CIM Wizard: A Modular and Automated Framework for Standardized Urban Energy Modeling from Sparse Data

Ali Taherdoustmohammadi, Daniele Salvatore Schiera, Luca Barbierato, Edoardo Patti, Lorenzo Bottaccioli, Pietro Rando Mazzarino

***Abstract*—Urban Energy Modelling (UEM) is increasingly vital for supporting local energy-saving strategies through quantitative analysis and simulation. However, practitioners face significant challenges due to limited data availability, heterogeneity, and the lack of standardized formats for both inputs and outputs, which limits reproducibility across different simulation tools. This work presents an enhanced, modular framework for generating standardized City Information Models (CIMs) from sparse and inconsistent urban datasets. The workflow is designed to be flexible and extensible, performing automated data enrichment and gap-filling through theoretical models and statistical inference. The tool supports persistent spatial storage in PostGIS and outputs CityJSON-compliant CIMs extended with energy and utility network data. The resulting models are simulation-ready and interoperable with co-simulation platforms, enabling dynamic analysis of buildings, systems, and grids. Validated on an Italian case study, the framework demonstrates its capacity to transform fragmented urban data into rich, reproducible digital models suitable for urban energy planning, scenario analyses, and digital twins.**

***Index Terms*—Urban Energy Modeling, UEM, UBEM, Urban Data Automation, Digital Twin, City Information Model (CIM)**

I. Introduction

The shift toward low-carbon and resilient cities increasingly depends on the ability to model and optimize interactions among buildings, infrastructures, and energy networks. In this context, Urban Energy Modelling (UEM) and its related fields, such as Urban Building Energy Modelling (UBEM) and Urban Energy System Modelling (UESM), have become key instruments for planning sustainable interventions, improving energy efficiency, and supporting climate mitigation strategies. As highlighted in several literature reviews [1]–[4], UEM plays a strategic role in policy design, while authors in [5] demonstrate its relevance for evaluating large-scale retrofit scenarios. UEM frameworks are broadly classified, into top-down and bottom-up approaches [2]. Top-down methods, as described in [1], use aggregated socio-economic or sectoral data to estimate urban energy demand, typically lacking spatial detail at the building scale. Conversely, bottom-up approaches model energy use from the ground up, using detailed physical, geometric, and operational data at the individual building level. These methods are increasingly preferred for their ability to produce high-resolution, scenario-specific analyses [5]. However, as noted by authors in [6], their effectiveness is often constrained by inconsistent data quality and fragmented sources [7] as well as the lack of common standards [8].

Focusing on bottom-up UEM that uses simulation for case study assessment, the general approach can be depicted in Figure 1. When choosing a real-world Urban Energy System (UES), all the needed information must be gathered and processed. In Figure 1, we have introduced the concept of City Information Model (CIM) as a results of these steps, although most of the UEM frameworks do not explicitly use standardized information models for data management, all of them use structured ways of storing the raw information. The CIM or whatever other data structure is used is the starting point for instantiating and performing simulations. The vast majority of frameworks available cover all the steps, while, as better described later, our solution focuses on the first three.

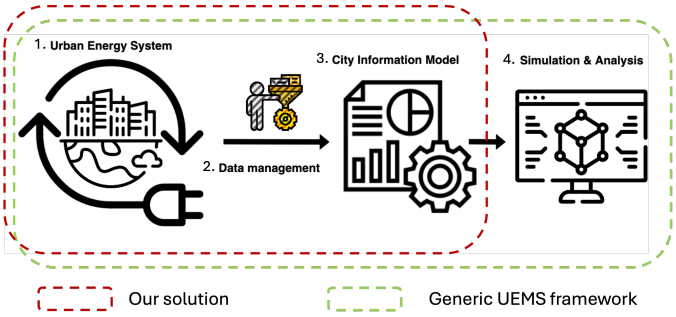

Fig. 1. Conceptual steps of UESM

Several modelling platforms have significantly advanced the field; however, notable limitations remain. Frameworks such as CityBES [9], UBEM.io [10], City Energy Analyst (CEA) [11], Simstadt [12], and UrbanOpt [13] are designed as all-in-one platforms or tightly integrated toolsets that guide users through the entire process of urban energy scenario analyses. These frameworks differ in their focus and intended applications, making their use highly context-specific. As a result, they often employ custom data models that are closely tied to specific simulation engines. This tight coupling restricts interoperability and hampers the reuse of data across different platforms.

To overcome these limitations, it is necessary to decouple the data management and processing from the simulation phase. Complementary, it is necessary to adopt standardized and widely used CIMs or data structures to ensure interoperability and consistency within different simulation contexts with different focuses and spatio-temporal

resolutions. Thereby enabling independent development and greater flexibility. Achieving this requires two key conditions: (i) the availability of modular, plug-and-play, and configurable frameworks, often based on co-simulation principles, capable of managing the simulation and analysis tasks [14]–[17]; and (ii) the standardization of CIM outputs to ensure seamless integration with co-simulation platforms as data sources.

A promising candidate for CIM standardization is CityGML [18], including its extensions (Application Domain Extensions, ADEs [19]), its simplified form (CityJSON [20]), and associated database structures (3DcityDB [21]). While some existing or updated frameworks can use CityGML models as inputs for simulation, none of them currently supports the generation of standardized CityGML outputs covering the first three steps illustrated in Figure 1. Standardized CIMs are also essential for advancing research on urban digital twins, as highlighted in [22]. A critical barrier remains the absence of standardized, interoperable database architectures, as noted in [23].

From a literature review of the state of the art, it is possible to highlight the following main challenges, needs or gaps: (1) **Flexibility and adaptability**: there is a need for flexible and extensible data architectures and processing methods that support heterogeneous inputs and deal with possible data scarcity [24], [25]; (2) **Standardization**: there is a need for generating standardized outputs to ensure data exchange and reusability in different applications, standardization should be central, not only for outputs generation but also in both data storing and process calculations [26]–[28]; (3) **Application independence**: There is a need for tools that can facilitate the creation of a configurable city information model independently to the application of the study and the chosen simulation platform or models [29], [30]. These tools should be at least customizable to embed data to adapt to the specific model parameters requirements (e.g. if the values of wall transmittance are required, the tool could give the user the possibility to estimate it or to embed a user-defined process for it).

In this context, with our previous work [31] we proposed an automated pipeline for City Information Model generation. With this article, we aim to extend our previous work by presenting a more complete and functional novel methodology that is implemented into **CIM Wizard**, a new service-based, flexible, and configurable UEM framework. CIM Wizard generates GIS-based city information models of a chosen case study area, enabling different workflows and different user interactions. It decouples data management, execution logic, and feature computation, enabling flexible orchestration of modeling steps via HTTP/REST interfaces. It supports runtime method selection based on available inputs and dependency resolution and integrates both public and private data sources—including Open Street Map (OSM), Digital Surface Models (DSMs), cadastral databases, and utility networks. The system also offers native export to CityJSON enriched with Energy and Utility Application Domain Extensions (ADEs). By bridging flexibility and standard compliance, CIM Wizard represents a significant step forward in enabling robust, context-aware UEM pipelines ready to be deployed across diverse urban environments and connected digital infrastructures.

The remainder of this paper is organized as follows. Section II presents the proposed methodology and the default workflow. Section III describes the components of the CIM wizard and the overall system architecture. Sections IV and V introduce the selected demonstrative case study and its simulation, illustrating the potential of employing a standardized City Information Model to describe and subsequently simulate an energy-related scenario. Finally, Section VI summarizes the main findings and outlines the conclusions.

II. Framework Methodology and Workflow

This section describes how configuration governs the framework's dynamic execution. The architecture of CIM Wizard is designed for flexibility and easy extension and better described in Section III. By defining fundamental features, such as building height or census population, these elements can be reused across multiple processing pipelines and combined into custom workflows. Each defined feature is associated with a feature calculator that is a module capable of estimating the feature value using different methods. By leveraging a central orchestration mechanism and a configuration catalog, the system enables seamless integration of modules and streamlined workflows. The orchestrator and catalog are described in detail in Section III-B. This modular approach enables a "plug-and-play" customization style, facilitating the creation of tailored solutions with minimal effort.

A. Pipeline Orchestration and Execution

Our methodology enables dynamic and highly customizable pipeline orchestration for CIM generation. A pipeline is defined as a sequence of feature calculators, which may execute synchronously or asynchronously depending on their input dependencies. The resulting framework allows users to construct pipelines manually, chaining feature-method pairs to reflect specific project needs, or to rely on predefined milestones that encapsulate common processing stages. These milestones, such as the initialization of geometry and demographic features, serve as modular execution bundles, ensuring that feature dependencies are resolved and calculation order is preserved (e.g., height before number of floors). Independent calculators can be executed in parallel, while dependent calculators are scheduled sequentially. Figure 2 illustrates an example in which methods, from a sample of feature calculators, are chained across calculators, with green and red arrows representing different user-defined execution paths. Advanced users may further customize the pipeline by defining explicit chains using a dedicated syntax (e.g., `feature.method | feature.method | ...`), granting them fine-grained control over the execution logic. This "chainable" interface supports iterative development, algorithm comparison, and research prototyping—capabilities that are rarely available in traditional UESM platforms. At the core of this orchestration capability lies a robust

fallback mechanism. If a method fails, for example, because it cannot compute a feature for all buildings or its inputs are unavailable, the orchestrator automatically invokes lower-priority methods as specified in the configuration catalog. This adaptive strategy enhances resilience to missing data and enables modular, incremental data enrichment. In addition, the configuration catalog is exposed as a service for storing all custom information related to feature calculation and execution characteristics, thereby enabling customization with minimal effort.

B. Context-Aware Data Management

The core of a pipeline is the context manager that dynamically manages input data, dependencies, and intermediate results across all feature calculators. This context object acts as a shared memory space, receiving and distributing inputs from multiple sources, including user uploads, database records, external services, and default assumptions. Each feature calculator interacts with the context manager to retrieve the required inputs and store outputs without explicit coupling to other modules. This architecture allows individual calculators to operate independently while maintaining access to a common execution environment. If a method requires a feature that has not yet been computed, the orchestrator automatically resolves this by invoking the appropriate upstream calculators in the correct order. This context-based execution not only facilitates robustness and traceability but also allows runtime introspection, where users can inspect and debug intermediate states, validate assumptions, or inject custom values. It is particularly useful in urban data environments characterized by heterogeneity, partial completeness, and evolving data availability.

C. Flexible Data Input Options

The framework is designed to accommodate both richly detailed and minimally specified input datasets. Users may upload building footprints, census data, energy attributes, or utility network layouts using standard formats such as GeoJSON, Shapefiles, pandapower, or tabular CSV with configuration metadata. These inputs are validated, mapped to the internal schema, and integrated into the pipeline via the context manager. In cases where only minimal input is available, such as a boundary polygon defining the study area, the system invokes external data services to retrieve necessary information. OpenStreetMap (OSM) is queried to obtain building geometries, census APIs provide demographic attributes, and elevation services support height estimation. If no data is found, fallback mechanisms are triggered to populate fields using national defaults or statistical estimates, such as the TABULA archetypes for building stock.

This flexible input strategy ensures that our methodology can operate under a wide range of data conditions, making it suitable for both research and operational planning across varied urban contexts.

D. Adding New Feature Calculators

A core strength of our methodology is its extensibility: it allows developers to introduce new computational logic for existing features as well as adding new features without altering the internal orchestration engine. This is achieved through a modular design in which each feature calculator is defined as a standalone class in object-oriented programming. These classes adhere to a common interface and are structured to encapsulate one or more estimation methods. Each method corresponds to a valid strategy for computing a specific feature (e.g., building height, population inside a building, or the thermal archetype) and can incorporate data-driven models, heuristic rules, or external service integrations.

To register a new calculator, the developer simply declares the class in a configuration catalog, that defines the available methods, input dependencies, execution priorities, and expected outputs. This configuration is dynamically loaded at runtime and serves as the central decision-making layer for pipeline execution. Importantly, no changes are required in the core logic or orchestration engine; the new class becomes automatically available in the pipeline once the configuration is updated. This decoupled approach accelerates prototyping, encourages experimentation, and enables parallel development of new features without compromising system stability.

E. Method Resolution and Fallback Logic

When the pipeline is triggered for a specific feature, the orchestrator follows a multi-step resolution procedure. It begins by consulting the configuration catalog to identify all registered methods associated with the target feature. These methods are then ranked by priority as defined by the developer. For each method in descending order of priority, the system evaluates the availability of required inputs, which may include spatial geometries, demographic attributes, or previously computed feature values. If all dependencies are satisfied, the method is executed. If the result is valid and non-null, it is stored; otherwise, the system proceeds to the next method in the queue. This fallback-driven mechanism ensures resilience to missing data and supports the integration of multi-tiered estimation logic. For example, a building height feature may first attempt to use satellite elevation data, fall back to OpenStreetMap tags if unavailable, and ultimately default to a standardized height assumption. By treating each method as a candidate in a prioritized decision tree, CIM Wizard adapts dynamically to the quality and completeness of input data. This flexibility is especially critical in urban energy modeling contexts, where data sources vary significantly in granularity and coverage.

F. Runtime Priority Overrides

In addition to static configuration, our methodology provides a runtime override mechanism for method prioritization. This is particularly useful during sensitivity analysis, calibration studies, or comparative assessments of different estimation strategies. Through a dedicated API endpoint, users or developers can inject temporary priority

settings for selected features without altering the underlying configuration file, as shown in Figure 2. These overrides are interpreted at runtime and take precedence over the default priority hierarchy defined in the catalog.

This capability enhances the framework's usability in both research and operational scenarios. It allows practitioners to tailor execution logic to specific case studies or data conditions, and it enables developers to test new methods in isolation before formally integrating them into the main configuration. By combining declarative configuration with dynamic control, our methodology supports a wide range of workflows, from stable production pipelines to experimental urban modeling scenarios.

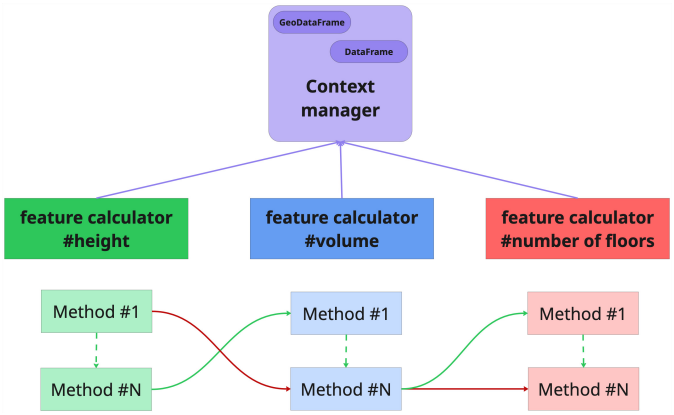

Fig. 2. Possible workflow pipelines. Each feature calculator supports multiple methods and writes directly to the Context Manager. Users can orchestrate different pipelines by configuring both the methods and their execution order (red and green lines). In particular, the green dashed lines illustrate a fallback mechanism, where multiple methods of the same feature calculator can be chained with a defined priority, ensuring sequential fallback if one method fails.

III. Framework Architecture and Components

The solution introduced in Section II is implemented into CIM Wizard, our open-source tool that follows a micro-service architecture [32]. Micro-service architectures are based on the composition of small independent and deployable services to compose a large infrastructure. CIM Wizard is designed to create comprehensive City Information Models from sparse data sources, while gathering all the necessary information for subsequent energy simulations. Throughout this work, we refer to these data elements as features of the case study or its components (e.g., buildings). A key characteristic of CIM Wizard is its extensibility and modular design, which enables the straightforward addition of new features to address diverse requirements. The CIM Wizard architecture is organized into four interconnected component layers that together enable the transformation of sparse urban datasets into standardized City Information Models. The *Data Source Layer* manages all incoming data, including public repositories, user uploads, and live web services such as OpenStreetMap, national census APIs, and TABULA construction data. The *Core CIM Wizard* layer handles orchestration and configuration, managing execution logic, pipeline control, and context propagation across the framework. The *Feature Calculator Modules* layer performs the core data enrichment operations, computing features through modular calculators, each equipped with fallback strategies and extensibility for external services. Finally, the *Representation Layer* converts the enriched data into GeoServer, GeoJSON and 3DCityDB compliant structures, embedding Energy ADE and Utility ADE semantics for expanding knowledge in a structured fashion. The rest of this Section will in-depth discuss each layer.

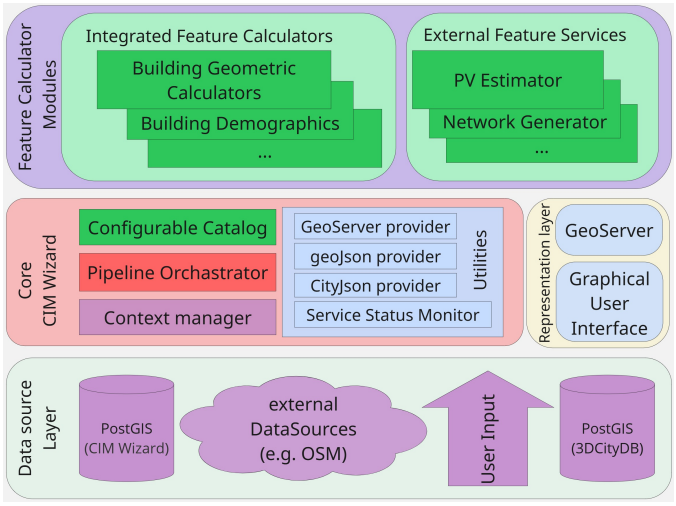

Fig. 3. High-level architecture schema

A. Data Source Layer

The Data Source Layer in the CIM Wizard architecture is composed of four key components: (1) *PostGIS (CIM Wizard)*, (2) *External Data Sources*, (3) *User Input*, and (4) *PostGIS (3DCityDB)*. Each of these components plays a distinct role in managing the spatial and semantic data required for scenario creation, feature enrichment, and export into standardized formats such as CityJSON and 3DCityDB.

PostGIS (CIM Wizard) is the central operational database that supports the entire data processing workflow of the framework. It stores both spatial geometries, such as building footprints and project boundaries, and the semantic attributes computed throughout the enrichment process, including physical properties, demographic characteristics, and typological classifications. The database schema follows a normalized structure: geometric entities are maintained separately from their associated descriptive data, which are organized into modular feature sets. Each enriched record is indexed using a composite key linking it to the geometric object and then to a classification schema that consists of project and scenario identifiers. As shown in Figure 4, it is possible to create different projects and within one project, different scenarios. With this logic, the same geometric entity may have different feature data across multiple scenarios. Thus, it is possible to reuse the spatial entities across multiple simulation contexts while minimizing redundancy. Through an incremental update strategy, only modified attributes are stored in scenario variants, thus supporting efficient memory

usage, rapid scenario switching, and comparative simulation workflows across time or configuration space.

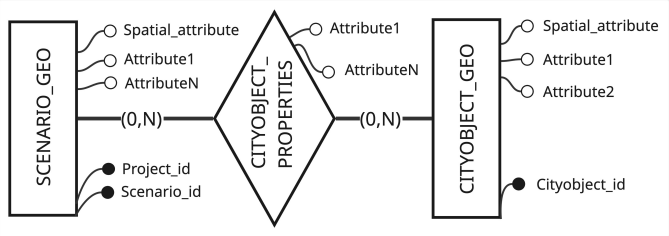

Fig. 4. Generic entity-relations diagram CIM Wizard PostGIS database schema

External Data Sources are used to bootstrap or enrich projects when no user input is available. These sources are grouped into two categories: global and local. Global sources, such as OpenStreetMap (OSM), provide universal coverage and are accessed through Python-based libraries like `osmnx` and the Overpass API. Local sources are tailored to the regional context and may include (i) building footprints obtained from official geospatial portals (accessed through WFS or similar services), (ii) demographic data from national or regional census repositories (retrieved via API endpoints), (iii) high-resolution Digital Surface Models (DSM) for building height estimation, and (iv) archetype datasets providing construction-year-specific building envelope properties such as the TAUBULA one for Europe. All data pipelines are wrapped in modular parsers or data services, ensuring that each source can be updated or replaced independently without disrupting the pipeline logic.

User Input represents another vital source of data for the CIM Wizard framework. Developers or simulation users may upload their own project boundaries, building footprints, or feature attributes in GeoJSON or tabular formats. The framework accommodates such inputs through a configuration-based mapping system, which aligns the uploaded schema with the internal CIM Wizard database structure. Users are also able to predefine fallback priorities for their data, override defaults, and define explicit field-to-field correspondences. This ensures flexibility for integration with third-party surveys, digital twin platforms, or external GIS workflows, without sacrificing semantic consistency.

PostGIS (3DCityDB) acts as the long-term storage and export layer for simulation-ready CIMs. This instance of 3DCityDB has been extended to support relevant Application Domain Extensions (ADEs), including the **Energy ADE** and the **Utility Network ADE**. These extensions enable the database to represent thermal zones, energy demand, construction details, and energy system assignments as defined by the Energy ADE, as well as grid connectivity and infrastructure logic through the Utility ADE. The modules used by CIM Wizard to export in CityGML compliant format map the intermediate tables in PostGIS to CityGML/CityJSON records in the 3DCityDB schema. This mapping ensures that every enriched object can be transformed into a standard city object for simulation, visualization, or semantic reasoning.

Together, these four components provide a flexible, scalable, and interoperable data foundation for CIM Wizard's operations. The design accommodates both default and user-defined workflows, integrates heterogeneous data from multiple sources, and supports standardized urban model exports for advanced co-simulation platforms.

B. CIM Wizard Core Layer

The CIM Wizard Core layer governs the execution logic and data flow of the entire system. It is composed of modular components responsible for orchestrating feature computation, managing system configuration, and ensuring robust interaction with the database.

In the middle of this layer, there is the *Pipeline Orchestrator*, which dynamically constructs and executes processing workflows (i.e. the pipelines described in Section II) based on user-defined objectives and configuration rules. In rough words, a pipeline is the sequential set of calls that allows to fill in the data of the city information model, the workflow of this process is already explained in Section II. From a high-level perspective, the pipeline must invoke the appropriate feature calculators, handle their dependencies (e.g., calculating volume requires height), and implement fallback strategies (e.g., if a calculator does not populate all entities using its primary method, it sequentially triggers the subsequent methods until the process is completed). Additionally, it ensures the correct execution order across tasks, whether sequential or parallel.

Complementing this, there is the *Context Manager*, a component that abstracts all database interactions. It acts as the central interface for reading inputs and persisting outputs during pipeline execution. The context manager also enforces schema consistency and supports automatic migrations, enabling modular services to interact seamlessly within a microservice-oriented architecture.

The *Configuration Catalog* harmonizes all microservices. It stores their endpoints and data source locations, as well as the configuration of each external feature service. This enables central management of any changes to the default settings. It also defines the path and method prioritization for all integrated feature calculators. Optionally, it can include method parameters and the semantic layer for the CityGML mapping schema.

Lastly, the *Utility* block contains a set of various methods for consistency and completeness checking, *Service Status Monitor* to return all available services and integrated feature calculators available in CIM Wizard framework and a set of data format convertors, *geoJSON provider* for react-based libraries' GUI, *GeoServer provider* to publish data layers on GeoServer for OGC Standard-based services like WMS and *CityJson Provider* to map the data to 3DCityDB schema by using the predefined semantic layer in configuration Catalog. It will consider any missing feature in the semantic layer as a generic attribute.

C. Features Calculator Modules layer

The Features Calculator Modules layer of the CIM Wizard system is composed of two distinct but complementary

categories: *Integrated Feature Calculators* and *External Feature Services*. This dual structure supports both fine-grained attribute computation and large-scale domain-specific modeling, enabling the platform to adapt to a wide range of urban energy simulation contexts.

Integrated Feature Calculators are embedded directly within the CIM Wizard core and are invoked by the pipeline orchestrator. These calculators implement specific building or scenario-level features, each as an independent Python class, encapsulating one or more estimation methods. The system supports developer extensibility by allowing new feature calculators to be added through a well-defined interface inheritance. Each class must define one or more method functions, which are registered and prioritized via the Configuration Catalog.

This design enables developers to create new logic modules without modifying the core orchestration engine. By simply declaring a new calculator class and appending it to the configuration catalog, users can integrate alternative data sources, computational heuristics, or fallback strategies. Execution priorities and method dependencies are managed through the configuration catalog, allowing runtime resolution of the optimal method based on data availability and developer-defined logic.

This section details some of the most critical and methodologically interesting feature calculators integrated by default in the *CIM Wizard* engine.

The **Scenario geometry calculator** initiates each project by defining the spatial context, project boundary, or buildings' footprint. If a user-defined project boundary is provided via the user interface, the system processes it to extract a unique project boundary, determine its centroid, and set metadata including coordinate reference system (CRS), zoom level, and assign a unique project and scenario identifiers to that. In the absence of such input, the system derives the project boundary algorithmically by applying a convex hull operation to available building footprints, thus ensuring a minimal and accurate spatial boundary for the project area, and then assigns the universal unique project and scenario identifier.

In parallel, the **Building Geometry Calculator** validates or acquires the building footprint layer. When no initial building data is supplied, the system queries OpenStreetMap (OSM) using both the `osmnx` library and the Overpass API to retrieve buildings within the project boundary. To ensure semantic relevance, it integrates an intelligent classification layer that preserves European mixed-use patterns (e.g., residential units above retail shops) while filtering out buildings that do not meet geometric configurable criteria (e.g., height < 8 m or area < 100 m^2). If a user uploads a building layer in GeoJSON format, the calculator normalizes the schema, assigns unique building identifiers, and performs geometry validation. This dual-source logic supporting both externally-sourced, and user-defined datasets, enhances the system's adaptive capacity.

Each building is associated with a **BuildingProperties** instance, which acts as the semantic backbone for feature storage. The calculator initializes or updates these entries to link each building with its scenario. This data structuring enables downstream calculators to persist values consistently across simulation workflows.

Among the most critical calculations, there is the **Building Height Calculator**, which implements a three-tiered fallback system. The preferred method queries a raster elevation service to compute building height as the difference between the Digital Surface Model (DSM) and the Digital Terrain Model (DTM). To ensure scalability, it batches requests in groups of 100 with coordinate transformation and error management. This number is a configurable parameter based on computational power. If raster data is unavailable, the calculator extracts height values from OSM tags, employing regex-based parsing to handle inconsistent formats (e.g., meters, feet, or free-text). When all else fails, a standardized assumption of 12 meters is used reflecting a building with 4 floors with under a 3.0 m/floor heuristic.

The **Building Area Calculator** complements this by projecting building geometries from geographic coordinates (WGS84) to the local projected CRS, for accurate area computation. It handles both Polygon and MultiPolygon types, applying fallback approximations if projections are infeasible. With height and area known, the **Building Volume Calculator** computes volumetric space via a simple product, storing validated results through atomic transaction handling. The number of floors is then estimated by applying a height-based heuristic (1 floor per 3 meters, this is building height calculator's hyperparameter), rounded to the nearest integer. This method, though approximate, enables rapid floor count estimation suitable for energy simulation archetype assignment. Another key component is the **Scenario Census Boundary Calculator**, which queries a remote census API using the project's spatial extent. It retrieves all intersecting census zones, applies a geometric union with buffered boundaries to avoid over-dilution, and stores relevant demographic variables such as total population, family count per type of family, and residential building construction epochs. These data points are parsed by the **Census Population Calculator**, which extracts population counts and prepares them for distribution.

Building usage classification is performed via the **Building Usage Type Calculator**, which filters buildings through a two-step process. First, it removes explicitly non-residential types using OSM tags (e.g., schools, hospitals), but retains mixed-use buildings when residential use is inferred. Second, it applies geometric filters to exclude structures unlikely to house residents based on size and height. Once residential buildings are identified, the **Building Population Calculator** allocates population across them proportionally to their volume thereby approximating population density per building. Family counts are then inferred from this allocation using a simple division by the average family size (e.g., 3 people), producing realistic integer values. The **Building Demographic Calculator** combines these methods into a comprehensive pipeline. It orchestrates census acquisition, building geometry queries, height and volume computation, usage classification, and demographic allocation. It also includes zone-level accuracy reporting and random assignment of construction year using weighted distributions from census periods. This enables buildings to be tagged with plausible construction epochs,

enhancing downstream simulation capabilities.

To ensure compatibility with the Energy ADE schema, the CIM Wizard framework supports the generation of thermal boundaries and the assignment of construction and energy system parameters to each building or thermal zone. This process is essential for enabling scalable simulation of urban energy systems using standardized formats such as CityJSON and 3DCityDB.

The process begins with the **Building Geo LoD 1.2 Calculator**, which constructs semantic surfaces including `RoofSurface`, `WallSurface`, and `GroundSurface` by extruding 2D building footprints according to calculated building heights. Each surface is further enriched with geometric metadata such as orientation, area, and azimuth angle, resulting in a LoD 1.2-compliant CityJSON geometry structure. To populate these surfaces with construction-related thermal properties, the `TABULA building type` calculator is employed. This module classifies each building into a representative typology based on number of floors, family count, and usage pattern. The classification is then mapped to a city-specific construction table derived from TABULA and local building physics references, providing U-values, R-values, construction materials, and refurbishment class identifiers. These parameters are applied to the respective building surfaces through the **Thermal Boundary Calculator**, resulting in a thermally enriched geometric model consistent with the Energy ADE construction module.

In addition to thermal boundaries, the Energy ADE schema requires specification of energy conversion systems at the building or thermal zone level. For this purpose, we developed an **Energy System Calculator**, integrated as an internal feature module within the CIM Wizard framework. This calculator assigns representative heating and renewable systems, such as gas boilers, heat pumps, and photovoltaic arrays, based on typological and temporal indicators derived from the TABULA archetypes. Each system is characterized by attributes such as fuel type, nominal efficiency, installed capacity, and age class. The assignment logic is rule-based and supports batch processing across the entire scenario. The outcome of this process is enriched with both thermal and system-level metadata, aligned with a simplified but valid subset of the 3DCityDB + Energy ADE schema. Altogether, these feature calculators exhibit a robust blend of geospatial reasoning, semantic enrichment, and fault-tolerant fallback logic. Their modular architecture allows for parallel development, testing, and extension, thereby enabling CIM Wizard to perform under diverse urban data conditions while remaining compliant with open standards like CityJSON and Energy ADE.

The *External Feature Services*, on the other hand, are implemented as independent web services due to their computational and architectural complexity. Indeed, each *Integrated feature calculators* follows, as discussed in Section III-C a common design in other to be easily integrated in the pipeline orchestrator, while the *External Feature Services* are stand-alone services that can be run on top of the intermediary outputs generated by the integrated calculators and works for the full scenario. Examples include photovoltaic (PV) system estimators and power network generators. These services operate outside the pipeline orchestrator but are linked to the CIM Wizard database via HTTP/REST interfaces. Unlike internal calculators, feature services do not rely on standard dependency resolution and they are not invoked through the method selection chain. Instead, they act autonomously, either reading precomputed external outputs or generating their own, and map results back into the CIM Wizard schema.

So far, the CIM Wizard ecosystem includes two external feature services: i) A **Network Generator and Integrator**, which imports and links distribution network data in pandapower or geojson format (e.g., the power grid or the district heating); ii) A **PV Estimator and Integrator**, which can estimate PV installation data based on roof surfaces and typical meteorological data [33].

The PV estimator services are built upon the redesign and integration of a tool originally proposed in [33]. At this stage, the network integrator for the power grid solely merges a given power grid case study into the existing information model, whereas for district heating, it can additionally generate a synthetic virtual district heating network by integrating the approach presented in [34]. Future work will focus on integrating services for the power grid synthetic and realistic generation using aggregated load data and real world GIS constraints (i.e. substations, street network) such as it has [35].

The modular orchestration design outlined in Section II-A ensures seamless integration of additional feature services, whether internally computed or externally linked, thus maintaining the scalability and extensibility of the overall system.

D. Representation layer

The representation layer in our architecture consists of two main components, *GeoServer* – for serving geospatial data over standard protocols. *Graphical User Interface* (GUI) – a React-based frontend for end-user interaction. A detailed explanation of this layer is beyond the scope of this article. However, it is important to note that CIM Wizard is capable of publishing vector data to GeoServer and providing OGC-compliant services such as WMS, WFS, and others.

IV. Case study application

To illustrate the practical use of the proposed City Information Model (CIM), we present a co-simulation case study built using an instance of this information model. This section outlines the simulation environment, including the chosen simulators, co-simulation platform, and data exchange mechanisms, highlighting how the CIM can serve as a flexible and comprehensive data backbone for complex simulation workflows, supporting multidisciplinary exploration and scenario evaluation. The selected study area, for which the CIM was developed, is a portion of a neighbourhood in the city of Turin, Northern Italy. The dataset includes building footprints from the *Geoportale Piemonte* [36], the regional geospatial portal, a DSM of the area, and census information from the Italian National Institute

of Statistics (ISTAT) [37]. The total number of buildings in the case study is 552, with 453 residential Buildings (mostly apartment blocks). In addition to the building-level information, a power grid network case file has been linked and mapped to the area, as illustrated in the GIS-based representation of the case study in Figure 5.

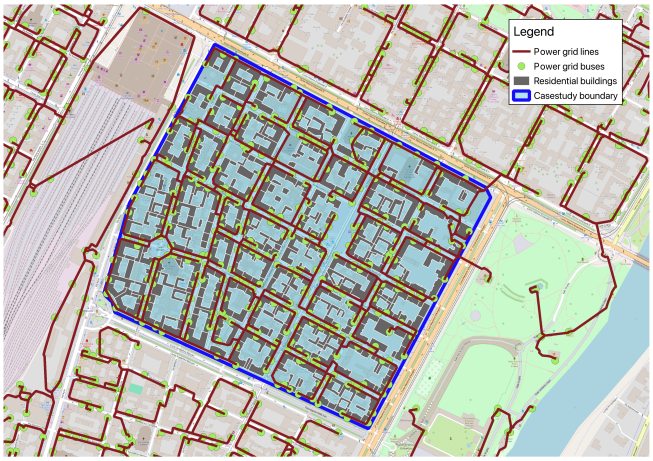

Fig. 5. GIS-based representation of the case study. This is obtained using CIM wizard and visualized in QGIS, It is the actual representation fo the generated City Information model.

Using the CIM Wizard in its default configuration, we selected a specific area and generated a CIM that includes buildings, heating systems, and a power network. Based on this model, we developed an energy scenario implemented through co-simulation using the platform from [16]. Leveraging the estimated granular data, we parameterized and instantiated five types of simulators to accurately model and simulate each real-world component. The co-simulation approach enables scenario decomposition, allowing control over the desired level of granularity and representing real-world entities through multiple specialized simulators when needed. For this demonstrative application case, we employed the simulators described in the following.

Weather file reader is a simulator that reads the weather file for the analyzed area. This case study uses a typical meteorological year (TMY) file for Turin. **Scheduler for indoor setpoints** is a simulator that sends setpoint signals to each building. For this case study, a uniform schedule was applied to all buildings, setting the indoor thermal comfort temperature to 20°C during the heating season. **Occupancy file readers** is a simulator that reads predefined CSV files containing occupancy data, providing values for people's presence and internal gains to the envelope models. **Envelope** simulate buildings by FMU-wrapped Modelica models, generated and parameterized for each building based on its individual thermal and geometrical properties. These simulators estimate the heat demand required to maintain indoor comfort and can receive an actuation signal representing the actual heat power delivered to the thermal zone. **Heat Pump** is a Python-based simulator of a heat pump. Each instance represents a heat pump connected to a specific building, sized according to the building's area and energy class. **Power grid model** is a simulator based on the Python library `pandapower`, which takes a network case file and simulates the power flow of the electrical grid, receiving connected loads at each simulation step. Data exchange begins with the weather simulator, which provides boundary conditions to the envelope simulators and heat pumps. Specifically, it sends the external temperature to the heat pumps and a broader set of parameters—including temperature, humidity, and solar irradiation—to the envelope simulators. The envelope simulators then estimate the thermal demand and communicate it to the corresponding heat pump models, which compute the actual thermal power output and the resulting electrical consumption. The thermal power is sent back to the envelope simulators for actuation, while the electrical load is forwarded to the power grid model at the appropriate bus.

For this case study, we ensured uniform conditions across all buildings in order to isolate the impact of the CIM Wizard's data modeling process on the simulation results. Specifically, the same weather file was applied to all buildings, the occupancy schedule was standardized (though it could be customized for each building based on demographic information generated by the CIM Wizard), and the indoor temperature setpoint schedule was fixed at a constant value of 10°C. No control logic for the heat pump was implemented; only physical constraints related to realistic sizing were considered. The simulation covered a 15-day period in early January, using a 10-minute timestep.

V. Experimental Results

This section is organized into an initial examination of the validation of the estimated data using the default methods of the CIM Wizard, followed by the presentation of the co-simulation results.

A. Data Validation

Three key features were validated against available ground-truth data: building height, number of floors, and year of construction. The features were estimated using the default CIM wizard methods that could be easily customized or replaced. Building heights were derived from DSM based measurements. The error distribution in Figure 6-(a) shows a mean bias close to +1 m and limited dispersion, indicating that the method provides sufficiently accurate volumetric information for urban-scale thermal modeling.

The number of floors was obtained by dividing the estimated height by a constant representing the average storey height. Figure 6-(b) reports the error distributions obtained for different constant values, highlighting that a thorough understanding of the prevailing building types in the area of interest can significantly reduce the systematic overestimation observed with the default setting. For example in our case study an average assumption of 3.7 meter per floor lead to the best agreement between estimated and observed floor counts. Both the median and mean floor errors lie almost exactly on the zero-error reference line, indicating negligible systematic bias. Still this kind of fuzzy methods generates outliers

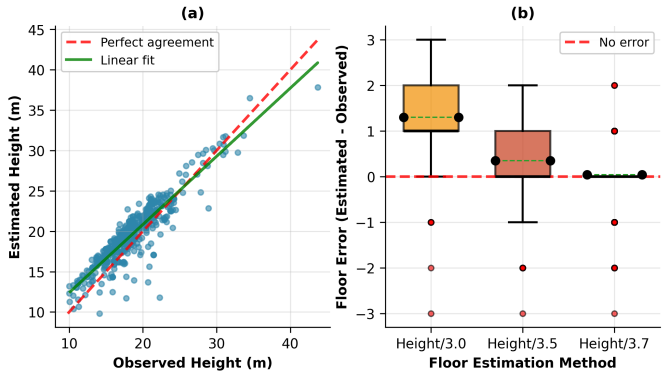

Fig. 6. Error distribution for features: height and number of floors

which are those buildings that manifest different construction archetypes from the majority of the building stock.

The year of construction was validated at the aggregation level of ISTAT census time slots, rather than at the individual-building scale. This reflects the temporal resolution typically available in official datasets and ensures that the modeled age distribution statistically matches the actual building stock composition as shown in Figure 7.

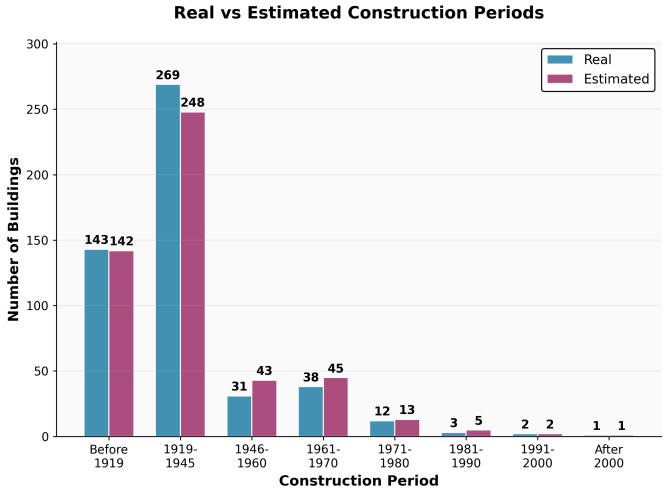

Fig. 7. Estimated vs Real construction periods

B. Simulation Results

The generated City Information Model was used to produce a simple demonstrative co-simulation scenario. The purpose of this exercise is illustrative, showing the capability of the framework to flexibly create and organize information for a further sparse usages.

Figure 8 presents the simulated load curves for a selection of buildings. Even under uniform boundary conditions and operational schedules, notable variability emerges due to differences in volume, construction period, and thermal characteristics, evidencing the effectiveness of archetype-based parameterization in reproducing building stock heterogeneity. Figure 9 highlights the joint effect of building volume and construction year on annual heating demand. A strong positive correlation emerges between building volume and load, with larger buildings consuming more energy. Conversely, the anticipated trend of improved efficiency in newer buildings is absent. This reflects the characteristics of the Italian building stock, where, over time, construction shifted toward faster and cheaper methods without significant advances in envelope performance.

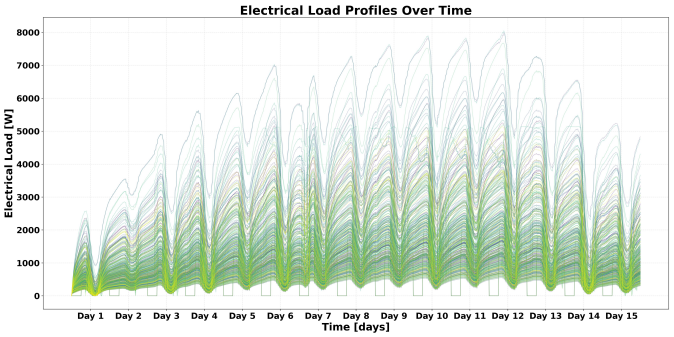

Fig. 8. Electrical load profiles for all the simulated residential buildings in the first 15 days of January in Turin

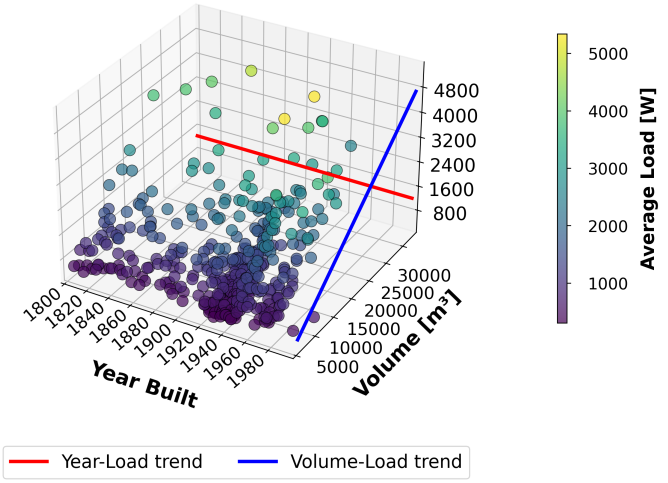

Fig. 9. 3D scatter visualization for average load, volume and building year construction correlation

The co-simulation integration with the electrical network enabled the assessment of the interactions between buildings and power infrastructures. For example in Figure 10 voltage levels at selected buses are presented showing the direct impact of building loads on network conditions. While this represents a simplified case study, it demonstrates the potential of the proposed workflow: starting from sparse geospatial data, it is possible to generate coherent multi-domain simulation models. This approach can be extended to more complex and large-scale analyses, enabling integrated planning and operation studies across multiple energy vectors.

VI. Conclusion

This work introduced CIM Wizard, a modular microservice-based framework for automating the generation of

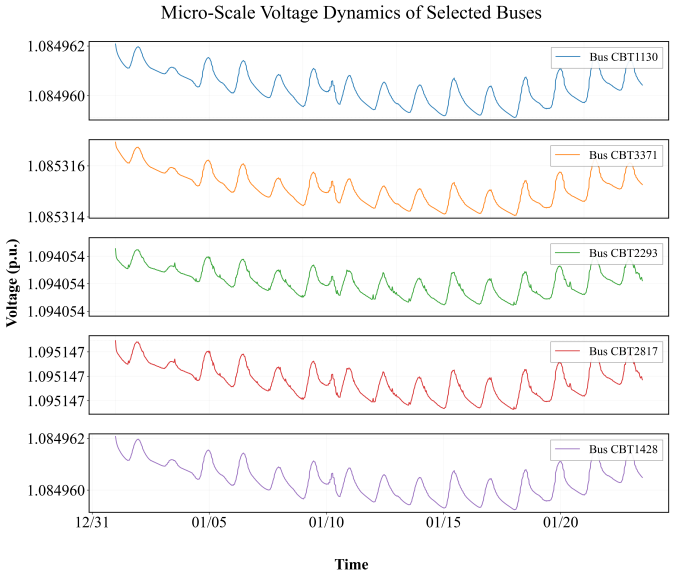

Fig. 10. Voltage dynamics for some example buses

standardized City Information Models from sparse and heterogeneous urban datasets. Capable of operating with either user-provided inputs or no manual data at all, it enriches and harmonizes information through robust fallback strategies, producing outputs compliant with CityJSON and extended with Energy and Utility ADEs. By decoupling data management from simulation and offering a flexible, customizable architecture, the framework overcomes key limitations in existing approaches, enabling reproducible and interoperable models from the building to the city scale. In doing so, it provides a robust starting point for Urban Energy Modelling processes, particularly in data-scarce contexts, and supports the creation of scalable, simulation-ready digital twins for integrated urban planning. Future development aims to evolve these into fully self-generating agents, capable of estimating features without relying on external preprocessing. This transformation will support deeper integration of high-fidelity energy generation and distribution models into the CIM framework and reduce dependency on static data inputs.

REFERENCES

[1] L. G. Swan and V. I. Ugursal, "Modeling of end-use energy consumption in the residential sector: A review of modeling techniques," *Renewable and sustainable energy reviews*, vol. 13, no. 8, pp. 1819–1835, 2009.

[2] N. Abbasabadi and M. Ashayeri, "Urban energy use modeling methods and tools: A review and an outlook," *Building and Environment*, vol. 161, p. 106270, 2019.

[3] I. Hijazi and A. Donaubauer, "Integration of building and urban information modeling–opportunities and integration approaches," *Geoinformationssysteme*, pp. 42–56, 2017.

[4] C. F. Reinhart and C. C. Davila, "Urban building energy modeling–a review of a nascent field," *Building and Environment*, vol. 97, pp. 196–202, 2016.

[5] J. Keirstead, M. Jennings, and A. Sivakumar, "A review of urban energy system models: Approaches, challenges and opportunities," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 6, pp. 3847–3866, 2012.

[6] J. Allegrini, K. Orehounig, G. Mavromatidis, F. Ruesch, V. Dorer, and R. Evins, "A review of modelling approaches and tools for the simulation of district-scale energy systems," *Renewable and Sustainable Energy Reviews*, vol. 52, pp. 1391–1404, 2015.

[7] Y. Li and H. Feng, "Integrating urban building energy modeling (ubem) and urban-building environmental impact assessment (ub-eia) for sustainable urban development: A comprehensive review," *Renewable and Sustainable Energy Reviews*, vol. 213, p. 115471, 2025.

[8] T. Hong, Y. Chen, X. Luo, N. Luo, and S. H. Lee, "Ten questions on urban building energy modeling," *Building and Environment*, vol. 168, p. 106508, 2020.

[9] T. Hong, Y. Chen, S. H. Lee, and M. A. Piette, "Citybes: A web-based platform to support city-scale building energy efficiency," *Urban Computing*, vol. 14, p. 2016, 2016.

[10] Y. Q. Ang, Z. M. Berzolla, S. Letellier-Duchesne, V. Jusiega, and C. Reinhart, "Ubem. io: A web-based framework to rapidly generate urban building energy models for carbon reduction technology pathways," *Sustainable Cities and Society*, vol. 77, p. 103534, 2022.

[11] J. A. Fonseca, T.-A. Nguyen, A. Schlueter, and F. Marechal, "City energy analyst (cea): Integrated framework for analysis and optimization of building energy systems in neighborhoods and city districts," *Energy and Buildings*, vol. 113, pp. 202–226, 2016.

[12] R. Nouvel, K.-H. Brassel, M. Bruse, E. Duminil, V. Coors, U. Eicker, and D. Robinson, "Simstadt, a new workflow-driven urban energy simulation platform for citygml city models," in *Proceedings of International Conference CISBAT 2015 Future Buildings and Districts Sustainability from Nano to Urban Scale*. LESO-PB, EPFL, 2015, pp. 889–894.

[13] R. El Kontar, B. Polly, T. Charan, K. Fleming, N. Moore, N. Long, and D. Goldwasser, "Urbanopt: An open-source software development kit for community and urban district energy modeling," National Renewable Energy Lab.(NREL), Golden, CO (United States), Tech. Rep., 2020.

[14] C. Steinbrink, M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. P. Ramírez Acosta, T. Raub, J. S. Schwarz, S. Stark, A. Nieße, and S. Lehnhoff, "Cpes testing with mosaik: Co-simulation planning, execution and analysis," *Applied Sciences*, vol. 9, no. 5, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/5/923

[15] T. D. Hardy, B. Palmintier, P. L. Top, D. Krishnamurthy, and J. C. Fuller, "Helics: A co-simulation framework for scalable multi-domain modeling and analysis," *IEEE Access*, vol. 12, pp. 24 325–24 347, 2024.

[16] D. S. Schiera, L. Barbierato, A. Lanzini, R. Borchiellini, E. Pons, E. Bompard, E. Patti, E. Macii, and L. Bottaccioli, "A distributed multimodel platform to cosimulate multienergy systems in smart buildings," *IEEE Transactions on Industry Applications*, vol. 57, no. 5, pp. 4428–4440, 2021.

[17] P. R. Mazzarino, M. Capone, E. Guelpa, L. Bottaccioli, V. Verda, and E. Patti, "A modular co-simulation platform for comparing flexibility solutions in district heating under variable operating conditions," *IEEE Transactions on Sustainable Computing*, vol. 10, no. 2, pp. 408–417, 2025.

[18] Citygml. [Online]. Available: https://www.ogc.org/standard/citygml

[19] O. CityGML. (2023) Citygml ades. [Online]. Available: https://www.citygmlwiki.org/index.php/CityGML-ADEs

[20] Cityjson specifications 1.1.3. [Online]. Available: https://www.cityjson.org/specs/1.1.3

[21] Z. Yao, C. Nagel, F. Kunde, G. Hudra, P. Willkomm, A. Donaubauer, T. Adolphi, and T. H. Kolbe, "3dcitydb-a 3d geodatabase solution for the management, analysis, and visualization of semantic 3d city models based on citygml," *Open Geospatial Data, Software and Standards*, vol. 3, no. 1, pp. 1–26, 2018.

[22] T. Kutzner, K. Chaturvedi, and T. H. Kolbe, "Citygml 3.0: New functions open up new applications," *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 88, no. 1, pp. 43–61, 2020.

[23] P. Wate and V. Coors, "3d data models for urban energy simulation," *Energy Procedia*, vol. 78, pp. 3372–3377, 2015.

[24] O. K. Iseri, A. Duran, I. Canlı, C. M. Akgul, S. Kalkan, and I. G. Dino, "A method for zone-level urban building energy modeling in data-scarce built environments," *Energy and Buildings*, vol. 337, p. 115620, 2025.

[25] D. Perez, "A framework to model and simulate the disaggregated energy flows supplying buildings in urban areas," Ph.D. dissertation, EPFL, 2014.

[26] G. Agugiaro, J. Benner, P. Cipriano, and R. Nouvel, "The energy application domain extension for citygml: enhancing interoperability for urban energy simulations," *Open Geospatial Data, Software and Standards*, vol. 3, pp. 1–30, 2018.

[27] N. Luo, X. Luo, M. Mortezazadeh, M. Albettar, W. Zhang, D. Zhan, L. Wang, and T. Hong, "A data schema for exchanging information between urban building energy models and urban microclimate models in coupled simulations," *Journal of Building Performance Simulation*, vol. 18, no. 3, pp. 333–350, 2025.

[28] F. Rehmann, M. Mosteiro-Romero, C. Miller, and R. Streblow, "Enhancing urban energy modeling: A case study of data acquisition, enrichment, and evaluation in berlin," *Energy and Buildings*, p. 116070, 2025.

[29] M. Issermann, F.-J. Chang, and P.-Y. Kow, "Interactive urban building energy modelling with functional mockup interface of a local residential building stock," *Journal of Cleaner Production*, vol. 289, p. 125683, 2021.

[30] T. Charan, C. Mackey, A. Irani, B. Polly, S. Ray, K. Fleming, R. El Kontar, N. Moore, T. Elgindy, D. Cutler *et al.*, "Integration of open-source urbanopt and dragonfly energy modeling capabilities into practitioner workflows for district-scale planning and design," *Energies*, vol. 14, no. 18, p. 5931, 2021.

[31] P. R. Mazzarino, S. Finocchiaro, L. Barbierato, D. S. Schiera, L. Bottaccioli, and E. Patti, "An automated tool for urban building energy modelling: from sparse datasets to cityjson," in *2024 IEEE International Conference on Environment and Electrical Engineering and 2024 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 2024, pp. 1–7.

[32] L. De Lauretis, "From monolithic architecture to microservices architecture," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 93–96.

[33] L. Bottaccioli, E. Patti, E. Macii, and A. Acquaviva, "Gis-based software infrastructure to model pv generation in fine-grained spatio-temporal domain," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2832–2841, 2017.

[34] P. R. Mazzarino, M. Capone, E. Guelpa, V. Verda, L. Bottaccioli, and E. Patti, "A multi-agent framework to evaluate energy flexibility in district heating networks," in *2022 IEEE International Conference on Environment and Electrical Engineering and 2022 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, 2022, pp. 1–6.

[35] $\mathrm{open}_e Go{-}Team$, "$ding0 : Distributionnetworkgenerator|documentation($ $//dingo.readthedocs.io/en/dev/, revision656f85b6. Accessed$: $2025-08-14$.

[36] Regione Piemonte, "Geoportale piemonte," accessed: 2025-08-14. [Online]. Available: https://www.geoportale.piemonte.it/

[37] Istituto Nazionale di Statistica (Istat), "Istatdata — piattaforma per la diffusione dei dati," istatData è la nuova piattaforma; i dati da I.Stat (dati.istat.it) sono stati migrati qui. Accessed: 2025-08-14. [Online]. Available: https://esploradati.istat.it/