors and interactions, making it suitable for identifying potential performance bottlenecks within the analysis workflow. The validation process is detailed in the step-by-step user guide in Section 11.8, which demonstrates the workflow using this toy model.

11.6.2 Earthquake Resilience of Palazzo Poniatowski Guadagni

The module was subsequently applied to a more complex, real-world scenario: the structural assessment of the strategic heritage building Palazzo Poniatowski Guadagni in Florence. Unlike the toy example, this historic structure with multiple floors features non-orthogonal wall configurations, and intricate architectural details, making the modeling and analysis more challenging. This application served to demonstrate the module's robustness and scalability in handling realistic structural systems, especially in the context of municipal asset management and cultural heritage preservation. The use of this pilot, carried out under the BUILDCHAIN Use Case 4, aimed to support the municipality of Florence in enhancing transparency and accountability in seismic resilience assessments of its strategic buildings.

11.7 Step by Step Library Package User Guide for the Module

The Python library developed for Module 4 of the TRACE-STRUCTURES platform enables the automated evaluation of seismic resilience for masonry buildings. The package is designed to support transparent, scriptable workflows for creating equivalent frame models, running structural analyses, and extracting safety indicators in line with European standards. It integrates seamlessly with BIM or tabular input data and accommodates both linear static and nonlinear pushover analyses.

To support users in understanding and applying the package, two annotated Jupyter notebooks are provided and included in Annex C.4:

- **Notebook 1: Toy Model Demo** (described in Section 11.6.1 and notebook presented in Annex Section C.4.2): Demonstrates the full pipeline on a simplified three-story building with orthogonal walls. This notebook highlights the step-by-step process of generating the model, performing analyses, and interpreting results, making it ideal for validation and testing.

- **Notebook 2: Real-World Pilot** (desribed in Section 11.6.2 and notebook presented in Annex Section C.4.2): Applies the methodology to the Palazzo Poniatowski Guadagni in Florence—part of the BUILDCHAIN Pilot 3 initiative. It showcases the use of realistic geometry, complex wall orientations, and municipal hazard data to derive both local and global safety insights for a cultural heritage asset.

These notebooks serve both as validation tools and practical templates, allowing engineers and researchers to adopt the module's computational workflow for other case studies involving seismic assessment of masonry buildings. The scripts are modular and documented, encouraging further extension or integration into broader digital twin pipelines.

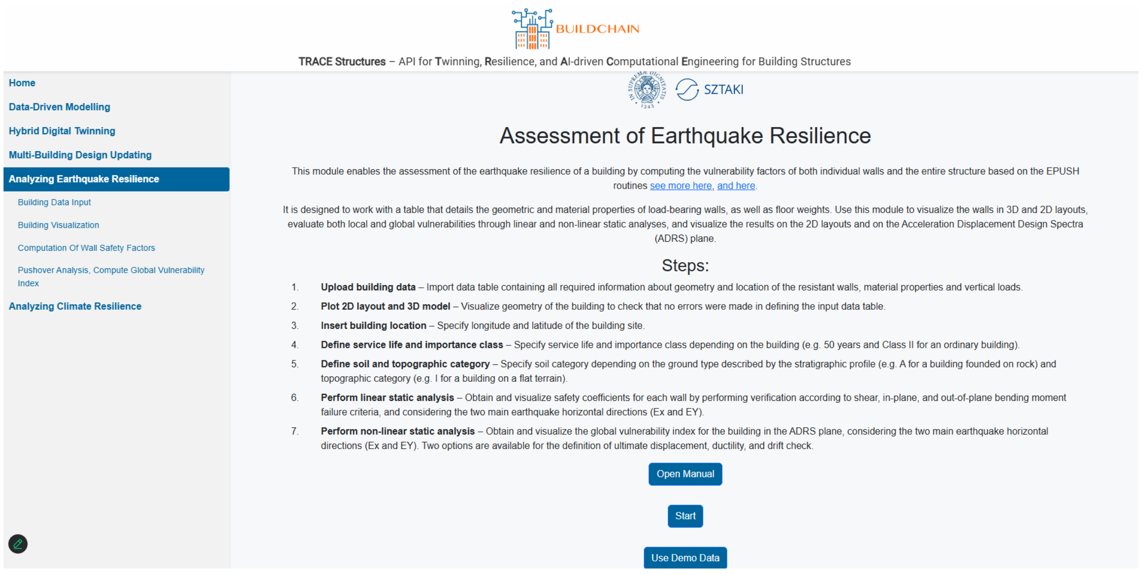
11.8 Step by Step API User Guide for the Module

11.8.1 Step 1: Access the BUILDCHAIN Web Interface

Visit the TRACE-Structures interface at `https://buildchain.ilab.sztaki.hu` and click on `Start app`. Navigate to the "Analyzing Earthquake Resilience" section by either clicking the blue `Analyzing Earthquake Resilience` button or selecting the corresponding item from the left-hand navigation menu (see upper panel of Fig 8.4).

11.8.2 Step 2: Module Selection

On the first page of the module (see Fig 11.4) click on the `Start` button and continue with Step 3. By clicking on `Use Demo Data`, you can load the built-in file of the toy building. The screenshots provided in the following steps of the UI correspond to this example. You can also access the scientific paper of linked to the analysis for theoretical background by clicking `Open Manual`.



BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
SZTAKI
Open Manual
Start
Use Demo Data

Assessment of Earthquake Resilience

This module enables the assessment of the earthquake resilience of a building by computing the vulnerability factors of both individual walls and the entire structure based on the EPUSH routines see more here, and here.

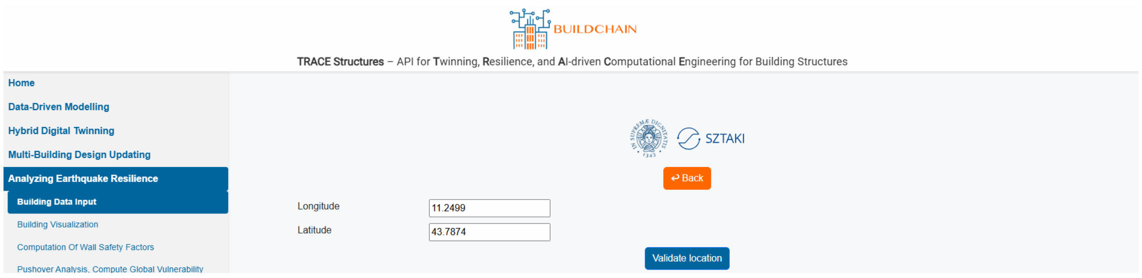
It is designed to work with a table that details the geometric and material properties of load-bearing walls, as well as floor weights. Use this module to visualize the walls in 3D and 2D layouts, evaluate both local and global vulnerabilities through linear and non-linear static analyses, and visualize the results on the 2D layouts and on the Acceleration Displacement Design Spectra (ADRS) plane.

Steps:

1. **Upload building data** – Import data table containing all required information about geometry and location of the resistant walls, material properties and vertical loads.
2. **Plot 2D layout and 3D model** – Visualize geometry of the building to check that no errors were made in defining the input data table.
3. **Insert building location** – Specify longitude and latitude of the building site.
4. **Define service life and importance class** – Specify service life and importance class depending on the building (e.g. 50 years and Class II for an ordinary building).
5. **Define soil and topographic category** – Specify soil category depending on the ground type described by the stratigraphic profile (e.g. A for a building founded on rock) and topographic category (e.g. I for a building on a flat terrain).
6. **Perform linear static analysis** – Obtain and visualize safety coefficients for each wall by performing verification according to shear, in-plane, and out-of-plane bending moment failure criteria, and considering the two main earthquake horizontal directions (Ex and EY).
7. **Perform non-linear static analysis** – Obtain and visualize the global vulnerability index for the building in the ADRS plane, considering the two main earthquake horizontal directions (Ex and EY). Two options are available for the definition of ultimate displacement, ductility, and drift check.

Figure 11.4: Step 2 of module 5 – Getting started

11.8.3 Step 3: Building data input



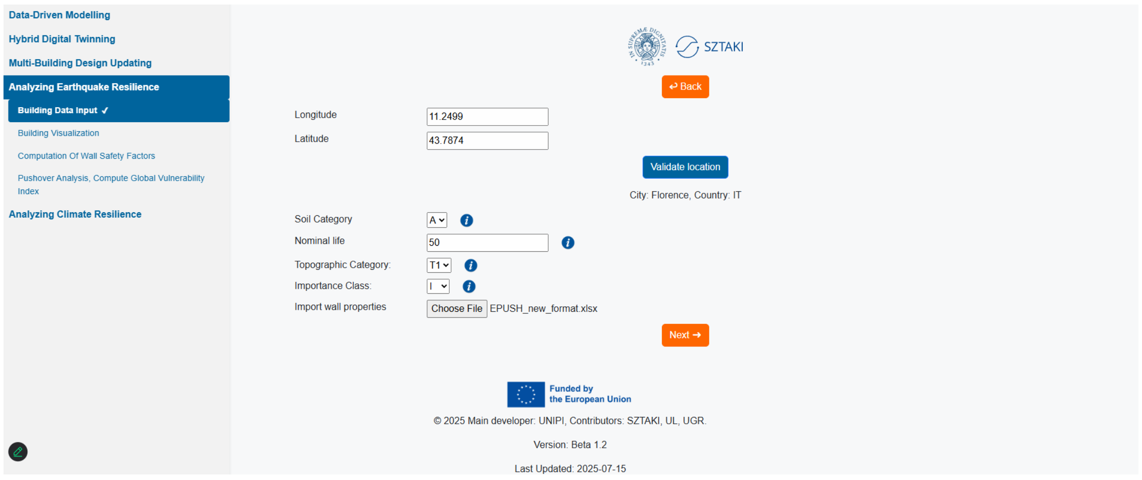
BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability
SZTAKI
↩ Back
Validate location

Longitude 11.2499
Latitude 43.7874

Figure 11.5: Step 3 of mudule 4 – specification and validation of building location.

Click on `Building Data Input` in the left-hand navigation menu if it is not already highlighted. Specify the location of the building (longitude and latitude in decimal degrees), and click on `Validate location` (see upper panel of Fig 11.5).

After validation the UI shows the city and its country that is the closest to the given coordinate. If the shown city looks good, go to the next step, otherwise give a new coordinate. When location successfully given, specify the following (see Fig 11.6): i) soil category, from A (hard rock) to E (very soft soil) according to Eurocode (EC) 8; ii) nominal life, intended design service life expressed in years; iii) topographic category, from T1 (flat terrain) to T4 (upper half of a ridge) according to EC 8; iv) importance class, from Class I (Low importance, agricultural or temporary buildings) to IV (Essential facilities and structures of vital importance for civil protection), according to EC 8. The user specification of parameters defined in the EC 8, such as soil category, nominal life, topographic category and importance class, is guided with relevant information provided in the web module. Finally, upload the excel file containing the data needed to create the structural model and run the analysis.


Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
SZTAKI
↩ Back
Validate location
Next ➔
Choose File
Funded by the European Union

Longitude	11.2499
Latitude	43.7874

City: Florence, Country: IT

Soil Category	A
Nominal life	50
Topographic Category:	T1
Importance Class:	I
Import wall properties	EPUSH_new_format.xlsx

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Figure 11.6: Step 3 of module 4 – building location and functionality specific inputs

11.8.4 Step 4: Building Visualization

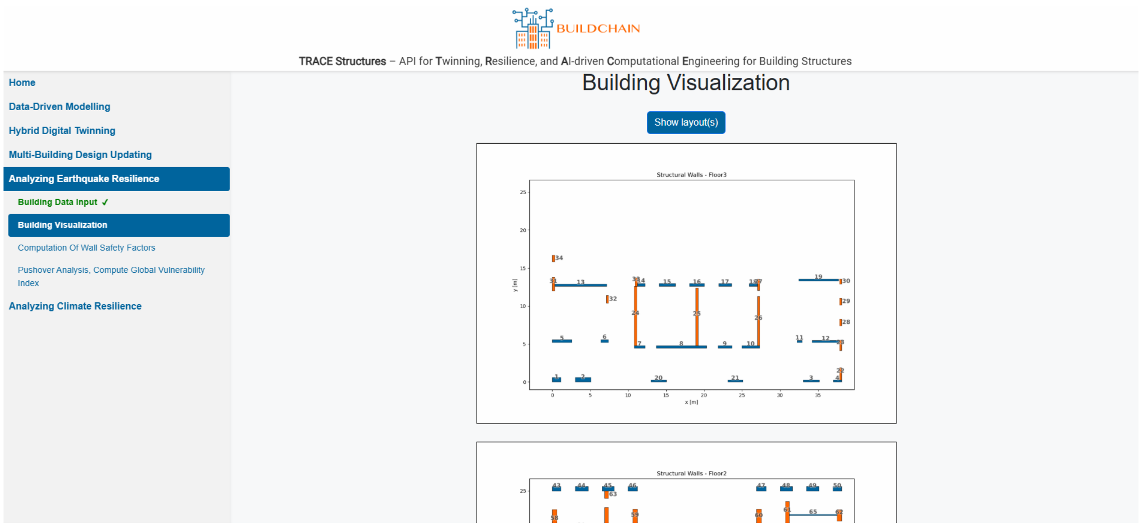

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
Building Visualization
Show layout(s)
Structural Walls - Floor3
y [m]
x [m]
Structural Walls - Floor2

Figure 11.7: Step 4 of module 4 - building visualization, show building layout(s).

Click on `Building Visualization` in the left navigation panel or simply the `Next` button of the previous page and then `Show layout(s)` to obtain the layout of load bearing walls for each floor (Fig 11.7) or `Show 3D model` to obtain a navigable 3D view of the structural model (Fig 11.8).

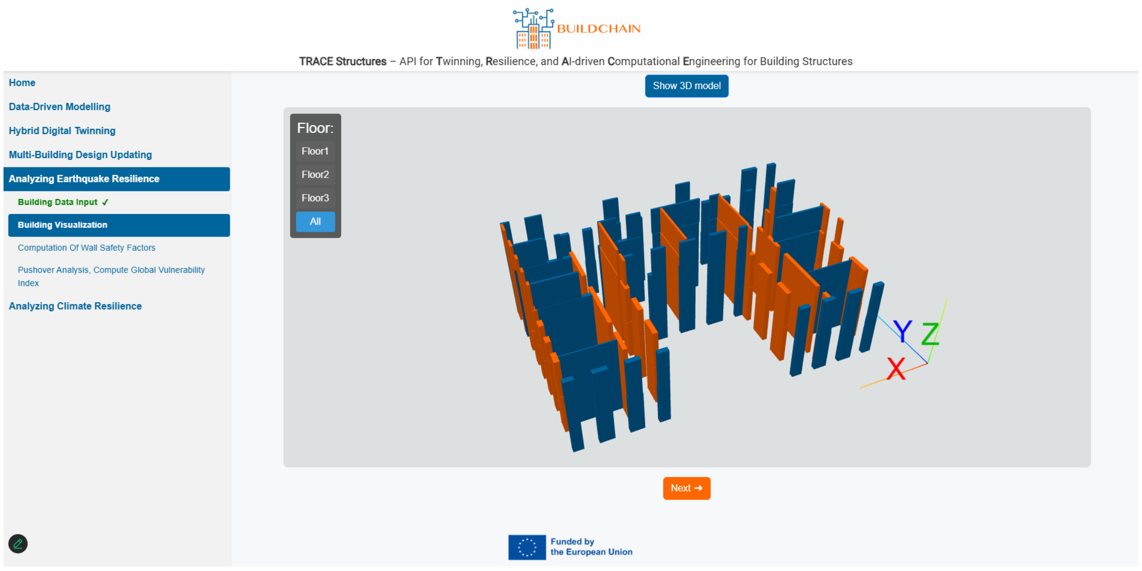
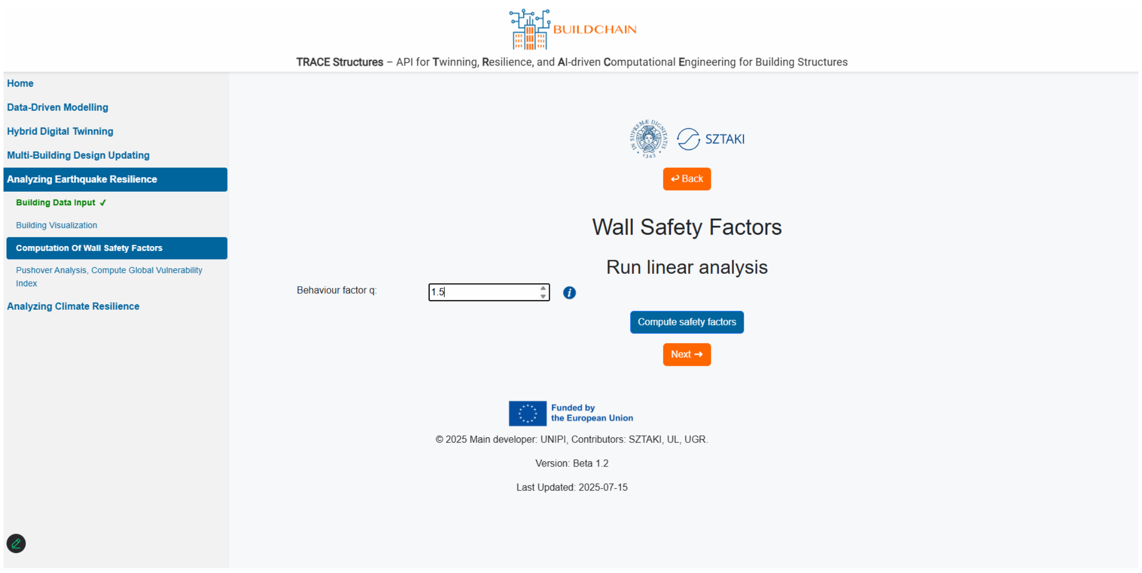

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
Show 3D model
Floor:
Floor1
Floor2
Floor3
All
Y
Z
X
Next →
Funded by the European Union

Figure 11.8: Step 4 of module 4 – building visualization, show 3D model (lower panel) in Building Visualization

11.8.5 Step 5: Computation of Wall Safety Factors

Click on `Computation Of Wall Safety Factors`, and specify the behavior factor *q* which quantifies the expected reduction in seismic demand due to the building's ability to undergo inelastic deformations without collapse (see Fig 11.9). Then, click on `Compute safety factors` to run the linear static analysis. After a few seconds, the possibility to visualize and save wall safety factors will appear (see Fig 11.10).


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
SZTAKI
↩ Back
Wall Safety Factors
Run linear analysis
Compute safety factors
Next →
Funded by the European Union

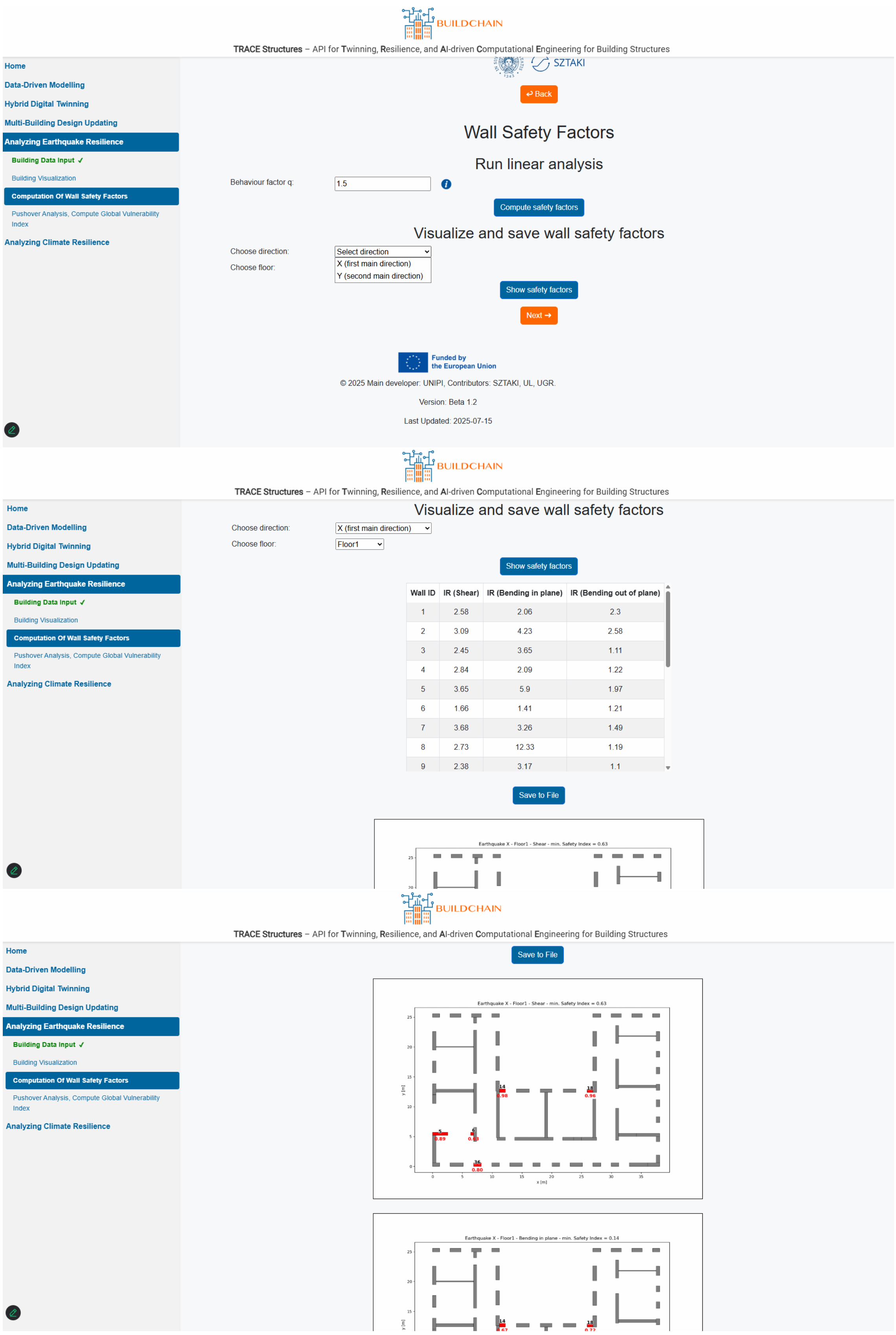
Behaviour factor q: 1.5

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Figure 11.9: Step 5 of module 4 – setting behavior factor for computing wall safety factors


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
SZTAKI
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
↩ Back
Compute safety factors
Show safety factors
Next ➜
Save to File
Funded by the European Union
Earthquake X - Floor1 - Shear - min. Safety Index = 0.63
Earthquake X - Floor1 - Bending in plane - min. Safety Index = 0.14
y [m]
x [m]
14
0.98
18
0.96
5
0.89
6
0.63
36
0.80

Wall Safety Factors

Run linear analysis

Behaviour factor q: 1.5

Visualize and save wall safety factors

Choose direction: Select direction

X (first main direction)

Y (second main direction)

Choose floor:

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Visualize and save wall safety factors

Choose direction: X (first main direction)

Choose floor: Floor1

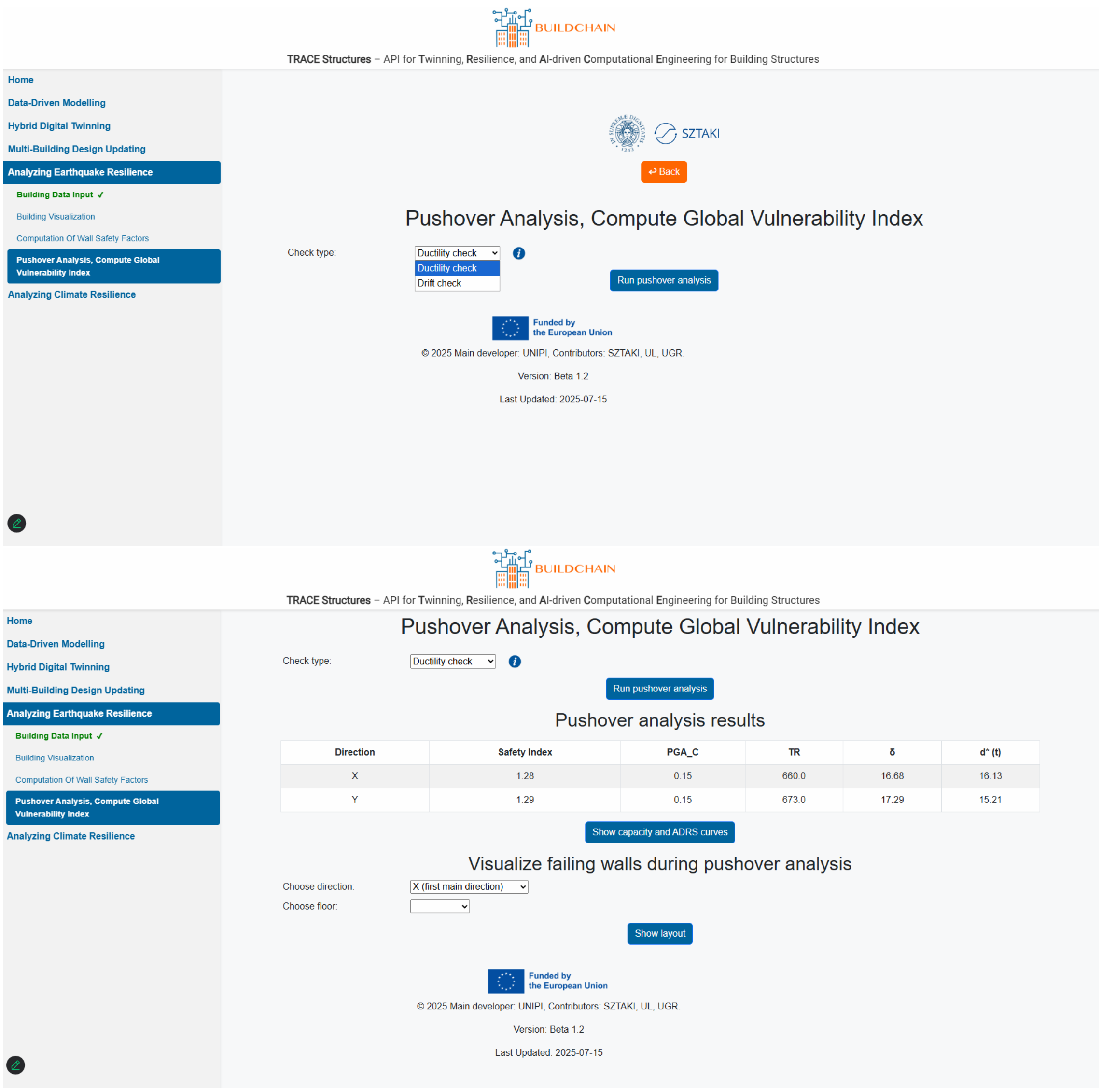
Wall ID	IR (Shear)	IR (Bending in plane)	IR (Bending out of plane)
1	2.58	2.06	2.3
2	3.09	4.23	2.58
3	2.45	3.65	1.11
4	2.84	2.09	1.22
5	3.65	5.9	1.97
6	1.66	1.41	1.21
7	3.68	3.26	1.49
8	2.73	12.33	1.19
9	2.38	3.17	1.1

Figure 11.10: Step 5 of module 4 – choosing main direction and floor for visualizations (upper panel), table of the safety factors (middle panel) and floor layouts highlighting walls with low safety factors (lower panel)

11.8.6 Step 6: Pushover Analysis, Compute Global Vulnerability Index

Click on `Pushover Analysis, Compute Global Vulnerability Index` in the left-hand navigation panel—or simply proceed using the `Next` button from the previous page—to access the pushover analysis settings. Begin by specifying the desired type of verification, referred to as the *"check type"*, which determines how the ultimate displacement of the walls will be assessed. You can choose between a *Drift Check* or a *Ductility Check*, depending on the structural behavior you want to analyze. Once the check type is selected, click on `Run pushover analysis` to initiate the computation (see upper panel of Fig. 11.11). After a brief processing period, results will be displayed in a summary table. This includes the peak ground acceleration at building capacity (PGA$_C$), the corresponding return period (T$_{R,C}$), and the computed seismic safety index (see lower panel of Fig. 11.11).

In addition to the numerical outputs, the interface allows you to i) visualize the results in the Acceleration-Displacement Response Spectrum (ADRS) plane (see upper panel of Fig. 11.12); ii) display the locations of structural failure by highlighting the collapsed walls at the end of the analysis (see lower panel of Fig. 11.12).


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
SZTAKI
↩ Back
Run pushover analysis
Ductility check
Drift check
Funded by the European Union
Show capacity and ADRS curves
Show layout

Pushover Analysis, Compute Global Vulnerability Index

Check type: Ductility check

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Pushover Analysis, Compute Global Vulnerability Index

Check type: Ductility check

Pushover analysis results

Direction	Safety Index	PGA_C	TR	δ	d* (t)
X	1.28	0.15	660.0	16.68	16.13
Y	1.29	0.15	673.0	17.29	15.21

Visualize failing walls during pushover analysis

Choose direction: X (first main direction)

Choose floor:

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Figure 11.11: Step 6 of module 4 – choosing the *check type* of the ultimate displacement of the pushover analysis (upper panel) and the table of the results of the computation (lower panel)

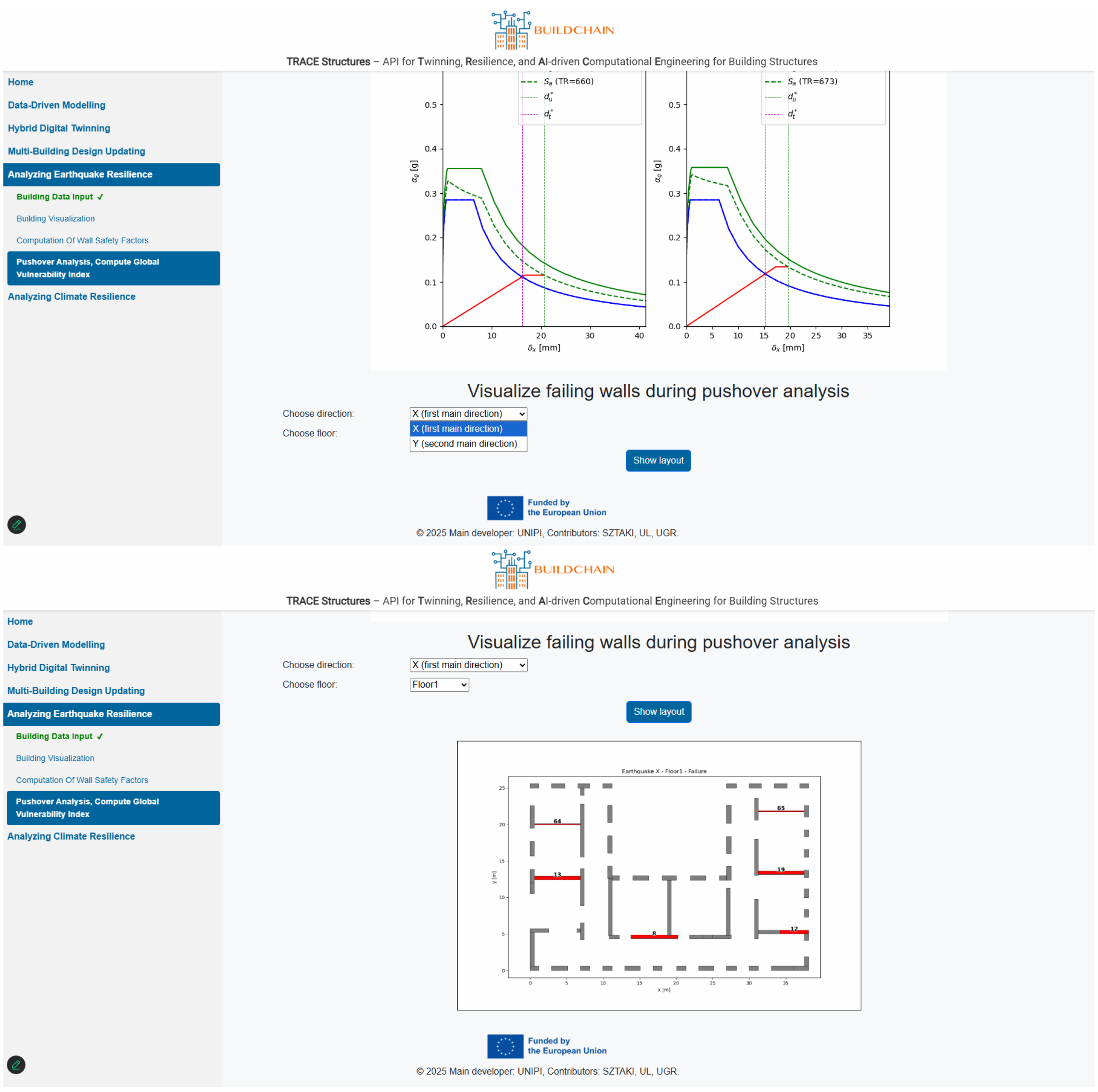

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation Of Wall Safety Factors
Pushover Analysis, Compute Global Vulnerability Index
Analyzing Climate Resilience
S_a (TR=660)
d_u^*
d_t^*
S_a (TR=673)
a_g [g]
δ_x [mm]
Visualize failing walls during pushover analysis
Choose direction:
X (first main direction)
Y (second main direction)
Choose floor:
Floor1
Show layout
Earthquake X - Floor1 - Failure
y [m]
x [m]
Funded by the European Union
© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Figure 11.12: Module 4 - Analyzing Earthquake Resilience: Step 6: Visual results of the pushover analysis: building capacity and seismic demand on the ADRS plane (upper panel) and floor layout with the collapsed walls (lower panel)

11.9 Results of Testing, Interpretation of Outputs

11.9.1 Toy Model, a Low Complexity Benchmark Building

The initial validation of the Earthquake Resilience Module was performed using a simplified toy example—an orthogonal masonry building with three floors. This model was deliberately chosen to strike a balance between analytical tractability and computational relevance. Its geometry allows for easy hand calculations, enabling rapid validation of the module's numerical outputs, while still being complex enough to highlight performance bottlenecks related to multi-storey interaction and wall layout complexity.

The toy model was used to demonstrate the full analysis workflow from importing building and hazard data to visualizing linear safety checks and conducting pushover analysis. Safety coefficients for individual walls were computed using linear static analysis under predefined loading conditions, and results were displayed both numerically and visually through wall layout highlights (see Fig. 11.10).

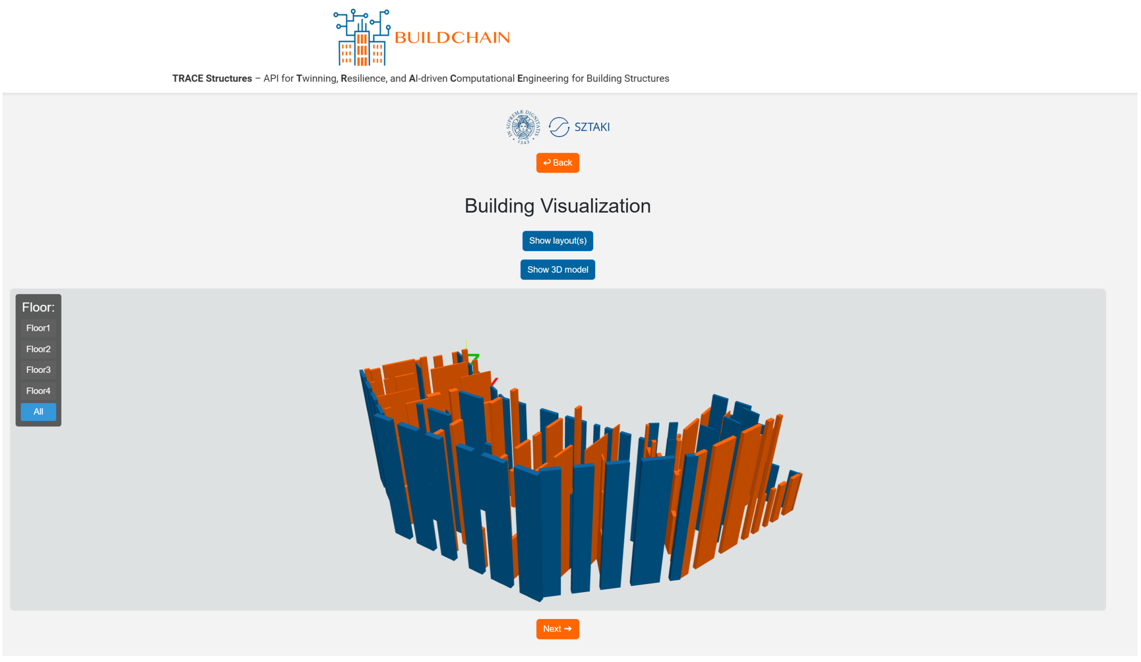

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
SZTAKI
Back
Building Visualization
Show layout(s)
Show 3D model
Floor:
Floor1
Floor2
Floor3
Floor4
All
Next

Figure 11.13: 3D model of Pilot 3, Palazzo Poniatowski Guadagni

Subsequently, a non-linear pushover analysis was performed to compute the global vulnerability index of the structure. This included visualization of results on the Acceleration-Displacement Response Spectrum (ADRS) plane, and identification of collapsed walls based on the defined failure criteria (see Fig. 11.12).

Through this controlled test case, the module's ability to perform end-to-end structural assessment—including input handling, model generation, multi-criteria evaluation, and visualization—was confirmed. The complete sequence of steps can be followed in the user interface walkthrough in Section 11.8, where the toy model serves as the running example.

11.9.2 Pilot 3: Palazzo Poniatowski Guadagni

The Earthquake Resilience Module was also applied to a real-world case study: the historic Palazzo Poniatowski Guadagni in Florence, a key building analyzed within the BUILDCHAIN project under Pilot 3. This pilot focused on demonstrating the use of digital tools for improving the seismic safety transparency of strategic heritage buildings managed by public authorities.

Figure 11.13 illustrates the 3D structural model of the building as rendered in the TRACE-Structures web interface. Unlike the toy model, this building exhibits considerable geometric complexity—including multiple floors and non-orthogonal walls—which posed more realistic modeling challenges. These characteristics made the pilot an ideal test case to evaluate the robustness and flexibility of the module.

The module was used to perform both linear static and non-linear pushover analyses. The linear analysis results revealed wall-level safety factors for different failure mechanisms (shear, in-plane and out-of-plane bending), identifying critical walls with safety coefficients below acceptable thresholds. These results were visualized in both numerical tables and colored wall layouts, highlighting vulnerable sections of the structure.

Following the linear checks, a pushover analysis was conducted to evaluate the building's global seismic performance. This included computing the Peak Ground Acceleration capacity (PGA$_C$), associated return period ($T_{R,C}$), and seismic safety index. Figures 11.14

and presents the corresponding outputs, including the Acceleration-Displacement Response Spectrum (ADRS) visualization and identification of walls reaching collapse under lateral loading.

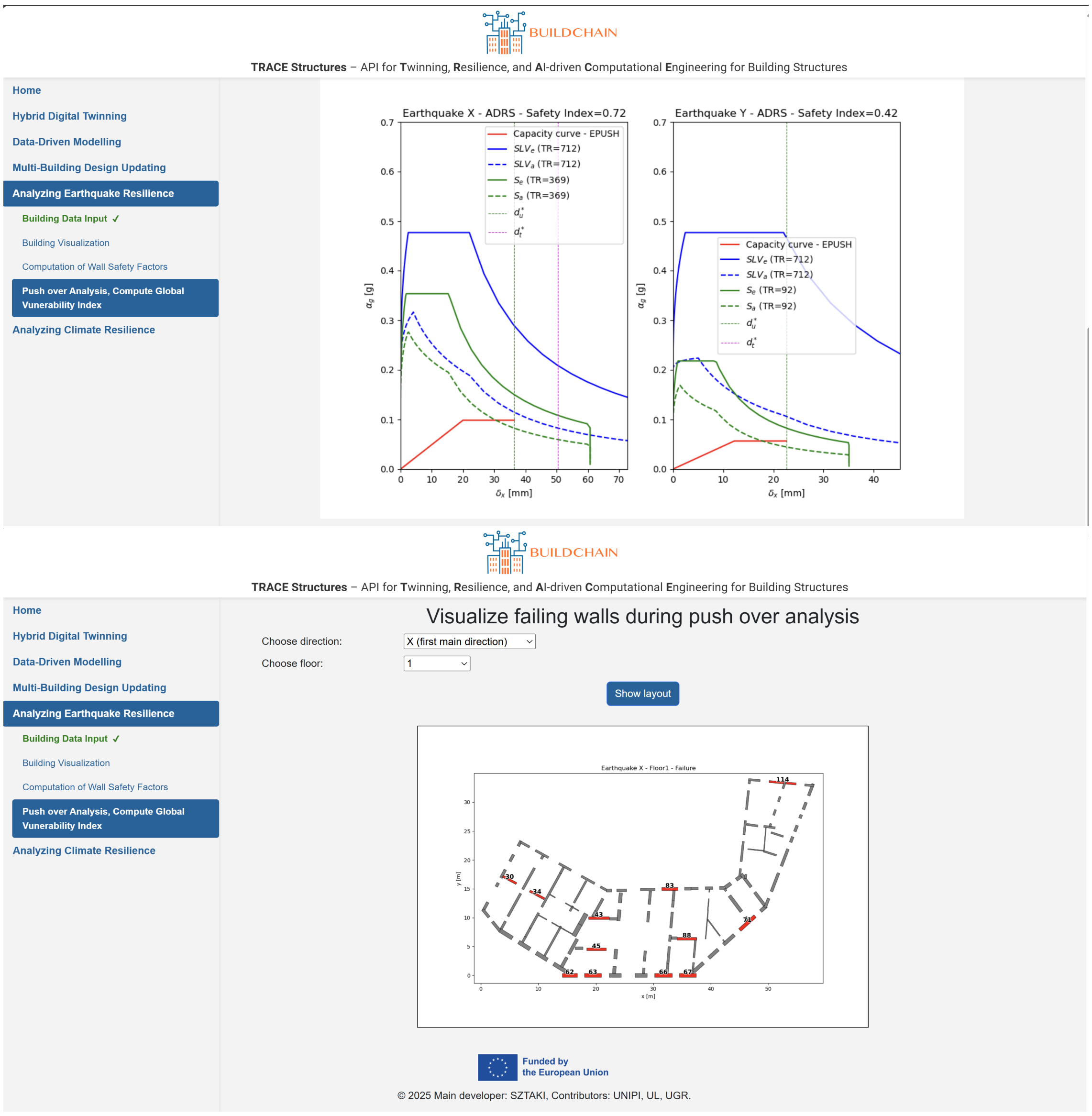

BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Hybrid Digital Twinning
Data-Driven Modelling
Multi-Building Design Updating
Analyzing Earthquake Resilience
Building Data Input ✓
Building Visualization
Computation of Wall Safety Factors
Push over Analysis, Compute Global Vunerability Index
Analyzing Climate Resilience
Earthquake X - ADRS - Safety Index=0.72
Earthquake Y - ADRS - Safety Index=0.42
Capacity curve - EPUSH
SLV_e (TR=712)
SLV_a (TR=712)
S_e (TR=369)
S_a (TR=369)
S_e (TR=92)
S_a (TR=92)
d_u^*
d_t^*
a_g [g]
δ_x [mm]
Visualize failing walls during push over analysis
Choose direction:
X (first main direction)
Choose floor:
1
Show layout
Earthquake X - Floor1 - Failure
y [m]
x [m]
Funded by the European Union
© 2025 Main developer: SZTAKI, Contributors: UNIPI, UL, UGR.

Figure 11.14: Pilot 3 - Palazzo Poniatowski Guadagni: building capacity and seismic demand on the ADRS plane (upper panel) and first floor layout with the collapsed walls (lower panel)

Overall, the results validated the module's capability to process complex structural configurations and to deliver actionable insights on seismic vulnerability, supporting data-informed decision-making for risk mitigation and heritage preservation.

11.10 Strengths and Limitations

The Earthquake Resilience Module offers a transparent, user-friendly environment for the structural assessment of masonry buildings under seismic loads. Its key strengths include:

- **Web-based accessibility and intuitive interface:** The module is integrated into the TRACE-Structures platform, providing a guided workflow for users with varying technical backgrounds.

- **Support for both linear and non-linear static analyses:** Users can compute wall-level safety factors and assess global seismic vulnerability through pushover analysis and ADRS verification.
- **Rapid prototyping of equivalent frame models:** The module supports streamlined model creation using tabular building input formats and visualizes results directly within the platform.
- **Compliance with European structural regulations:** The safety evaluation procedures are aligned with the Eurocodes and national standards.

Despite its robust capabilities, several limitations remain:

- **Lack of enriched IFC integration:** Although IFC models can be used as input, the current implementation does not yet support automatic extraction of structural and material properties from enriched BIM models. As a result, users must manually fill an Excel table to define these inputs.
- **Limited geometric complexity:** The module currently supports only sraight wall partitions and does not handle buildings with curved or irregularly shaped walls.
- **Simplified structural modeling approach:** The module is based on an equivalent frame model and static analysis methods, which do not capture the full complexity of dynamic finite element (FE) simulations. While this simplification may limit the precision of results for highly detailed analyses, it offers a significant advantage in terms of speed and scalability. The approach enables fast, approximate assessments of seismic resilience across a large number of buildings. This balance between simplicity and usability—particularly when enriched BIM support is fully integrated—is a core strength of the module. It empowers municipalities to consistently evaluate, compare, and prioritize their building stock based on standardized earthquake resilience metrics.

Future updates will aim to address these limitations by integrating richer BIM interoperability and extending geometric and material modeling capabilities.

Chapter 12

Module 5: Analyzing Climate Resilience

12.1 Problem Statement and Needs Addressed

Natural hazards pose a growing threat to people, property, and cultural heritage sites. Across Europe, buildings and infrastructure are becoming increasingly vulnerable to the effects of extreme weather. In fact, the frequency and magnitude of extreme events are increasing due to climate change[17], and projections suggest that the situation will continue to worsen in the coming decades [18]. This growing pressure necessitates that climate risks be directly addressed in the design, renovation, and management of the built environment [19]. For this reason, having a safe, strong, and resilient built environment has become one of the most challenging tasks of our time. This can be done, as suggested by the EU Guideline on the adaptation of buildings to climate change [20], by conducting a Climate Vulnerability and Risk Assessment (CVRA), which analyzes the sensitivity, exposure, and vulnerability of existing structures. From this point forward, when significant risks are identified, appropriate and timely adaptation measures should be implemented.

12.2 Tool Overview, purpose, core functionality

The developed module can help assess climate, particularly in assessing the exposure of buildings. The Python-based libtrary package climate-resilience enables the extraction of climate data from the Copernicus Data Store [21], customized according to the building's geographic location. In particular, high-resolution climate projections from the EURO-CORDEX initiative [22], covering the period from 1951 to 2100 and the RCP4.5 and RCP8.5 emission scenarios, are utilized. The climate variables are analysed through the Extreme Value Theory (EVT) using the block maxima method. Then, characteristic values are determined based on different probabilities of exceedance, for example, 2 percent per year, in line with EN 1990 recommendations for climatic actions on structures. To better understand the trend of characteristic values in the future, a factor change approach is applied [23]. The user will be able to select various parameters, including geographical coordinates, climate variables, experiment type, time windows, and return period. As the final output, the user will be able to view the trend of the factor of change under a specific scenario of climate variables for the selected location. This result provides an initial assessment aimed at understanding the future climate-related trend in the selected location. The module is designed to integrate with the BUILDCHAIN DBL by the possibility of importing annual climate data directly from the DBL database using oracles, ensuring that the data is trustworthy and traceable, rather than sourcing it directly from Copernicus. Furthermore, the user interface allows the processed results to be exported back to the decentralized knowledge graph based DBL. This supports a shared knowledge base, providing engineers with characteristic climate values projected over the coming century to inform structural design and decision-making.

12.3 Key Features and Capabilities

This module, integrated into the TRACE-STRUCTURES platform, provides the following core functionalities:

- Automated climate data extraction from the Copernicus platform via the Copernicus API using the geographic coordinates (latitude and longitude)
- Extreme value analysis obtaining characteristic values, factor of change of the characteristic value, mean value, coefficient of variance, yearly maxima, and Gumbel probability paper of the climate models
- Support for different scenarios selecting various climate variables (e.g. maximum temperature), emissions scenarios (e.g. RCP 4.5 and RCP 8.5), time-windows (e.g., 40-year windows shifted every 10 years), return periods (e.g., 50 years), and statistical methods (e.g., Method of moments)

The key features and functionalities are:

- **User-friendly interface**: allow intuitive input by selecting the option directly from the interface.
- **Token-based access to Copernicus**: connection through the Copernicus data-store using the personal API token provided directly from the Copernicus website.
- **Token-based access to the DKG**: Enables secure access to climate data via oracles through the BUILDCHAIN Distributed Knowledge Graph (DKG), ensuring traceability. Also supports exporting estimated future climate impacts on buildings back to the DKG.
- **Scenario-based analysis**: modifying different features allows for various analyses combination.
- **Analysis scalability** using data from the Copernicus platform ensures robust and spatially accurate climate extension, making it scalable for different European cities.

12.4 Methodology, Conceptual Approach

This section outlines the theoretical foundations and computational methodology underpinning the climate resilient buildings module. The approach integrates the acquisition of climate projections from climate models, the extreme value analysis and the use of factors of change method to describe climatic actions trends until the end of the century. The main steps of the module is presented in Fig 12.1.

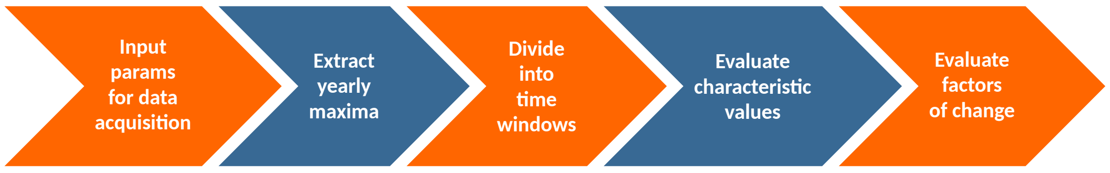

Input params for data acquisition
Extract yearly maxima
Divide into time windows
Evaluate characteristic values
Evaluate factors of change

Figure 12.1: Methodology, conceptual approach of using change factors method with the aim of quantify climate risk under different future scenarios

12.4.1 Algorithms, Models, and Assumptions

Input parameters

Daily projections from climate models are available within the Copernicus CDS dataset ”*Cordex regional climate model data on single levels*” [24]. To obtain these data, two requests for each single combination of model, variables, and scenario should be formulated by means of Earthkit, an open-source Python library. The first request (*request_orography*), related to the variable orography, is to access to all the coordinates of the centroid of the grid cell of the model. Then, the python function named *find_extract_coords* allows to convert the geographical coordinates of the building (lat,lon) into rotated coordinates of latitude and longitude (rlat,rlon) of the closest point of the grid. These rotated coordinates are placed into the “area” parameter of the second request (*request_model*), that one dedicated to the acquisition of projections, by means of ”area”: coordinates.get(”area_orography_florence”).
The first request must follow the format as below:

```
requests_orography = [
    (
        "orography",
        {
            "domain": "europe",
            "experiment": "historical",
            "horizontal_resolution": "0_11_degree_x_0_11_degree",
            "temporal_resolution": "fixed",
            "variable": ["orography"],
            "gcm_model": gcm,
            "rcm_model": rcm,
            "ensemble_member": ensemble_orography,
        },
    ),
]
```

Listing 12.1: Input parameter of request for orography data

The second request must follow the format as below:

```
request_models = [
            (
                model,
                {
                    "domain": "europe",
                    "experiment": ["historical", "rcp_8_5"],
                    "horizontal_resolution": "0_11_degree_x_0_11_degree",
                    "temporal_resolution": "daily_mean",
                    "variable": "maximum_2m_temperature_in_the_last_24_hours",
                    "gcm_model": gcm,
                    "rcm_model": rcm,
                    "ensemble_member": ensemble_model,
                    "start_year": start_years,
```

Table 12.1: Overview of input parameters for request

GCM	RCM	Experiment	Ensemble	Period
ICHEC-EC-EARTH	RACMO22E	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
MOHC-HadGEM2-ES	RACMO22E	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2099
NCC-NorESM1-M	HIRHAM5	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
NCC-NorESM1-M	REMO2015	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
MOHC-HadGEM2-ES	REMO2015	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2099
CNRM-CM5	RACMO22E	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
MPI-ESM-LR	CCLM4-8-17	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
MOHC-HadGEM2-ES	HIRHAM5	Historical, RCP 4.5, RCP 8.5	r1i1p1	1951–2100
ICHEC-EC-EARTH	HIRHAM5	Historical, RCP 4.5, RCP 8.5	r3i1p1	1951–2100
MOHC-HadGEM2-ES	CCLM4-8-17	Historical, RCP 4.5, RCP 8.5	r3i1p1	1951–2099
ICHEC-EC-EARTH	CCLM4-8-17	Historical, RCP 4.5, RCP 8.5	r12i1p1	1951–2100

```
                "end_year": end_years,
                "area": coordinates.get(
                    "area_orography_florence"
                ),
            },
        ),
    ]
```

Listing 12.2: Input parameter of request for model projections

Since many variables are investigated, the inputs for the parameter ”variable” can be: ”minimum_2m_temperature_in_the_last_24_hours”, ”10m_wind_speed” and ”mean_precipitation_flux”. In general, several variables differ for the input value of some parameters, i.e. the ensemble member or the years can vary from maximum to minimum temperature. Some request parameters are the same between different models and variables, i.e. ”domain”, “horizontal_resolution”, “temporal_resolution”. For the other, it is possible to choose between several options, as shown in the table above (Table 12.1), which is valid for maximum temperature.

Climate projections from the selected dataset are provided in a temporal aggregation of 5 years, so the “start_year” and “end_year” parameters are list of years started from 1951 and 1955 and shifted by 5, respectively. There are some exceptions.

Yearly maxima and time windows

In accordance with extreme value analysis, carried out following the block maxima approach, yearly maximum values should be extracted from daily data series. Since climate projections are more reliable when expressed in relative rather than absolute terms, yearly maxima are then divided into time windows to reduce associated uncertainties. A 40-year window length, shifted progressively by 10 years, is often adopted to capture the influence of climate change on the analyzed variables: 1950-1989, 1960-1999, ..., 2060-2099. Within these intervals, the assumption of stationary climate conditions is adopted.

Characteristic values

Characteristic values are evaluated in accordance with the extremal types theorem, which states that the distribution of block maxima converges to one of three possible limiting distributions, usually referred as Gumbel (extreme value type I), Fréchet (extreme value type II), or Weibull (extreme value type III) distributions. Despite their distinct behaviour, these distributions are encompassed within a unified family known as the Generalized Extreme Value (GEV) distribution, also referred to as the Fisher-Tippett distribution [25]. The selection of the appropriate distribution depends on various factors, including the nature of the variable under study and its geographical context. Following the methodology adopted for estimating characteristic climatic values throughout Europe, the Gumbel distribution (Type I) has been chosen to represent the behavior of extreme climate events. Type I distribution is a subset of GEV family, occurred when the shape parameter is null. The mathematical formulation of the Gumbel distribution is provided in the following section. So, if Extreme value type I distribution is assumed as the limiting distribution for maxima, the value of the characteristics of the climatic action, c_k is determined by the inverse formula, as proposed in 12.4.2.

Factors of change

Once results from extreme value analysis are obtained, it's possible to evaluate the factors of change, which are concise representations of changes of characteristic values of climatic actions, based on the elaboration of data over subsequent time windows (n) of constant length [25]. To define Factor of Change (FC) for each request it's necessary to compare the characteristic values of each time window with the one referred to the first one. FCs are calculated differently for each variable.

12.4.2 Calculations and Formulas

The cumulative distribution function of Gumbel distribution, assumed the convergence to a Gumbel distribution of block maxima is

$$F(x < X) = \text{exp}\left\{-\,\text{exp}\left[-\left(\frac{x-\mu}{\sigma}\right)\right]\right\}, \quad \mu \in \mathbb{R};\ \sigma > 0 \tag{12.1}$$

where the location parameter, μ, specifies the centre of the distribution and the scale parameter, σ, determines the size of deviations around the location parameter. Parameters can be estimated by means of several method, indicate respectively as MLM for Maximum Likelihood Method, LSM for Least Square Method, and MOM for Method of Moment. To evaluate characteristic value, c_k, following inverse formula is used

$$c_k = \mu + \sigma \cdot \{-\ln\left[-\ln(1-p)\right]\}, \tag{12.2}$$

According to EN 1990, the probability of exceedance in one year p is equal to 0.02. Factors of change are evaluated in terms of differences (delta changes), for temperature

$$FC_k(n) = c_k(n) - c_k(n=1). \tag{12.3}$$

and in terms of ratio, for other variables

$$FC_k(n) = c_k(n)/c_k(n=1). \tag{12.4}$$

where n represents the n-th time window.

12.5 Procedures, Data Workflows

12.5.1 Procedures, User Workflow

This module provides a structured workflow for assessing the future climate-related effects that buildings may be subjected to. While it does not perform a full resilience analysis, it supports such evaluations by offering critical input data on expected climate impacts. The UML diagram in Figure 12.2 illustrates how users can interact with the TRACE-STRUCTURES interface to conduct this assessment.

12.5.2 Data Flow, Data Formats

A more detailed overview of the data flow is provided in Fig. 12.3. The figure distinguishes between user-provided input data, data imported from either the BUILDCHAIN DKG or local storage, and the outputs resulting from the climate resilience computation pipeline. Each of these components is described in detail below.

12.6 Demo, Alignment with BUILDCHAIN Goals

This module directly supports one of the BUILDCHAIN project's goals by enabling climate analysis as a means to support a more resilient built environment in the face of extreme events, which are increasingly emphasized due to climate change. Through an intuitive interface, the module enables users to perform a climate analysis using data directly from the Copernicus website. This is related to other objectives of the project, such as trustworthiness, transparency, and traceability of logbook data. A demo use case was performed for the city of Florence, related to Pilot 4 of the project (e.g., school buildings in the city center of Florence), where the module was used to conduct extreme value analysis to understand the future climate trend. Especially, the trend of climate variables for the Florence area is evaluated under various scenarios, enabling an analysis of the city's exposure to extreme events. These results provide a solid foundation for conducting a Climate Vulnerability and Risk Assessment (CVRA) of the building, as recommended by EU guidelines, which involves exposure, sensitivity, adaptive capacity, likelihood, and impact.

12.7 Step by Step Library Package User Guide for the Module

The Python library developed for Module 5 of the TRACE-STRUCTURES platform supports the analysis of future climate exposure for buildings using downscaled climate projections. It provides a structured, scriptable workflow to acquire, process, and interpret high-resolution climate data from the Copernicus Climate Data Store (CDS). The package is specifically designed to estimate characteristic values and factors of change of key climatic variables, such as temperature, wind, and precipitation, based on Extreme Value Theory (EVT).

The tool enables engineers and planners to quantify the impact of future climate scenarios on building exposure, supporting early vulnerability screening and adaptation planning. While it does not perform structural simulations, it plays a key role in identifying trends in climatic actions that may affect buildings in the coming decades.

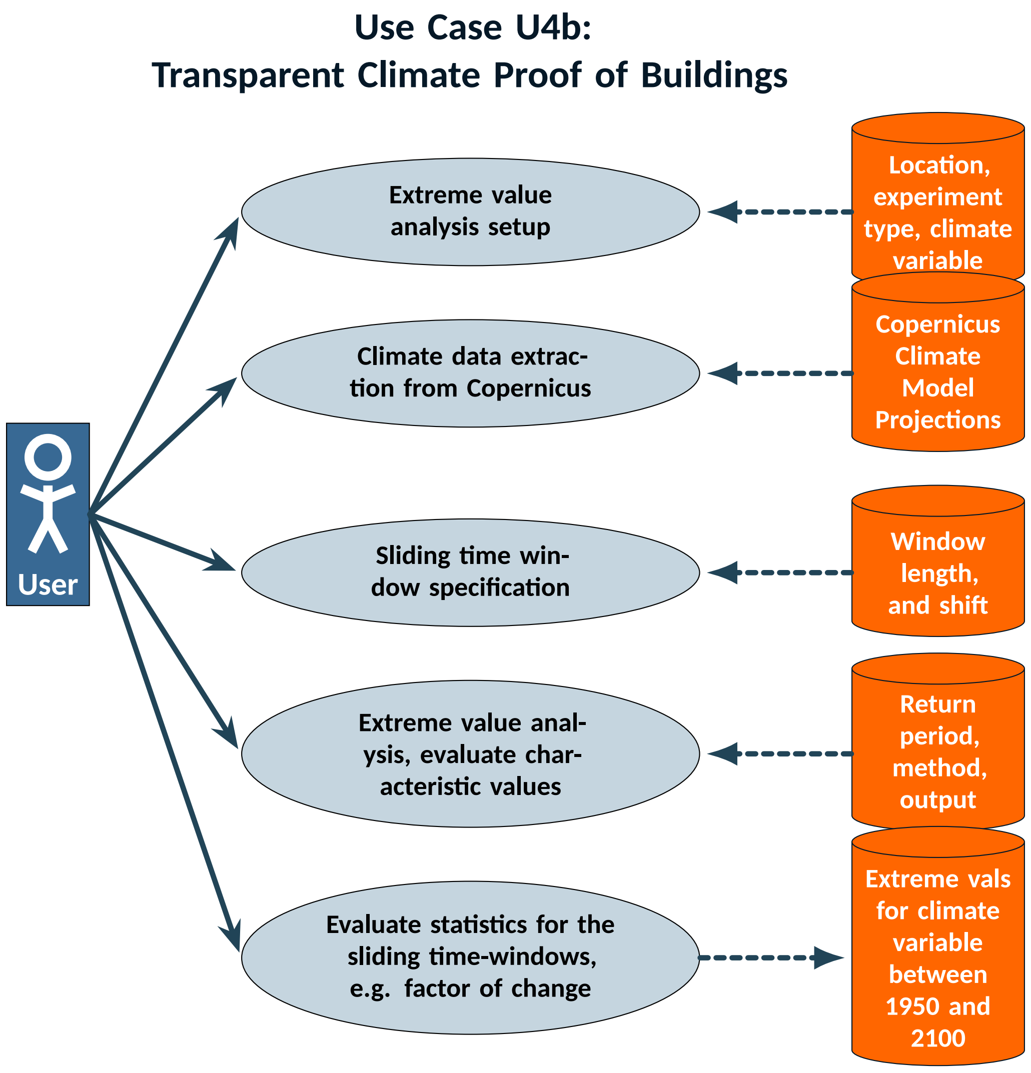

Use Case U4b:
Transparent Climate Proof of Buildings
User
Extreme value analysis setup
Climate data extraction from Copernicus
Sliding time window specification
Extreme value analysis, evaluate characteristic values
Evaluate statistics for the sliding time-windows, e.g. factor of change
Location, experiment type, climate variable
Copernicus Climate Model Projections
Window length, and shift
Return period, method, output
Extreme vals for climate variable between 1950 and 2100

Figure 12.2: Workflow and data flow for the Climate Resilience module. This diagram illustrates the user interaction with the system: starting from the specification of input parameters (e.g., location, variable, emission scenario), followed by the extraction of yearly extreme values (e.g., maxima), the definition of sliding time-window settings, and the selection of statistical methods. The final step involves computing and visualizing climate statistics—such as characteristic values or factors of change—across the defined time windows for the selected climate variable.

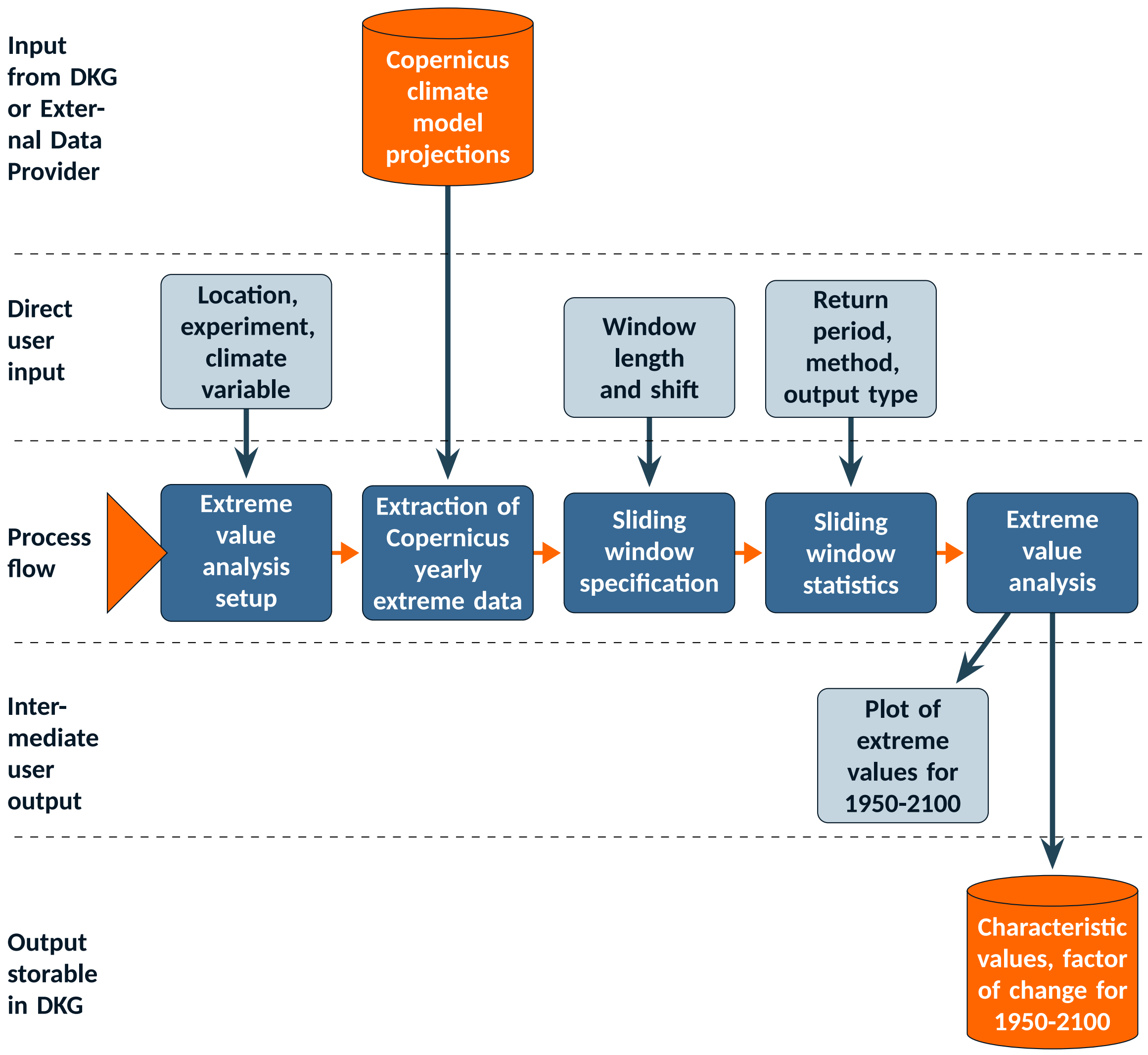

Input from DKG or External Data Provider
Copernicus climate model projections
Direct user input
Location, experiment, climate variable
Window length and shift
Return period, method, output type
Process flow
Extreme value analysis setup
Extraction of Copernicus yearly extreme data
Sliding window specification
Sliding window statistics
Extreme value analysis
Intermediate user output
Plot of extreme values for 1950-2100
Output storable in DKG
Characteristic values, factor of change for 1950-2100

Figure 12.3: Workflow of the Climate Resilience Module.

To guide users in applying the tool and reproducing its workflow, an annotated Jupyter notebook is included and referenced in Annex C.5. The notebook applies the climate resilience pipeline to a real cultural heritage site in Florence. Climate data is downloaded, transformed, and visualized to show the projected evolution of temperature extremes over the next decades under different emission scenarios.

This notebook serves as both validation example and ready-to-use template for assessing climate exposure trends. The Python codebase is modular, transparent, and easily extensible, supporting integration into broader digital twin or CVRA (Climate Vulnerability and Risk Assessment) frameworks.

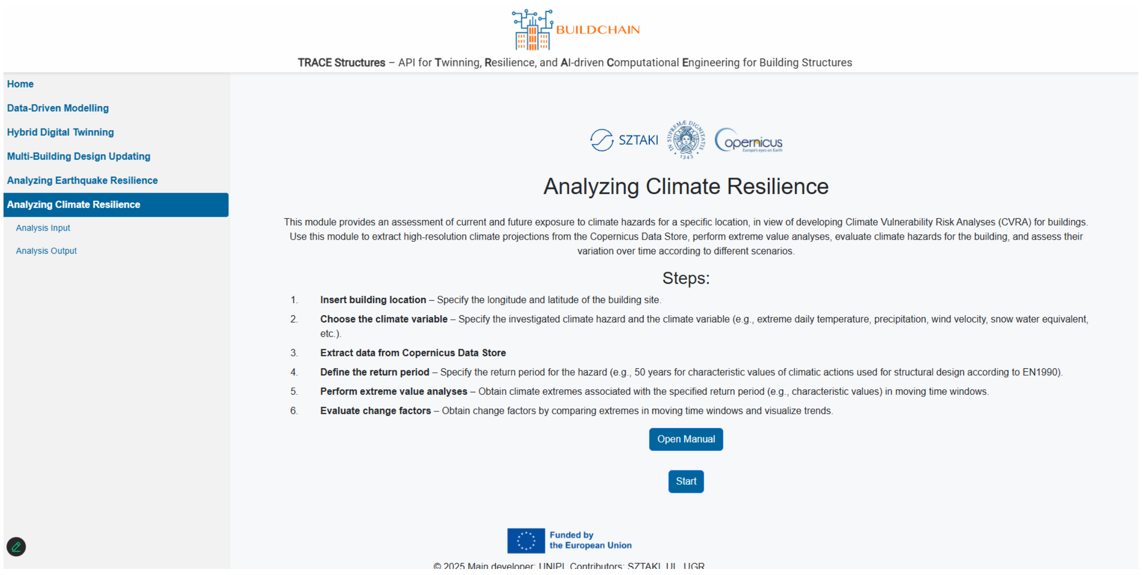
12.8 Step by Step API User Guide for the Module

12.8.1 Step 1: Access the BUILDCHAIN Web Interface

Visit the TRACE-Structures interface at `https://buildchain.ilab.sztaki.hu` and click on `Start app`. Navigate to the ”Analyzing Climate Resilience” section by either clicking the blue `Analyzing Climate Resilience` button or selecting the corresponding item from the left-hand navigation menu (see upper panel of Fig 8.4).

12.8.2 Step 2: Module Selection

On the first page of the module (see Fig 12.4) click on the `Start` button and continue with Step 3. By clicking on `Open Manual`, you can access the scientific paper linked to the analysis for theoretical background.



TRACE Structures – API for **T**winning, **R**esilience, and **A**I-driven **C**omputational **E**ngineering for Building Structures

Analyzing Climate Resilience

This module provides an assessment of current and future exposure to climate hazards for a specific location, in view of developing Climate Vulnerability Risk Analyses (CVRA) for buildings. Use this module to extract high-resolution climate projections from the Copernicus Data Store, perform extreme value analyses, evaluate climate hazards for the building, and assess their variation over time according to different scenarios.

Steps:

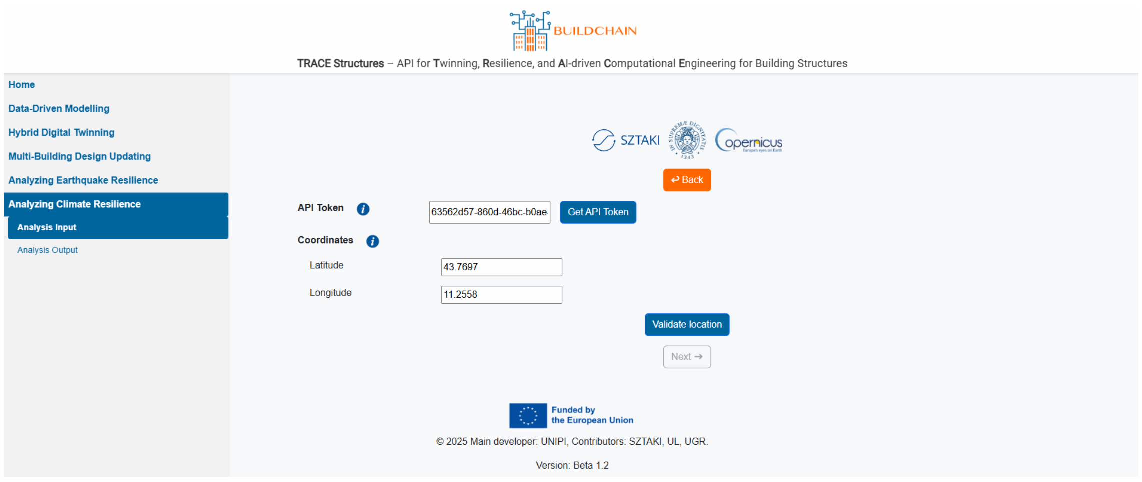
1. **Insert building location** – Specify the longitude and latitude of the building site.
2. **Choose the climate variable** – Specify the investigated climate hazard and the climate variable (e.g., extreme daily temperature, precipitation, wind velocity, snow water equivalent, etc.).
3. **Extract data from Copernicus Data Store**
4. **Define the return period** – Specify the return period for the hazard (e.g., 50 years for characteristic values of climatic actions used for structural design according to EN1990).
5. **Perform extreme value analyses** – Obtain climate extremes associated with the specified return period (e.g., characteristic values) in moving time windows.
6. **Evaluate change factors** – Obtain change factors by comparing extremes in moving time windows and visualize trends.

© 2025 Main developer: UNIPI. Contributors: SZTAKI, UL, UGR

Figure 12.4: Step 2 of module 5 - Getting started

12.8.3 Step 3: User inputs - building location

Click on `Analysis input` in the left-hand navigation menu if it is not already highlighted. To request data from Copernicus, you need to have an API Token first. If you do not have an API token, you can follow the `Get API Token` button to login/register on the Copernicus website. After registration, and signed in to Copernicus website, navigate to ”Your Profile”, copy your API token and paste it into the corresponding field. Afterwards, specify the location of the building (longitude and latitude in decimal degrees), and click on `Validate location` (see Fig 12.5). After validation the UI shows the city and its country that is the closest to the given coordinate. If the shown city looks good, go to the next step, otherwise give a new coordinate.



TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

API Token: 63562d57-860d-46bc-b0ae

Coordinates
Latitude: 43.7697
Longitude: 11.2558

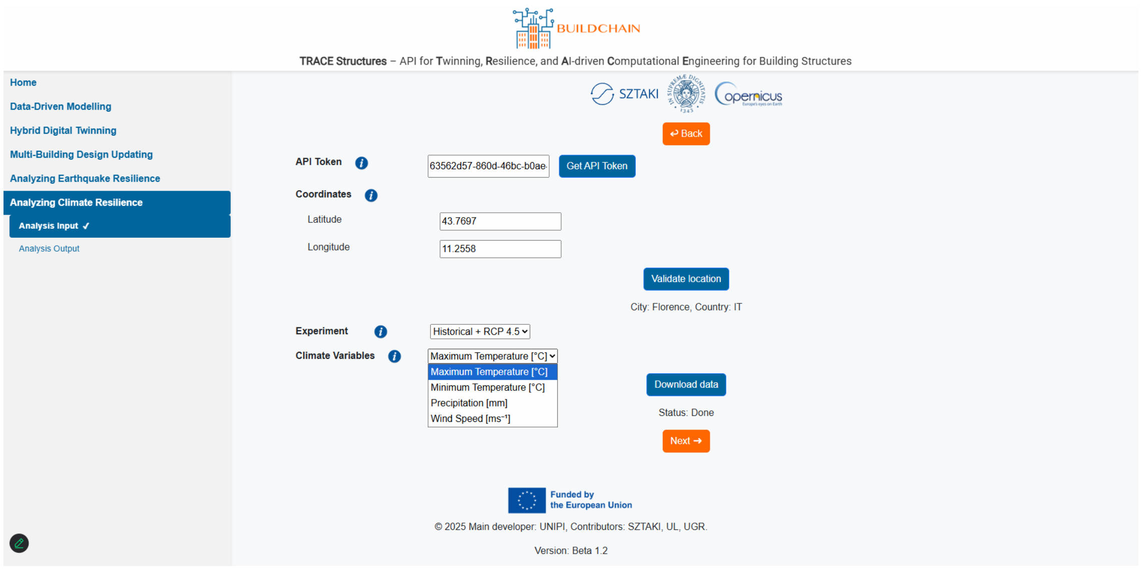
© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Figure 12.5: Step 3 of module 5 - Input API Token and building coordinates

12.8.4 Step 4: Specification of Experiment and Climate Variable, Extraction of Climate data

When location successfully given, specify the experiment (*Historical + RCP 4.5* or *Historical + RCP 8.5*)[1] and the climate variable to be analyzed (possible choices are *Minimum Temperature*, *Maximum Temperature*, *Precipitation* and *Wind Speed*) for the data request from Copernicus (see Fig 12.6).



TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

API Token: 63562d57-860d-46bc-b0ae

Coordinates
Latitude: 43.7697
Longitude: 11.2558

City: Florence, Country: IT

Experiment: Historical + RCP 4.5

Climate Variables: Maximum Temperature [°C]
- Maximum Temperature [°C]
- Minimum Temperature [°C]
- Precipitation [mm]
- Wind Speed [ms⁻¹]

Status: Done

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Figure 12.6: Step 4 of module 5 - Choosing experiment and variable for data request.

[1]**RCP 4.5**: This is a Representative Concentration Pathway, a scenario used for future climate projections. RCP 4.5 **assumes moderate emissions**, with greenhouse gases stabilizing by 2100 due to climate policies and technology changes. Future scenario is **RCP 8.5**, which **assumes high greenhouse gas emissions continuing into the future**, with little or no mitigation efforts.

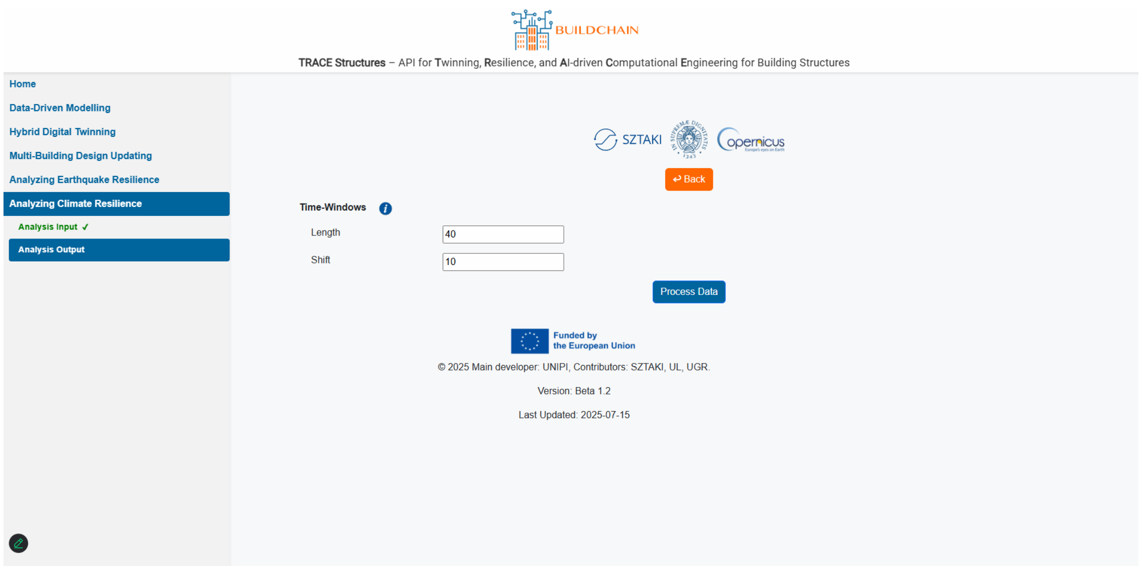
By clicking on `Download data`, a request is sent through the Copernicus API using the provided API token and selected experiment to retrieve daily data for the chosen variable within the specified coordinates, covering the years 1950 to 2100. The download process may take several minutes. You can proceed to Step 5 once the download is complete (the status will change to "Done" when finished) by clicking on the `Next` button.

12.8.5 Step 5: Sliding Time-Window Specification

In this step, the user specifies how the climate data will be segmented over time to assess long-term trends and future climate risks. Define two key parameters: the **length** and **shift** of the time window (see Fig. 12.7).

- **Length**: This sets the duration (in years) of each analysis window. For example, setting the length to 40 means each statistical evaluation will be conducted over 40-year periods.
- **Shift**: This defines how much the window moves forward between evaluations. A shift of 10 means each successive window starts 10 years after the previous one, resulting in overlapping intervals (e.g., 1950–1989, 1960–1999, 1970–2009, etc.).

After defining the time-window settings, click on the `Process Data` button. This triggers the internal segmentation of the previously extracted and uploaded climate data, dividing it into sliding blocks according to the specified length and shift. Each window is then prepared for further statistical analysis in the subsequent steps.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Analysis Input ✓
Analysis Output
SZTAKI
Copernicus
↩ Back
Process Data
Funded by the European Union

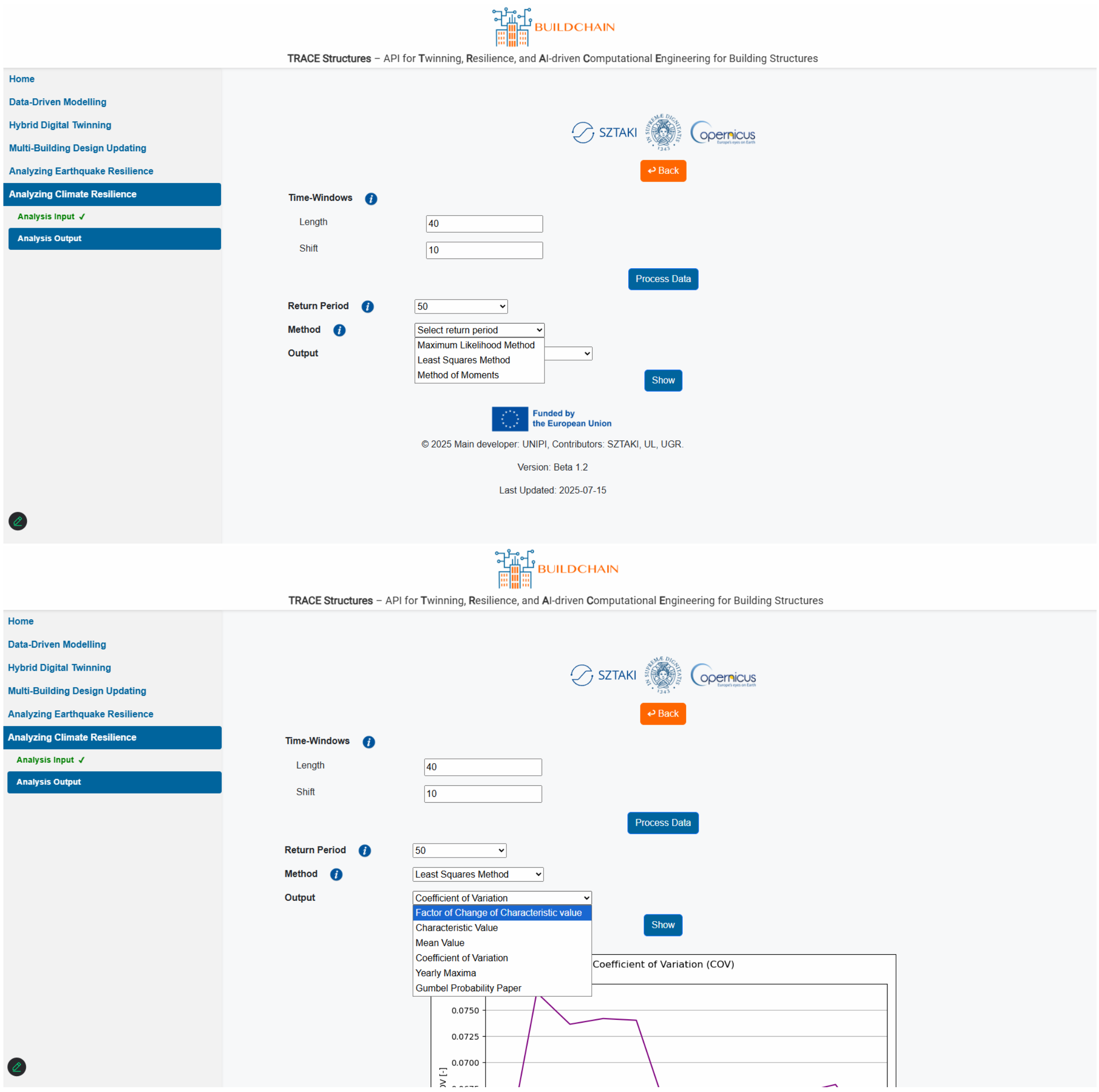
Time-Windows
Length: 40
Shift: 10

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.
Version: Beta 1.2
Last Updated: 2025-07-15

Figure 12.7: Step 5 of Module 5 – Time-window specification for analysis.

12.8.6 Step 6: Return Period, Statistical Method, and Output Type Selection

In this step, you configure the statistical parameters used in the extreme value analysis. Begin by specifying the desired **return period** (e.g., 50 years), which determines the recurrence interval for the climatic event of interest—such as extreme temperature or wind. This setting will be used in the computation of characteristic values that correspond to rare but significant events.


BUILDCHAIN
TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures
Home
Data-Driven Modelling
Hybrid Digital Twinning
Multi-Building Design Updating
Analyzing Earthquake Resilience
Analyzing Climate Resilience
Analysis Input ✓
Analysis Output
SZTAKI
opernicus
Back
Process Data
Show
Funded by the European Union
Maximum Likelihood Method
Least Squares Method
Method of Moments
Factor of Change of Characteristic value
Characteristic Value
Mean Value
Coefficient of Variation
Yearly Maxima
Gumbel Probability Paper
Coefficient of Variation (COV)
0.0750
0.0725
0.0700

Time-Windows
Length 40
Shift 10
Return Period 50
Method Select return period
Output

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Version: Beta 1.2

Last Updated: 2025-07-15

Time-Windows
Length 40
Shift 10
Return Period 50
Method Least Squares Method
Output Coefficient of Variation

Figure 12.8: Step 6 of Module 5 – Specification of return period, statistical method, and desired output.

Next, select the **statistical method** for parameter estimation. Available methods include:

- **Maximum Likelihood Method (MLM)** – Suitable for large datasets and provides efficient parameter estimates.
- **Least Squares Method (LSM)** – Fits the model to minimize the squared difference between observed and theoretical values.
- **Method of Moments (MOM)** – Matches sample moments (mean, variance) to the distribution's moments.

Finally, choose the **output metric** you wish to visualize. The module offers several analysis outputs:

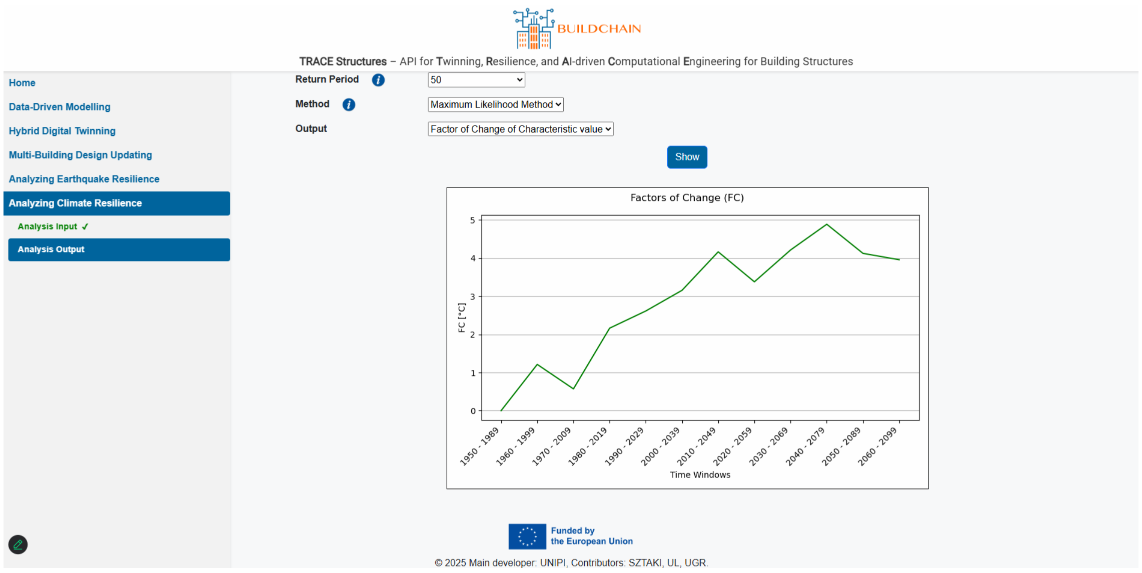
- **Factor of Change of Characteristic Value** – Shows how the characteristic value evolves across time windows.
- **Characteristic Value** – The value associated with the specified return period under the assumed distribution.

- **Mean Value and Coefficient of Variation (CV)** – Descriptive statistics of the extreme values in each time window.
- **Yearly Maxima** – Visualization of raw annual maximum values.
- **Gumbel Probability Paper Plot** – Diagnostic plot used to assess the fit of the Gumbel distribution to the data.

12.8.7 Step 7: Visualize Results

After all settings are configured, proceed to the visualization tab to explore the computed results. Based on the selected climate variable, return period, statistical method, and time-window configuration, the tool generates a plot showing the evolution of the selected metric over time (see Fig. 12.9).

These visualizations allow users to observe how extreme climatic conditions—such as maximum temperature or wind speed—are projected to change in the future under different emission scenarios. This step provides valuable insights for climate risk assessments and decision-making regarding structural resilience and adaptation planning.



TRACE Structures – API for Twinning, Resilience, and AI-driven Computational Engineering for Building Structures

Return Period: 50

Method: Maximum Likelihood Method

Output: Factor of Change of Characteristic value

© 2025 Main developer: UNIPI, Contributors: SZTAKI, UL, UGR.

Figure 12.9: Step 7 of Module 5 – Visualization of selected statistical output across time windows.

12.9 Analysis of Climate Effects for Buildings in Florence

Natural hazards have a significant impact on the built environment, as seen by recent extreme events. This module represents an initial tool designed to incorporate reliable climate projections of several climate variables over time into the decision-making processes concerning the built environment, especially for public authorities. The output of this tool coincides with the first step of a complete assessment of climate-risk conducted within the framework of Climate Vulnerability and Risk Assessment (CVRA) as suggested buy EU Guidelines on adaptation of buildings to climate change. Several steps are needed but the identification and evaluation of hazards that may affect building assets is surely the first step. Then, other steps are needed to reach a complete evaluation of the building. Intrinsic characteristic of the buildings and its materials will be taken into consideration to understand how they can react in front of the extreme events. This methodology will be

Time-Window	Characteristic Value	Factor of Change of Characteristic Value	Mean Value
1950–1989	40.1	0.0	33.5
1960–1999	41.3	1.2	34.0
1970–2009	40.6	0.6	34.3
1980–2019	42.2	2.2	34.6
1990–2029	42.7	2.6	34.8
2000–2039	43.2	3.2	35.2
2010–2049	44.2	4.2	35.5
2020–2059	43.4	3.4	35.4
2030–2069	44.3	4.2	35.8
2040–2079	45.0	4.9	36.1
2050–2089	44.2	4.1	36.0
2060–2099	44.0	4.0	36.4

Table 12.2: Characteristic Value, Factor of Change, and Mean Value for each time window

applied to several schools' buildings in the city center of Florence. Here, climatic hazard under two scenarios (RCP 4.5 and RCP 8.5) is evaluated separately for several climate phenomena, which are maximum and minimum temperatures, wind speed, and precipitation. Trends of these variables through time are defined in terms of factors of change. Almost all the area of the municipality of Florence is "inside" the same cell of the model grid, so that even all the school buildings. Some of the results in terms of characteristic values, factors of change and mean values for the municipality of Florence are shown in Table 12.2.

The goal is to support public authorities dealing with the management of large building stocks, as the Municipality of Florence, in planning interventions that enhance the climate resilience of existing buildings, thereby improving the management of financial resources, particularly when resources are limited.

12.10 Strengths and Limitations

The potential of this tool lies in providing a practical approach to understanding how the climate is changing in a specific location, serving as a starting point for exposure analysis in the CVRA for buildings across Europe. One of its main strengths is the direct connection to the Copernicus Climate Data Store, which enables working with high-quality data. Another strong point is that the results provided to the user follow engineering standards, such as the Extreme Value Theory, which aligns with the most common procedure used in Europe to calculate the climatic actions. The interface is simple and intuitive, allowing even non-experts to conduct climate analyses by selecting geographical coordinates and some parameters. One of the tool's greatest strengths is its scalability, thanks to its connection to the Copernicus Climate Data Store (CDS). By inserting the coordinates of a city or building, it is possible to perform analysis across all of Europe. However, it provides an assessment of exposure to climate hazards; it does not yet account for the vulnerability of a building to extreme events. Currently, the tool produces an analysis only for a single climate projection model, but further development will extend the possibility to evaluate multiple climate projection models simultaneously, as required by the literature. Lastly, a more holistic, multi-hazard approach would require users to conduct analyses to obtain indices that take into account not only the severity of extreme events but also their frequency.

Chapter 13

Troubleshooting and FAQs

13.1 Common Issues

1. **Long-running computations appear to freeze**
   Problem: Complex computations may seem stalled due to lack of a visible progress indicator.
   Resolution: We are working to improve progress feedback. Please do not refresh the page while a task is running.

2. **Unexpected timeouts for large datasets, upload fails silently**
   Problem: Large file uploads can trigger timeouts. No feedback is shown when an upload fails.
   Resolution: Ensure your file format is valid. Please, contact the support team if such a problem arises.

3. **Limited UI support for training configuration and monitoring**
   Problem: The UI offers basic training capabilities but does not support custom metrics, real-time loss tracking, or fine-grained control.
   Resolution: Use the TRACE-STRUCTURES Python library, which enables verbose training mode, training/validation loss monitoring, visualization of convergence, and other advanced setups.

4. **Advanced preprocessing is limited in the UI**
   Problem: The UI is designed for weather/modal or preprocessed data. It lacks features for handling missing values, merging complex datasets, or encoding categorical data.
   Resolution: The Python library allows complete preprocessing flexibility using pandas or built-in utilities. The UI's capabilities are being expanded to support more preprocessing options in future updates.

5. **UI behaves unexpectedly or freezes**
   Problem: Occasionally, the user interface may act strangely, such as buttons not responding, in case of server errors, views not updating, or output appearing inconsistent.
   Resolution: When in doubt, refresh the page. Reloading usually fixes temporary UI issues caused by cached states or incomplete loading.

6. **Inconsistent ordering between parameter samples and quantities of interest (QoIs)**
   Problem: When uploading pregenerated parameter sets and samples, the system only performs a simplified check for consistency with the parameter distributions.

However, there is currently no way to ensure that the uploaded quantities of interest (QoI) table is in the same row order as the parameter samples.
Why it matters: In parallel or distributed computation workflows, mismatched row ordering between parameter samples and QoIs may result in incorrect model evaluations or misinterpretation of results.
Resolution: Users must take care to verify that QoI rows are correctly aligned with the parameter samples, especially when merging outputs from parallel runs.
Future plan: We plan to introduce a hashcode column for parameter samples, which will allow automatic consistency checks between samples and QoIs.

13.2 Frequently Asked Questions (FAQ), Error Messages and Solutions

1. **How can I know if a computation is still running?**
   A visible indicator (e.g., spinner) may appear during processing. We are working on improving visibility for long tasks. Please avoid refreshing the page.

2. **What file formats are supported?**
   Supported formats include JSON, XLSX, CSV, SM, DDPM and UNV files that adhere to our schema. For more details on each file type and their expected structure, refer to the documentation section **"Data Flow, Data Formats"** under `modulechapter#.5.2`.

3. **What is the API token in Module 5 and how do I get it?**
   To request climate data from the Copernicus Climate Data Store in **Module 5**, you must first provide an **API token**. This token ensures that the data is accessed according to the Copernicus usage policies and licensing terms — it links your requests to your registered account.

   (a) Click on `Analysis input` in the left navigation menu if it is not already selected.

   (b) If you do not have an API token, click the `Get API Token` button. This will take you to the Copernicus website, where you can log in or register.

   (c) After registration and login, go to `Your Profile` on the Copernicus website, copy your API token, and paste it into the corresponding input field in the UI.

4. **How long does it take to download data through Copernicus?**
   Usually, it takes approximately 3-5 minutes, but the duration can vary based on several factors such as the size of the dataset, your internet connection speed, and current server load.

5. **Is there a size limit for uploaded files?**
   Yes, the UI limit is typically around 400MB. However, using the Python library, it is also possible to work with larger files.

6. **How can I report a bug or get support?**
   If you encounter a bug or need assistance, please send an email to `friedrron@gmail.com`. To help us diagnose the issue quickly, include the exact error message, a brief description of the step where the problem occurred, and details about any imported data used.

13.3 Support Contact

For questions or support regarding the TRACE-STRUCTURES modules, please refer to the contact details below:

Primary Support Contact

- Áron Friedman — `friedrron@gmail.com`

Secondary Support Contacts

- Bence Popovics — `popbence@gmail.com`
- Emese Vastag — `vastag.emese@sztaki.hun-ren.hu`
- Noémi Friedman — `n.friedman@ilab.sztaki.hu`

Affiliation

AI Laboratory,
HUN-REN Institute for Computer Science and Control (SZTAKI)
`https://sztaki.hun-ren.hu/`

With module-specific theoretical questions, please contact

- **Module 1 (Data-Driven Modeling):** Emese Vastag — `vastag.emese@sztaki.hun-ren.hu`
- **Module 2 (Hybrid Digital Twinning):** Bence Popovics — `popbence@gmail.com`
- **Module 3 (Multi-Model Design Updating):** Noémi Friedman — `n.friedman@ilab.sztaki.hu`
- **Module 4 (Seismic Resilience):** Filippo Landi — `filippo.landi@ing.unipi.it`
- **Module 5 (Climate Exposure):** Costanza Guarducci — `costanza.guarducci@unifi.it` and Giulia Pierotti — `giulia.pierotti@phd.unipi.it`

Chapter 14

Updates and Maintenance

14.1 Version History

The following table outlines the major versions of the system, including release dates and notable changes or improvements.

Version	Release Date	Notes
Beta 1.0	April 2025	Initial release with core functionality.
Beta 1.1	June 2025	Improved performance, minor bug fixes added Module 5.
Beta 1.2	July 2025	Import from and export to DBL using the BUILDCHAIN API, minor bug fixes, making Module 5 more efficient, adding exploratory data analysis.
Beta 1.3	**July 2025**	Improved BUILDCHAIN API interoperability, minor bug fixes, adding background image.

Table 14.1: System Version History

14.2 Planned Features or Fixes

Future development aims to expand functionality and improve stability based on user feedback.

- Creating python packages from the three library packages for easier use
- Adding climate effect estimations using an ensemble of climate models;
- Extend Exploratory Data Analysis features in Modules 1 and 2;
- Adding the possibility to streamline wall property table for Earthquake analysis from BIM imported from the BUILDCHAIN DBL;
- Possibility to filter input features/parameters and quantity of interests before generating the predictive model in Modules 4 and 5;
- Better explanations of necessary steps on the user interface;
- Bug fix: export crash when large datasets are used (Planned patch 1.2.1).

Contributions and feature requests are welcomed via the official repository or support email.

Chapter 15

Licensing and Credits, Potential for Replication or Scaling

15.1 Licensing and Credits

The library packages that the TRACE-STRUCTURES user interface relies on was developed in alignment with FAIR principles (Findable, Accessible, Interoperable, Reusable), supporting open data workflows and extensibility for future modules.

The library packages are released under the **GNU General Public License**, allowing reuse with attribution and minimal restriction.

The API and the user interface are openly accessible for all interested users.

Credits:

- Lead Developer: Noémi Friedman
- Contributors: (see a more detailed explanation of contributions at the beginning of this document) Bence Popovics, Filippo Landi, Áron Friedman, Emese Vastag, András Urbanics, Constanza Guarducci, Giulia Pierotti, Giada Bartolini, Blaz Kurent, Bostjan Brank, Juan Chiachio Ruano, Emilio Daroch, Uros Bohinc, Stevan Nesovic
- Funding: by the European Commission within the HORIZON project BUILDCHAIN PROJECT - BUILDing knowledge book in the blockCHAIN distributed ledger. Trustworthy building life-cycle knowledge graph for sustainability and energy efficiency (https://buildchain-project.eu/, Grant agreement ID: 101092052)

15.2 Potential for Replication or Scaling

The Pythong library packages have been designed with modularity and scalability in mind. It can be replicated or extended in the following ways:

- **Deployment in different environments:** Compatible with both cloud and on-premise infrastructure.
- **Customization potential:** Open-source modules allow for feature expansion and integration with third-party systems.

Organizations interested in adapting or extending the system are encouraged to contact the development team or refer to the technical documentation repository.

Chapter 16

Security, Privacy, and Performance Considerations

16.1 Security Measures

The server implements multiple security-related headers and initialization parameters to mitigate attacks such as cross-site scripting (XSS), clickjacking, MIME sniffing, and man-in-the-middle (MITM) attacks. These safeguards are mostly implemented via response headers using Tornado's secure handling functionality.

16.1.1 Security HTTP Headers

The following response headers are set to improve browser security:

- **X-Frame-Options:** `DENY` — Prevents clickjacking by blocking the page from being embedded in a frame.
- **X-XSS-Protection:** `1; mode=block` — Enables browser XSS filtering mode.
- **X-Content-Type-Options:** `nosniff` — Prevents content-type sniffing.
- **Referrer-Policy:** `strict-origin-when-cross-origin` — Provides fine-grained control over the amount of referrer information sent with requests.
- **Strict-Transport-Security:** Enforces secure (HTTPS) connections to the server.
- **Content-Security-Policy:** Restricts resource loading to trusted sources, significantly reducing the risk of content injection attacks.

16.1.2 Secure Cookies

To ensure the confidentiality and integrity of user sessions, the application uses secure HTTP-only cookies:

- A random `UUID4` session ID is generated and stored in a secure, HTTP-only, `SameSite=Lax`, and `Secure` cookie.
- Tornado's `set_secure_cookie` function is used in combination with the `cookie_secret`, which is stored in an environment variable.

16.2 Privacy Considerations

The application enforces privacy by adhering to the following principles:

- No personally identifiable information (PII) is stored or logged.
- Session identification is based solely on randomly generated UUIDs.
- Cookie attributes are configured to prevent access from client-side JavaScript.

16.3 Logging and Audit Trails

Logging is stored in a dedicated logs directory. The handler ensures that:

- Log file size is limited to 10MB.
- A maximum of 5 log files are retained.
- All standard Tornado loggers (access, app, and gen) are integrated.

All server activity is appropriately logged without revealing sensitive request or session metadata.

16.4 Performance Considerations

To ensure high performance and scalability, the server employs the following strategies:

- **Asynchronous Socket.IO Server:** The application makes use of `socketio.AsyncServer` configured for Tornado's non-blocking I/O engine.
- **Template and Static File Caching:** Static and template paths are defined to streamline I/O access.
- **Efficient Notebook Browsing:** The `DocsHandler` dynamically loads available notebook files and generates HTML pages with minimal overhead.
- **Large File Transfers Using Chunking:** To support the transmission of large files over WebSocket connections, the server implements an efficient chunked transfer mechanism. Rather than sending the entire file in a single payload—which can exceed memory or transport limits—files are sent in smaller, sequential chunks and reassembled on the server.

Chapter 17

Conclusions, Outlook

The development of the **BUILDCHAIN** platform represents a significant advancement in leveraging digital twin technology for transparent, data-informed management of building structures. Unlike purely data-driven approaches, the platform ensures that engineering knowledge and underlying physical principles are preserved and integrated alongside observational data. By combining open-source Python library packages with a user-friendly, web-based interface, the TRACE-Structures toolset caters to both technically proficient users and non-programming stakeholders. It enables structural analysis, real-time monitoring, and resilience assessments through hybrid simulation techniques that blend machine learning with physics-based modeling.

At the core of this toolset is its deep integration with the **Digital Building Logbook (DBL)** system, which enables seamless two-way communication with trusted sources of building data. Sensor measurements, simulation models, and metadata can be ingested into the system, processed using advanced computational pipelines, and translated into new digital knowledge assets—such as calibrated digital twins, performance indicators, or early-warning alerts—that can then be pushed back to the DBL. This closed-loop data cycle supports ongoing enrichment of building information and ensures long-term relevance across a building's lifecycle.

The platform has been successfully demonstrated through multiple pilot studies and use cases, including real-life applications involving CLT (cross-laminated timber) buildings. Through these pilots, advanced methods such as surrogate modeling, Bayesian updating, sensitivity analysis, and multi-building parameter inference were implemented to bridge the gap between as-designed and as-built performance.

Looking ahead, future development will focus on enhancing automation, expanding modularity, and increasing interoperability with additional data standards and simulation tools. The BUILDCHAIN platform thus lays the groundwork for a broader ecosystem of AI- and simulation-driven building lifecycle management, advancing sustainability, resilience, and transparency in the built environment.

17.1 Lessons Learned

Several key lessons emerged during the development and integration of the TRACE-Structures library packages and the associated web-based API and UI:

- **Balancing usability and flexibility:** Providing both a Python-based interface for advanced users and a graphical UI for non-programmers helped bridge diverse user needs, but required careful architectural planning to ensure consistent functionality across both layers.

- **Data and model quality and availability are bottlenecks:** Successful implementation of digital twin workflows relies heavily on high-quality sensor data and simulation models. Integration with the DBL helped mitigate this by offering the usage of traceable and trustworthy data source, but inconsistencies in field data still may pose practical challenges.

- **Surrogate modeling is a powerful enabler:** For computationally intensive simulations, surrogate models—such as generalized polynomial chaos or machine learning-based emulators—proved invaluable for real-time calibration and inference without much compromising accuracy.

- **Multi-building inference is promising but complex:** Extending digital twinning from individual buildings to portfolios introduces opportunities for generalization and learning across similar structures. However, it also increases the complexity and dimension of the problem.

- **Close alignment with DBL architecture adds long-term value:** Ensuring compatibility with the DBL ecosystem not only simplifies data management but also supports regulatory compliance, enhances transparency, and promotes future extensibility.

- **Data format fragmentation and lack of standardization remain key challenges in full automation:** The use of multiple formats across tools and platforms complicates integration, especially when working with enriched IFC files. Streamlining these for digital twin workflows is not yet trivial and often requires manual preprocessing or tool-specific customization.

Continued collaboration across disciplines—combining structural engineering, data science, and software development—remains key to realizing the full potential of digital twin technology in the built environment.

Chapter 18

Glossary

API (Application Programming Interface) A set of rules and protocols that allow different software components to communicate. In BUILDCHAIN, APIs connect the TRACE-Structures libraries to the web-based interface and to external systems such as the Digital Building Logbook (DBL).

BIM (Building Information Model) A digital representation of the physical and functional characteristics of a building, used for planning, design, construction, and operation.

Bayesian Updating A statistical inference method used to update uncertain model parameters based on observed data, enhancing model fidelity through calibration.

Composite Metric (summed_metric) A custom scalar score combining Kendall's Tau, Pearson r, and Spearman's ρ, penalized by relative RMSE. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

CLT (Cross-Laminated Timber) An engineered wood product composed of layers of lumber boards glued perpendicularly, widely used in tall timber construction.

DBL (Digital Building Logbook) A structured repository that stores verified building data—such as sensor measurements, BIM/FE models, and metadata—and facilitates bidirectional data exchange with digital twin platforms.

Digital Twin A dynamic virtual representation of a physical building system that integrates real-time data, simulation models, and historical records for monitoring, analysis, and decision support.

EC Eurocode. A set of European standards (EN 1990–EN 1999) that provide a common approach to structural design across EU member states. The Eurocodes cover structural safety, serviceability, durability, and fire resistance for different materials and types of structures.

EC8 Part of the Eurocode series, EC8 (EN 1998) deals specifically with the design of structures for earthquake resistance. It outlines rules and methodologies for assessing seismic actions, ensuring structural ductility, and safeguarding building occupants during seismic events.

FE Model (Finite Element Model) A discretized computational model used to simulate structural behavior under loads, often used as the baseline for digital twin development.

GPC (Generalized Polynomial Chaos) A spectral method used for uncertainty quantification and surrogate modeling, enabling efficient approximation of complex model responses.

Hybrid Modeling An approach that combines data-driven models (e.g., ML) with physics-based simulations to improve accuracy and generalizability.

IFC (Industry Foundation Classes) An open and standardized file format used for sharing building and construction data across different software platforms.

Kendall's Tau (τ) Measures ordinal association by comparing concordant and discordant value pairs.. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

Mean Absolute Error (MAE) Computes the average of absolute differences between predictions and true values. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

Mean Squared Error (MSE) Measures the average of the squared differences between predicted values and actual target values. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

ML (Machine Learning) A subset of artificial intelligence focused on building models that learn patterns from data. In TRACE, ML models are used for surrogate modeling, classification, and attribution analysis.

Module A logical and functional unit within the TRACE-Structures library, designed to implement a specific capability—such as data-driven modeling (Module 1), hybrid updating (Module 2), or resilience assessment (Modules 4–5).

Normalized MAE (NMAE) MAE normalized by the range of the target variable. . See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

Normalized RMSE (NRMSE) RMSE normalized by the range of the target variable. . See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

PCE (Polynomial Chaos Expansion) A mathematical framework for constructing surrogate models and performing sensitivity and uncertainty analyses, often used in combination with Bayesian inference.

Pearson Correlation Coefficient (r) Measures the linear correlation between predicted and actual values.

Relative RMSE (rel_RMSE) RMSE scaled by the standard deviation of the training labels.

RMSE (Root Mean Square Error) A standard measure of prediction error that quantifies the average magnitude of residuals between observed and predicted values. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

R^2 (Coefficient of Determination) A statistical measure that indicates how well the model predictions match observed data. An R^2 value closer to 1 implies a better fit. See the *Evaluation of the trained surrogate model* subsection in Section 8.4.2.

SHAP (SHapley Additive exPlanations) A method for interpreting machine learning models by assigning feature-level importance scores, based on cooperative game theory.

SHM (Structural Health Monitoring) The practice of instrumenting structures with sensors to monitor performance over time and detect potential damage or degradation.

Surrogate Model A computationally efficient approximation of a high-fidelity simulation (e.g., an FE model), used to support tasks like optimization, sensitivity analysis, and Bayesian updating.

TRACE API A modular application programming interface developed within the BUILDCHAIN project that exposes core functionality of the digital-twinning, climate-resilience and eathquake-resilience library packages for external use via a web-based UI.

UQ (Uncertainty Quantification) A methodological framework for quantifying uncertainty in model inputs and outputs, often used in risk-informed decision-making.

UI (User Interface) The web-based graphical front end through which users interact with the TRACE platform. It enables access to functionality such as model configuration, data upload, simulation control, and results visualization.

Bibliography

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 1st edition, 2001.

[2] Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[3] Dongbin Xiu and George E. Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, 2002.

[4] Noémi Friedman, Claudia Zoccarato, Elmar Zander, and Hermann G. Matthies. A worked-out example of surrogate-based bayesian parameter and field identification methods. In Juan Chiachio-Ruano, Manuel Chiachio-Ruano, and Shankar Sankararaman, editors, *Bayesian Inverse Problems: Fundamentals and Engineering Applications*, pages 155–203. CRC Press, Boca Raton, FL, 2021.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[6] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774, 2017.

[7] Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145(2):280–297, 2002.

[8] Jon Herman and Will Usher. Salib: An open‐source python library for sensitivity analysis. *Journal of Open Source Software*, 2(9):97, 2017.

[9] Bruno Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering & System Safety*, 93(7):964–979, 2008.

[10] Matej Šodan, András Urbanics, Noémi Friedman, Andjelka Stanić, and Mijo Nikolić. Comparison of machine learning and gpc-based proxy solutions for an efficient bayesian identification of fracture parameters. *Computer Methods in Applied Mechanics and Engineering*, 436:117686, 2025. Published March 2025.

[11] Noémi Friedman, Blaž Kurent, Bence Popovics, Stevan Nesovic, Branimir Rakic, Gregor Karlovsek, Ema Lovsin, Nenad Milosevic, Igor Osmokrovic, Katja Malovrh Rebec, Rok Rudolf, and Boštjan Brank. Digital twinning for enhancing transparency in sustainability, safety, health and serviceability of tall timber structures. In *Proceedings of the 14th World Conference on Timber Engineering (WCTE 2025)*, pages 2260–2269, Brisbane, Australia, June 2025. World Conference on Timber Engineering.

[12] Blaž Kurent, Noemi Friedman, Angelo Aloisio, Dag Pasca, Roberto Tomasi, and Boštjan Brank. Bayesian model updating of eight-storey clt building using modal data. *Probabilistic Engineering Mechanics*, 77:103642, 2024.

[13] Blaž Kurent, Noemi Friedman, Angelo Aloisio, Dag Pasca, Roberto Tomasi, and Boštjan Brank. Bayesian model updating of eight-storey clt building using modal data. *Probabilistic Engineering Mechanics*, 77:103642, 2024.

[14] Pietro Croce, Filippo Landi, and Paolo Formichi. Probabilistic seismic assessment of existing masonry buildings. *Buildings*, 9(12), 2019.

[15] Peter Fajfar. A nonlinear analysis method for performance-based seismic design. *Earthquake spectra*, 16(3):573–592, 2000.

[16] Edoardo Cosenza, Ciro Del Vecchio, Marco Di Ludovico, Mauro Dolce, Claudio Moroni, Andrea Prota, and Emanuele Renzi. The italian guidelines for seismic risk classification of constructions: technical principles and validation. *Bulletin of Earthquake Engineering*, 16:5905–5935, 2018.

[17] Conall Heussaff, Johannes Emmerling, Gunnar G Luderer, Robert C Pietzcker, Severin Reissl, Renato Rodrigues, and Rupert Way. Europe's 2040 climate target: Four critical risks and how to manage them. Technical report, Bruegel Policy Brief, 2024.

[18] Giovanni Forzieri, Luc Feyen, Simone Russo, Michalis Vousdoukas, Lorenzo Alfieri, Stephen Outten, Mirco Migliavacca, Alessandra Bianchi, Rodrigo Rojas, and Alba Cid. Multi-hazard assessment in europe under climate change. *Climatic Change*, 137(1):105–119, 2016.

[19] S Binz, M Brombacher, and V Propach. On the development of a standardised method for climate vulnerability and risk assessment (cvra) for buildings and properties. In *IOP Conference Series: Earth and Environmental Science*, volume 1363, page 012110. IOP Publishing, 2024.

[20] Linda Toledo and European Commission Directorate-General for Climate Action. Eu-level technical guidance on adapting buildings to climate change. Technical report, Publications Office of the European Union, 2023. Technical guidance report, March 2023.

[21] Carlo Buontempo, Samantha N Burgess, Dick Dee, Bernard Pinty, Jean-Noël Thépaut, Michel Rixen, Samuel Almond, David Armstrong, Anca Brookshaw, Angel Lopez Alos, et al. The copernicus climate change service: climate science in action. *Bulletin of the American Meteorological Society*, 103(12):E2669–E2687, 2022.

[22] Daniela Jacob, Juliane Petersen, Bastian Eggert, Antoinette Alias, Ole Bøssing Christensen, Laurens M Bouwer, Alain Braun, Augustin Colette, Michel Déqué, Goran Georgievski, et al. Euro-cordex: new high-resolution climate change projections for european impact research. *Regional environmental change*, 14(2):563–578, 2014.

[23] Douglas Maraun. Bias correcting climate change simulations-a critical review. *Current Climate Change Reports*, 2(4):211–220, 2016.

[24] Copernicus Climate Change Service. CORDEX regional climate model data on single levels. https://doi.org/10.24381/cds.bc91edc3, 2019.

[25] Pietro Croce, Paolo Formichi, and Filippo Landi. Climate change: Impacts on climatic actions and structural reliability. *Applied Sciences*, 9(24):5416, 2019.

Appendix A

Data Format Manual

A.1 Module 1-3

The following excerpt shows a portion of a UNV (Universal File Format) file containing various datasets. This example includes three dataset types:

- **Type 164** – Defines the unit system (e.g., `mm/newton`).
- **Type 15** – Contains nodal coordinate data (node IDs and their spatial positions).
- **Type 55** – Represents complex dataset results, such as mode shapes from a modal analysis.

Below the raw UNV snippet, the parsed output of the `read_sets()` Python function is shown. This function reads and converts the structured UNV data into a dictionary-based Python format, making it easy to process the contents programmatically. Each dictionary corresponds to a dataset block in the UNV file, with key-value pairs describing its type and contents.

```
  -1
 164
    1 mm/newton 1
 1.0000000000000000D+00 1.0000000000000000D+00 1.0000000000000000D+00
 1.0000000000000000D+00
  -1
  -1
  15
    1 0 0 0 1.00000E+00 0.00000E+00 0.00000E+00
    2 0 0 0 2.00000E+00 0.00000E+00 0.00000E+00
    3 0 0 0 3.00000E+00 0.00000E+00 0.00000E+00
    4 0 0 0 4.00000E+00 0.00000E+00 0.00000E+00
    5 0 0 0 5.00000E+00 0.00000E+00 0.00000E+00
    6 0 0 0 6.00000E+00 0.00000E+00 0.00000E+00
  -1
  -1
  55
OMA id: 1
Mode id: 1
Time from 2025-02-04 14:23:48 to 2025-02-04 14:38:48
Xi: 0.032889582965781865
None
    1 2 2 8 5 3
    2 4 1 1
 3.86939e+00 0.00000e+00 0.00000e+00 0.00000e+00
    1
 5.37656e-01 -1.62873e-02 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
       2
 8.22407e-02 -8.87557e-02 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
       3
 4.52694e-01 7.94008e-03 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
       4
 1.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
       5
 1.32852e-01 -2.28746e-01 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
       6
 9.72776e-01 3.46153e-03 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
   -1
```

Listing A.1: Excerpt from a UNV file

```
{'type': 164,
 'units_code': 1,
 'units_description': 'mm/newton',
 'temp_mode': 1,
 'length': 1.0,
 'force': 1.0,
 'temp': 1.0,
 'temp_offset': 1.0}
```

Listing A.2: Parsed result of UNV type 164 (Units)

```
{'type': 15,
 'node_nums': [1.0, 2.0],
 'def_cs': [0.0, 0.0],
 'disp_cs': [0.0, 0.0],
 'color': [0.0, 0.0],
 'x': [9124.0, 4529.0],
 'y': [13576.0, 12295.0],
 'z': [17721.0, 17721.0]}
```

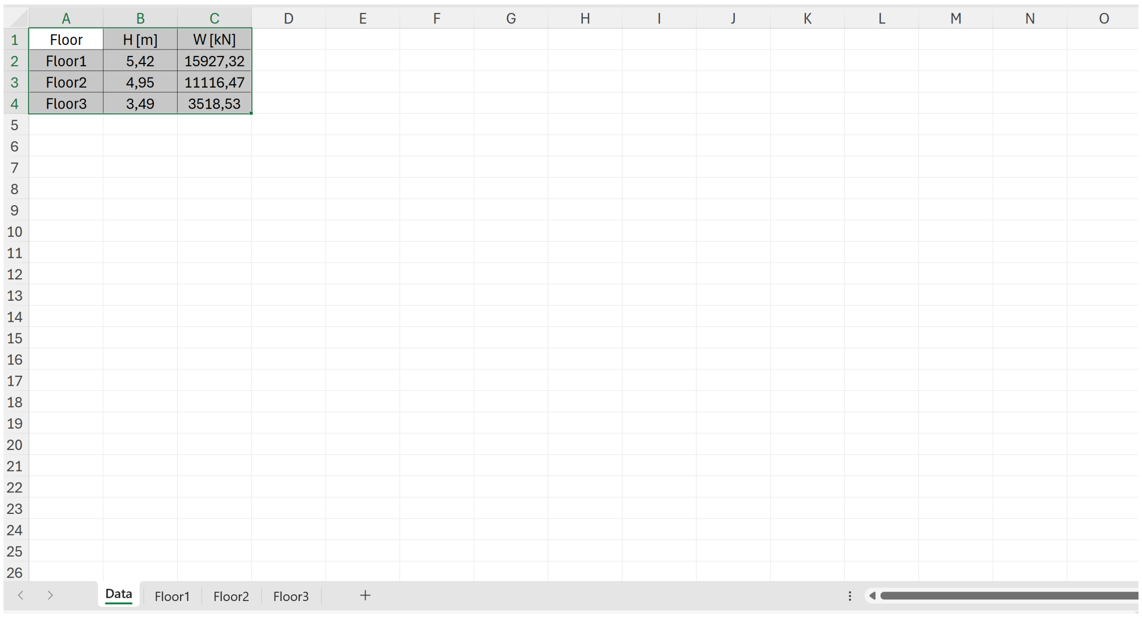
Listing A.3: Parsed result of UNV type 15 (Node coordinates)

```
{'type': 55,
 'id1': 'OMA id: 1',
 'id2': 'Mode id: 1',
 'id3': 'Time from 2025-02-04 14:23:48 to 2025-02-04 14:38:48',
 'id4': 'Xi: 0.032889582965781865',
 'id5': 'None',
 'model_type': 1,
 'analysis_type': 2,
 'data_ch': 2,
 'spec_data_type': 8,
 'data_type': 5,
 'n_data_per_node': 3,
 'load_case': 1,
 'mode_n': 1,
 'freq': 3.86939,
 'modal_m': 0.0,
 'modal_damp_vis': 0.0,
 'modal_damp_his': 0.0,
 'node_nums': array([1, 2, 3, 4, 5, 6]),
 'r1': array([0.537656 -0.0162873j , 0.0822407-0.0887557j ,
       0.452694 +0.00794008j, 1. +0.j ,
       0.132852 -0.228746j , 0.972776 +0.00346153j]),
 'r2': array([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j]),
 'r3': array([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j])}
```

Listing A.4: Parsed result of UNV type 55 (Complex mode shape)

A.2 Module 4

The wall properties for the Earthquake Resilience Modulus are retrieved from an excel Table (`.xls` or `.xlsx` file) uploaded from the local computer on the web API. The excel Table should be prepared according to the format shown in Figure A.1 for a three-story building. The first datasheet, named ”Data”, contains a table with general building data: floor heights, H, in [m] and total floor weights, W, in the equivalent Multi-Degree Of Freedom (MDOF) system of the building.



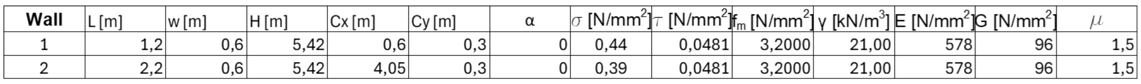
Floor	H [m]	W [kN]
Floor1	5,42	15927,32
Floor2	4,95	11116,47
Floor3	3,49	3518,53

Figure A.1: Example of excel format for a three-story building.

Then, there is one datasheet for each floor i, named ”Floor$i$”, which collects all relevant wall properties as shown in Figure A.2, where each column corresponds to the following property of the walls:

- L is the wall length in [m]
- w is the wall width in [m]
- H is the wall height in [m]
- Cx is x-coordinate of the center of gravity of the wall in [m]
- Cy is y-coordinate of the center of gravity of the wall in [m]
- α is the rotation angle around the x-axis in [degree]
- σ is the compressive stress on the wall induced by the quasi-permanent load combination as defined in EN1990, in [N/mm^2]
- τ is the masonry shear strength in [N/mm^2]

- f_m is the masonry compressive strength in [N/mm^2]
- γ is the masonry unit weight in [kN/m^3]
- E is the masonry elastic modulus in [N/mm^2]
- G is the masonry shear modulus in [N/mm^2]
- μ is the ductility factor adopted in pushover analysis for ductility check verification mode



Wall	L [m]	w [m]	H [m]	Cx [m]	Cy [m]	α	σ [N/mm^2]	τ [N/mm^2]	f_m [N/mm^2]	γ [kN/m^3]	E [N/mm^2]	G [N/mm^2]	μ
1	1,2	0,6	5,42	0,6	0,3	0	0,44	0,0481	3,2000	21,00	578	96	1,5
2	2,2	0,6	5,42	4,05	0,3	0	0,39	0,0481	3,2000	21,00	578	96	1,5

Figure A.2: Example of excel table for wall properties.

A.3 Module 5

A summary of the available options for each input and output parameter is provided in this appendix. They are selected through the user interface and allow to carry out the same analysis but considering the variation of some input.
The first input parameter is the API Token, obtained after the registration on Copernicus CDS website. Then all the other parameters must be entered.

- **Coordinates**: enter the values of latitude and longitude of the building in decimal degrees
  Latitude: 43.7697 Longitude: 11.2558

- **Experiment**: historical and RCPs scenarios are combined to obtain data series from 1950 to 2100
  Historical + RCP 4.5
  Historical + RCP 8.5

- **Climate Variables**: select just one among those proposed
  Maximum temperature [°C]
  Minimum temperature [°C]
  Precipitation [mm]
  Wind Speed [ms-1]

- **Time-Windows**: define the length of the period of time in which the climate conditions are stationary
  Length: 40
  Shift: 10

- **Return Period**: select a value of return period of actions expressed in years
  10
  50

100
200

- **Method**: select one method or the estimation of parameters of the Gumbel distribution
  Maximum Likelihood Method
  Least Square Method
  Method of Moments

- **Output**: the graphical format of the results will be displayed
  Factor of Change of Characteristic value
  Characteristic value
  Mean value
  Coefficient of Variation
  Yearly Maxima
  Gumbel Probability Paper

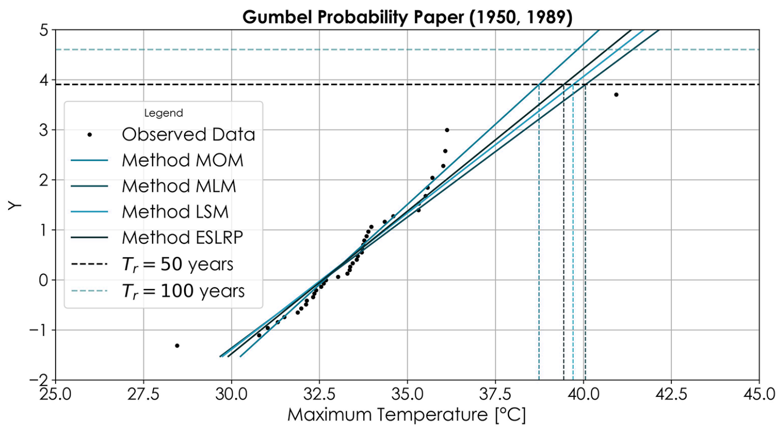


Figure A.3: Example of Gumbel Probability Paper Output

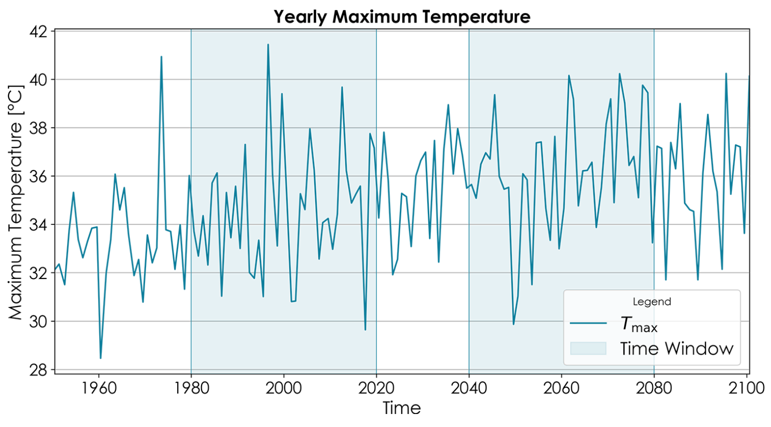


Figure A.4: Example of Yearly Maxima Output

Appendix B

Dependencies, Classes and Methods of the Tools

B.1 Dependencies

B.1.1 Dependencies of the digital-twinning library package

External Python modules used in the package:

`abc`, `asyncio`, `catboost`, `copy`, `emcee`, `IPython`, `lightgbm`, `matplotlib`, `numpy`, `os`, `pandas`, `pickle`, `plotly`, `pyuff`, `re`, `SALib`, `scipy`, `seaborn`, `shap`, `sklearn`, `sys`, `time`, `torch`, `warnings`, `xgboost`

B.1.2 Dependencies of the earthquake-resilience library package

External Python modules used in the package:

`copy`, `folium`, `json`, `math`, `matplotlib`, `numpy`, `os`, `pandas`, `reverse_geocoder`, `scipy`

B.1.3 Dependencies of the climate-resilience library package

External Python modules used in the package:

`cdsapi`, `cftime`, `earthkit`, `folium`, `IPython`, `json`, `math`, `matplotlib`, `numpy`, `os`, `pandas reverse_geocoder`, `scipy`, `time`, `xarray==0.21.0`

B.2 Classes and Methods of the Digital Twinning Library Package

Figure B.1 provides an overview of the main classes and methods implemented in the `digital-twinning` library package, presented as a UML class diagram. This package underpins the functionality of three core modules: ***Data-Driven Modeling***, ***Hybrid Digital Twinning***, and ***Multi-Building Updating***. Together, these components form the foundational backend of the system accessed through the first three modules of the user interface. A more detailed description of the individual classes is presented in the UML diagrams shown in Figures B.2–B.12.

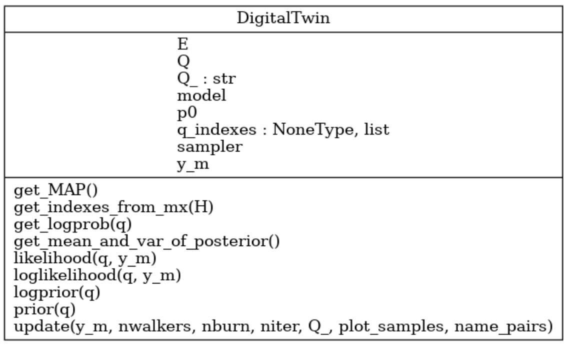


Figure B.2: UML diagram of the DigitalTwin class

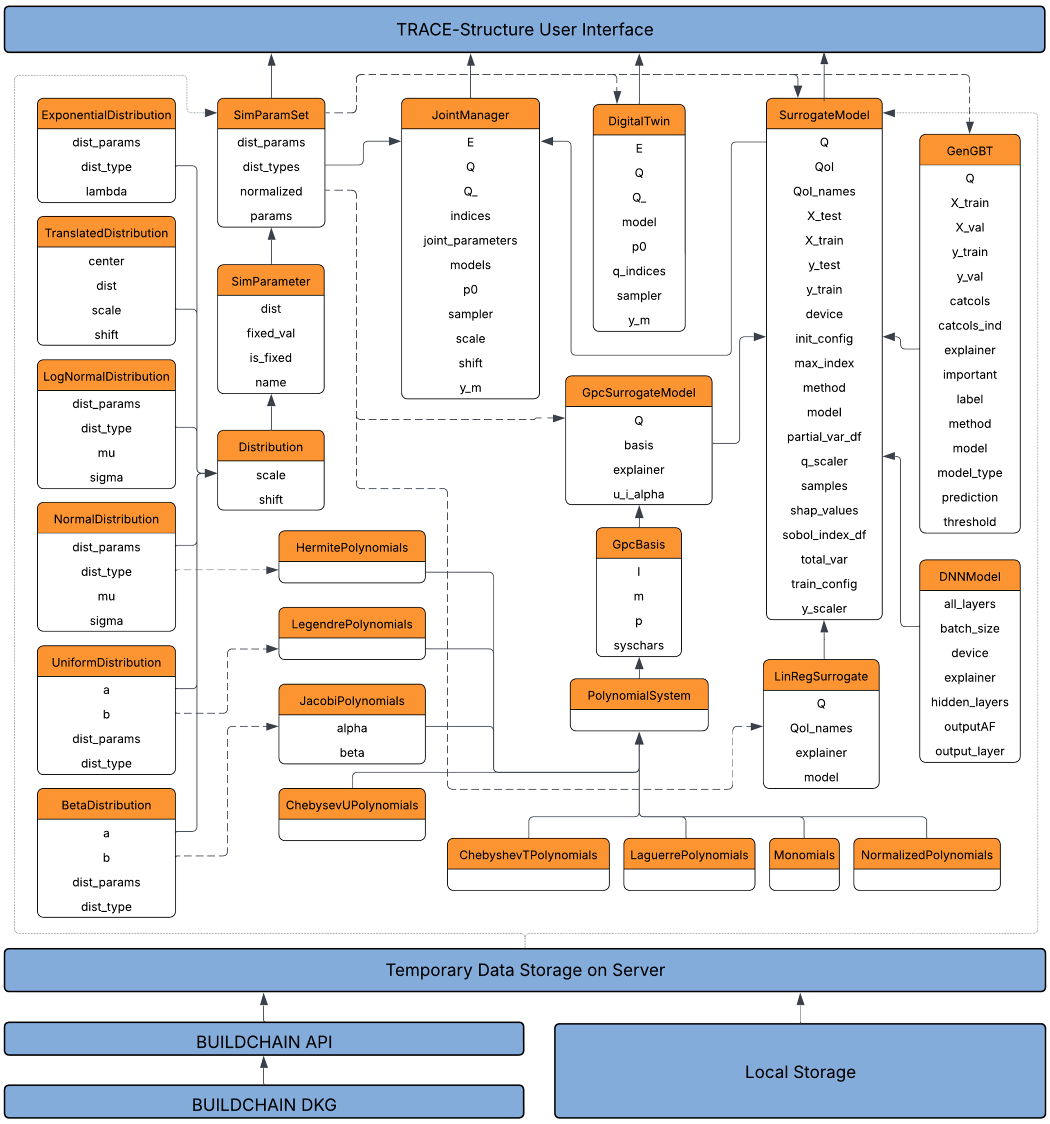


Figure B.1: Architecture of the library package

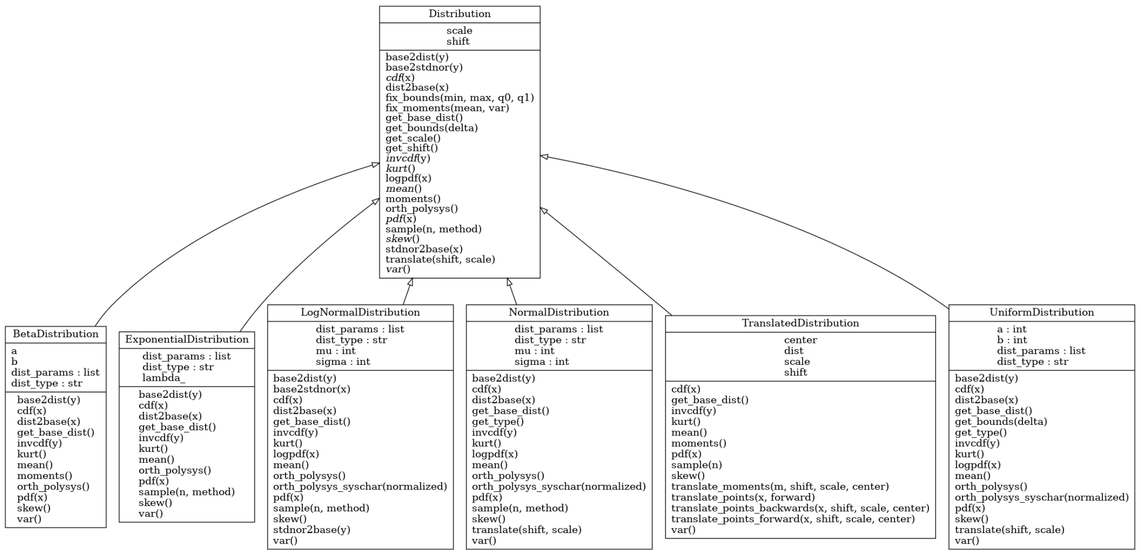


Figure B.3: UML diagram of the Distribution classes

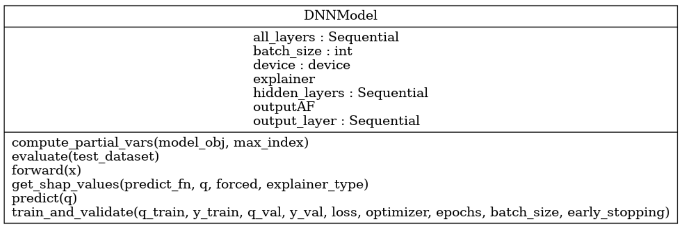

DNNModel
all_layers : Sequential
batch_size : int
device : device
explainer
hidden_layers : Sequential
outputAF
output_layer : Sequential
compute_partial_vars(model_obj, max_index)
evaluate(test_dataset)
forward(x)
get_shap_values(predict_fn, q, forced, explainer_type)
predict(q)
train_and_validate(q_train, y_train, q_val, y_val, loss, optimizer, epochs, batch_size, early_stopping)

Figure B.4: UML diagram of the DNN Surrogate Model

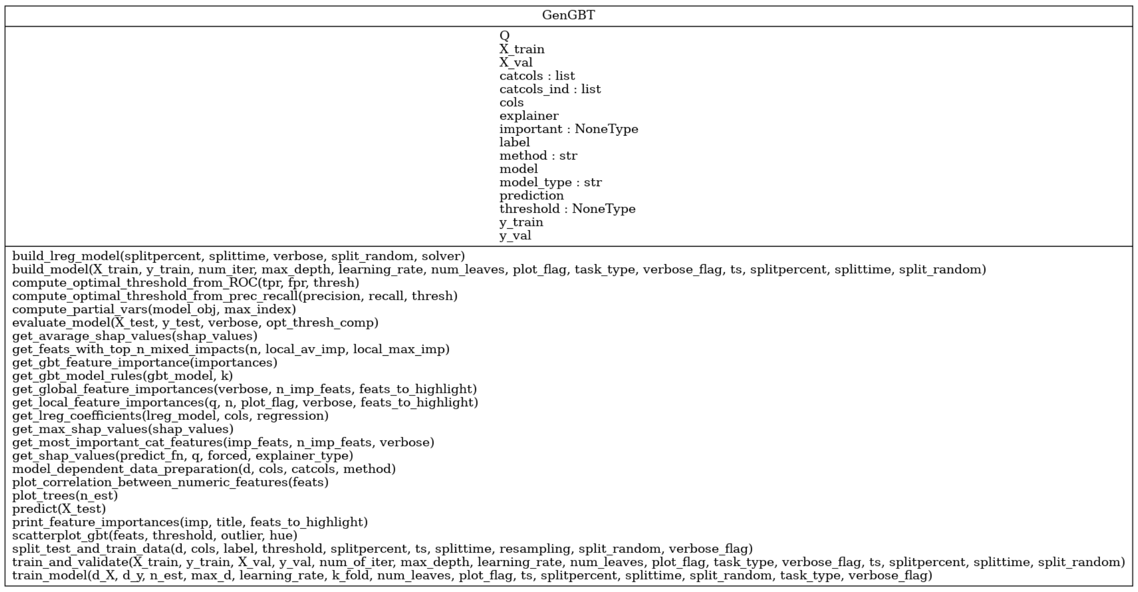

GenGBT
Q
X_train
X_val
catcols : list
catcols_ind : list
cols
explainer
important : NoneType
label
method : str
model
model_type : str
prediction
threshold : NoneType
y_train
y_val
build_lreg_model(splitpercent, splittime, verbose, split_random, solver)
build_model(X_train, y_train, num_iter, max_depth, learning_rate, num_leaves, plot_flag, task_type, verbose_flag, ts, splitpercent, splittime, split_random)
compute_optimal_threshold_from_ROC(tpr, fpr, thresh)
compute_optimal_threshold_from_prec_recall(precision, recall, thresh)
compute_partial_vars(model_obj, max_index)
evaluate_model(X_test, y_test, verbose, opt_thresh_comp)
get_avarage_shap_values(shap_values)
get_feats_with_top_n_mixed_impacts(n, local_av_imp, local_max_imp)
get_gbt_feature_importance(importances)
get_gbt_model_rules(gbt_model, k)
get_global_feature_importances(verbose, n_imp_feats, feats_to_highlight)
get_local_feature_importances(q, n, plot_flag, verbose, feats_to_highlight)
get_lreg_coefficients(lreg_model, cols, regression)
get_max_shap_values(shap_values)
get_most_important_cat_features(imp_feats, n_imp_feats, verbose)
get_shap_values(predict_fn, q, forced, explainer_type)
model_dependent_data_preparation(d, cols, catcols, method)
plot_correlation_between_numeric_features(feats)
plot_trees(n_est)
predict(X_test)
print_feature_importances(imp, title, feats_to_highlight)
scatterplot_gbt(feats, threshold, outlier, hue)
split_test_and_train_data(d, cols, label, threshold, splitpercent, ts, splittime, resampling, split_random, verbose_flag)
train_and_validate(X_train, y_train, X_val, y_val, num_of_iter, max_depth, learning_rate, num_leaves, plot_flag, task_type, verbose_flag, ts, splitpercent, splittime, split_random)
train_model(d_X, d_y, n_est, max_d, learning_rate, k_fold, num_leaves, plot_flag, ts, splitpercent, splittime, split_random, task_type, verbose_flag)

Figure B.5: UML diagram of the GBT Surrogate Model

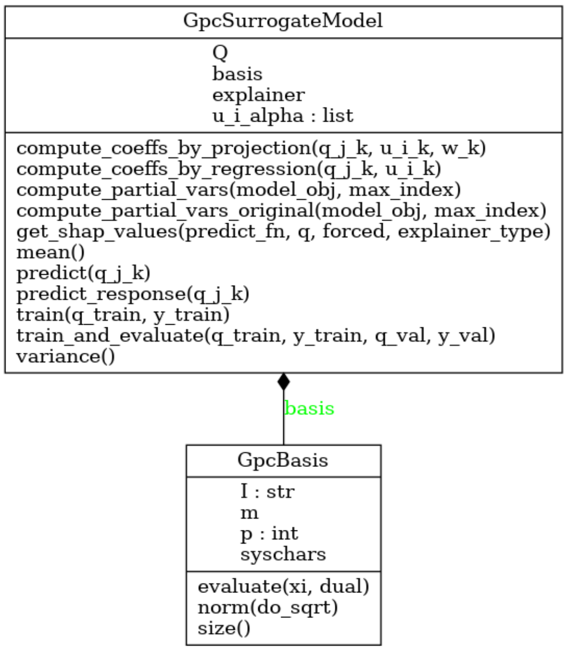

GpcSurrogateModel
Q
basis
explainer
u_i_alpha : list
compute_coeffs_by_projection(q_j_k, u_i_k, w_k)
compute_coeffs_by_regression(q_j_k, u_i_k)
compute_partial_vars(model_obj, max_index)
compute_partial_vars_original(model_obj, max_index)
get_shap_values(predict_fn, q, forced, explainer_type)
mean()
predict(q_j_k)
predict_response(q_j_k)
train(q_train, y_train)
train_and_evaluate(q_train, y_train, q_val, y_val)
variance()
basis
GpcBasis
I : str
m
p : int
syschars
evaluate(xi, dual)
norm(do_sqrt)
size()

Figure B.6: UML diagram of the GPC Surrogate Model

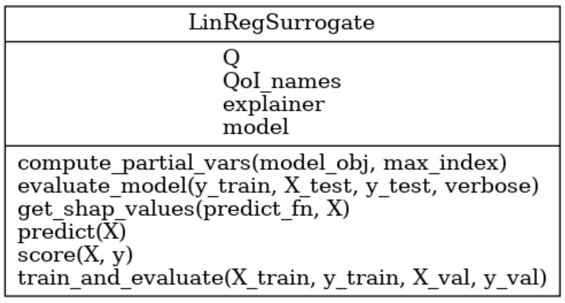

LinRegSurrogate
Q
QoI_names
explainer
model
compute_partial_vars(model_obj, max_index)
evaluate_model(y_train, X_test, y_test, verbose)
get_shap_values(predict_fn, X)
predict(X)
score(X, y)
train_and_evaluate(X_train, y_train, X_val, y_val)

Figure B.7: UML diagram of the Linear Regression Surrogate Model

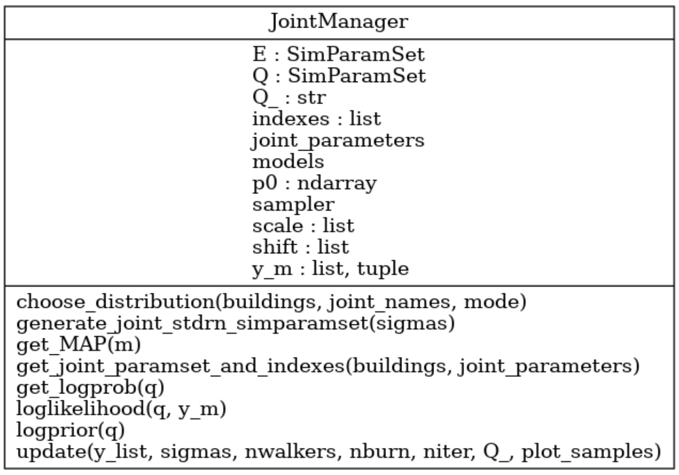

JointManager
E : SimParamSet
Q : SimParamSet
Q_ : str
indexes : list
joint_parameters
models
p0 : ndarray
sampler
scale : list
shift : list
y_m : list, tuple
choose_distribution(buildings, joint_names, mode)
generate_joint_stdrn_simparamset(sigmas)
get_MAP(m)
get_joint_paramset_and_indexes(buildings, joint_parameters)
get_logprob(q)
loglikelihood(q, y_m)
logprior(q)
update(y_list, sigmas, nwalkers, nburn, niter, Q_, plot_samples)

Figure B.8: UML diagram of the JointManager class

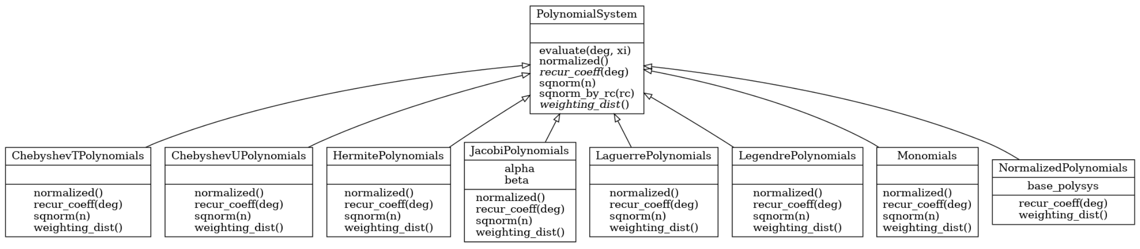


Figure B.9: UML diagram of the Polynomial System

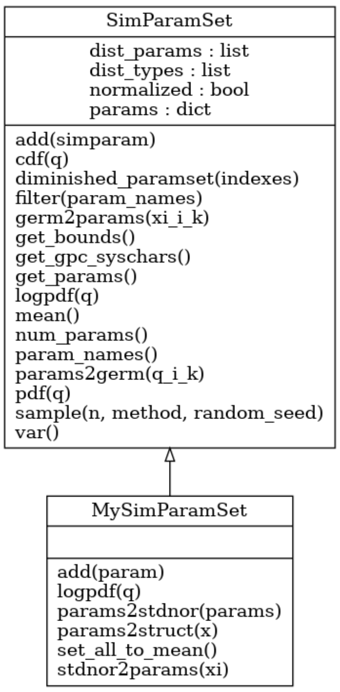


Figure B.10: UML diagram of the SimParamSet class

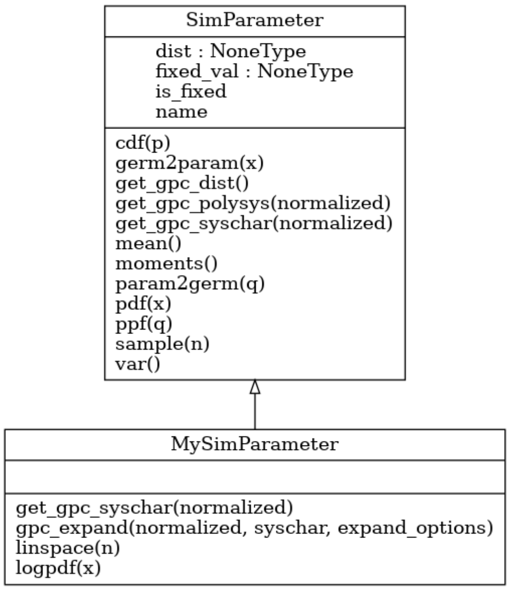
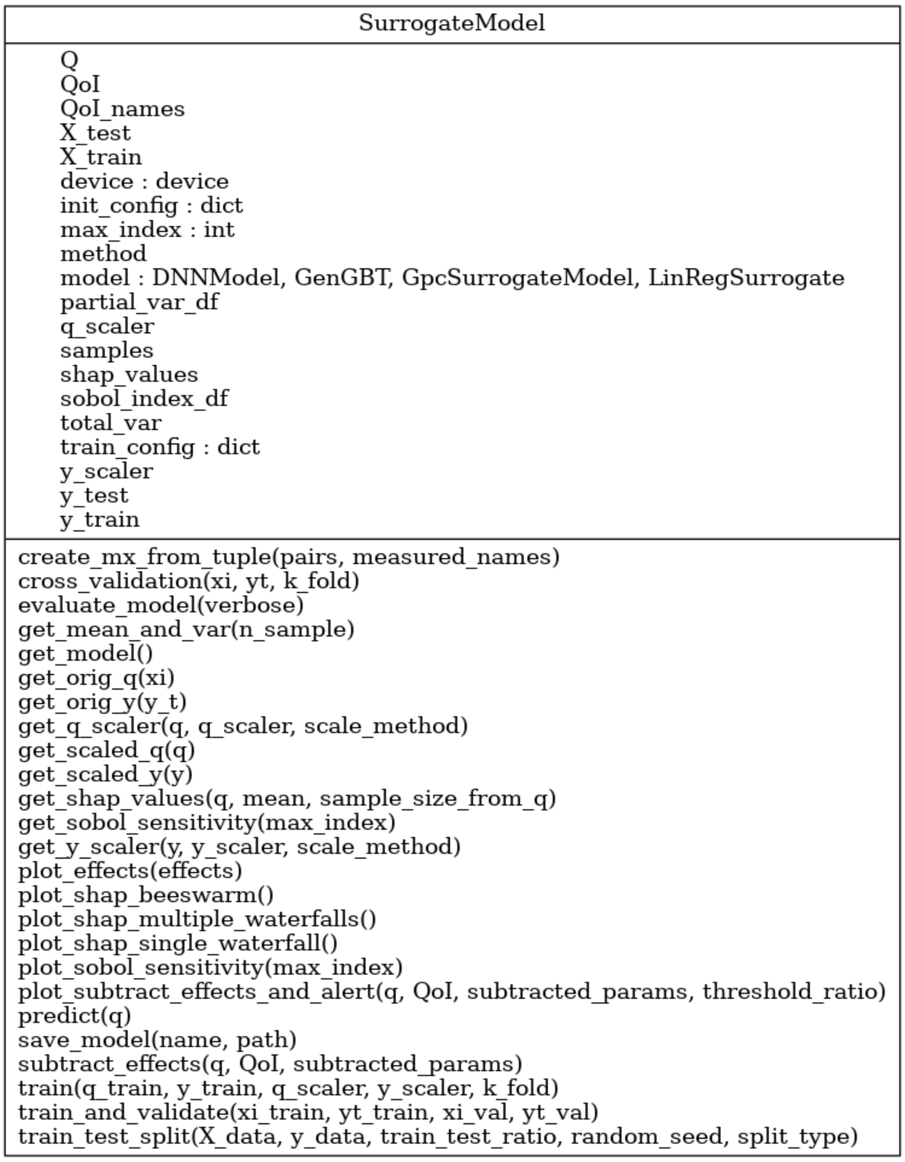


Figure B.11: UML diagram of the SimParameter class



```
SurrogateModel
--------------------------------------------------------------
    Q
    QoI
    QoI_names
    X_test
    X_train
    device : device
    init_config : dict
    max_index : int
    method
    model : DNNModel, GenGBT, GpcSurrogateModel, LinRegSurrogate
    partial_var_df
    q_scaler
    samples
    shap_values
    sobol_index_df
    total_var
    train_config : dict
    y_scaler
    y_test
    y_train
--------------------------------------------------------------
create_mx_from_tuple(pairs, measured_names)
cross_validation(xi, yt, k_fold)
evaluate_model(verbose)
get_mean_and_var(n_sample)
get_model()
get_orig_q(xi)
get_orig_y(y_t)
get_q_scaler(q, q_scaler, scale_method)
get_scaled_q(q)
get_scaled_y(y)
get_shap_values(q, mean, sample_size_from_q)
get_sobol_sensitivity(max_index)
get_y_scaler(y, y_scaler, scale_method)
plot_effects(effects)
plot_shap_beeswarm()
plot_shap_multiple_waterfalls()
plot_shap_single_waterfall()
plot_sobol_sensitivity(max_index)
plot_subtract_effects_and_alert(q, QoI, subtracted_params, threshold_ratio)
predict(q)
save_model(name, path)
subtract_effects(q, QoI, subtracted_params)
train(q_train, y_train, q_scaler, y_scaler, k_fold)
train_and_validate(xi_train, yt_train, xi_val, yt_val)
train_test_split(X_data, y_data, train_test_ratio, random_seed, split_type)
```

Figure B.12: UML diagram of the SurrogateModel class

B.3 Earthquake Resilience

- `get_Parameters()` — Retrieves seismic parameters for the given location.

  Inputs:

  - `latitude` — *float*, latitude in decimal degrees
  - `longitude` — *float*, longitude in decimal degrees

  Outputs:

  - `ParaTR` — *pandas.DataFrame*, seismic parameters (the ground acceleration on type A ground, the maximum value of the amplification factor, and the reference period needed for the evaluation of the constant segment of the spectrum for return periods 30, 50, 72, 101, 140, 201, 475, 975, 2475)
  - `city` — *string*, city name
  - `country_code` — *string*, country code

- `get_2D_layout()` — Retrieves 2D layout figures from Excel data.

  Input:

  - `xlsx` — *pandas.ExcelFile*, Excel file with information of the building

  Output:

  - `figures` — *list of matplotlib.pyplot.figures*, 2D layout figures for every floor

- `linear_static_analysis()` — Performs linear static analysis.

  Inputs:

  - `xlsx` — *pandas.ExcelFile*, Excel file with information of the building
  - `ParaTR` — *pandas.DataFrame*, seismic parameters
  - `soil_category` — *string*, soil category
  - `topographic_category` — *string*, topographic category
  - `service_life` — *integer*, nominal life of the building
  - `importance_class` — *string*, importance class of the building
  - `q` — *float*, behavior factor

  Outputs:

  - `IR_fess_X` — *list of arrays*, wall safety factors under Shear load in the first main direction
  - `IR_fess_Y` — *list of arrays*, wall safety factors under Shear load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-in-plane load in the first main direction
  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-in-plane load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the first main direction

  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the second main direction

- `get_linear_dataframes()` — Retrieves linear analysis output in *pandas.DataFrame* format for the chosen floor and direction

  Inputs:

  - `direction` — *string*, main direction
  - `IR_fess_X` — *list of arrays*, wall safety factors under Shear load in the first main direction
  - `IR_fess_Y` — *list of arrays*, wall safety factors under Shear load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-in-plane load in the first main direction
  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-in-plane load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the first main direction
  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the second main direction
  - `chosen_floor` — *integer*, index of the chosen floor

  Output:

  - `dataframe` — *pandas.DataFrame*, merged data frame of the wall safety factors with the Shear, Bending-in-plane and Bending-out-of-plane loads in the chosen floor and direction

- `choose_linear_analysis_plot()` — Retrieves 2D floor layouts highlighting the walls with safety factor < 1 for all load types in the chosen floor and direction

  Inputs:

  - `xlsx` — *pandas.ExcelFile*, Excel file with information of the building
  - `direction` — *string*, main direction
  - `IR_fess_X` — *list of arrays*, wall safety factors under Shear load in the first main direction
  - `IR_fess_Y` — *list of arrays*, wall safety factors under Shear load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-in-plane load in the first main direction
  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-in-plane load in the second main direction
  - `IR_pf_X` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the first main direction
  - `IR_pf_Y` — *list of arrays*, wall safety factors under Bending-out-of-plane load in the second main direction
  - `chosen_floor` — *integer*, index of the chosen floor

Output:

- `figures` — *list of matplotlib.pyplot.figures*, 2D layout figures of the wall safety factors for each load type in the chosen floor and direction, highlighting the walls with safety factor < 1

- `pushover_analysis()` — Runs nonlinear pushover analysis and retrieves results.

  Inputs:

  - `xlsx` — *pandas.ExcelFile*, Excel file with information of the building
  - `ParaTR` — *pandas.DataFrame*, seismic parameters
  - `soil_category` — *string*, soil category
  - `topographic_category` — *string*, topographic category
  - `service_life` — *integer*, nominal life of the building
  - `importance_class` — *string*, importance class of the building
  - `check_type` — *string*, type of the resilience check

  Outputs:

  - `failed_walls` — *list of arrays*, indices of the failed walls
  - `dataframe_results` — *pandas.DataFrame*, data frame with the results of the analysis (Safety Index, PGA_C, TR, δ, d* (t) in both directions)
  - `figures` — *list of matplotlib.pyplot.figures*, capacity curve and ADRS figures

- `plot_wall_failures()` — Retrieves 2D floor layout highlighting the failed walls after the pushover analysis in the chosen floor and direction

  Inputs:

  - `xlsx` — *pandas.ExcelFile*, Excel file with information of the building
  - `direction` — *string*, main direction
  - `walls` — *list of arrays*, indices of the failed walls
  - `chosen_floor` — *integer*, index of the chosen floor

  Output:

  - `figure` — *matplotlib.pyplot.figure*, 2D layout of the chosen floor highlighting the failed walls in the chosen direction

B.4 Climate Resilience

- `coordinate_check()` — Retrieves the city name and the country code on the given coordinate. Sends an error message if the location is not in the European continent.

  Inputs:

  - `latitude` — *float*, latitude in decimal degrees
  - `longitude` — *float*, longitude in decimal degrees

Outputs:

- `city` — *string*, city name
- `country_code` — *string*, country code

- `request_data()` — Retrieves climate model data through the Copernicus API

  Inputs:

  - `api_key` — *string*, Copernicus API key
  - `model_name` — *string*, name of the requested model (only works with the ’IRNET’ model currently)
  - `experiment_option` — *string*, type of the requested experiment (’Historical + RCP 4.5’ or ’Historical + RCP 8.5’)
  - `variable_option` — *string*, name of the requested variable (’Minimum Temperature’, ’Maximum Temperature’, ’Precipitation’ or ’Wind Speed’)
  - `latitude` — *float*, latitude in decimal degrees
  - `longitude` — *float*, longitude in decimal degrees
  - `city` — *string*, city name

  Output:

  - `model_data` — *dictionary*, daily values of the requested variable in the given location between 1950 and 2100 obtained from the requested model and using the chosen experiment

- `processed_dataframe()` — Corrects the units in the climate data, calculates yearly values and organizes the data into time windows.

  Inputs:

  - `dfs_model` — *dictionary*, raw climate data
  - `window_length` — *integer*, length of the time windows in years
  - `shift` — *integer*, shift of the time windows in years

  Outputs:

  - `dfs_processed` — *dictionary of pandas.DataFrames*, processed yearly values of the data organized in time windows
  - `windows` — *list of tuples*, start and end years of the time windows

- `ev1_param()` — Retrieves the evaluation of the time windows with the chosen method and return period

  Inputs:

  - `dfs_processed` — *dictionary of pandas.DataFrames*, processed yearly values of the data organized in time windows
  - `T_r` — *integer*, return period in years
  - `method` — *string*, method type (’MLM’, ’LSM’ or ’MOM’)

  Output:

  - `results` — *dictionary*, results of the calculations for every time window (characteristic value, factors of change of characteristic value, standard deviation, mean value, coefficient of variance, parmhat, parmci)

- `model_output()` — Retrives the figure of the selected result value

  Inputs:

  - `results` — *dictionary*, results of the data evaluation
  - `value` — *string*, name of the result value ('Characteristic Value', 'Factors of Change of Characteristic value', 'Coefficient of Variance', 'Mean Value' or 'Gumbel Probability Paper')
  - `dfs_processed` — *dictionary of pandas.DataFrames*, processed yearly values of the data organized in time windows (optional, only used if `value` = 'Gumbel Probability Paper')
  - `T_r` — *integer*, return period in years (optional, only used if `value` = 'Gumbel Probability Paper')
  - `window` — *tuple*, start and end year of a time window (optional, only used if `value` = 'Gumbel Probability Paper')

  Outputs:

  - `figure` — *matplotlib.pyplot.figure*, figure of the selected value
  - `dataframe` — *pd.DataFrame*, data of the figure (returns only if `value` != 'Gumbel Probability Paper')

Appendix C

Notebooks

C.1 Module 1: Data-Driven Modeling with the Python Library Package

C.1.1 Toy Example: Modeling from Synthetic Data, Sensitivity Analysis, Explainability, Subtracting Affects

Data-Driven Modeling of a Sine Function

This notebook demonstrates the core functionality of the Data-Driven module using a simple analytical sine function as a toy model. The goal is to approximate a known function:

$$y = a\sin(bt + c)$$

using machine learning techniques based on synthetic data. The input parameters $\mathbf{x} = [a, b, c]^T$ are sampled from predefined distributions, and the outputs $\mathbf{y}$ are computed at $N = 9$ discrete time steps. Gaussian noise is added to simulate observational uncertainty.

A **gPCE (Generalized Polynomial Chaos Expansion)** model is trained to approximate the mapping from input parameters to function outputs.

Sensitivity analyses using Sobol indices are performed to assess the global influence of each input parameter on the response.

SHAP (SHapley Additive exPlanations) values are computed to explain the contributions of individual input features on model predictions.

The **effects** of input parameters are isolated and subtracted from the model response to better understand their influence.

Imports

In [1]:
```
import sys
import os
parent_dir = os.path.abspath("../../libraries/surrogate_modelling")
sys.path.insert(0, parent_dir)
```

In [2]:
```
from distributions import *
from simparameter import SimParameter
from simparameter_set import SimParamSet
from surrogate_model import SurrogateModel
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from digital_twin import DigitalTwin
import utils
from surrogate_model import *

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Load data

In [3]:
```
x_df = pd.read_csv('../data/sine_data/example_x_df.csv')
y_df = pd.read_csv('../data/sine_data/example_y_df.csv')
```

For training and sensitivity analysis

In [4]:
```
Q = utils.get_simparamset_from_data(x_df)
QoI_names = y_df.columns.to_list()

max_index = 2
QoI_param = 't_3'

subtracted_effects = ['a','b','c']
```

For measurements

In [5]:
```
z_m_df = pd.read_csv('../data/sine_data/example_z_m_df.csv')
sigma = pd.read_csv('../data/sine_data/example_sigma_df.csv')
```

For update

In [6]:
```
nwalkers = 64
nburn = 800
niter = 200

E = utils.generate_stdrn_simparamset(sigma.values.reshape(-1, 1))

q = np.load('../data/sine_data/example_q.npy')
q_df = pd.read_csv('../data/sine_data/example_q_df.csv')
```

Train model

Choosing the Model

In this notebook, we implement the model using one of the following methods: **gPCE**, **DNN**, **GBT**, or **LinReg**.

- **gPCE (Generalized Polynomial Chaos Expansion)**: Models uncertainty by expressing the output as a polynomial expansion, with coefficients determined by projecting the system's response onto orthogonal polynomials.
- **DNN (Deep Neural Networks)**: Machine learning models composed of interconnected neurons, capable of learning complex, non-linear patterns from data.
- **GBT (Gradient Boosting Trees)**: Combines weak decision trees into a strong predictive model, iteratively correcting errors from previous trees for robust performance.

- **LinReg (Linear Regression)**: A statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.

In [7]:
```
method = "gPCE"
```

In [8]:
```
# Model configurations
match method:
    # DNN model configurations
    case "DNN":
        config = {
            'init_config' : {
                'layers': [
                    {'neurons': 512, 'activation': 'relu', 'dropout': 0.2},
                    {'neurons': 256, 'activation': 'sigmoid', 'dropout': 0.2},
                    {'neurons': 128, 'activation': 'relu', 'dropout': None},
                    ],
                'outputAF': 'tanh'
                },
            'train_config' : {
                'optimizer': 'Adam',
                'loss': 'MSE',
                'epochs': 100,
                'batch_size': 32,
                'k_fold': None,
                'early_stopping': {
                    'patience': 25,
                    'min_delta': 0.0001}
                }
                }
    # gPCE model configurations
    case "gPCE":
        config = {
            'init_config' : {
            'p' : 5
            },
            'train_config' : {
                'k_fold': 5
                }
        }
    # GBT model configurations
    case "GBT":
        config = {
            'init_config' : {
                'gbt_method': 'xgboost'
            },
            'train_config' : {
                'max_depth': 3,
                'num_of_iter': 250,
                'k_fold': 5
                }
        }
```

In [9]:
```
split_config = {
    'train_test_ratio': 0.2,
    'random_seed': 1997,
    'split_type': 'no_shuffle'
    }
```

In [10]:
```
# Initialize surrogate model
# This creates an instance of the SurrogateModel class using the provided sampli
model = SurrogateModel(Q, QoI_names, method, **config['init_config'])
# Split data into training and testing sets
model.train_test_split(x_df, y_df, **split_config)
# Train the model
model.train(model.X_train, model.y_train, **config['train_config'])
```

```
----- Training started for 'gPCE' model -----
Fold 1/5
Fold 2/5
Fold 3/5
Fold 4/5
Fold 5/5
Average train loss: 0.00000000000000, Average valid loss: 0.00000000000000
----- Training ended for 'gPCE' model -----
```

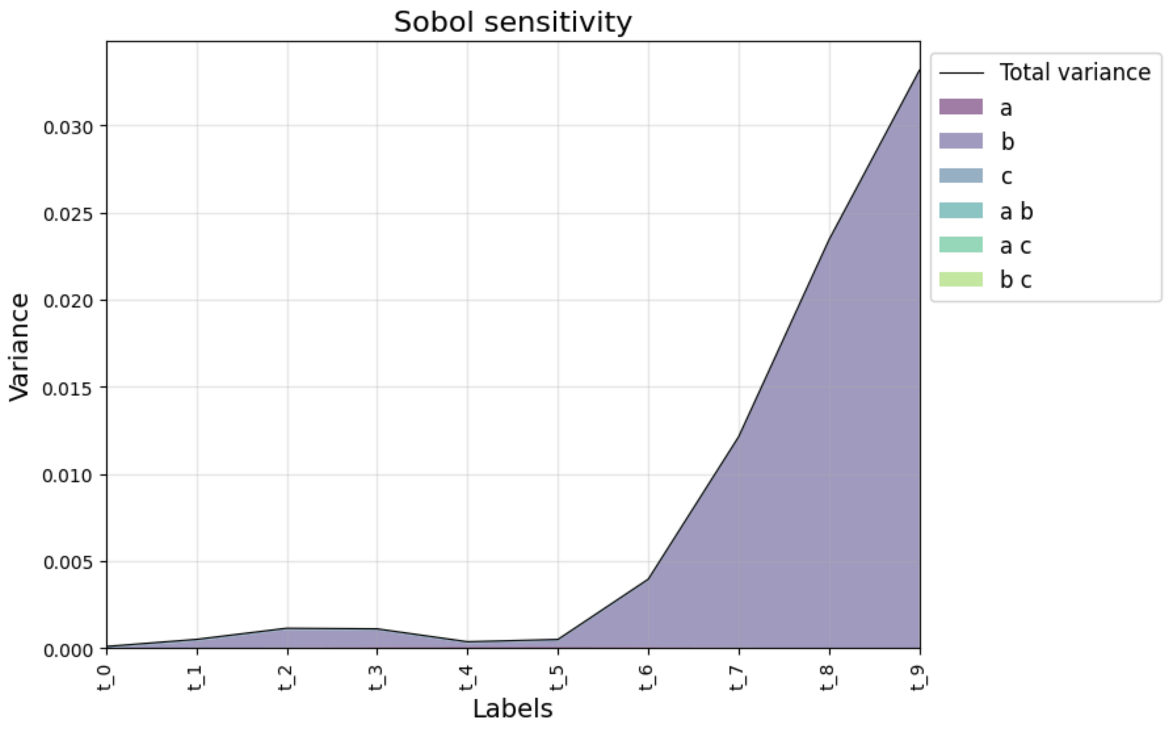
In [11]:
```
# get mean and variance of surrogate model
mean, var = model.get_mean_and_var()
mean, var
```

Out[11]:
```
(tensor([5.7904e-04, 3.4249e-01, 6.4277e-01, 8.6492e-01, 9.8207e-01, 9.8027e-0
1,
         8.6017e-01, 6.3689e-01, 3.3810e-01, 5.3171e-04], dtype=torch.float64),
 tensor([9.7192e-05, 4.6615e-04, 1.0875e-03, 1.0816e-03, 3.9445e-04, 5.1472e-0
4,
         3.8145e-03, 1.1676e-02, 2.2656e-02, 3.2258e-02], dtype=torch.float64))
```

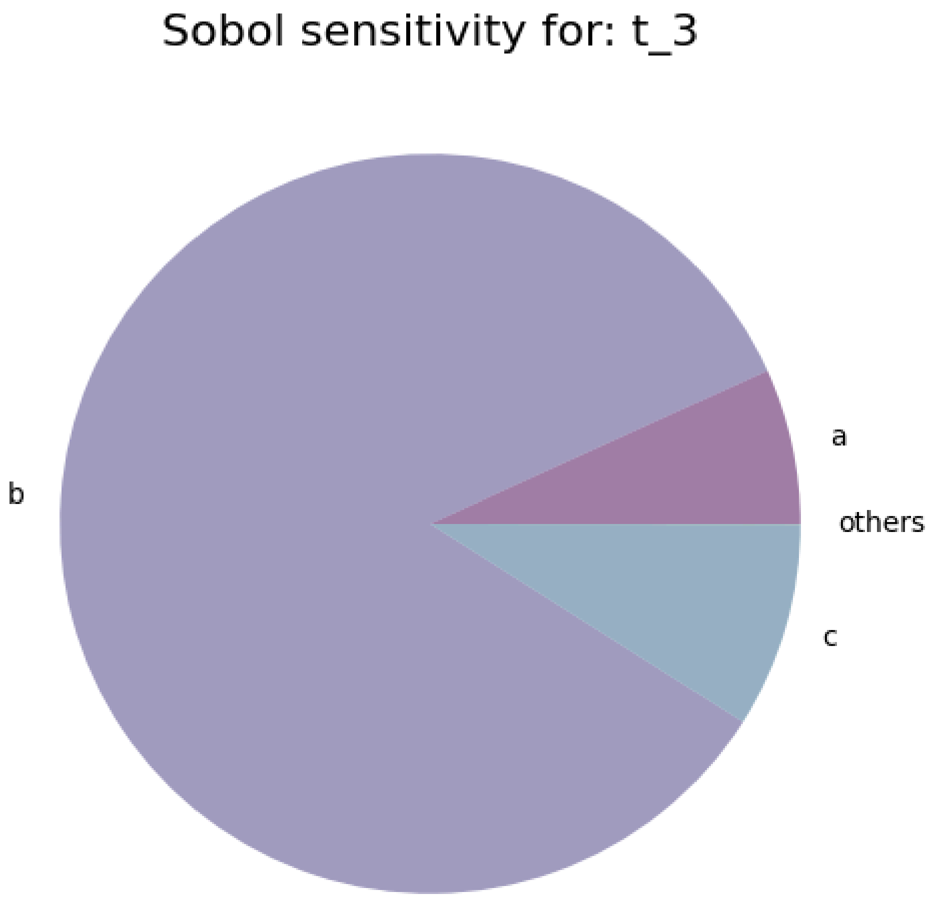
Sobol sensitivities

In [12]:
```
# set max_index for Sobol sensitivity analysis
max_index = 2
partial_variance, sobol_index = model.get_sobol_sensitivity(max_index)
```

In [13]:
```
# Plot Sobol Sensitivity Index
# The 'plot_sobol_sensitivity' method generates a plot of Sobol Sensitivity indi
# the contribution of each input parameter to the output uncertainty.
# - max_index: Specifies the maximum index of parameters to consider in the plot
# - param_name: The name of the quantity of interest (QoI) for which sensitivity
fig = model.plot_sobol_sensitivity()
```



```
In [14]: fig = model.plot_sobol_sensitivity(max_index=max_index, param_name=QoI_param)
```



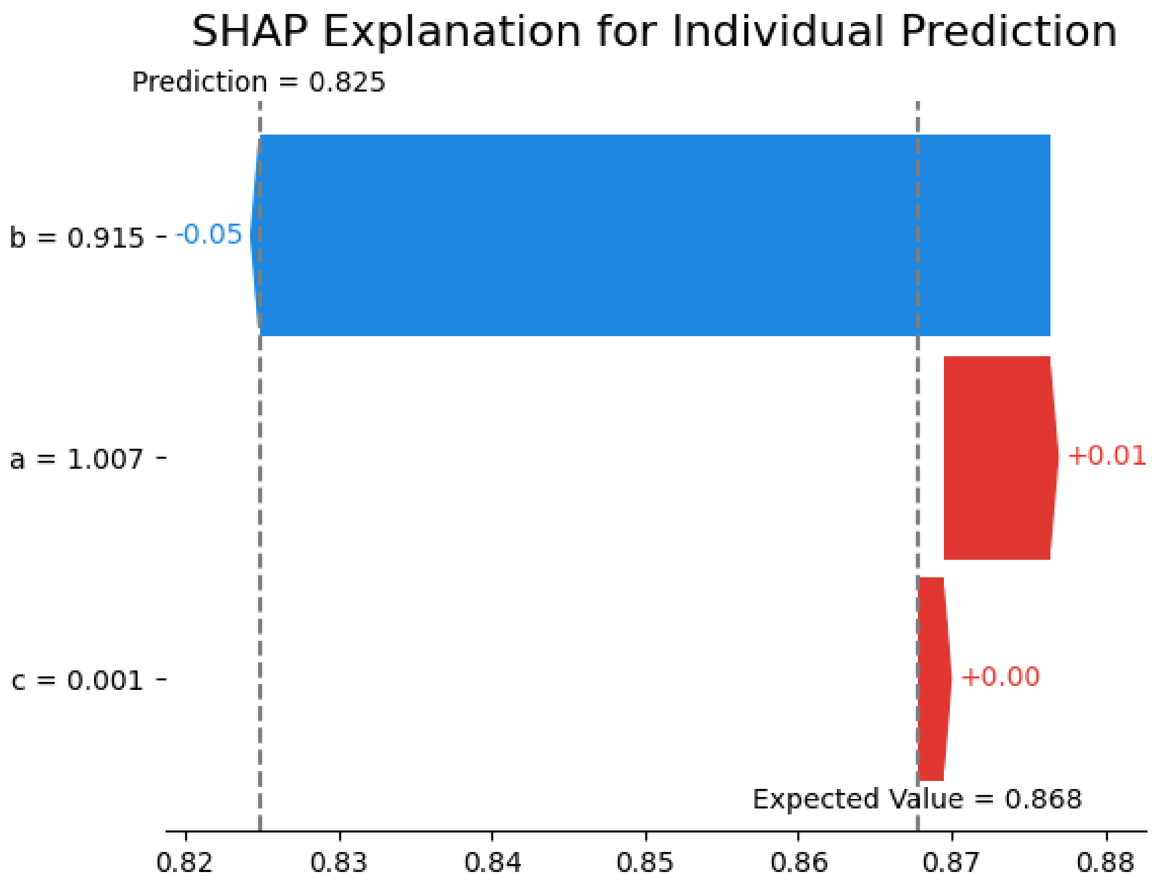
SHAP values

```
In [15]: # Calculate SHAP (Shapley Additive Explanations) values
         # The 'get_shap_values' method computes the SHAP values for the model's test dat
         # SHAP values explain the contribution of each input feature to the prediction f
         # SHAP values help to interpret the model's decisions and understand the impact
         shap_values = await model.get_shap_values(model.X_test)
```

```
Message: sample size for shap values is set to 100.
  0%|          | 0/100 [00:00<?, ?it/s]
```

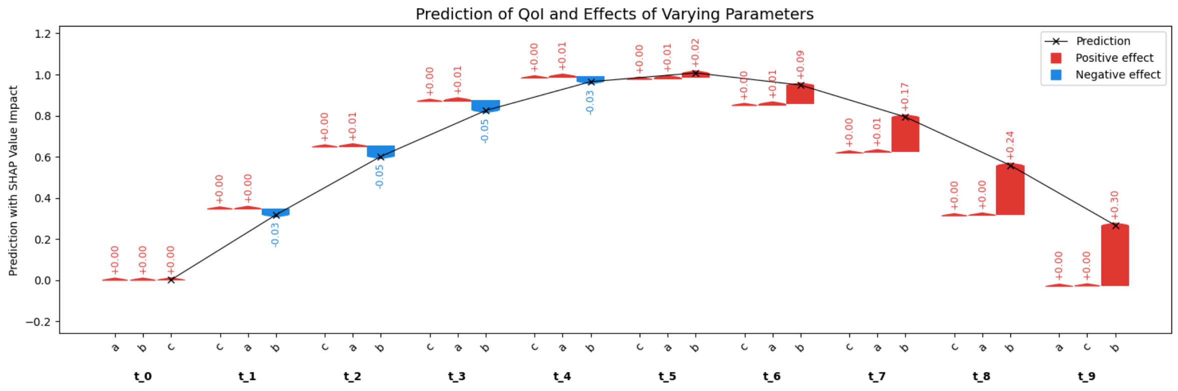
```
In [16]: # Plot SHAP Single Waterfall Plot
         # The 'plot_shap_single_waterfall' method generates a SHAP waterfall plot for a
         # visualizing how each feature contributes to the model's prediction for that sa
         # - q: A DataFrame containing the sample data (e.g., a specific parameter set) f
         # - param_name: The name of the quantity of interest (QoI) being analyzed ('t_3'
         # The SHAP waterfall plot shows the cumulative effect of each feature on the mod
         fig = await model.plot_shap_single_waterfall(q=q_df, param_name=QoI_param)
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```

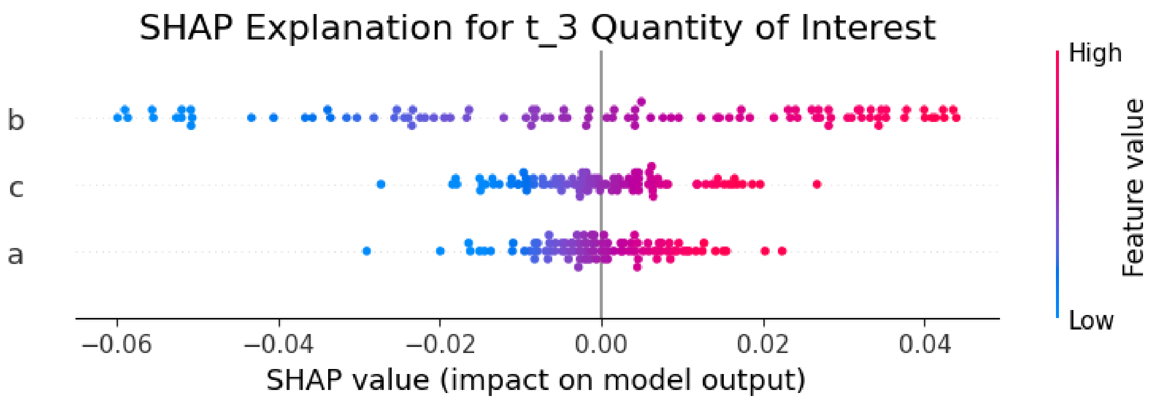


```
In [17]: fig = await model.plot_shap_multiple_waterfalls(q=q_df)
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```



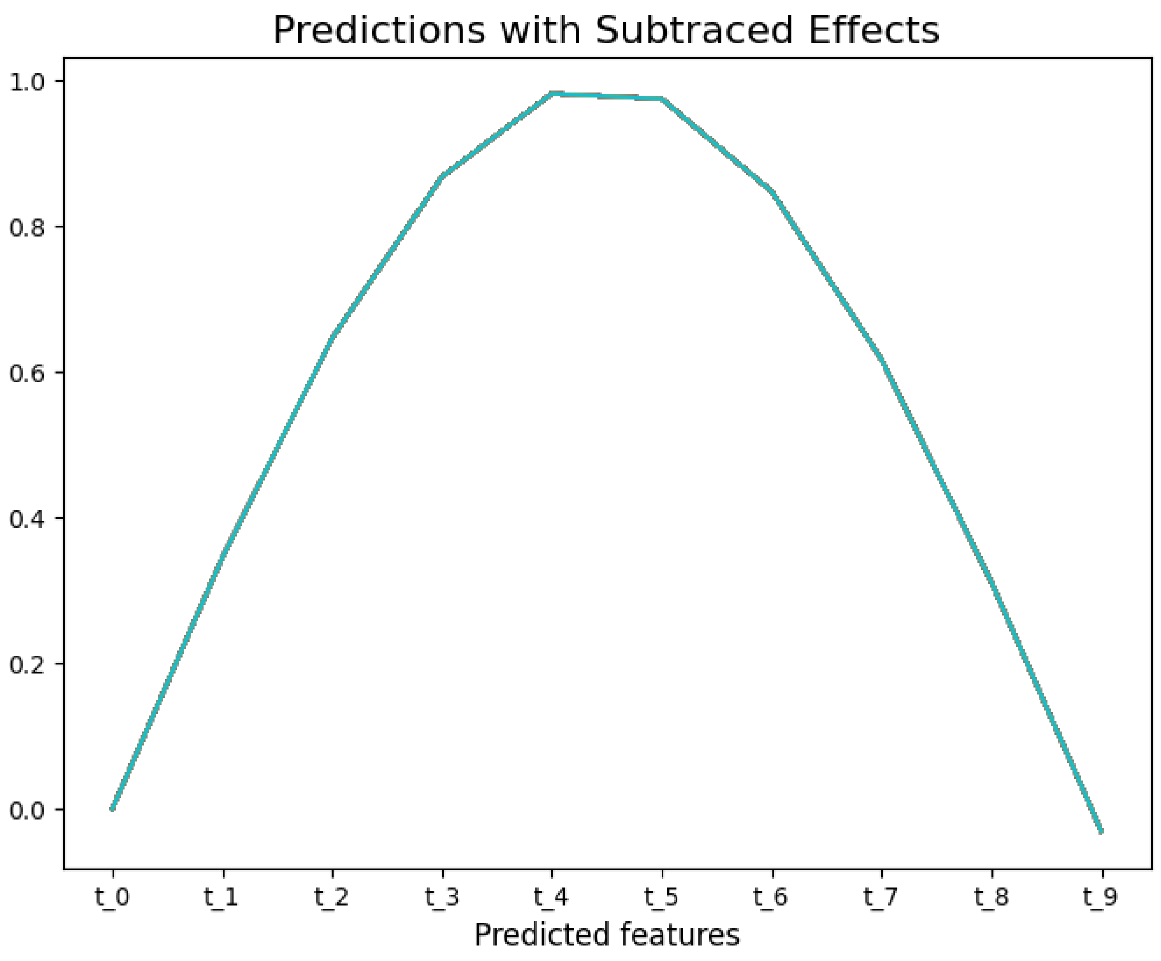
In [18]:
```
# Plot SHAP Beeswarm Plot
# The 'plot_shap_beeswarm' method generates a SHAP beeswarm plot, which visualiz
# for a range of test samples, showing the impact of each feature on the model’s
# - q: The test data (model.X_test.iloc[:100]) is selected for which SHAP values
# - param_name: The name of the quantity of interest (QoI) for which SHAP values
# The SHAP beeswarm plot shows how different input features influence the predic
# helping to understand the global feature importance and the relationships betw
fig = await model.plot_shap_beeswarm(param_name=QoI_param)
```



Effects

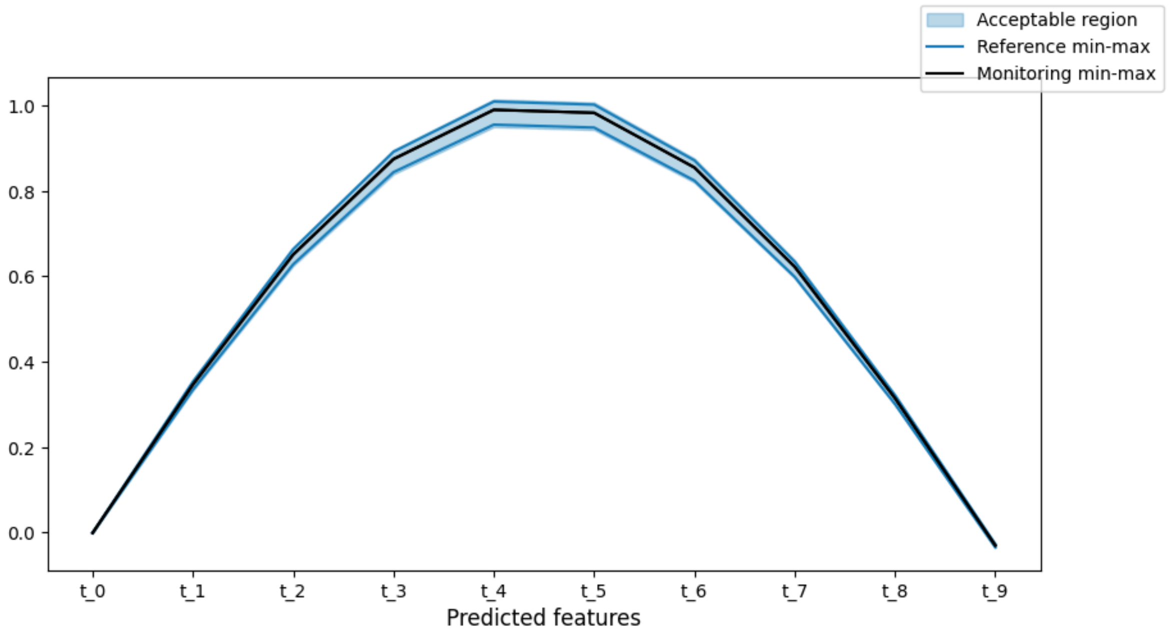
In [19]:
```
effects = await model.subtract_effects(model.X_test.iloc[:100], model.y_test.ilc
fig = await model.plot_effects(effects)
```

```
Message: sample size for shap values is set to 100, which is the number of sample
s.
  0%|          | 0/100 [00:00<?, ?it/s]
```



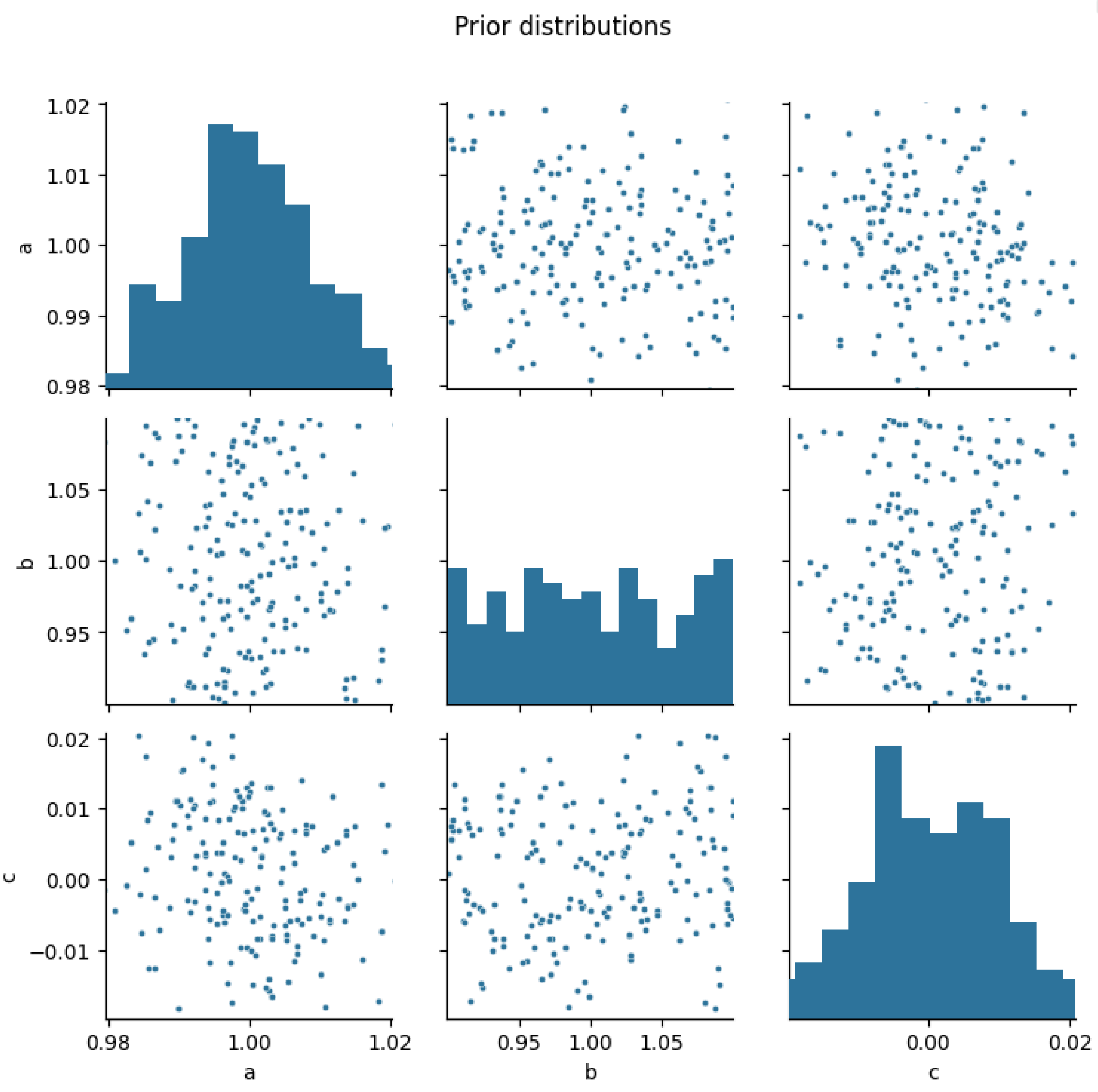
```
In [20]: figure, alert, remaining_effects, error_ratio, outliers = await model.plot_subtr
```

```
Message: sample size for shap values is set to 200, which is the number of sample
s.
  0%|          | 0/200 [00:00<?, ?it/s]
Message: sample size for shap values is set to 1, which is the number of samples.
  0%|          | 0/1 [00:00<?, ?it/s]
```



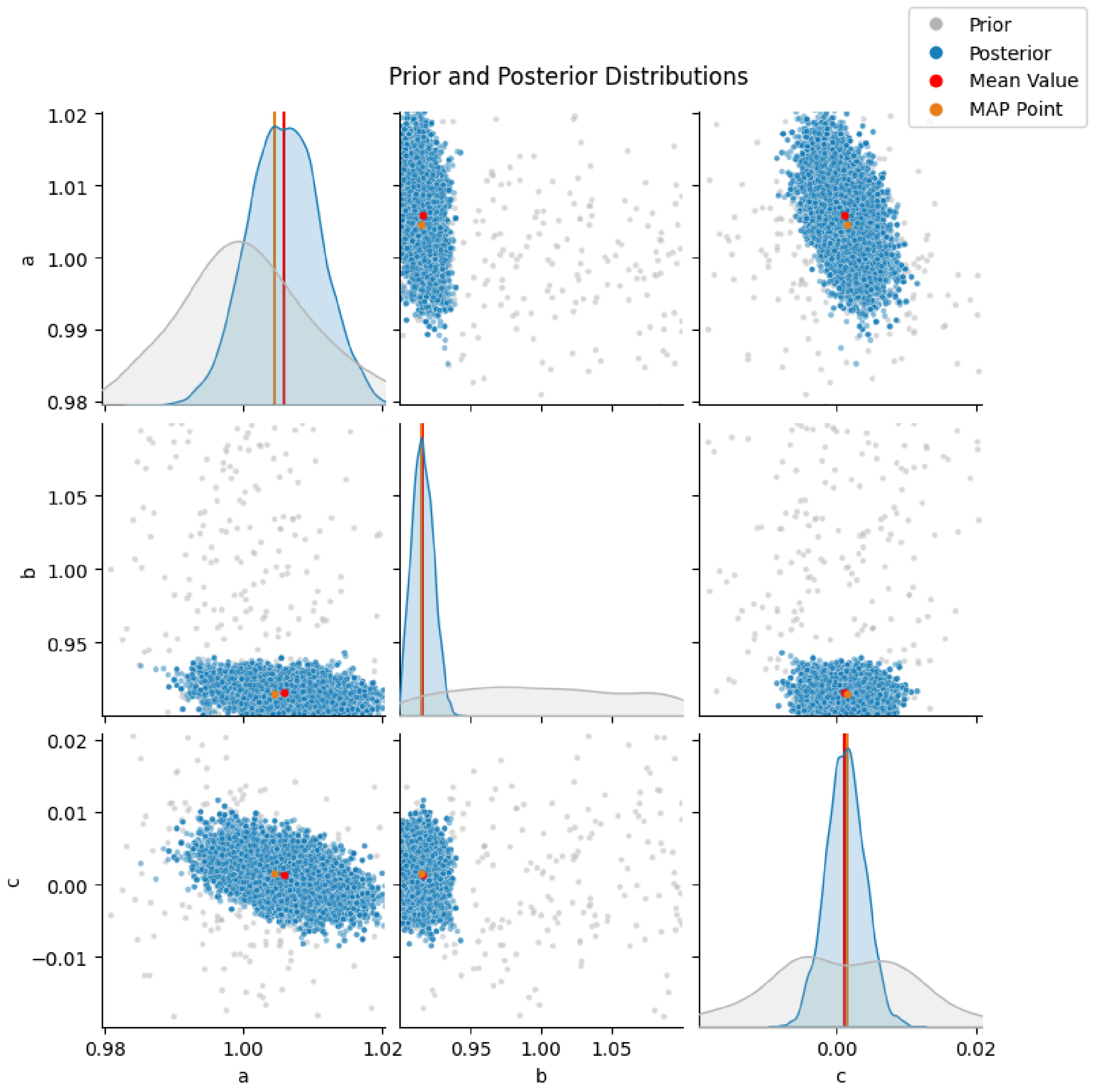
Update

```
In [21]: fig = utils.plot_prior(model)
```



```
In [22]: DT = DigitalTwin(model, E)
         DT.update(z_m_df, nwalkers=nwalkers, nburn=nburn, niter=niter)
         DT.get_mean_and_var_of_posterior()
         gpc_map = DT.get_MAP()
         fig = utils.plot_MCMC(model, DT, nwalkers=nwalkers, map_point=gpc_map)
```

```
MCMC creating
Burning period
100%|██████████| 800/800 [01:07<00:00, 11.77it/s]
MCMC running
100%|██████████| 200/200 [00:16<00:00, 11.90it/s]
--- 84.87607264518738 seconds ---
```


Prior and Posterior Distributions
Prior
Posterior
Mean Value
MAP Point
a
b
c
0.98
1.00
1.02
0.95
1.00
1.05
0.00
0.02
1.01
0.99
-0.01
0.01

C.1.2 Pilot1: Granada Hospital Real – Subtracting Environmental Effects from SHM Data

Exploratory Analysis on Modeshapes

Imports

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

import sys
import os
parent_dir = os.path.abspath("../../libraries/surrogate_modelling")
sys.path.insert(0, parent_dir)

from preprocess_modeshape_data import *
from plotting_functions import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
/home/vemese/miniforge3/envs/Buildchain_sandbox/lib/python3.10/site-packages/shap/plo
ts/colors/_colorconv.py:819: DeprecationWarning: Converting `np.inexact` or `np.float
ing` to a dtype is deprecated. The current result is `float64` which is not strictly
correct.
  if np.issubdtype(dtype_in, np.dtype(dtype).type):
<frozen importlib._bootstrap>:241: RuntimeWarning: numpy.ufunc size changed, may indi
cate binary incompatibility. Expected 216 from C header, got 232 from PyObject
```

Load data

In [2]:
```
df_modeshapes = df_from_unv('../data/OMA hospital/hospital_data_06_01.unv')
weather_df = preprocess_weather_data('../data/OMA hospital/granada_weather_data.csv')
```

In [3]:
```
merged_df = merge_dataframes(df_modeshapes, weather_df)
merged_df['mode_id'].value_counts()
```

Out[3]:
```
mode_id
 3    1802
 1    1799
 2    1279
-1    1255
 0     356
 6     350
-2     169
Name: count, dtype: int64
```

Mode ids -1 and -2 correspond to unrecognised or duplicate modes.

```
In [4]: modeshapes_dict = separate__and_flip_modes(merged_df)
        modeshapes_df = merge_modes(modeshapes_dict)
        modeshapes_df.head(2)
```

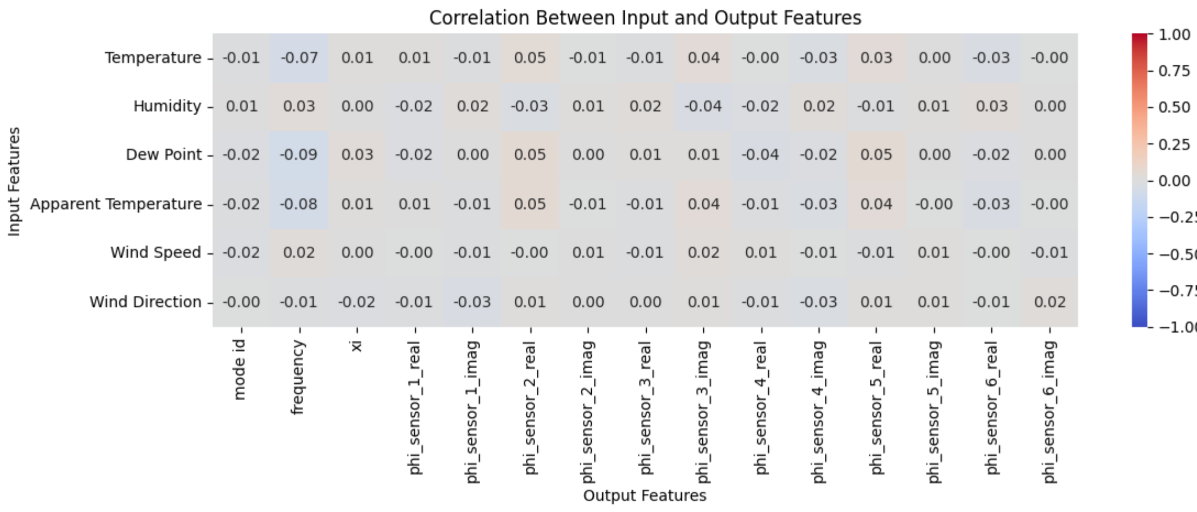
Out[4]:

	phi_sensor_1_real	phi_sensor_1_imag	phi_sensor_2_real	phi_sensor_2_imag	phi_sensor_3_real	phi_sens
0	0.010114	0.029889	0.53301	-0.036790	0.090962	
1	0.067700	0.073652	0.57011	-0.038071	0.069899	

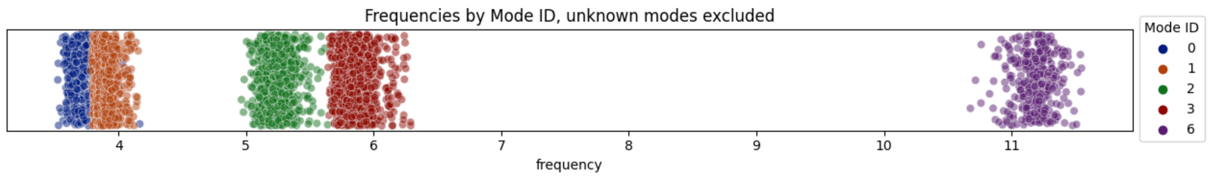
2 rows × 22 columns

Frequencies

```
In [5]: fig = plot_correlation(weather_df, df_modeshapes, mode='input-output')
```

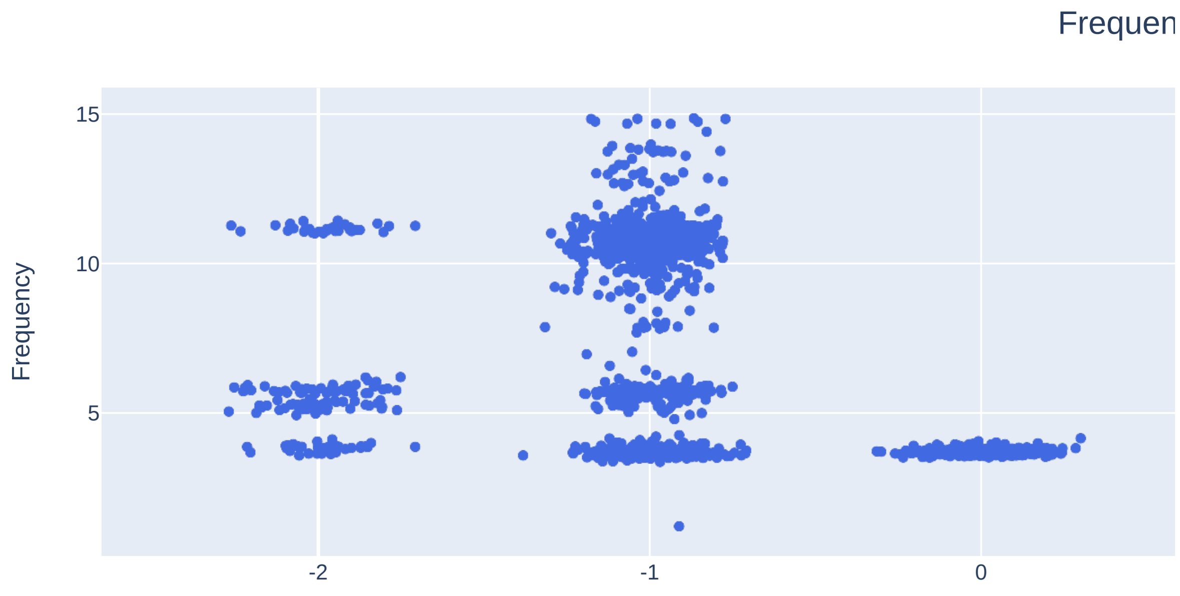


```
In [6]: data = modeshapes_df[~modeshapes_df['mode_id'].isin([-1, -2])].copy()

        y_jitter = np.random.uniform(-0.1, 0.1, size=len(data))

        plt.figure(figsize=(12, 2))
        sns.scatterplot(x='frequency', y=y_jitter, hue='mode_id', data=data, palette='dark', a
        plt.yticks([])
        plt.xlabel('frequency')
        plt.title('Frequencies by Mode ID, unknown modes excluded')
        plt.tight_layout()
        plt.legend(title='Mode ID', bbox_to_anchor=(1, 1.2))
        plt.show()
```

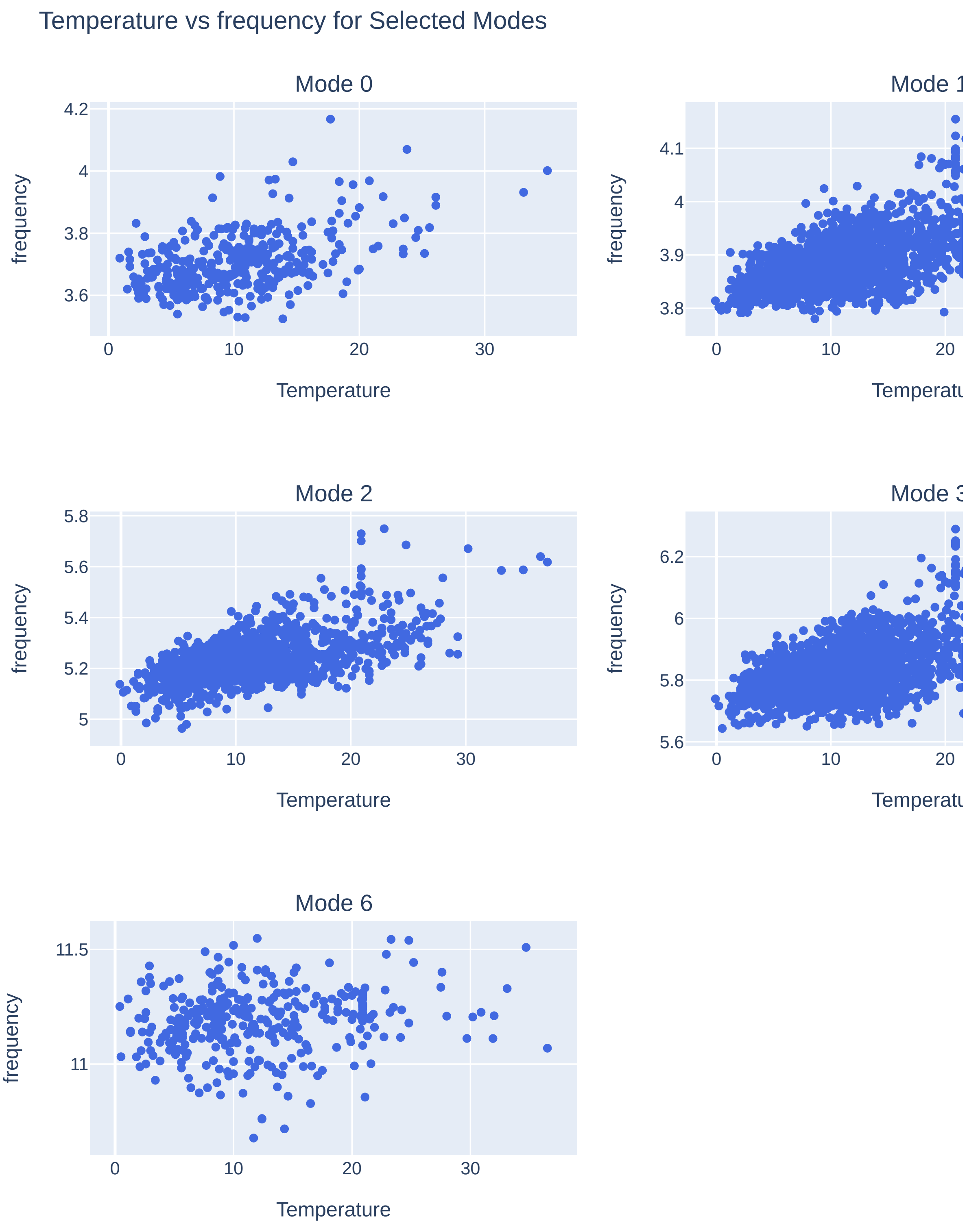


In [7]:
```python
# Sort the mode_id categories
sorted_modes = sorted(modeshapes_df['mode_id'].unique())
mode_id_mapping = {mode: i for i, mode in enumerate(sorted_modes)}

# Apply mapping and add jitter
modeshapes_df['jitter'] = modeshapes_df['mode_id'].map(mode_id_mapping) + np.random.nc

# Create plot
fig = px.scatter(modeshapes_df, x='jitter', y='frequency', width=1500, height=400)
fig.update_traces(marker=dict(color='royalblue'))

# Update layout with title and x-axis labels
fig.update_layout(
    title=dict(
        text='Frequencies by Mode ID with Jitter',
        x=0.5,
        xanchor='center',
        font=dict(size=18)
    ),
    xaxis=dict(
        tickmode='array',
        tickvals=list(mode_id_mapping.values()),
        ticktext=list(mode_id_mapping.keys()),
        title='Mode ID'
    ),
    yaxis=dict(title='Frequency')
)

fig.show()
```

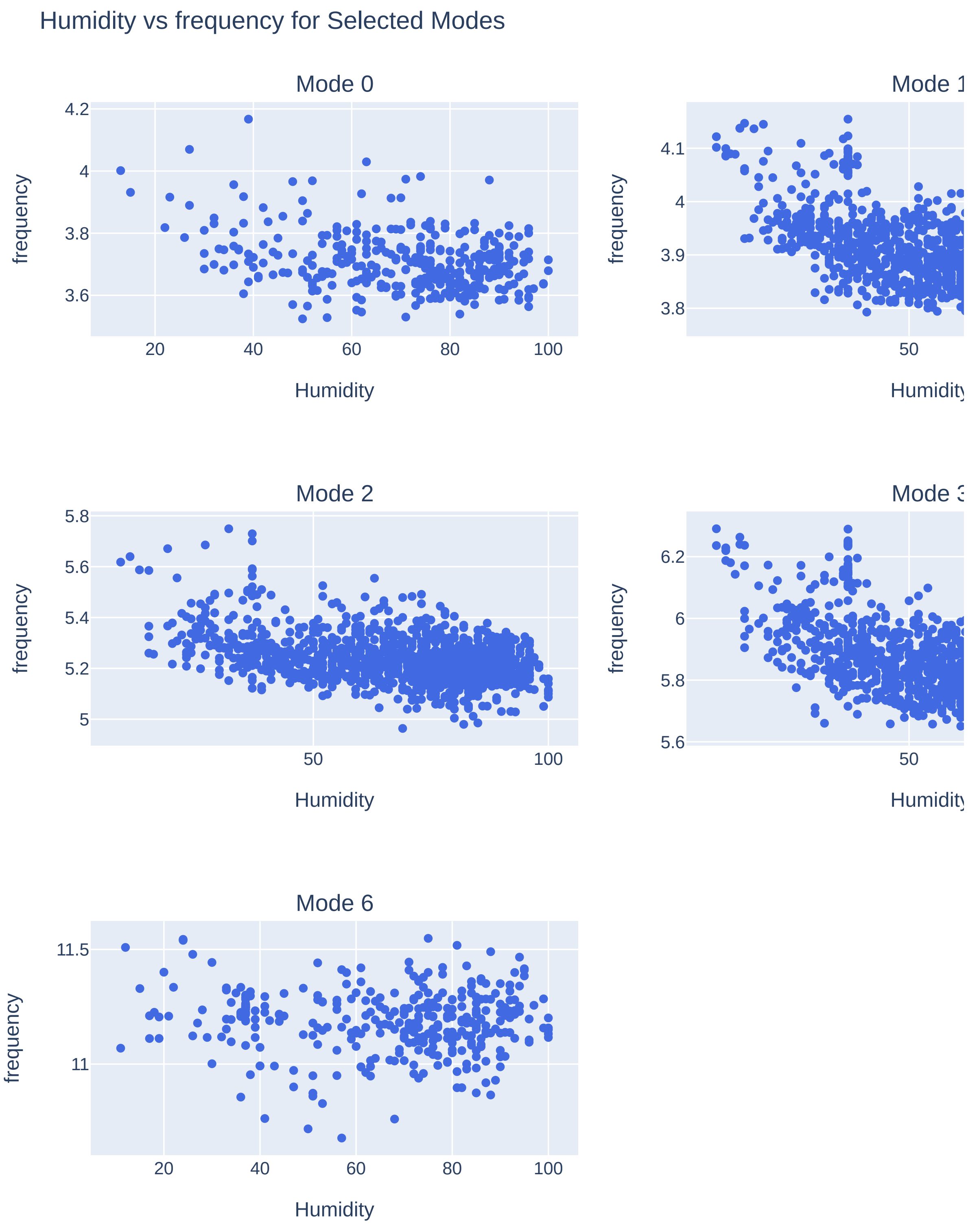
```
/home/vemese/miniforge3/envs/Buildchain_sandbox/lib/python3.10/site-packages/plotly/i
o/_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [8]:
```python
fig = scatter_plot_modes_subplots(modeshapes_dict, x_key='Temperature', y_key='frequer
fig.show()
```



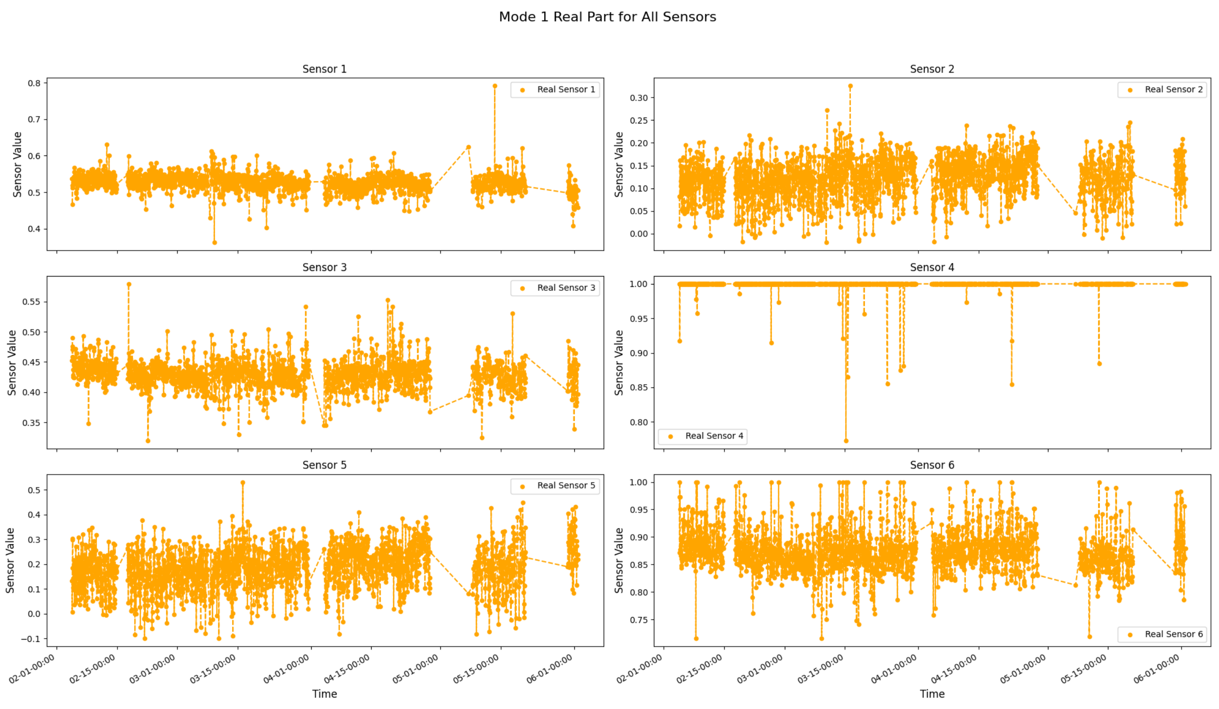
```
In [9]: fig = scatter_plot_modes_subplots(modeshapes_dict, x_key='Humidity', y_key='frequency'
        fig.show()
```

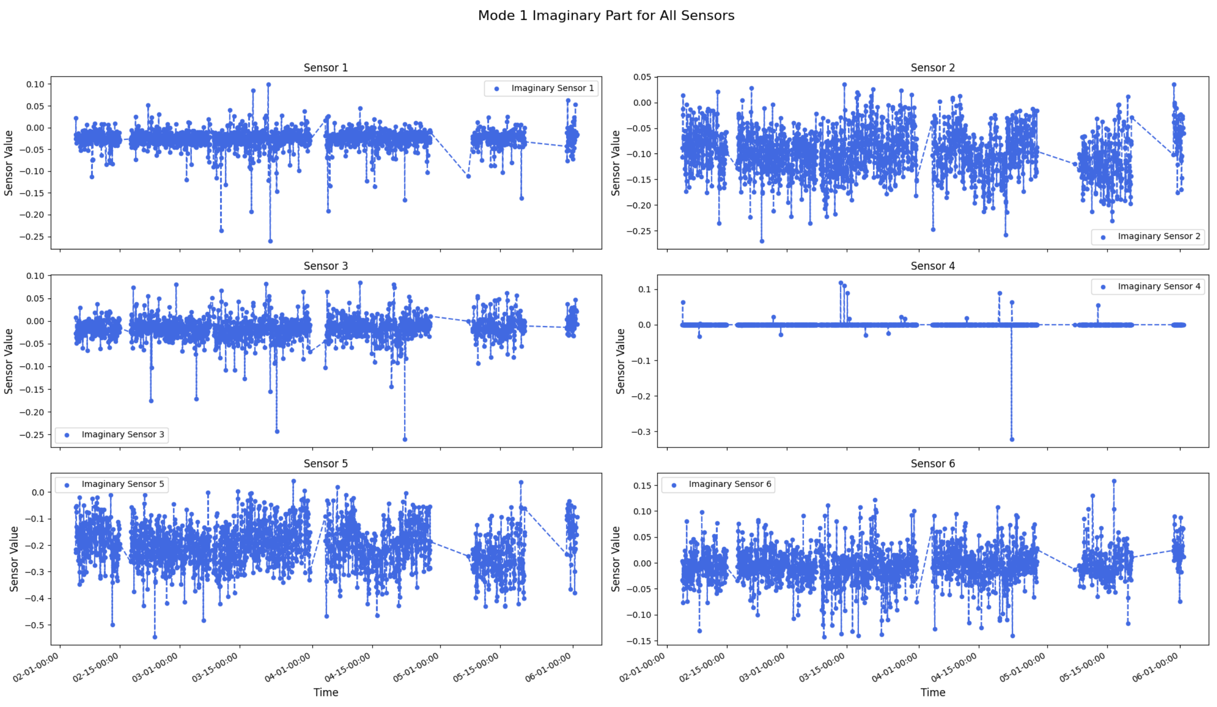


Mode 1 plots

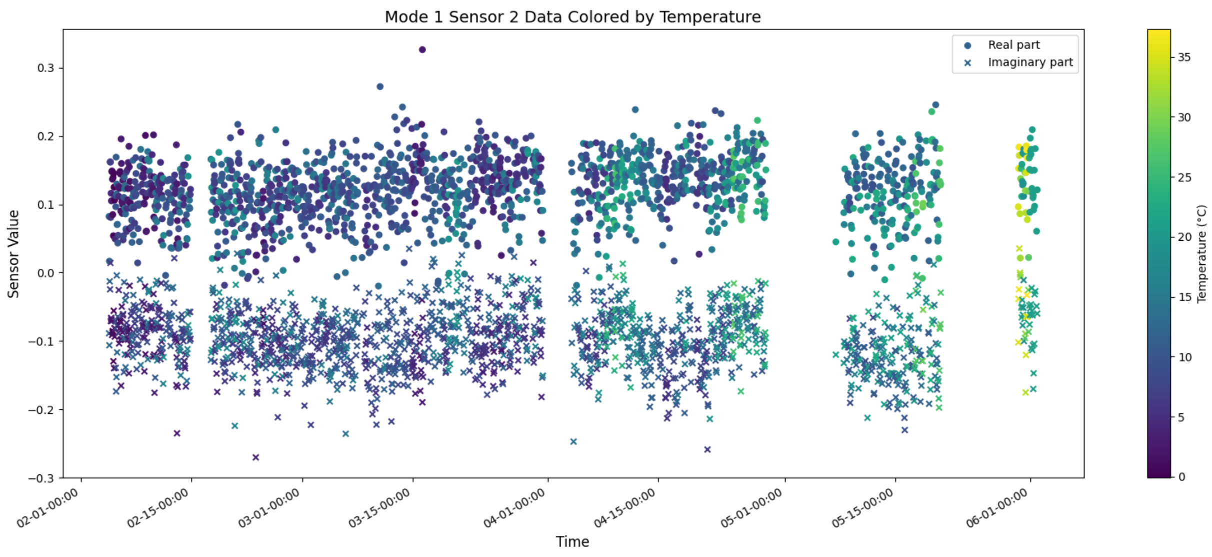
```
In [10]: mode = 1
```

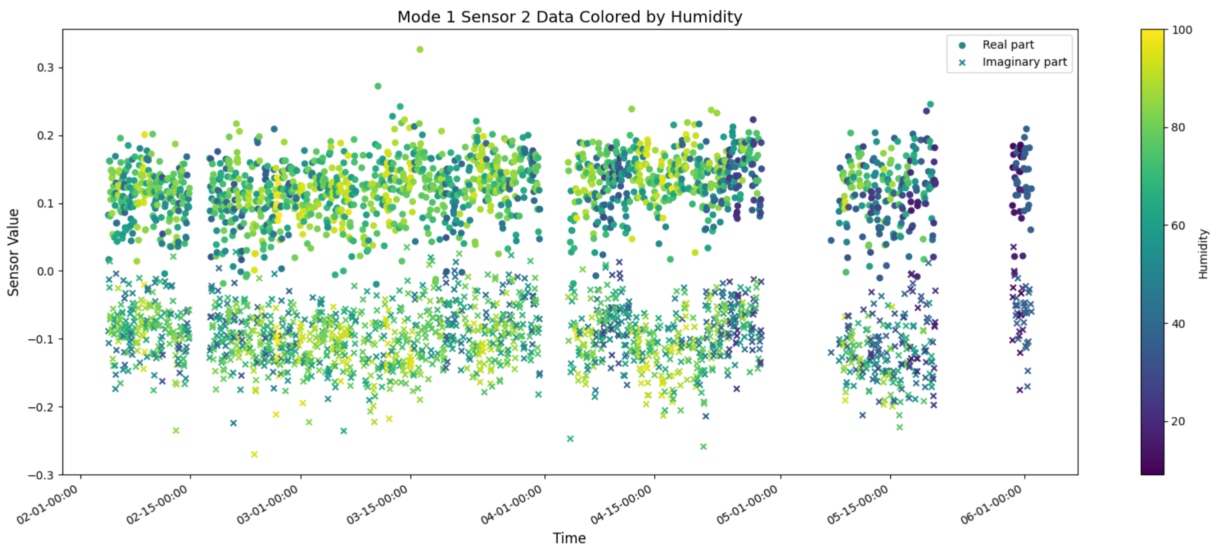
```
In [11]: fig_real, fig_imag = plot_parts(modeshapes_dict, mode)
```



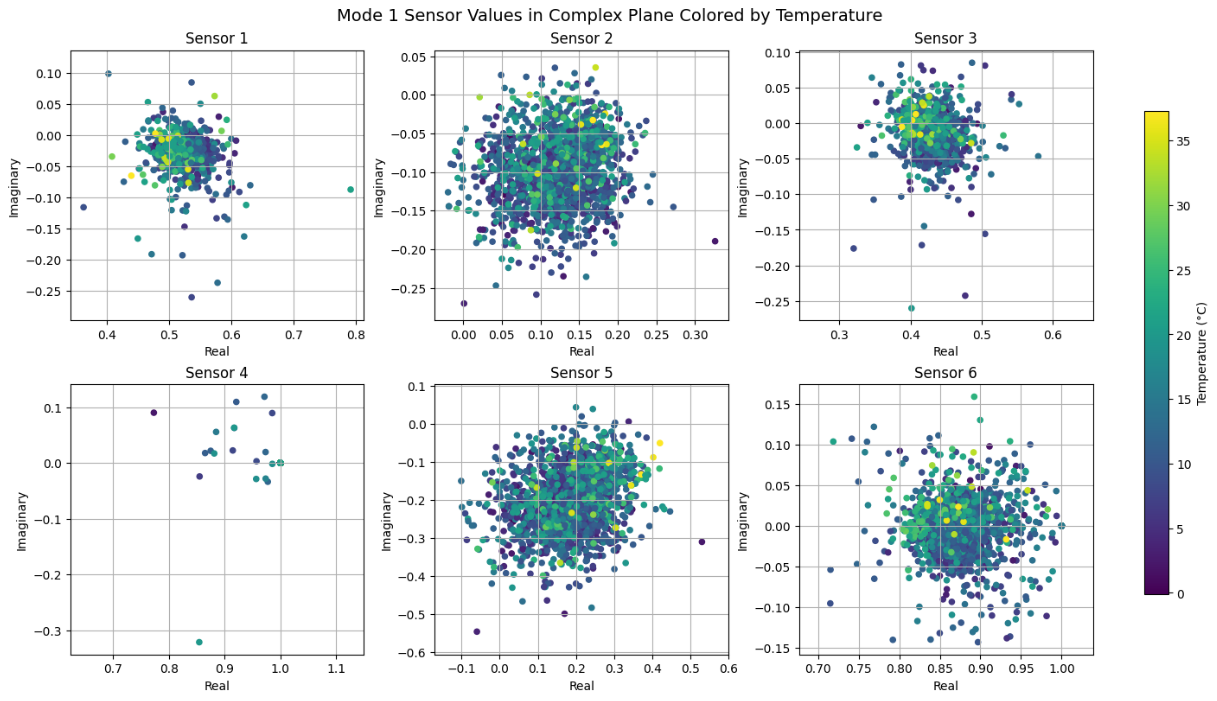


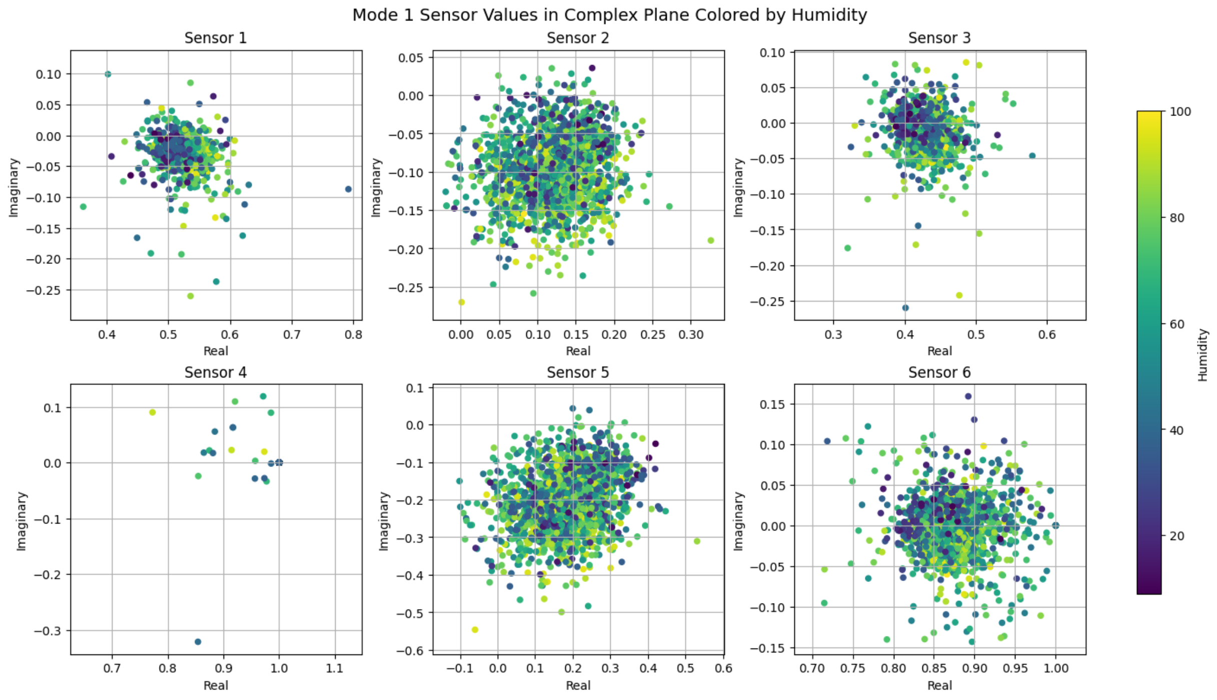
In [12]:
```
fig_temp = plot_sensor_complex_colored(modeshapes_dict, mode, sensor=2, color_by='Temp
fig_hum = plot_sensor_complex_colored(modeshapes_dict, mode, sensor=2, color_by='Humic
```





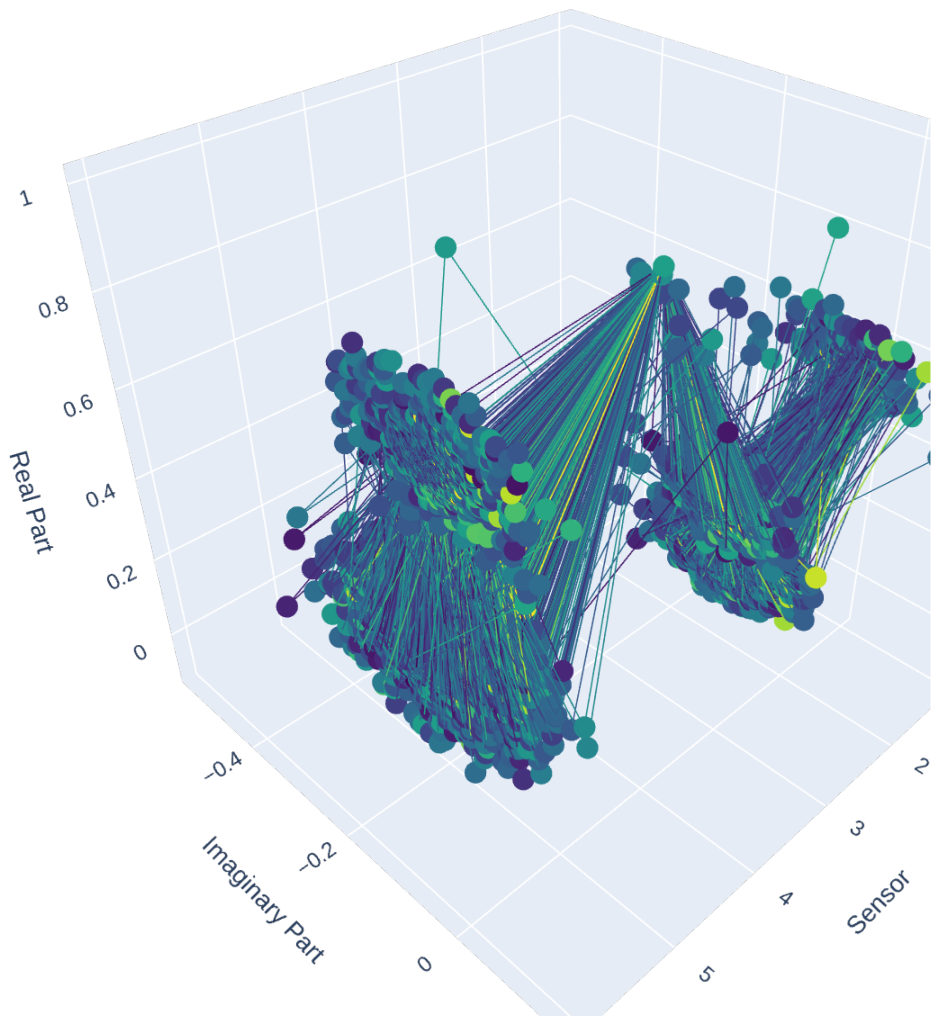
```
In [13]: fig_temp = plot_complex_plane_all_sensors(modeshapes_dict, mode, color_by='Temperature
         fig_hum = plot_complex_plane_all_sensors(modeshapes_dict, mode, color_by='Humidity')
```





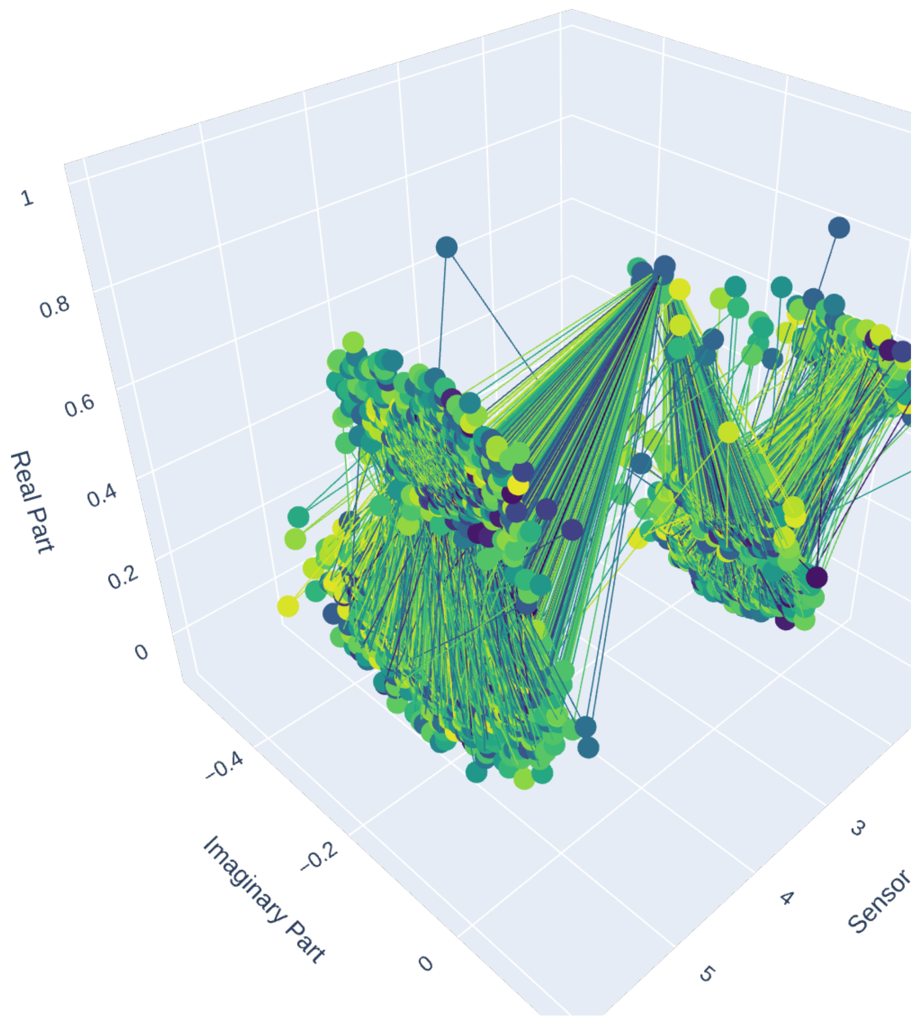
In [14]:
```python
fig = plot_mode_shapes_3d(modeshapes_dict, mode, color_by='Temperature')
fig.show()
```

3D Mode Shape (Mode 1) Colored by Temperature



In [15]:
```python
fig = plot_mode_shapes_3d(modeshapes_dict, mode, color_by='Humidity')
fig.show()
```

3D Mode Shape (Mode 1) Colored by Humidity



C.2 Module 2: Hybrid Digital twinning with the python library package

C.2.1 Toy example: Calibrating a 1D Oscillator Model

Spring-Mass System with Hybrid Digital Twinning

This Jupyter Notebook demonstrates how to use a **Hybrid Digital Twinning** model with a simple **spring system** as a toy example. The model considers **mass (m)** and **stiffness (k)** as variable parameters to showcase the twinning process.

System Overview

The spring-mass system follows Hooke's Law and Newton's Second Law:

$$F = -kx$$

where:

- (k) is the **spring stiffness**,
- (x) is the **displacement from equilibrium**,
- (m) is the **mass of the attached object**.

Analytical Solution

For an ideal undamped system, the displacement at a given time step is given by:

$$u(T) = u_0 \cos\left(\sqrt{\frac{k}{m}}T\right) + \frac{v_0}{\sqrt{\frac{k}{m}}} \sin\left(\sqrt{\frac{k}{m}}T\right)$$

where:

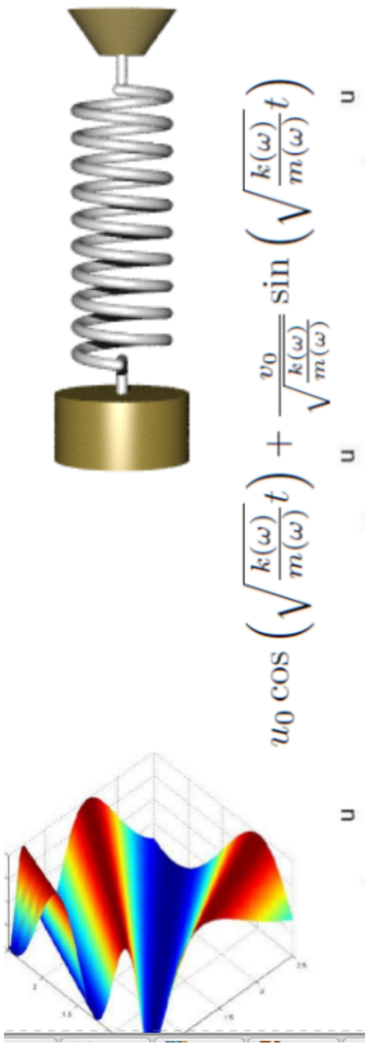
- (u_0) is the **initial displacement**,
- (v_0) is the **initial velocity**,
- (k) and (m) can change dynamically in the twinning process,
- **(T = 10) is the single time step at which the system is evaluated**.

Hybrid Digital Twinning Approach

- The model updates parameters ((k, m)) dynamically.
- It integrates **physical equations** with **real-time data**.
- Simulations showcase how the **digital twin adapts** to variations at a fixed moment in time (**T = 10**).

Visualizations

This notebook includes:
✅ **System state at (T = 10) only** rather than a full time evolution.
✅ **Phase-space representation** (velocity vs. displacement at (T = 10)).
✅ **3D plots** to visualize parameter influence at this specific time step.


$u_0 \cos\left(\sqrt{\frac{k(\omega)}{m(\omega)}}t\right) + \frac{v_0}{\sqrt{\frac{k(\omega)}{m(\omega)}}} \sin\left(\sqrt{\frac{k(\omega)}{m(\omega)}}t\right)$

Imports

In [1]:
```
import sys
import os
parent_dir = os.path.abspath("../../libraries/surrogate_modelling")
sys.path.insert(0, parent_dir)
```

In [2]:
```
from distributions import *
from simparameter import SimParameter
from simparameter_set import SimParamSet
from surrogate_model import SurrogateModel
import pandas as pd
from digital_twin import DigitalTwin
import utils

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Generate data

Function to calculate

```
In [3]: def spring_solve(q_i, x0=1, v0=0, d=0, T=10):
            # Solves the displacement of a damped spring-mass system at a given time ste
            m = q_i[:,0]  # Mass values from input array
            k = q_i[:,1]  # Stiffness values from input array
            alpha = d / m  # Damping per unit mass
            D = (k / m) - (alpha ** 2)  # Adjusted natural frequency squared
            omega = np.sqrt(D)  # Damped angular frequency

            # Compute displacement at time T using the damped harmonic motion formula
            xt = np.exp(-alpha * T) * (x0 * np.cos(omega * T) + (v0 + alpha * x0) / omeg

            xt = xt.reshape(-1, 1)  # Ensure output is a column vector
            return xt

        # Example usage:
        x0, v0, d, T = 1, 0, 0, 10
        spring_solve(np.array([[2, 2]]), x0, v0, d, T)
```

```
Out[3]: array([[-0.83907153]])
```

Create SimParameters and SimParameterSet

```
In [4]: # Define simulation parameters for mass (m) and stiffness (k),
        # each following a uniform distribution within a given range.

        P1 = SimParameter('m', UniformDistribution(0.5, 2.5))  # Mass (m) sampled from a
        P2 = SimParameter('k', UniformDistribution(0.5, 2.5))  # Stiffness (k) sampled f

        # Create a set to store simulation parameters.
        Q = SimParamSet()

        # Add the defined parameters (mass and stiffness) to the parameter set.
        Q.add(P1)
        Q.add(P2)
```

Sampling

```
In [5]: # Generate samples of simulation parameters (mass 'm' and stiffness 'k')
        # We use the **Quasi-Monte Carlo (QMC)** sampling method with a **Halton sequenc
        # The random seed ensures reproducibility of the sample set.
        num_samples = 35000  # Define the number of samples to generate
        Q_i = Q.sample(num_samples, method='QMC_Halton', random_seed=1997)  # Generate p
        Q_df = pd.DataFrame(Q_i, columns=Q.param_names())  # Convert the parameter sampl

        # Generate target values (outputs) based on the parameter samples
        # These are the quantities of interest (QoI), which represent the system's respo
        time_steps = 1  # In this case, we're only evaluating the system at a single tim
        QoI_i = spring_solve(Q_i)  # Solve the spring system for the generated parameter

        # Define names for the QoI (displacement values) columns
        # Since we're evaluating only a single time step (T=10), the output columns are
        QoI_names = [f"t_{i+1}" for i in range(time_steps)]  # List comprehension for co
```

```
# Store the computed values of the quantities of interest (displacement at time
QoI_df = pd.DataFrame(QoI_i, columns=QoI_names)  # Convert the results into a Da
```

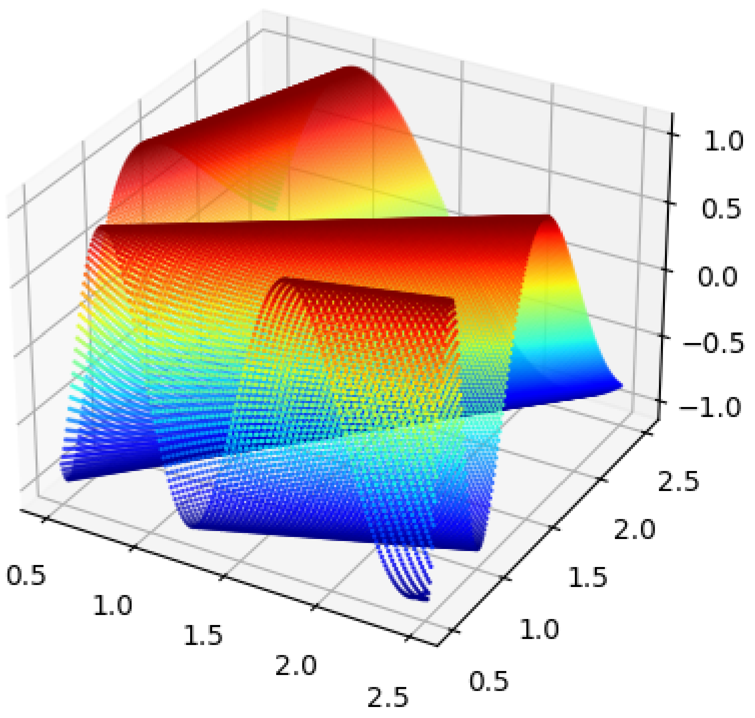
1 sample point

In [6]:
```
# Generate a single sample for the uncertain parameters (mass 'm' and stiffness
q_sample = Q.sample(1, random_seed=1)  # Sample a single set of parameters
qoi_sample = spring_solve(q_sample)  # Solve the spring system for the sampled p
q_sample_df = pd.DataFrame(q_sample, columns=Q.param_names())  # Convert the sam
```

Plot response surface

In [7]:
```
# Define a grid of values for the spring stiffness (k) and mass (m)
# Both k and m are sampled between 0.5 and 2.5, with 200 points for each.
k_grid = np.linspace(0.5, 2.5, 200)  # Create a grid of stiffness values (k)
m_grid = np.linspace(0.5, 2.5, 200)  # Create a grid of mass values (m)

# Create a meshgrid to generate all combinations of k and m for evaluation
# This results in two 2D arrays (K and M) where K corresponds to stiffness and M
K, M = np.meshgrid(k_grid, m_grid)  # Generate a grid of parameter combinations

# Solve the spring system for each combination of mass and stiffness in the grid
# The spring_solve function is called with each (k, m) pair flattened into a sin
Z_solver = spring_solve(np.array([[M.reshape(-1), K.reshape(-1)]]))  # Solve for
Z_solver = Z_solver.reshape(K.shape)  # Reshape the solution to match the grid s
```

In [8]:
```
fig = utils.plot_3Dscatter(K, M, Z_solver)
```



Model

Choosing the Model

In this notebook, we implement the model using one of the following methods: **gPCE**, **DNN**, or **GBT**.

- **gPCE (Generalized Polynomial Chaos Expansion)**: Models uncertainty by expressing the output as a polynomial expansion, with coefficients determined by projecting the system's response onto orthogonal polynomials.
- **DNN (Deep Neural Networks)**: Machine learning models composed of interconnected neurons, capable of learning complex, non-linear patterns from data.
- **GBT (Gradient Boosting Trees)**: Combines weak decision trees into a strong predictive model, iteratively correcting errors from previous trees for robust performance.

In [9]:
```
method = "gPCE"
```

Set configurations

In [10]:
```
# Model configurations
match method:
    # DNN model configurations
    case "DNN":
        config = {
            'init_config' : {
                'layers': [
                    {'neurons': 512, 'activation': 'relu', 'dropout': 0.2},
                    {'neurons': 256, 'activation': 'sigmoid', 'dropout': 0.2},
                    {'neurons': 256, 'activation': 'sigmoid', 'dropout': 0.2},
                    {'neurons': 128, 'activation': 'relu', 'dropout': None},
                    ],
                'outputAF': 'tanh'
                },
            'train_config' : {
                'optimizer': 'Adam',
                'loss': 'MSE',
                'epochs': 100,
                'batch_size': 32,
                'k_fold': None,
                'early_stopping': {
                    'patience': 25,
                    'min_delta': 0.0001}
                }
                }
    # gPCE model configurations
    case "gPCE":
        config = {
            'init_config' : {
            'p' : 29
            },
            'train_config' : {
                'k_fold': 3
                }
            }
    # GBT model configurations
    case "GBT":
        config = {
            'init_config' : {
                'gbt_method': 'xgboost'
            },
            'train_config' : {
                'max_depth': 6,
                'num_of_iter': 250,
                'k_fold': 9
                }
        }
```

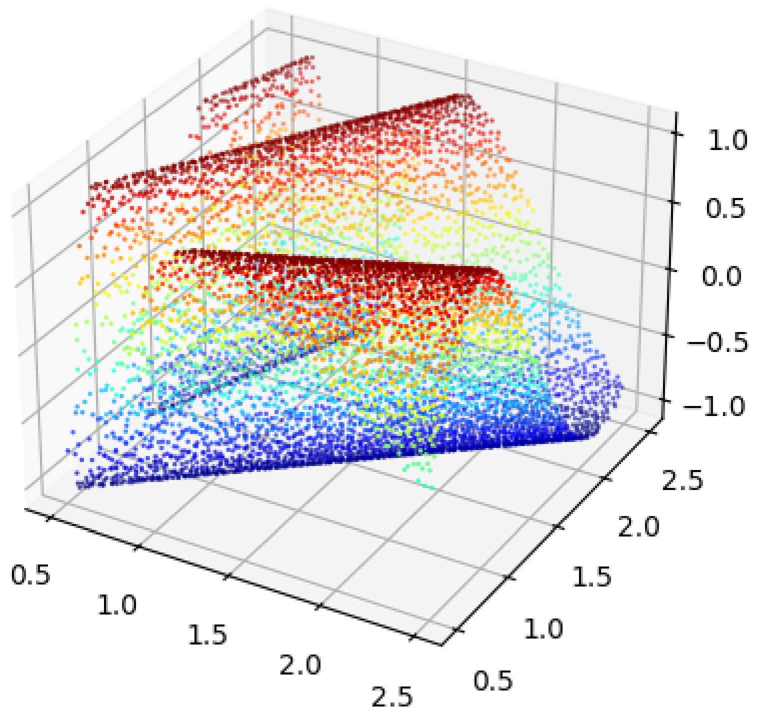
In [11]:
```
split_config = {
        'train_test_ratio': 0.2,
        'random_seed': 1997,
        'split_type': 'no_shuffle'
        }
```

Define model

In [12]:
```
# Initialize surrogate model
# This creates an instance of the SurrogateModel class using the provided sampli
model = SurrogateModel(Q, QoI_names, method, **config['init_config'])
# Split data into training and testing sets
model.train_test_split(Q_df, QoI_df, **split_config)
# Train the model
model.train(model.X_train, model.y_train, **config['train_config'])
```

```
----- Training started for 'gPCE' model -----
Fold 1/3
Fold 2/3
Fold 3/3
Average train loss: 0.00000000000091, Average valid loss: 0.00000000000538
----- Training ended for 'gPCE' model -----
```

In [13]:
```
fig = utils.plot_3Dscatter(model.X_train['m'].to_numpy(), model.X_train['k'].to_
```



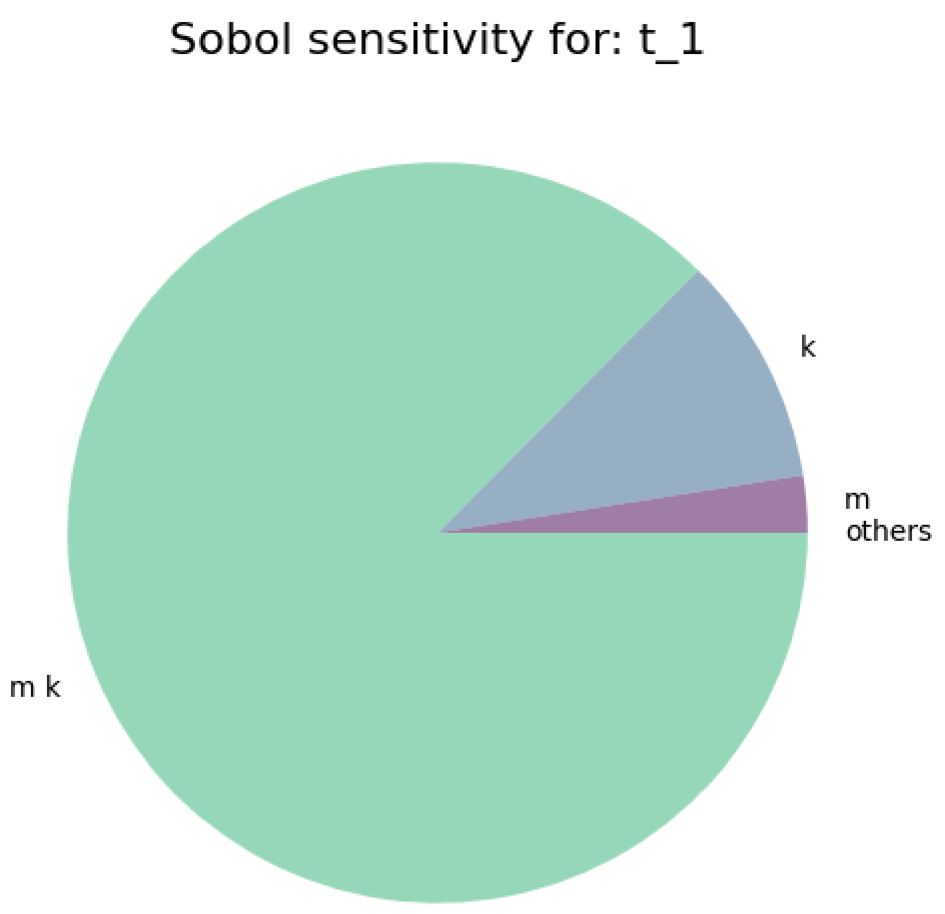
In [14]:
```
# get mean and variance of surrogate model
mean, var = model.get_mean_and_var()
mean, var
```

Out[14]:
```
(tensor([-0.0242], dtype=torch.float64), tensor([0.5085], dtype=torch.float64))
```

Calculate Sobol Sensitivities

In [15]:
```
# set max_index for Sobol sensitivity analysis
max_index = 2

partial_variance, sobol_index = model.get_sobol_sensitivity(max_index)
```

In [16]:
```
# Plot Sobol Sensitivity Index
# The 'plot_sobol_sensitivity' method generates a plot of Sobol Sensitivity indi
# the contribution of each input parameter to the output uncertainty.
# - max_index: Specifies the maximum index of parameters to consider in the plot
# - param_name: The name of the quantity of interest (QoI) for which sensitivity
fig = model.plot_sobol_sensitivity(max_index=max_index, param_name='t_1')
```



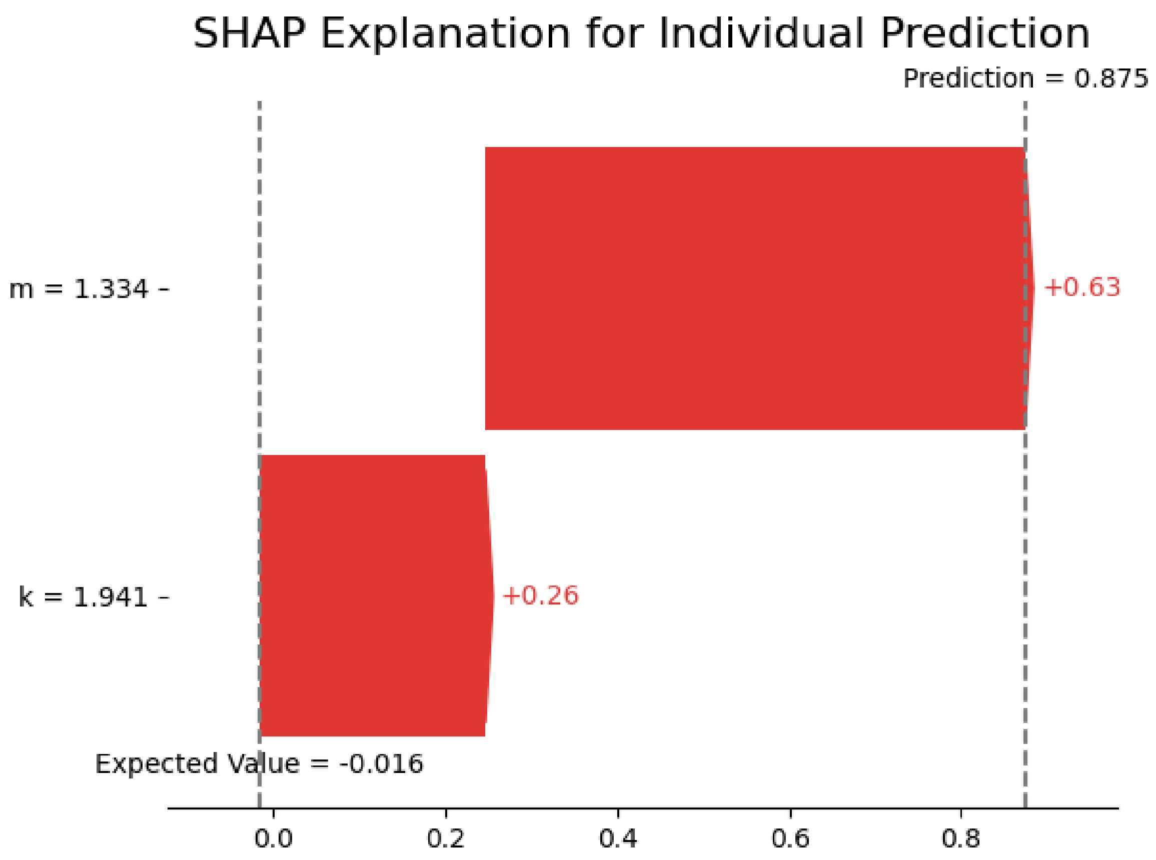
Calculate SHAP values

```
In [17]: # Calculate SHAP (Shapley Additive Explanations) values
         # The 'get_shap_values' method computes the SHAP values for the model's test dat
         # SHAP values explain the contribution of each input feature to the prediction f
         # - model.X_test.iloc[:100]: The first 100 samples from the test set are selecte
         # SHAP values help to interpret the model's decisions and understand the impact
         shap_values = await model.get_shap_values(model.X_test)
```

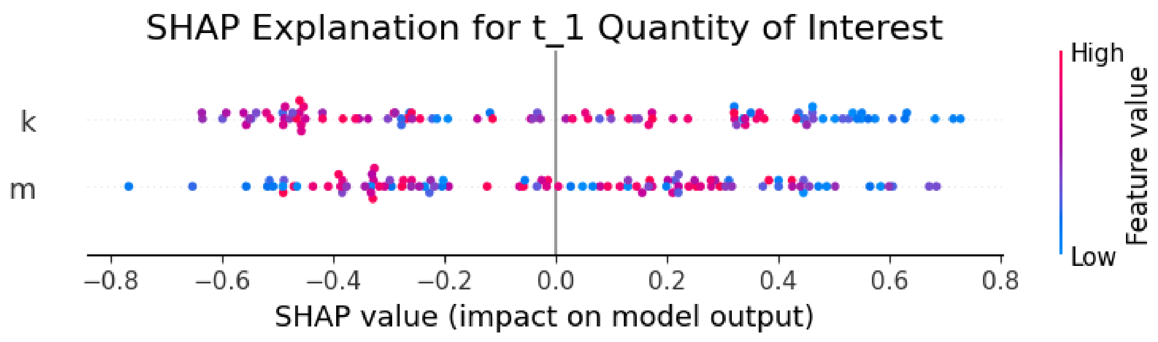
```
Message: sample size for shap values is set to 100.
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
In [18]: # Plot SHAP Single Waterfall Plot
         # The 'plot_shap_single_waterfall' method generates a SHAP waterfall plot for a
         # visualizing how each feature contributes to the model's prediction for that sa
         # - q_sample_df: A DataFrame containing the sample data (e.g., a specific parame
         # - param_name: The name of the quantity of interest (QoI) being analyzed ('t_1'
         # The SHAP waterfall plot shows the cumulative effect of each feature on the mod
         fig = await model.plot_shap_single_waterfall(q=q_sample_df, param_name='t_1')
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```



```
In [19]: # Plot SHAP Beeswarm Plot
         # The 'plot_shap_beeswarm' method generates a SHAP beeswarm plot, which visualiz
         # for a range of test samples, showing the impact of each feature on the model’s
         # - q: The test data (model.X_test.iloc[:100]) is selected for which SHAP values
         # - param_name: The name of the quantity of interest (QoI) for which SHAP values
         # The SHAP beeswarm plot shows how different input features influence the predic
         # helping to understand the global feature importance and the relationships betw
         fig = await model.plot_shap_beeswarm(param_name='t_1')
```



```
In [20]: # Make a prediction using the surrogate model
         # - q_sample_df: A DataFrame containing the input parameter values for which the
         model.predict(q_sample_df)

Out[20]: array([[0.87506346]])
```

Update

Synthetic measurement

```
In [21]: # Generate synthetic measurement data for the uncertain parameters (mass 'm' and
         q_true = Q.sample(1, random_seed=3)  # Sample a single set of uncertain paramete
         qoi_true = spring_solve(q_true)  # Solve the spring system for the synthetic par
         q_true_df = pd.DataFrame(q_true, columns=Q.param_names())  # Convert the synthet

         # Define measurement error
         s = 0.02  # Standard deviation of the measurement noise (2% error)
         sigma = (np.ones(time_steps) * s).reshape(1, -1)  # Create an array of measureme
         sigma = pd.DataFrame(sigma, columns=QoI_names)  # Convert the noise values into
         E = utils.generate_stdrn_simparamset(sigma.values.flatten())  # Generate the sta
         eps = E.sample(1)  # Sample a single set of random noise

         # Synthetic measurement (measurement noise + true value)
         qoi_m = qoi_true + eps  # Add the generated measurement noise to the true value
         qoi_m_df = pd.DataFrame(qoi_m, columns=QoI_names)  # Convert the noisy measureme

         print(q_true, qoi_true)
```

```
[[1.60159581 1.91629565]] [[-0.05711814]]
```

Set parameters

```
In [22]: # For Update
         # The standard deviation of the error is computed using the statistics from the
         sigma = QoI_df.describe().loc['std'] / 30  # Adjust standard deviation for each

         # Generate the simulation parameter set (random noise) based on the updated meas
         E = utils.generate_stdrn_simparamset(sigma.values.flatten())  # Generate random

         # Set up the parameters for the Markov Chain Monte Carlo (MCMC) simulation
         nwalkers = 64  # Number of walkers (parallel chains) in the MCMC process
         nburn = 500  # Number of burn-in steps: these are discarded, as they help the MC
         niter = 100  # Total number of iterations for the MCMC process
```

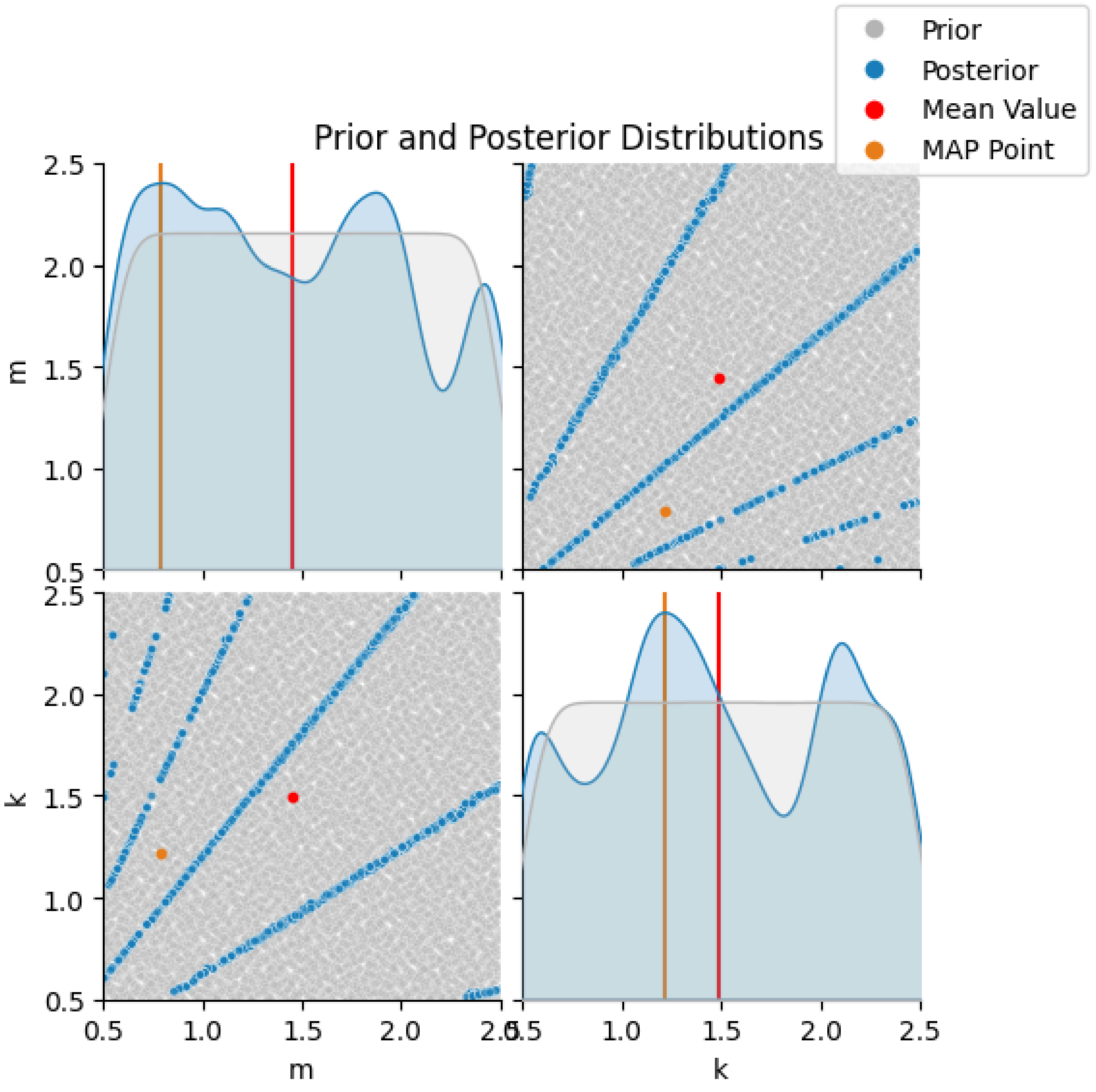
```
In [23]: # Initialize the Digital Twin model with the surrogate model and simulation para
         DT = DigitalTwin(model, E)  # Create a DigitalTwin instance that integrates the
```

```
In [24]: # Update the Digital Twin model with the synthetic measurement data and run MCMC
         DT.update(qoi_m_df, nwalkers=nwalkers, nburn=nburn, niter=niter)
         # The 'update' method updates the Digital Twin using the synthetic measurement d
         # and performs MCMC sampling with the specified parameters (number of walkers, b
```

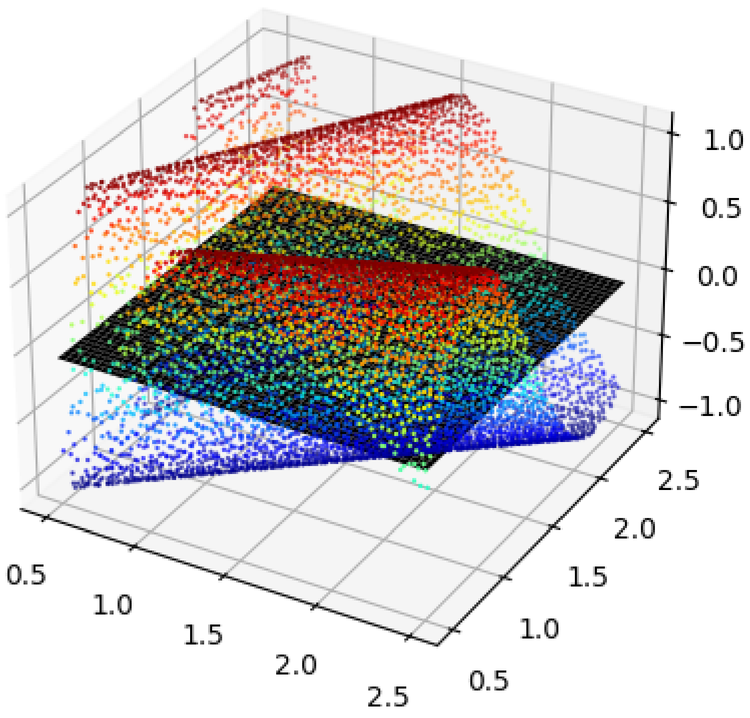
```
MCMC creating
Burning period
100%|██████████| 500/500 [00:38<00:00, 12.85it/s]
MCMC running
100%|██████████| 100/100 [00:06<00:00, 15.29it/s]
--- 45.52680158615112 seconds ---
```

```
In [25]: # Get the posterior mean and variance from the Digital Twin model
         mean_of_post, variance_of_post = DT.get_mean_and_var_of_posterior()
         # This returns the mean and variance of the posterior distribution after updatin

         # Get the Maximum A Posteriori (MAP) estimate of the parameters
         map = DT.get_MAP()
         # This retrieves the most likely parameter values based on the updated model.
```

```
In [26]: # Plot the MCMC sampling results along with the mean and MAP and true parameter
         fig = utils.plot_MCMC(model, DT, nwalkers=nwalkers, map_point=map)
```



```
In [27]: fig = utils.plot_3Dscatter(model.X_train['m'].to_numpy(), model.X_train['k'].to_
```


1.0
0.5
0.0
−0.5
−1.0
0.5
1.0
1.5
2.0
2.5
0.5
1.0
1.5
2.0
2.5

C.2.2 Digital Twin of the Yoker Building

https://buildchain.ilab.sztaki.hu/notebooks/1_hybrid_digital_twinning/hybrid_Yoker.html

C.3 Module 3: Multi Building Update with the python library package

C.3.1 Toy Example: Coupled Calibration of Two 1D Oscillator Experiments

Multi-Building Design Update of two Spring-Mass Systems

This Jupyter Notebook demonstrates how to use the joint **Multi-Model Update** with the example of two, previously trained 1D Spring-Mass Hybrid Models (see the notebook on Hybrid Digital Twinning here).

The main objective of the Multi-Building Updating is to calibrate uncertain parameters of simulation models across multiple buildings or experimental setups using shared measurement data. This ensemble-based calibration improves generalizability and robustness compared to single-structure updates.

System Overview

Both spring-mass systems follow Hooke’s Law and Newton’s Second Law:

$$F = -kx$$

where:

- (k) is the **spring stiffness**,
- (x) is the **displacement from equilibrium**,
- (m) is the **mass of the attached object**.

Analytical Solution

For an ideal undamped system, the displacement at a given time step is given by:

$$u(T) = u_0 \cos\left(\sqrt{\frac{k}{m}}T\right) + \frac{v_0}{\sqrt{\frac{k}{m}}}\sin\left(\sqrt{\frac{k}{m}}T\right)$$

where:

- (u_0) is the **initial displacement**,
- (v_0) is the **initial velocity**,
- (k) and (m) can change dynamically in the twinning process,
- **(T = 10) is the single time step at which the system is evaluated**.

Imports

In [1]:
```
import sys
import os
parent_dir = os.path.abspath("../../libraries/surrogate_modelling")
sys.path.insert(0, parent_dir)
```

In [2]:
```
import pickle
import pandas as pd
import utils
from multibuilding_update import JointManager

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Load models

In this example we simply load two previusly trained hybrid models of the two system. The chosen parameters of the models were:

- (m1 and m2) **mass of the attached object** and
- (k1 and k2) **spring stiffness**.

The models are connected through their **m** parameter. All parameters are defined with uniform distributions, both **m1** and **m2** between bounds [0.5, 2.5], while **k1** and **k2** between [1.73, 2.11] and [1.52, 2.0]

In [ ]:
```
#Load measurement data
z_spring1 = pd.read_csv('../data/spring_data/z_spring1_df.csv')
z_spring2 = pd.read_csv('../data/spring_data/z_spring2_df.csv')

#Load standard deviation of the measurement error
sigma_spring1 = pd.read_csv('../data/spring_data/sigma_spring1.csv')
sigma_spring2 = pd.read_csv('../data/spring_data/sigma_spring2.csv')

#Load surrogate models
with open('../data/spring_data/Spring1_model.sm', 'rb') as file:
    Spring1_model = pickle.load(file)
with open('../data/spring_data/Spring2_model.sm', 'rb') as file:
    Spring2_model = pickle.load(file)
```

Update

In [ ]:
```
#Define joint parameters
joint_parameters = {'m': ['m1', 'm2']}

#Define the joint manager
j = JointManager([Spring1_model, Spring2_model], joint_parameters)
print(j.Q.params)
```

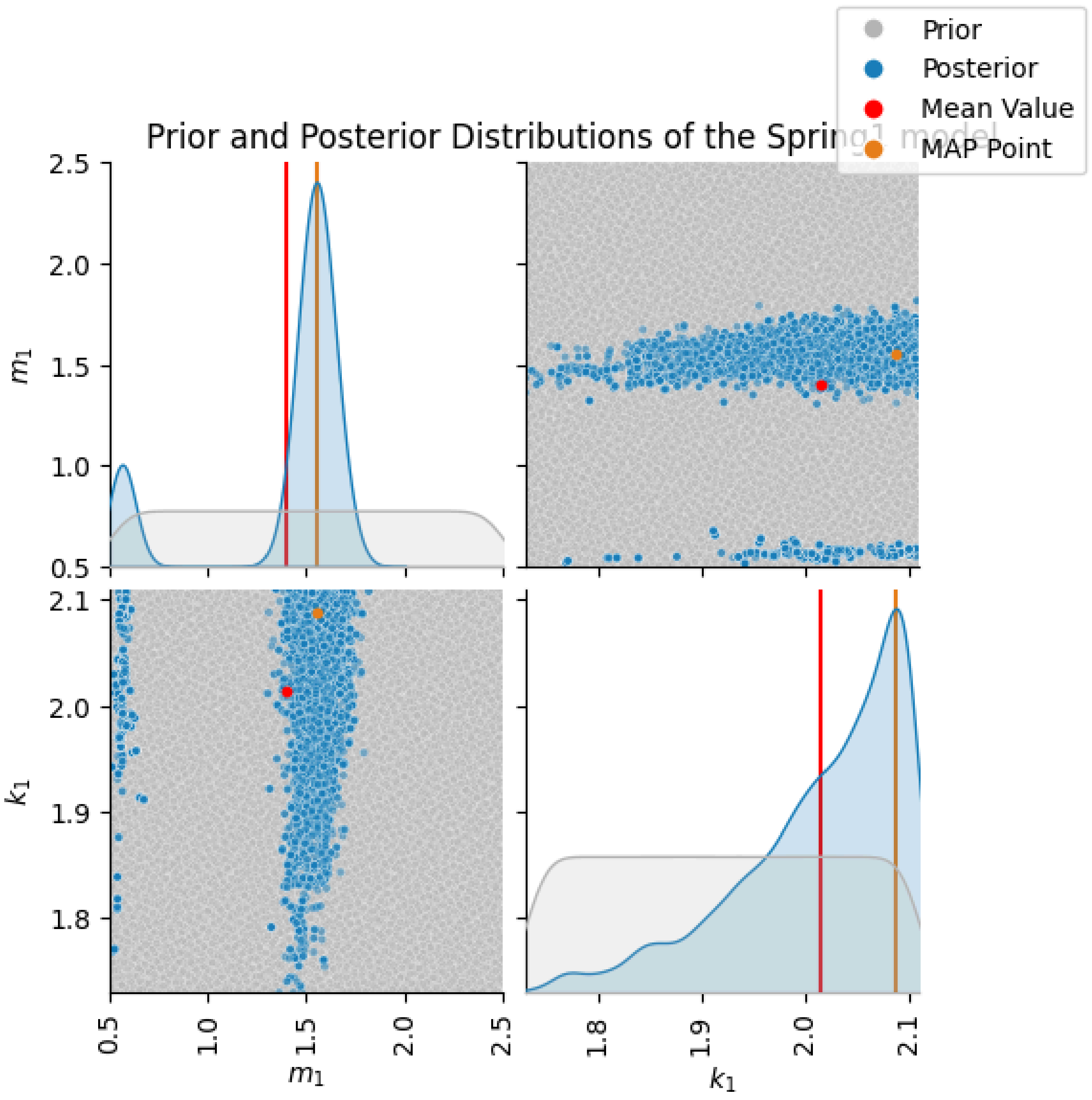
```
{'m': U(0, 1), 'k1': U(1.73, 2.11), 'k2': U(1.52, 2.0)}
```

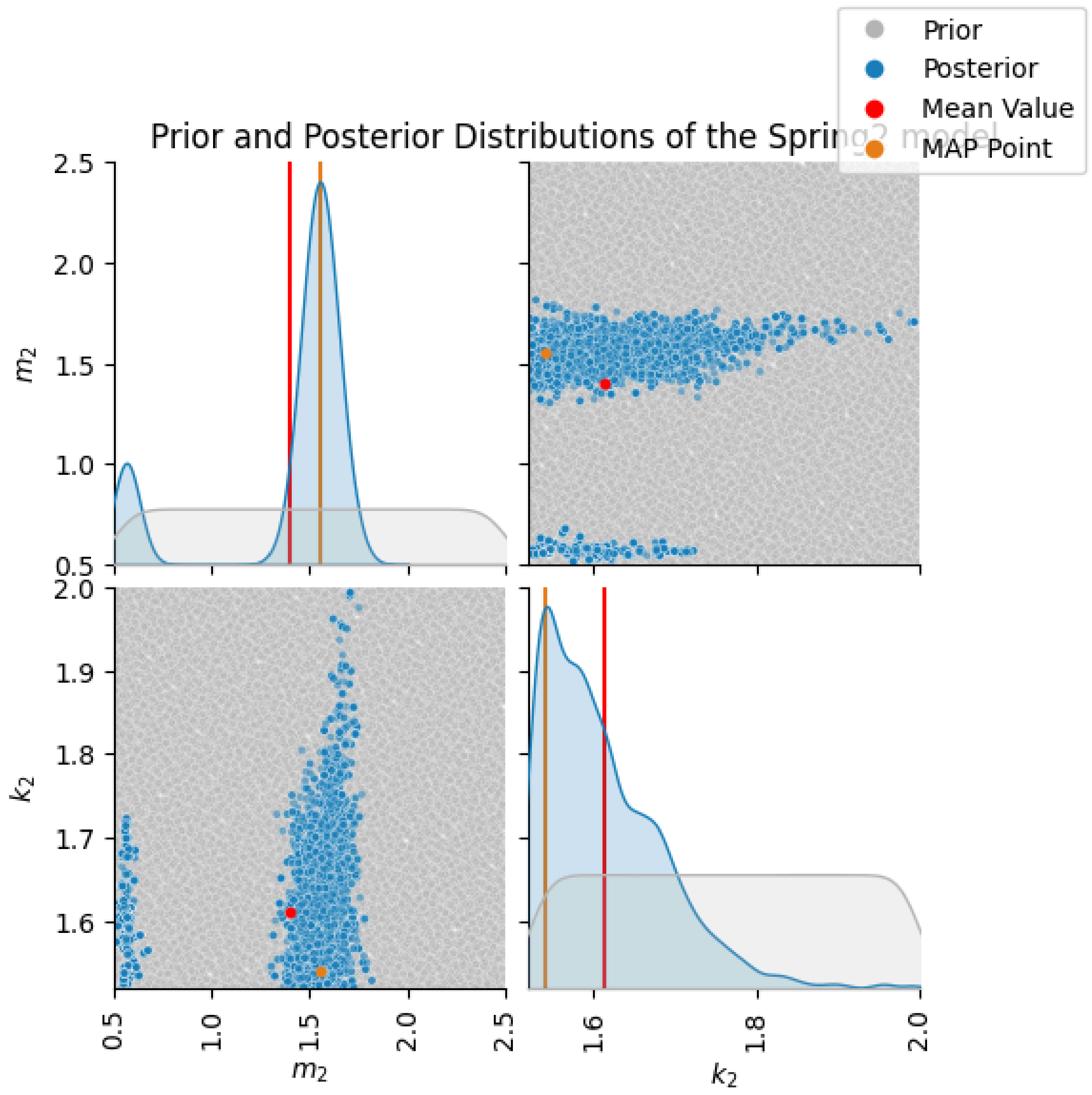
In [ ]:
```
#Update inputs
nwalkers=32
niter=200
nburn=1000

#Update
j.update([z_spring1, z_spring2], [sigma_spring1, sigma_spring2], nwalkers=nwalke
```

```
MCMC creating
Burning period
```

```
100%|██████████| 1000/1000 [04:06<00:00,  4.05it/s]
MCMC running
100%|██████████| 200/200 [00:55<00:00,  3.60it/s]
--- 302.52466464042664 seconds ---
```

```
In [ ]: # Plot the MCMC sampling results along with the mean and MAP points for each mod
        figs = utils.plot_multibuilding_MCMC(j, map_point=True, model_names=['Spring1',
```





Prior and Posterior Distributions of the Spring2 model
Prior
Posterior
Mean Value
MAP Point
m_2
k_2
0.5
1.0
1.5
2.0
2.5
1.6
1.7
1.8
1.9

C.3.2 Pilot 5: Design Parameter Updating based on measurements of two timber buildings

https://buildchain.ilab.sztaki.hu/notebooks/3_multi_building_design_updating/multimodel_update_Yoker_Palisaden.html

C.4 Module 4: Earthquake Resilience with the python library package

C.4.1 Toy Example, a Low Complexity Benchmark Building

Earthquake Resilience with low complexity building

This Jupyter Notebook demonstrates the functions of the **Assessment of Earthquake Resilience** module with the example of a low complexity, three-storey building.

The module implements linear and non-linear computations and visualizations for analyzing the earthquake resilience of a given building.

Imports

In [1]:
```
import sys
import os
parent_dir = os.path.abspath("../../libraries/earthquake_resilience")
sys.path.insert(0, parent_dir)

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [2]:
```
import eq_resilience_library
import pandas as pd
```

Data inputs

The following user inputs are needed for the calculations:

- latitude and longitude of the building (in decimal degrees)
- soil category - ranges from hard rock (`A`) to very soft or problematic soils (`E`)
- nominal life - the intended design service life of the structure (in years)
- topographic category - the influence of terrain features (such as hills and ridges) on wind or seismic effects. Ranges from `T1` (no inpact, flat terrain) to `T4` (upper half of a ridge, maximum amplification)
- importance class - the criticality of a structure, influences the level of seismic safety required
    - Class `I` : Low importance (e. g., agricultural or temporary building)
    - Class `II` : Ordinary structure (standard residential, commercial or office building)
    - Class `III` : Building with high occupancy or social importance (e.g., school, assembly hall)
    - Class `IV` : Essential facility and structure of vital importance for civil protection, which must remain operational after an earthquake (e.g., hospital, emergency center)
- `.xls` or `.xlsx` file with wall properties of the building in a predefined format

```
In [3]: latitude = 43.7874
        longitude = 11.2499

        soil_category = 'A'
        nominal_life = 50
        topographic_category = 'T1'
        importance_class = 'I'

        file_path = "../../demo/data/EQ_resilience/EPUSH_new_format.xlsx"
        file = pd.ExcelFile(file_path)
```

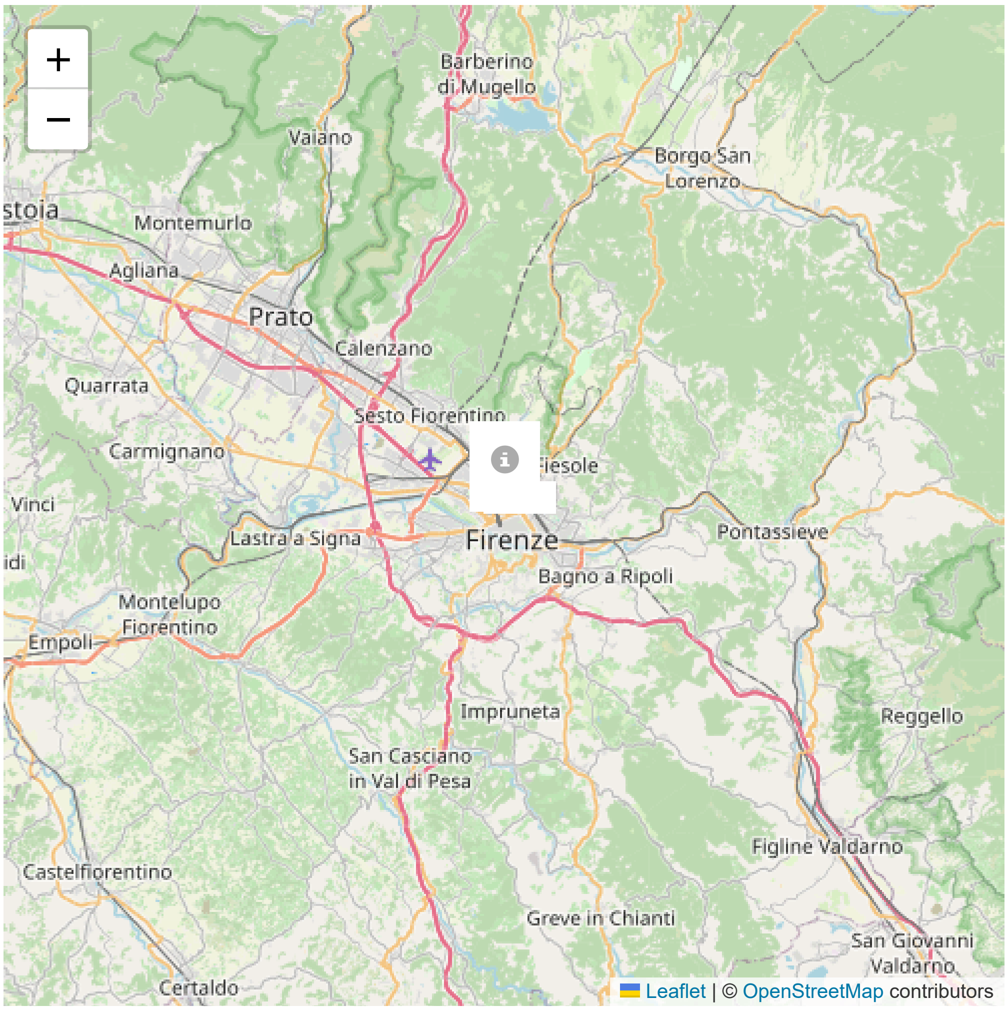
Coordinate check

```
In [4]: ParaTR, city, country = eq_resilience_library.get_Parameters(latitude, longitude
        print("{}, {}".format(city, country))
        if ParaTR is None:
            print("Earthquake hazard calculation is not supported in the selected region
```

```
Loading formatted geocoded file...
Florence, IT
```

```
In [5]: f = eq_resilience_library.get_map(latitude, longitude, city, country)
        f
```

Out[5]:



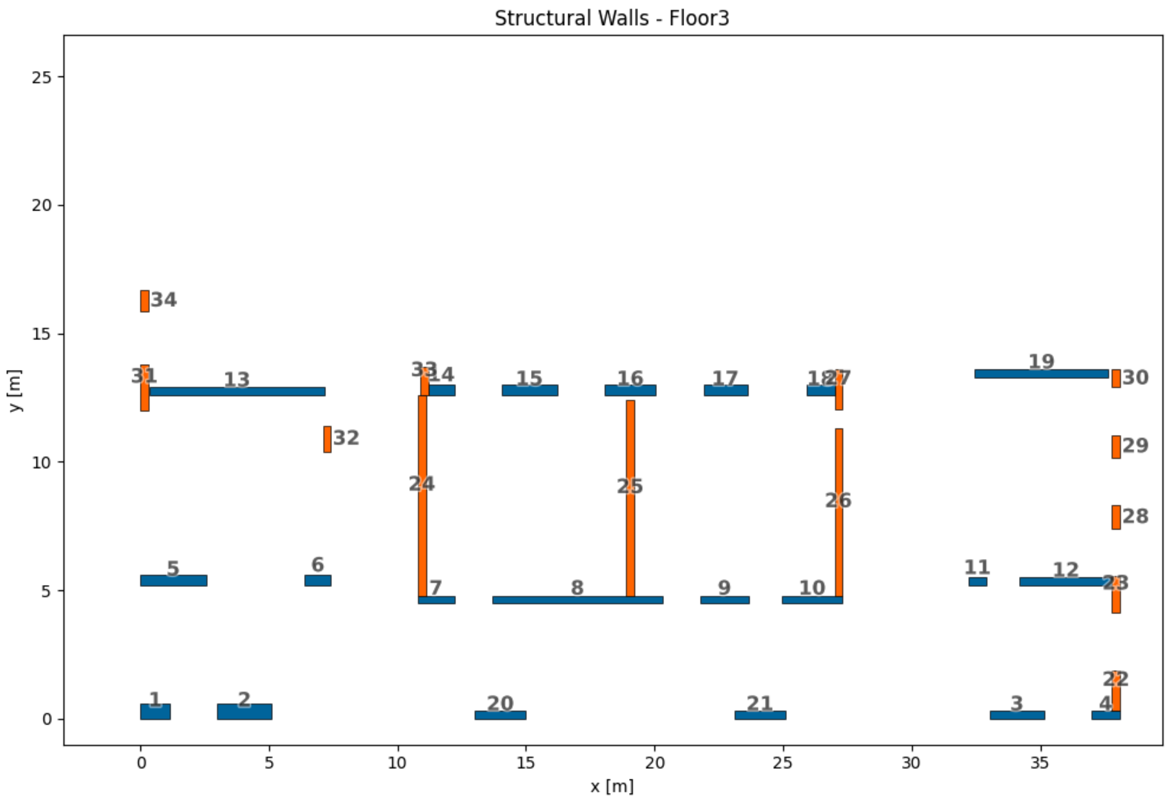
```
In [6]: ParaTR
```

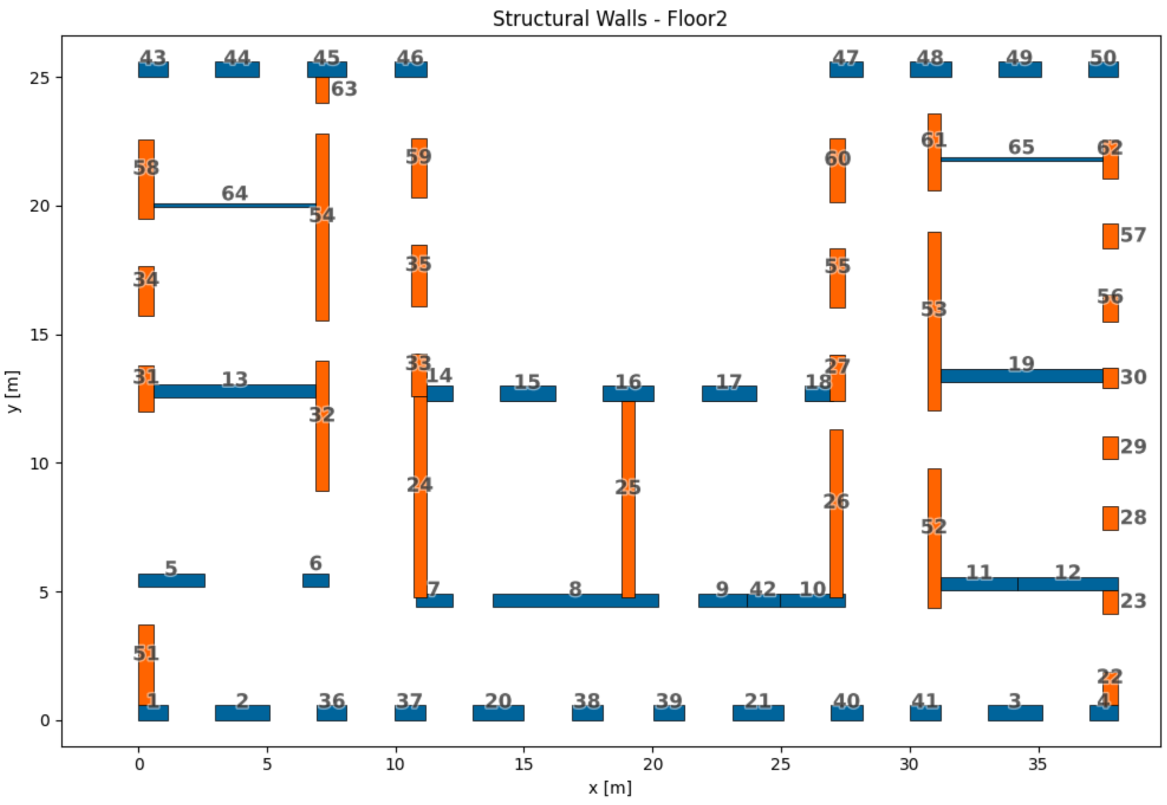
Out[6]:

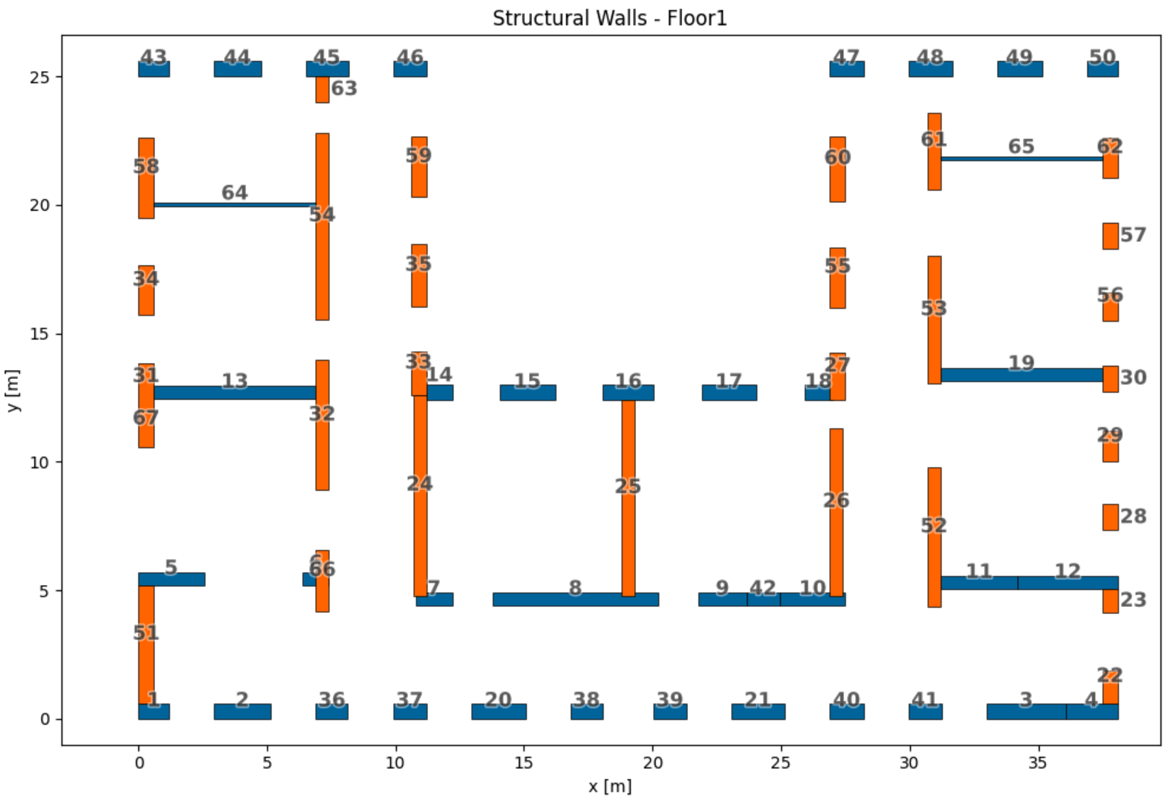
	Return Period	ag	Fo	Tc*
0	30.0	0.047018	2.551014	0.252808
1	50.0	0.056464	2.586081	0.267521
2	72.0	0.064100	2.593636	0.276378
3	101.0	0.072345	2.590411	0.281871
4	140.0	0.080685	2.597692	0.287275
5	201.0	0.093504	2.531818	0.293817
6	475.0	0.131090	2.420610	0.301819
7	975.0	0.167652	2.390029	0.310319
8	2475.0	0.221719	2.411010	0.318801

2D layout

In [7]:
```
figures = eq_resilience_library.get_2D_layout(file)
```







Linear analysis calculation

In [8]:
```
behaviour_factor_q = 1.5

IR_fess_X, IR_fess_Y, IR_pf_X, IR_pf_Y, IR_pf_ort_X, IR_pf_ort_Y = eq_resilience
```

In [9]:
```
direction = 'Y'
chosen_floor = 1
```

```
linear_dataframes = eq_resilience_library.get_linear_dataframes(direction, IR_fe
linear_dataframes
```

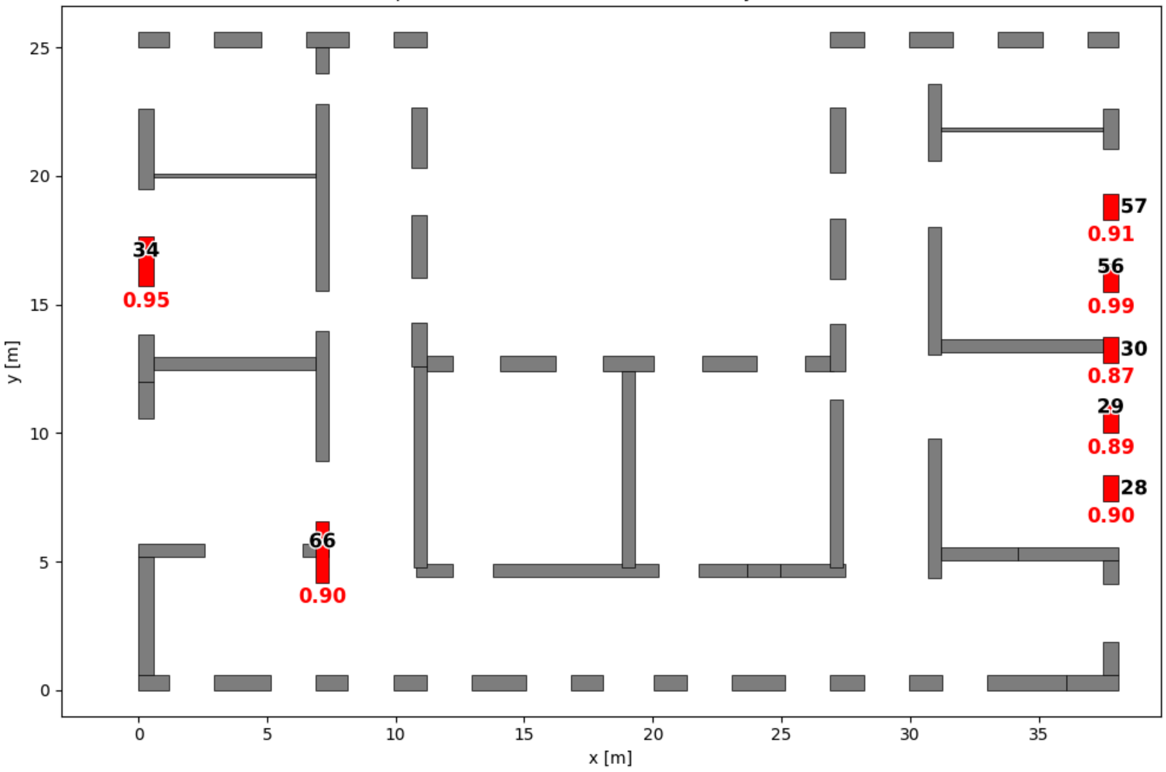
Out[9]:

Direction Y, floor 1	Wall ID	Shear	Bending in plane	Bending out of plane
0	21.0	1.18	1.05	1.85
1	22.0	1.11	0.69	1.75
2	23.0	1.21	6.41	1.57
3	24.0	1.25	6.43	1.61
4	25.0	1.09	4.90	1.44
5	26.0	1.00	1.27	1.57
6	27.0	0.90	0.59	1.36
7	28.0	0.89	0.70	1.34
8	29.0	0.87	0.56	1.29
9	30.0	1.04	1.32	1.63
10	31.0	1.03	3.56	1.34
11	32.0	1.01	1.18	1.59
12	33.0	0.95	1.24	1.46
13	34.0	1.06	1.75	1.67
14	50.0	1.24	3.88	1.93
15	51.0	1.11	4.11	1.45
16	52.0	1.21	4.09	1.57
17	53.0	1.17	5.79	1.52
18	54.0	1.22	1.95	1.90
19	55.0	0.99	0.74	1.55
20	56.0	0.91	0.60	1.37
21	57.0	1.15	2.45	1.81
22	58.0	1.22	1.95	1.90
23	59.0	1.23	2.09	1.91
24	60.0	1.08	2.22	1.41
25	61.0	1.01	1.07	1.58
26	62.0	1.09	0.75	1.43
27	65.0	0.90	1.39	1.13
28	66.0	1.30	1.26	1.99

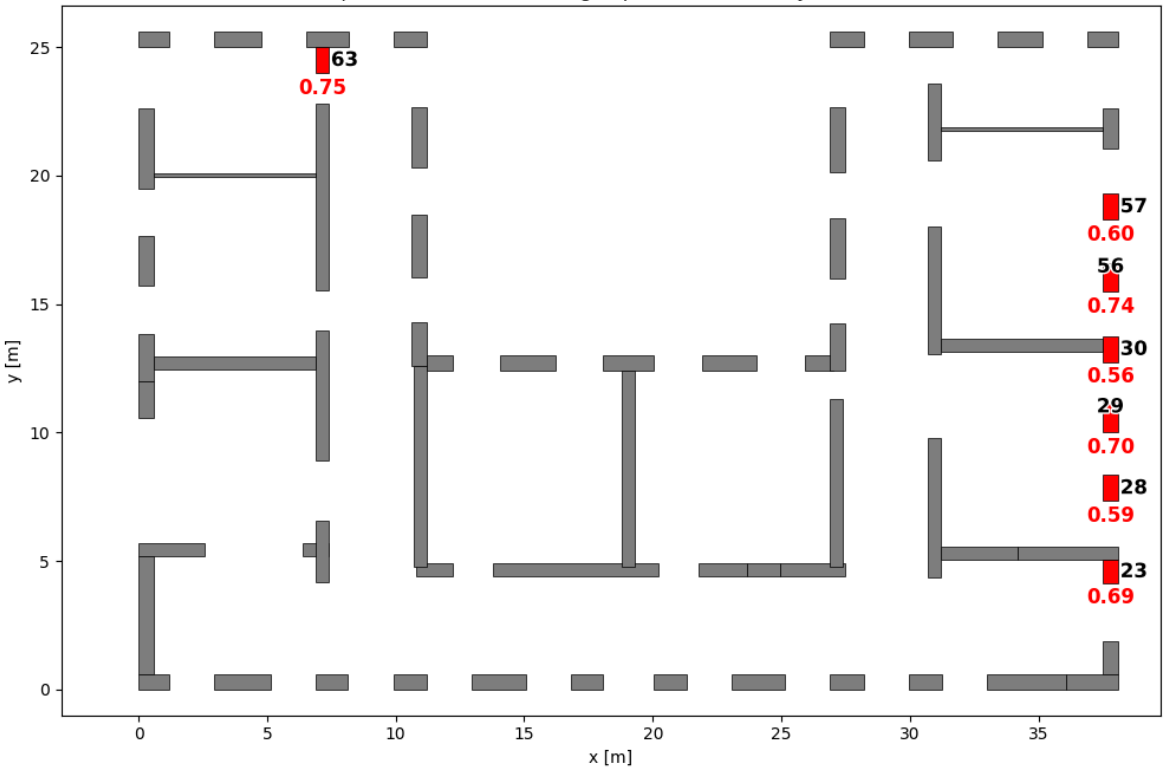
In [10]:

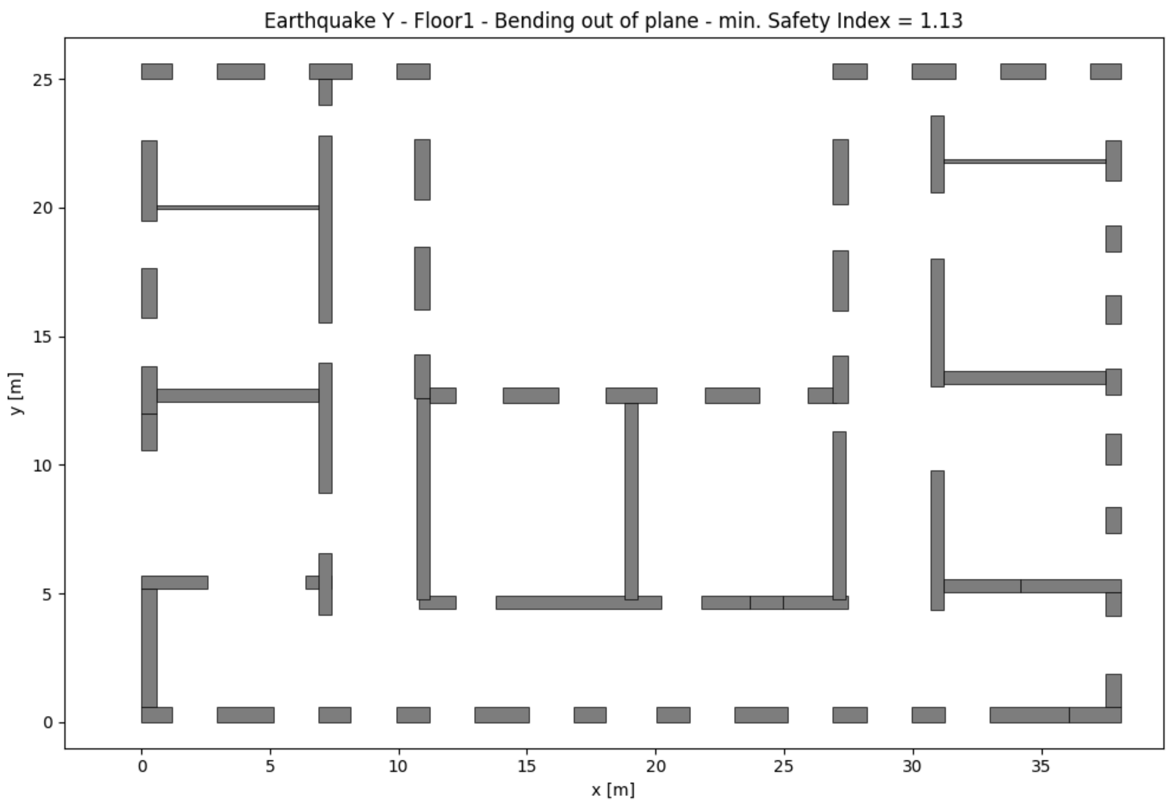
```
linear_analysis_figures = eq_resilience_library.choose_linear_analysis_plot(file
```

Earthquake Y - Floor1 - Shear - min. Safety Index = 0.87



Earthquake Y - Floor1 - Bending in plane - min. Safety Index = 0.56



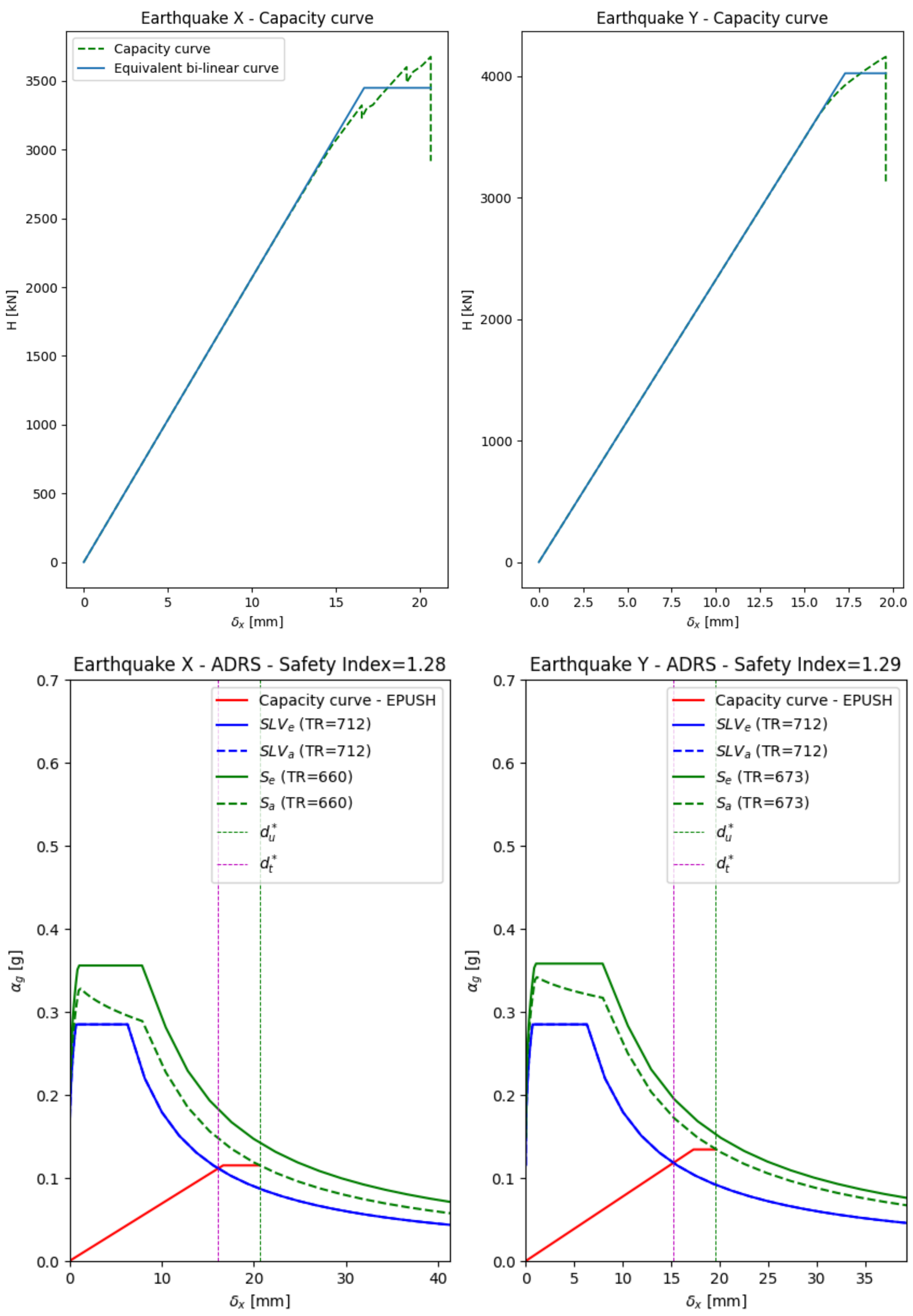


Pushover analysis calculation

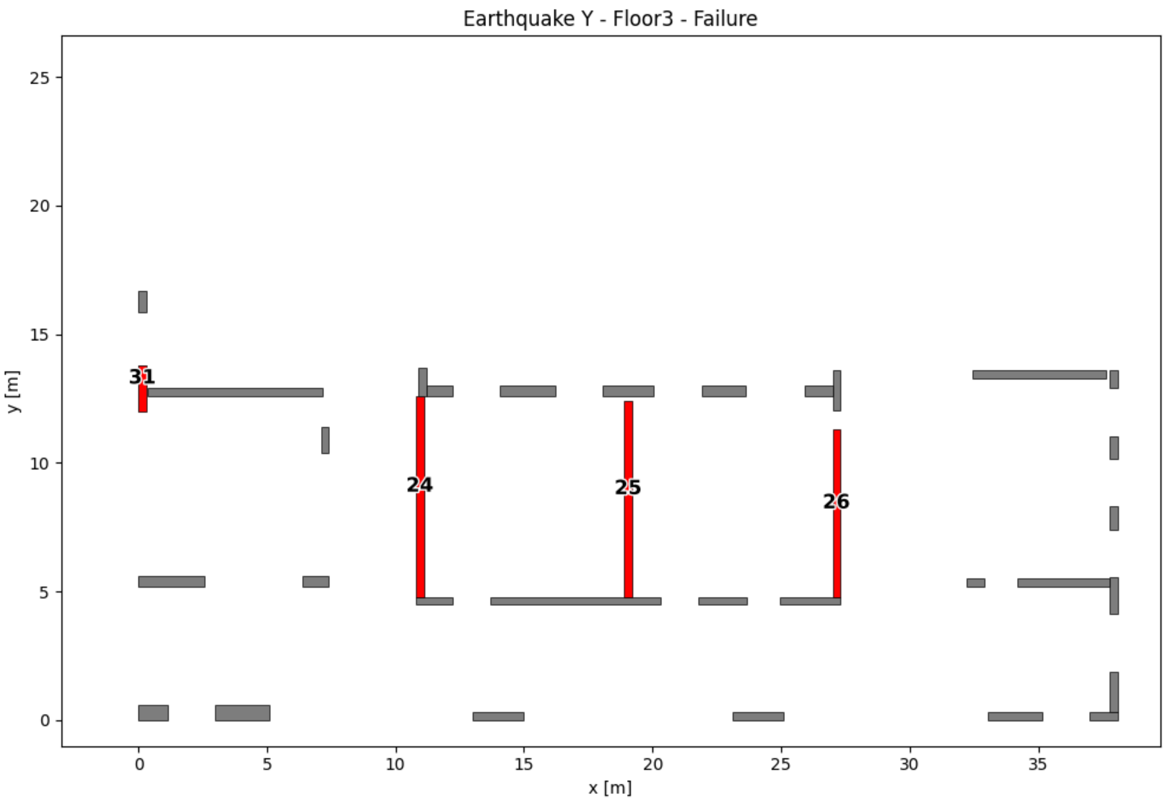
In [11]:
```
check_type = 'Ductility check'

L, pushover_dataframe, figures = eq_resilience_library.pushover_analysis(file, P
pushover_dataframe
```

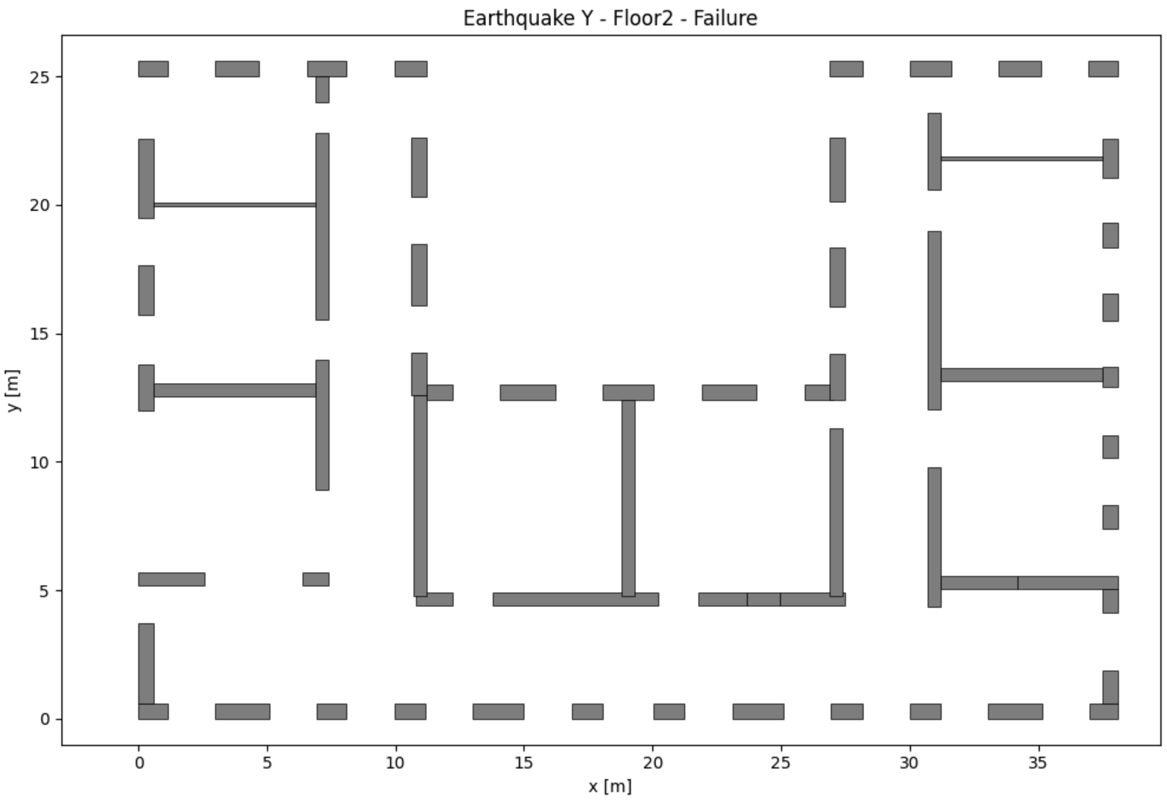
Out[11]:

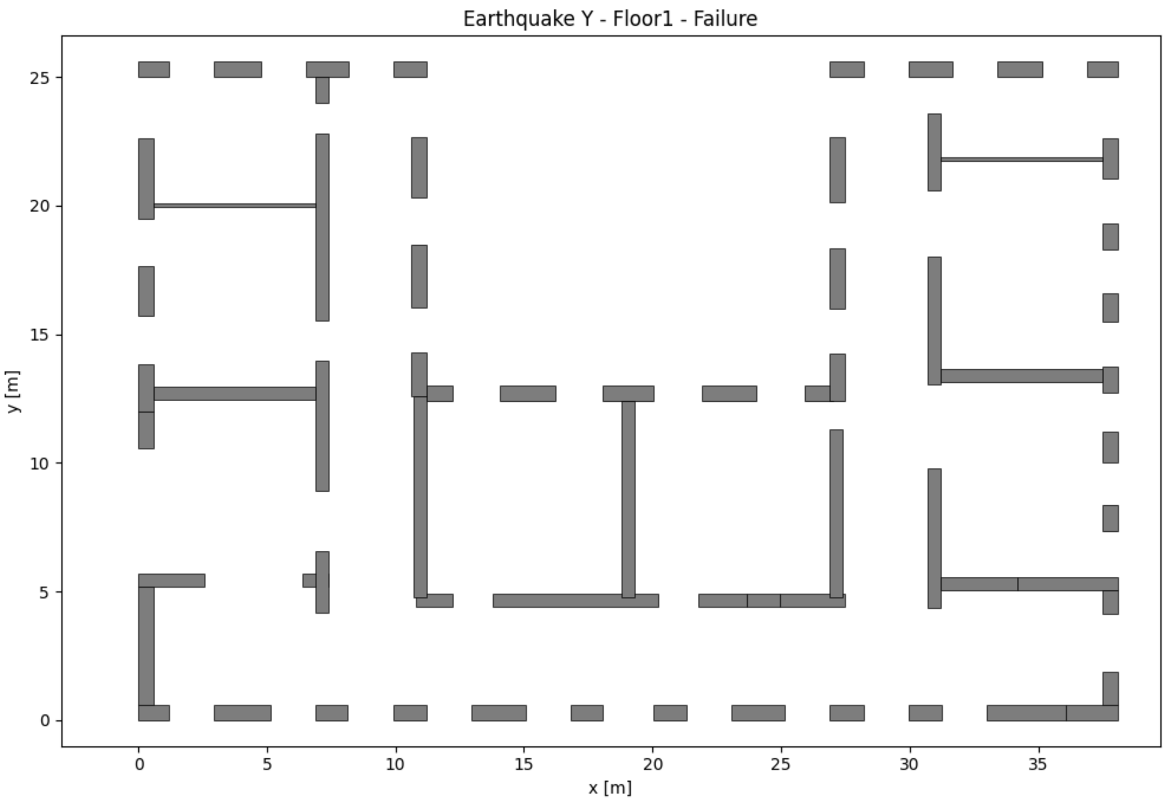
	Direction	Safety Index	PGA_C	TR	δ	d* (t)
0	X	1.28	0.15	660.0	16.68	16.13
1	Y	1.29	0.15	673.0	17.29	15.21



In [12]:
```
direction = 'Y'

figures = eq_resilience_library.plot_wall_failures(file, direction, L)
```


Earthquake Y - Floor3 - Failure
31
24
25
26
y [m]
x [m]
0
5
10
15
20
25
30
35


Earthquake Y - Floor2 - Failure
y [m]
x [m]
0
5
10
15
20
25
30
35


Earthquake Y - Floor1 - Failure
y [m]
x [m]
0
5
10
15
20
25
30
35

C.4.2 Pilot 4: Earthquake Resilience of Palazzo Poniatowski Guadagni

https://buildchain.ilab.sztaki.hu/notebooks/4_earthquake_resilience/earthquake_resilience_pilot_building.html

C.5 Module 5: Climate Effect Analysis

Climate Resilience in Florence

Imports

In [1]:
```
import sys
import os
parent_dir = os.path.abspath("../../libraries/climate_resilience")
sys.path.insert(0, parent_dir)

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [2]:
```
import climate_resilience as cr
```

User inputs

In [3]:
```
api_key = 'ffd7d363-d7bd-4314-acc7-e88827069c50'

latitude = 43.7697
longitude = 11.2558
```

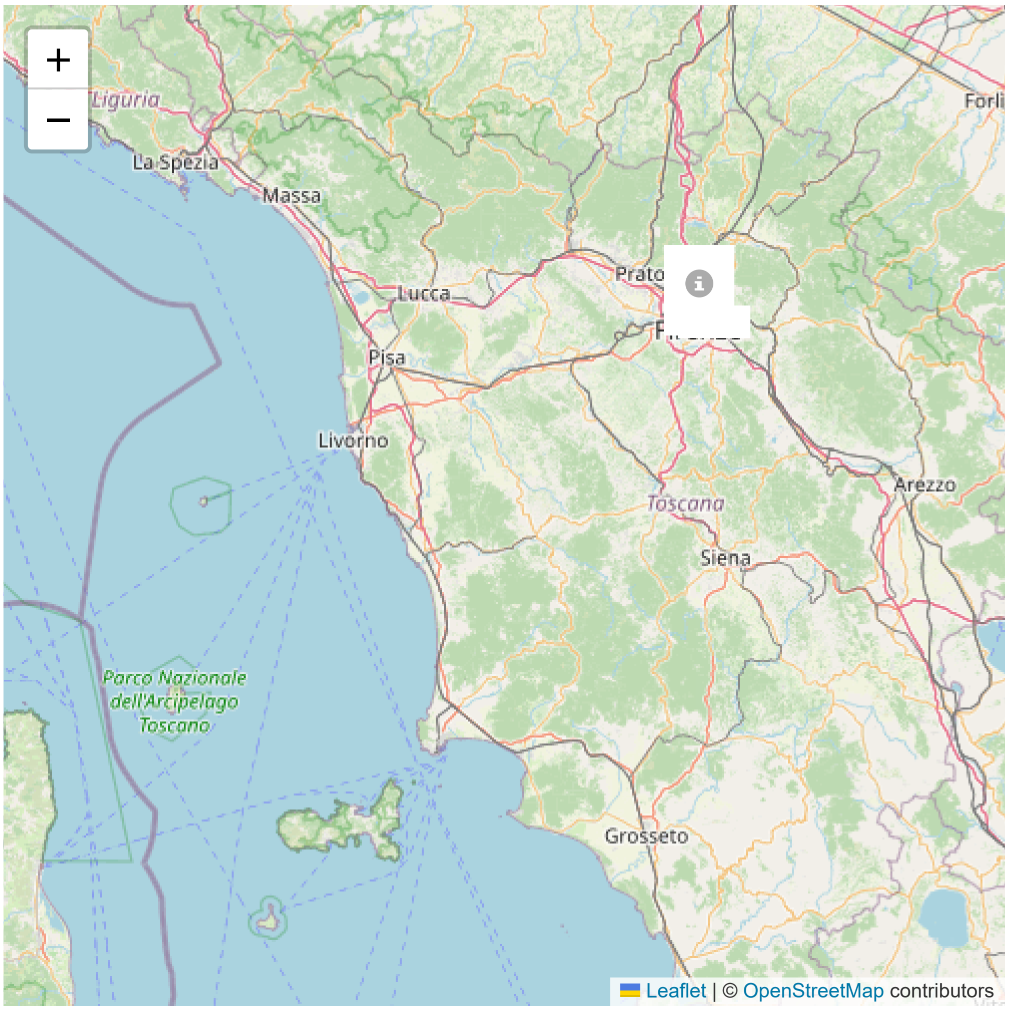
Coordinate check

In [4]:
```
city, country = cr.coordinate_check(latitude, longitude)
print("{}, {}".format(city, country))
```

```
Loading formatted geocoded file...
Florence, IT
```

In [5]:
```
cr.get_map(latitude, longitude, city, country)
```

Out[5]:


Liguria
La Spezia
Massa
Lucca
Prato
Pisa
Livorno
Toscana
Siena
Arezzo
Forlì
Parco Nazionale dell'Arcipelago Toscano
Grosseto
Leaflet | © OpenStreetMap contributors

Data request

```
In [6]: experiment_option = "Historical + RCP 4.5"
        variable_option = "Maximum Temperature"
        model_name = 'IRNET'
```

```
In [7]: model_data = cr.request_data(api_key, model_name, experiment_option, variable_op
```

```
2025-07-30 22:35:38,491 INFO [2024-09-26T00:00:00] Watch our [Forum](https://foru
m.ecmwf.int/) for Announcements, news and other discussed topics.
2025-07-30 22:35:38,667 INFO [2024-09-26T00:00:00] Watch our [Forum](https://foru
m.ecmwf.int/) for Announcements, news and other discussed topics.
2025-07-30 22:35:42,501 INFO Request ID is 9f272b8a-df50-47ed-9840-b7da68bb25b9
2025-07-30 22:35:42,574 INFO status has been updated to accepted
2025-07-30 22:35:51,261 INFO status has been updated to running
2025-07-30 22:35:56,371 INFO status has been updated to successful
2a99ed52dd64827fe9731e33c6dcf74a.zip:   0%|          | 0.00/2.22M [00:00<?, ?B/s]
  0%|          | 0/1 [00:00<?, ?it/s]
Dataset 'IRNET_orography' downloaded and added to dictionary
Data for 'IRNET_orography':
```

	rlon	rlat	rotated_pole	lon	lat	orog
0	-28.375	-23.375	b''	-10.063880	21.987829	265.612701
1	-28.375	-23.265	b''	-10.110996	22.088787	269.615082
2	-28.375	-23.155	b''	-10.158180	22.189732	274.739166
3	-28.375	-23.045	b''	-10.205431	22.290664	281.163361
4	-28.375	-22.935	b''	-10.252751	22.391581	289.754486

```
2025-07-30 22:36:05,081 INFO [2024-09-26T00:00:00] Watch our [Forum](https://foru
m.ecmwf.int/) for Announcements, news and other discussed topics.
==================================================


Processing city: Florence (Lat: 43.7697, Lon: 11.2558)
  Closest Point in IRNET_orography:
  Latitude: 43.757510, Longitude: 11.193669
  rlat: -6.765000, rlon: -4.945000
  Area area: [-6.765, -4.945, -6.765, -4.945]
coordinates:  {'area': [-6.765, -4.945, -6.765, -4.945]}
IRNET
2025-07-30 22:36:05,202 INFO [2024-09-26T00:00:00] Watch our [Forum](https://foru
m.ecmwf.int/) for Announcements, news and other discussed topics.
2025-07-30 22:36:08,961 INFO Request ID is c7a2104c-b1b2-4634-add2-328f660665a2
2025-07-30 22:36:09,048 INFO status has been updated to accepted
2025-07-30 22:36:17,690 INFO status has been updated to running
2025-07-30 22:36:22,848 INFO status has been updated to successful
2c35a538f5515a12d301fa7d00f48f7c.zip:   0%|          | 0.00/11.0M [00:00<?, ?B/s]
  0%|          | 0/31 [00:00<?, ?it/s]
```

```
Dataset 'IRNET' downloaded and added to dictionary
{'IRNET':                         time  bnds   rlat   rlon rotated_pole  time_bnds  
\
0       1950-01-01 12:00:00     0 -6.765 -4.945           b''  1950-01-01   
1       1950-01-01 12:00:00     1 -6.765 -4.945           b''  1950-01-02   
2       1950-01-02 12:00:00     0 -6.765 -4.945           b''  1950-01-02   
3       1950-01-02 12:00:00     1 -6.765 -4.945           b''  1950-01-03   
4       1950-01-03 12:00:00     0 -6.765 -4.945           b''  1950-01-03   
...                     ...   ...    ...    ...           ...         ...   
110299 2100-12-29 12:00:00     1 -6.765 -4.945           b''  2100-12-30   
110300 2100-12-30 12:00:00     0 -6.765 -4.945           b''  2100-12-30   
110301 2100-12-30 12:00:00     1 -6.765 -4.945           b''  2100-12-31   
110302 2100-12-31 12:00:00     0 -6.765 -4.945           b''  2100-12-31   
110303 2100-12-31 12:00:00     1 -6.765 -4.945           b''  2101-01-01   

            tasmax        lon        lat  height  
0       280.913208  11.193669  43.75751     2.0  
1       280.913208  11.193669  43.75751     2.0  
2       282.695435  11.193669  43.75751     2.0  
3       282.695435  11.193669  43.75751     2.0  
4       282.809937  11.193669  43.75751     2.0  
...            ...        ...        ...     ...  
110299  283.110413  11.193669  43.75751     2.0  
110300  283.725037  11.193669  43.75751     2.0  
110301  283.725037  11.193669  43.75751     2.0  
110302  285.548767  11.193669  43.75751     2.0  
110303  285.548767  11.193669  43.75751     2.0  

[110304 rows x 10 columns]}
```

Data processing

In [8]:
```
window_length=40
shift=10
```

In [9]:
```
dfs_processed, windows = cr.processed_dataframe(model_data, window_length=window
```

```
Dataset: IRNET, Window: 1950-1989
Dataset: IRNET, Window: 1960-1999
Dataset: IRNET, Window: 1970-2009
Dataset: IRNET, Window: 1980-2019
Dataset: IRNET, Window: 1990-2029
Dataset: IRNET, Window: 2000-2039
Dataset: IRNET, Window: 2010-2049
Dataset: IRNET, Window: 2020-2059
Dataset: IRNET, Window: 2030-2069
Dataset: IRNET, Window: 2040-2079
Dataset: IRNET, Window: 2050-2089
Dataset: IRNET, Window: 2060-2099
```

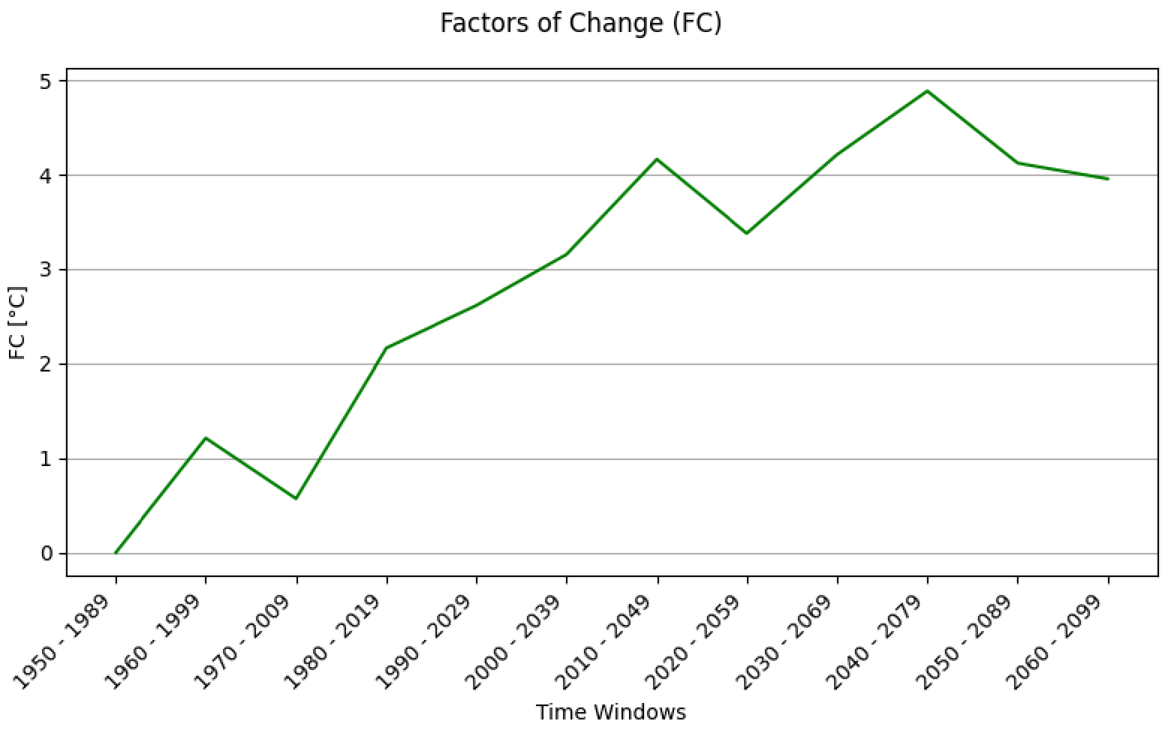
Calculating result values

In [10]:
```
T_r = 50
method = "MLM"
```

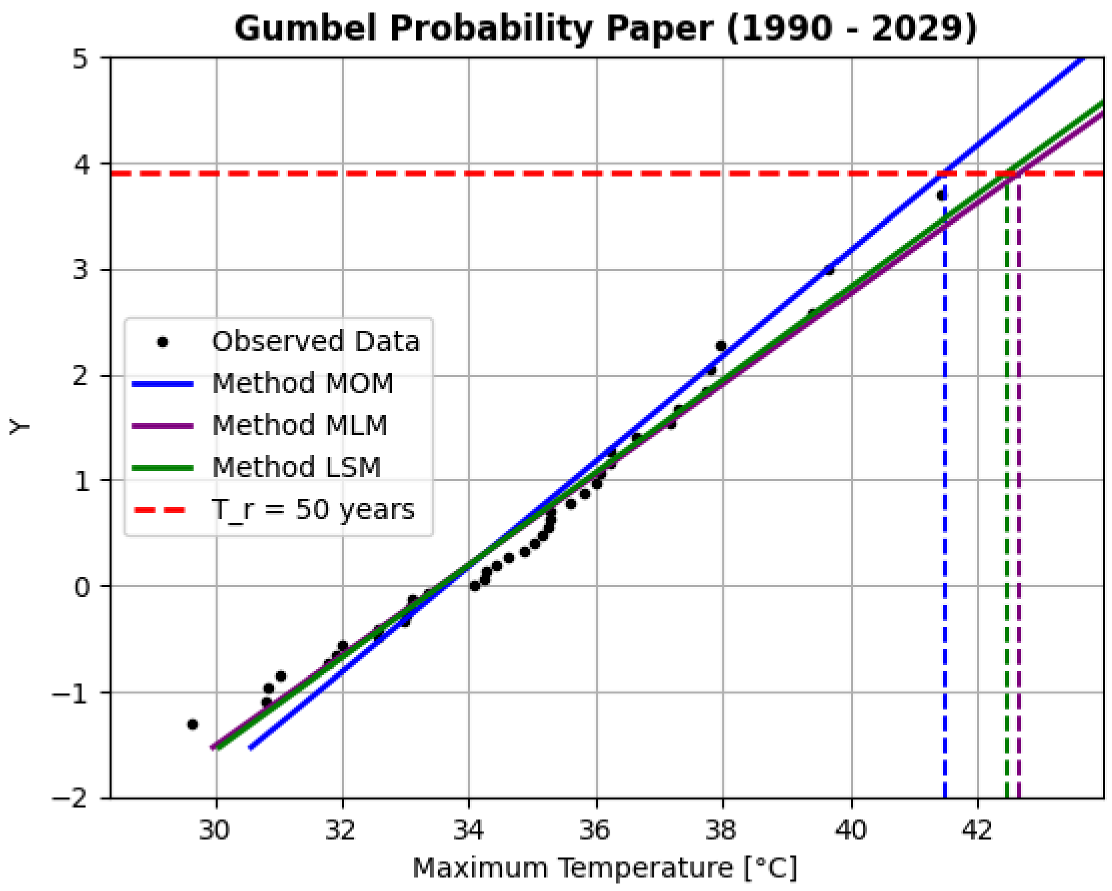
In [11]:
```
results = cr.ev1_param(dfs_processed, T_r, method)
```

Plot results

In [12]:
```
value = 'Factors of Change of Characteristic value'

fig, dataframe = cr.model_output(results, value, dfs_processed, T_r)
```



In [13]:
```
value = 'Gumbel Probability Paper'

window = windows[4]

fig = cr.model_output(results, value, dfs_processed, T_r, window)
```



Gumbel Probability Paper (1990 - 2029)
5
4
3
2
1
0
−1
−2
Y
30
32
34
36
38
40
42
Maximum Temperature [°C]
Observed Data
Method MOM
Method MLM
Method LSM
T_r = 50 years