A Lexical Approach to Predictive Carbon Analysis via Machine Learning Models: *Early-stage Carbon Observer*

(ECO)

Daniel Favour O. Oshidero

Master of Science in Computer Science
The University of Bath
2023/24

A Lexical Approach to Predictive Carbon Analysis via Machine Learning Models: *Early-stage Carbon Observer*

(ECO)

Submitted by: Daniel Favour O. Oshidero

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

The ambition to achieve sustainable architecture faces significant challenges, particularly during the early design stages where critical decisions often lack sufficient environmental analysis. This thesis addresses part of this gap by exploring a prototype of an Early-stage Carbon Observer (ECO), a machine learning (ML)-based tool designed to predict total embodied carbon emissions from architectural design descriptions. The approach involved generating synthetic datasets for the testing of various ML models. Histogram-based Gradient Boosting (HGB) was chosen for its observed robust performance, with an integrated system of Natural Language Processing (NLP) techniques to convert complex, unstructured text into structured data that the HGB model could process. Thorough evaluations of ECO's performance then demonstrated its consistent ability to rank design options by their carbon intensity, although a general tendency to overestimate absolute carbon values was observed.

Simply put, this study explores a prototype with potential to reshape the Architecture, Engineering, and Construction (AEC) industry by embedding sustainability considerations directly into the early design process via a lexical approach. Despite some identified areas for improvement, such as improved feature extraction, and potential further functionalities, such as backward prediction, ECO's initial version demonstrates high usability, earning a System Usability Scale (SUS) score of 84.74. This reflects strong user satisfaction and potential for integration into existing workflows, emphasising its promise as a tool for advancing sustainable architectural practices.

A paper on this work is currently being prepared for submission to *Building and Environment*.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Professor David Coley, Professor of Low Carbon Design at the University of Bath, and Professor Michael Tipping, Professor of Machine Learning, for their invaluable guidance and support throughout this research.

I am especially thankful to Professor David Coley for his expertise and insights, which were crucial in shaping the direction and focus of my work. I am also profoundly grateful to Professor Michael Tipping and the Computer Science Department for their encouragement and support wherever possible.

This thesis would not have been possible without their mentorship and assistance.

Additionally, I would like to extend my sincere thanks to the individuals who tested the tool described in this thesis, as well as those who assisted with proofreading.

Finally, I would like to express my heartfelt appreciation to my family for their unwavering support, understanding, and encouragement throughout this journey. Their belief in me has been a source of strength and motivation.

1. Introduction

There exists an elegant solution for providing realistic carbon grounding to the early, unrealised narrative of a building. Within the realm of architecture and construction, or more specifically sustainable construction, the ambition to create structures with minimal carbon footprint confronts a significant challenge: the early design stages often lack the necessary analysis to guide projects towards low-carbon outcomes. This challenge is rooted in the traditional separation of the conceptual design stage and environmental impact analysis. By the time a building's design is advanced enough to undergo detailed modeling for energy and carbon efficiency, significant architectural decisions — those with profound implications for the building's environmental footprint — have already been cemented (Weng, Ramallo-González and Coley, 2014).

With buildings accounting for approximately 37% of carbon release (see Figure 1.1), the Architecture, Engineering, and Construction (AEC) industry bears significant responsibility in the reduction of global emissions (United Nations Environment Programme and Yale Center for Ecosystems + Architecture, 2023). Greenhouse gas emissions (GHG) encompass both embodied emissions (EE) and operational emissions (OE), with EEs referring to GHGs emitted during material production and construction, and OEs referring to GHGs emitted during building use (Fenton, De Rycke and De Laet, 2023). Basic adjustments to design elements such as modifying the primary material, core design shape, or orientation can significantly boost energy efficiency, potentially reducing total GHG emissions by up to 40% (Wang, Rivard and Zmeureanu, 2006; Ying and Li, 2020). This disconnection between early design choices and their long-term environmental consequences leads to a paradox where buildings, despite initial low-carbon aspirations, may end up far removed from their sustainable targets.

A rise in accessible toolkits based on predefined formulas for GHG modeling has addressed part of this gap. Such tools provide designers with insights into the operational and embodied implications of their projects. However, while these formula-driven tools represent significant progress, they are based on static calculations rather than dynamically generating material masses or embodied carbon through data-driven narratives. This approach, while beneficial in later stages where feature details are more substantiated, still leaves room for integration into earlier design phases given the absence of forecasting capabilities. Such circumstances result in potential missed opportunities for enhanced efficiency from the design outset.

This paper's research aims to narrow the gap in sustainable architectural integrations with the investigation and development of a text-to-carbon machine learning (ML) pipeline, and integrated tool, for the highly accurate prediction of early-stage embodied GHG impacts and future absorption into existing architectural design processes.

As a sub-field of AI, machine learning models the relationships between vast amounts of data to solve problems effectively, offering a potential use-case in the reassimilation of sustainability principles into early-stage architecture (Helm et al., 2020). The intrinsic integration of computing machines across every industry, and the substantial increase in our volumes of data, a concept now termed *big data*, has allowed AI to evolve into a field that enables machines to solve increasingly complex problems through learning and reasoning (Xu et al., 2021; Neri et al., 2020). A well-known application of this in the AEC is the use of machine learning in smart building technologies to optimise energy consumption by analysing usage patterns and predicting future needs. This is an approach used to reduce waste and promote sustainability while maintaining comfort (Chen et al., 2023; Tien et al., 2022). Investigations of this nature clarify a more defined trajectory toward global sustainability ambitions.

The solution is inspired by the ZEBRA toolkit (2022), and the recognition, as discussed in research by Weng, Ramallo-González and Coley (2014), that the most impactful environmental decisions are made before pen touches paper, not after.

1.1 Objectives

ECO (Early-stage Carbon Observer) is proposed as an intelligent model capable of translating early-stage design descriptions into their associated embodied carbon footprints for each building lifecycle stage, from material production (A1-A3), through construction (A4-A5) and operational use (B1-B7), to the end-of-life processes (C1-C4). It draws from generated datasets, establishing an automated pipeline that employs natural language and regression methodologies for final estimations. The object is to integrate with existing workflows and overall design processes by providing designers with immediate and realistic feedback on the implications of their initial concepts. ECO's machine learning components are the backbone to achieve these goals, designed to enhance the architectural design process with predictive capabilities.

The tool aims to:

- Employ Natural Language (NL) extraction to convert textual design descriptions directly into actionable data, for simplified user interactions.
- Employ an ideal regression model(s), in conjunction with any further requisite algorithms, to produce precise estimates from the extracted data.

As a *proof of concept*, an end-user forecasting tool is envisioned, enabling designers to rationalise between their architectural imagination and their environmental responsibility. With buildings essentially existing as semi-permanent marks on our environmental landscape, the ability to presage the future carbon impact of our decisions empowers the creation of more sustainable designs from the outset.

The tool is fundamentally rooted in the application of Computer Science and Machine Learning (ML) to confront pressing environmental challenges within the field of architecture. Through the establishment of a comprehensive prediction pipeline, ECO aims to accurately calculate embodied emissions based on initial design prompts. It will utilise natural language approaches such as named entity recognition (NER) for identifying and classifying key terms related to building features, semantic similarity analysis for understanding and comparing the meanings of words and phrases, and regular expression-based text extraction. The extracted building features — material types, building dimensions, and general specifications — are then to be integrated into an appropriate predictive framework capable of handling non-linear relationships and patterns within the data.

An objective of particular interest is the emphasis on user experience. The tool must have an interface that is both intuitive and user-friendly, providing real-time and dynamic feedback to its users. The hope is to enable rapid iterations of designs, catering to a diverse range of users from novice students to seasoned designers. In ensuring success, it is critical to validate both the tools' predictions against real world data, and its applicability in typical workflows. Establishing frameworks to determine its error margins will support in refining the model's accuracy. This ensures that ECO can reliably support sustainable design decisions in practice. Testing is then necessary to confirm its usability within the design process.

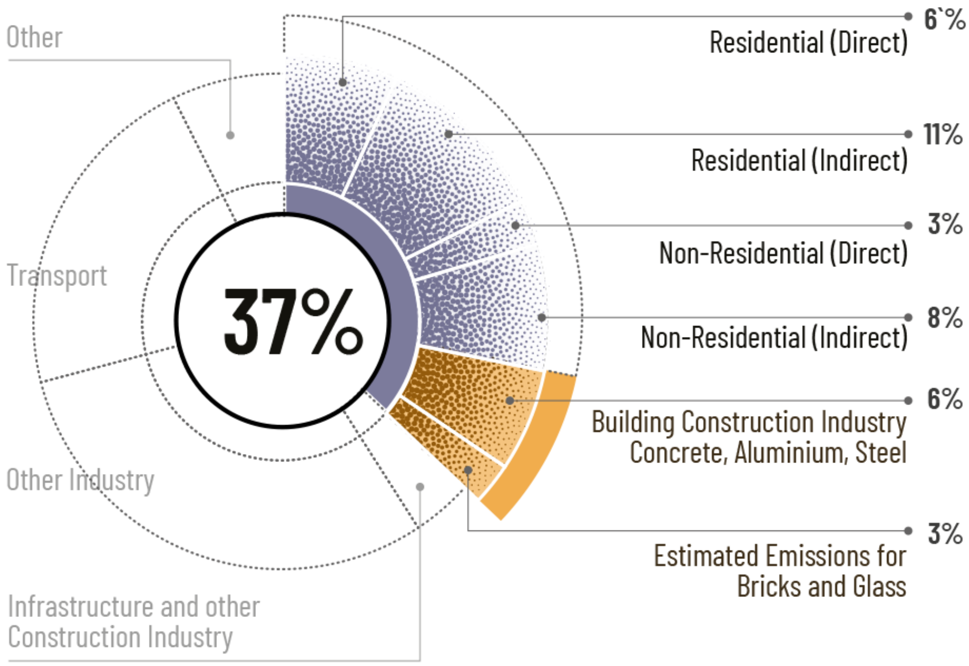

Figure 1.1: Global share of buildings and construction operational and process CO2 emissions, 2021.
(United Nations Environment Programme and Yale Center for Ecosystems + Architecture, 2023)

2. Literature and Technology Survey

2.1 Overview of Sustainable Architecture

Over the past two decades, sustainable architecture has emerged as an essential discipline that is integrally connected with the broader objectives of environmentally friendly infrastructure. This field reflects ongoing discourse within the architecture and design community concerning its significant influence on future trends and practices (Keitsch, 2012). Motivated by the need to address the challenges of environmental stewardship and the impacts of construction on the planet, sustainable design compels architects to reduce the operational and embodied footprint of their buildings. This includes enhancing energy efficiency and adapting the use of materials and development practices to align with local climates and geographic conditions (Keitsch, 2012; Donovan, 2020). The concept of sustainable architecture, however, is dynamic and resists any singular definition, embodying a complexity that defines the discipline (Donovan, 2020).

Sustainability generally extends past simple environmental efficiency, incorporating social and cultural dimensions that foster a holistic approach to development. From vernacular architecture that incorporates locally sourced materials and passive solar design, to modern innovations such as smart buildings and the use of recycled materials, these dimensions emphasise not only ecological and economic concerns but also the well-being and inclusivity of the communities that inhabit these spaces (Pons-Valladares and Nikolic, 2020; Donovan, 2020). This approach enables the creation of spaces that meet ecological standards whilst enhancing social inclusion and cultural connectivity (Keitsch, 2012). The architectural discipline views sustainable design not as a set of prescriptive measures but as an attitude or approach that should be inherently integrated into architectural practice.

Keitsch (2012) argues that the unification of sustainability into architectural education is crucial for cultivating a generation of architects who are adept in sustainable practices. This integration ensures that new professionals are not only skilled in traditional design but also in applying sustainable solutions that meet contemporary environmental and societal challenges. Key to this perspective is the application of interdisciplinary methods that address the complex sustainability challenges at a global scale. In the exploration of sustainable architecture, significant importance lies in the assessment and integration of various sustainability evaluation methodologies. Life Cycle Assessment (LCA), Multi-Criteria Decision Making (MCDM), and sustainability rating tools are extensively utilised to measure the environmental, social, and economic impacts of architectural projects (Pons-Valladares and Nikolic, 2020). These facilitate informed decision-making throughout the design and construction processes, ensuring that sustainability is embedded from the earliest stages of architectural development. Moreover, the integration of Building Information Modeling (BIM) with sustainability assessment tools represents a significant advancement in the field, enabling more precise and efficient design processes. BIM facilitates a deeper understanding of the lifecycle impacts of building projects, from construction through to demolition, thus promoting strategies that significantly reduce carbon footprints and enhance energy efficiency (Pons-Valladares and Nikolic, 2020).

However, despite these advancements, the challenge of sustainability in the architectural curriculum and professional practice remains. The ambiguity and variability in definitions and standards across different regions call for a more unified and clearer framework that can be universally adopted. This would not only streamline sustainability assessments but also enhance the global applicability of sustainable practices in architecture (Donovan, 2020).

2.2 Machine Learning in Sustainable Design

An integration of machine learning in sustainable architectural design not only focuses on simple technological advancements, but also represents a re-evaluation of the normative and ethical frameworks necessary for sustainable development. As suggested by Keitsch (2012), the utilisation of tool-based know-how such as life-cycle assessment and material-flow analysis, although technologically adept, may still require reconciliation with previously discussed normative principles to effectively guide sustainable practices. This reflects a broader debate in the field that while ML provides powerful tools for data-driven decisions, they must align with sustainable ethical standards to ensure that technological progress does not overshadow environmental or social equity concerns (Arroyo, Schöttle and Christensen, 2021).

In practical terms, ML's robust predictive capabilities offer significant advantages in sustainable architecture. Renewable energy forecasting (Seyedzadeh et al., 2018), accelerated materials discovery and enhanced solar harvesting (Sarker, 2021), as well as disaster management (Chen, Chiang and Storey, 2012) are among a number of studies which elucidate the efficacy of machine learning methods in resolving various sustainability tasks. Mohammed, Ahmed and Hacimahmud (2023) for example, outline a framework leveraging big data and ML to enhance current renewable energy forecasting and infrastructure efficiency. The idea utilises deep learning algorithms for uncovering complex, non-linear patterns by comparing historical data with real-time data, identifying model energy consumption patterns and predicting future needs with high precision. This approach highlights a key emphasis of this study: the broader trend of maximising data-driven insights for more accurate predictions and efficient resource management. Meanwhile, Seyedzadeh et al. (2018) review various ML techniques that optimise building energy consumption, demonstrating how neural networks, support vector machines, and clustering algorithms can substantially reduce energy usage and, by extension, carbon emissions. These methodologies not only predict but also actively inform strategies to enhance energy efficiency through their ability to recall learnt data, thus proving the potential of ML in unifying technological innovation and sustainable design, though on more specialised scales.

The contrast between the two approaches lies in their differing focuses: forecasting versus optimisation. Mohammed, Ahmed and Hacimahmud (2023) emphasise a prognostic approach, utilising supervised learning algorithms like regularised linear regression and support vector machines for tasks such as energy consumption foresight and biodiversity change detection. They argue that while deep learning models offer a state-of-the-art performance, simpler models like Decision Trees may be more suitable in the use case as the interpretability of the algorithms' decisions is crucial. This forward-looking strategy is crucial for design planning, with forecasting accuracy increasing as more data becomes available over the progression of work. This allows for iterative refinement of models and strategies based on emerging patterns. On the other hand, Seyedzadeh et al. (2018) concentrate on optimisation, specifically improving efficiency using existing data. Their methods are geared towards remediative enhancements within buildings, discussing several machine learning models that might be proficient in optimising energy use. For example, Genetic Algorithms (GA) can optimise HVAC control strategies by selecting the best operational parameters, while Artificial Neural Networks (ANN) can analyse energy usage patterns to enable system adjustments. They then propose a framework for selecting the optimal model for different use cases. However, while the methods are effective for producing immediate results, they remain inherently reactive due to a heavy reliance on current data without the provision for future changes.

2.3 Predictive Approach for Embodied Emissions: Case Studies

The emergence of predictive modeling has revolutionised the computation of EEs in the building sector through the integration of machine learning. As earlier noted, this shift is primarily driven by the recognised inadequacies within traditional late-stage assessments, which typically restrict the potential for implementing carbon reduction measures effectively (Su et al., 2024). These conventional approaches often necessitate detailed, volume-specific data, which may not be available during the early design stages. Recent discourse has witnessed diverse approaches in predictive modelling for embodied carbon, with each methodology typically commencing with the procurement of an extensive dataset comprising information such as building geometry, material specifications, and construction loads.

One rationale for adopting machine learning approaches lies in the ability to predict future emissions using instead what Fenton, De Rycke and De Laet (2023) refer to as “soft” features, including building type or gross floor area, among others. This approach applies multiple regression models, notably the development of "blender models" which assimilate the best-performing base models, attaining prediction accuracies as high as 91%. The technique is particularly effective in scenarios where detailed material data is unavailable, enabling early-stage assessments that can guide design decisions.

Similarly, Su et al.’s (2024) approach emphasises the influence of certain "critical" building materials on embodied carbon emissions. Their model evaluates 30 factors categorised across project, construction, and management levels, employing algorithms like ANN, Support Vector Regression (SVR), and Extreme Gradient Boosting (XGBoost) to predict the embodied carbon of materials such as concrete, steel, and wood. Validation using a dataset from buildings in the Yangtze River Delta, China, confirms the model’s accuracy, particularly highlighting the effectiveness of the XGBoost algorithm for predicting carbon emissions in concrete. This is a region-specific focus which calls attention to the importance of considering local construction practices and material availability when developing predictive models.

Pomponi et al. (2021) adopt a different methodology, replacing conventional BIM finite element analysis (FEA) methods — a computational technique used to simulate and analyse the physical behaviour of structures by breaking them down into smaller, simpler parts called finite elements — in favour of surrogate ML models to estimate EEs. By investigation of a range of regression models, including Random Forest (RF) and ANN, this approach first predicts structural masses, which then serve as the basis for calculating embodied carbon. The developed tool, available as both a SketchUp plugin and standalone software, integrates uncertainty analysis, allowing users to input probability distributions for embodied carbon coefficients. This real-time estimation capability is particularly valuable for architects and engineers, facilitating informed decision-making during the design process. Although, this approach might be seen as less predictive and more reactive in nature.

These tools demonstrate a progressive enhancement in the incorporation of ML into embodied carbon estimation processes. The utilisation of accessible soft features for early-stage predictions is particularly advantageous in scenarios where detailed material data is unavailable. This approach is crucial as a robust framework for guiding design decisions during phases where material or volumetric information is not available. Nevertheless, the consideration of all critical building materials and the inclusion of a wide range of influencing factors at various levels of a project facilitate a more nuanced understanding of how various elements contribute to embodied carbon. This is especially important in regions with distinct construction practices as local material availability and construction techniques must be considered.

Pomponi et al.'s approach stands out from the others with its real-time decision support capabilities. It enhances sustainability calculations by replacing current computationally expensive FEA methods with resource-saving ML models, reducing the time and computing power required for accurate assessments. It is a lot more practical for everyday use by professionals in the AEC. While distinct, these methodologies might complement each other in the creation of a multi-faceted approach to embodied carbon estimation. The continued refinement of the tools and their application across diverse design-construction contexts would be crucial in maximising their capabilities.

2.4 Evaluation of Existing Non-ML Toolkits

Building on the previous overview of formula-driven toolkits, a comparative analysis of prominent tools, such as the Zero Energy Building Reduced Algorithm (ZEBRA) suite and the FCBS Carbon Calculator might support in contextualising any current and future capabilities. With both tools being distributed as Excel spreadsheets, they offer the advantage of accessibility and ease of integration into existing workflows within architectural and design practices. This commonality underscores a recognition of the need for tools that can be easily adopted without significant skill development, changes to existing systems, or the necessity for complex software installations.

The ZEBRA suite is centered around two main tools, Zebra OC (Operational Carbon) and Zebra EC (Embodied Carbon), with an online version of Zebra OC offering limited functionality.

- **Zebra OC**: This tool is aimed at modeling the energy consumption and carbon emissions associated with the operational phase of a building. It aligns with standards used for designing certified Passivhaus buildings, indicating a focus on precision and adherence to recognised benchmarks in energy efficiency.
- **Zebra EC**: This focuses on the carbon footprint during the construction phase and demolition of a building, encompassing the whole life cycle but with particular attention to embodied carbon.

Both tools are designed to be accessible to a broad audience, including professionals and students without requiring specialised knowledge or software. This approach emphasises fast and approximate calculations that are sufficient for early-stage design decisions. The ZEBRA suite is optimised for early-stage design, allowing for quick evaluation of different options. The aim is to influence design decisions before they become too costly or complex to alter (University of Bath, 2022).

FCBS CARBON is a comprehensive tool designed for estimating the whole life carbon impact of a building. It also targets the early design stages to inform design decisions, using benchmarked data from the ICE Database and EPDs. It has a slightly friendlier user interface, producing graphical outputs to clearly communicate potential carbon impacts to the design team and stakeholders. This minimised approach aims to make complex carbon costing more understandable (FCBStudios, 2020).

Both toolkits are pivotal in advancing sustainable design, characterised by their accessibility and ease of use without necessitating specialised software. While ZEBRA facilitates rapid conceptual development through its modular approach and quick, approximate calculations, FCBS Carbon offers a holistic, comprehensive analysis from the project's inception, enriched with graphical outputs for more intuitive insights. Both tools are grounded in established benchmarks — ZEBRA with Passivhaus standards and FCBS Carbon with the ICE Database and EPDs — bolstering their credibility.

Additionally, several other resources are available: the AHMM Carbon Calculator (AHMM and redboxmedia, 2022) offers streamlined energy efficiency planning for buildings, "regenerate" — developed by the University of Sheffield (2020) — evaluates a building's adherence to circular economy principles, and the Passive House Planning Package (Passive House Institute, n.d.), also known as PHPP, is noted for its user-friendly approach to energy planning. These advancements in carbon impact assessment tools highlight the possibilities, and needs, for further integration with computational design software to allow dynamic adjustments as designs evolve. Advanced machine learning models capable of interpreting initial design inputs could significantly refine predictive analytics in early design stages.

3. Data Acquisition

For the purposes of this study, synthetic data generation was employed to compensate for the lack of publicly available real-world data pertaining to building carbon footprints. This process was executed through the utilisation of a VBA script integrated into the FCBS CARBON tool. The script facilitated the automation of the tool's typical input process, enabling the generation of a diverse dataset representing a vast range of building geometries and material specifications in relation to their respective carbon value. This permitted a thorough analysis despite the absence of actual empirical data. However, available real-world data was used for testing purposes to validate the final system's performance under various metrics.

3.1 Synthetic Data Generation

The script generates a comprehensive set of building attributes, encompassing various classifications such as sector (e.g., Housing, Office) and sub-sector (e.g., Flat/Maisonette, Single Family House). It also accommodates the selection of materials for essential structural elements, including piles, columns, roofs, and façades. Additionally, key physical dimensions are addressed, such as Gross Internal Area (GIA), building perimeter, floor height, the number of storeys above and below ground, and glazing ratios. Attribute selection operates by populating the calculator with random combinations in iterations. Each iteration represents a complete configuration of a building, where materials are selected for each element from a predefined set of options. This method ensures a broad exploration of possible combinations, contributing to the dataset's variability. For instance, during an iteration, the script might assign either "RC 32/40 (50kg/m^3 reinforcement)" or "Steel" to the "Piles" element. The script would also experiment with scenarios of missing elements, allowing blank selections for certain cells to address instances where a user might not provide an option. Logical constraints are then enforced to maintain realistic building configurations to some degree (refer to Appendix A.1 and A.2), such as preventing the selection of a "Raft" foundation if "Capping beams" or "Pile caps" have already been chosen, as a means of safeguarding against structural conflicts and ensuring that the selection process remains consistent with engineering best practices.

A combined total of 150,000 data points were generated for processing, distributed across three datasets, comprising two sets of 60,000 entries each and one set of 30,000 entries. Documentation of the loop responsible for generating these parameters is provided in Appendix G.1, with relevant code commencing on line 29.

3.2 Preprocessing Pipeline

During preprocessing, the multiple synthetic datasets were first combined into a single data frame, as a means of ensuring uniformity and facilitating simple management throughout. This was followed by normalising the data, which involved general cleaning to maintain data integrity, standardising terminology (e.g., renaming all "RC" typologies to “Reinforced Concrete"), and removing or simplifying superfluous information to focus on essential variables. Any information this extracted was then archived for future reference and verifications (refer to Appendix A.3). The final step then involved the preservation of the processed dataset in CSV format, prepared for any further inspection and training.

4. Methodology

4.1 Model Training and Validation

The trained regression model serves as one of the primary foci of this implementation, designed to meet the predictive expectations of ECO. The approach commences with the encoding of categorical features into a numerical format manageable by machine learning algorithms. It is then split into the refined feature set, **X_cleaned**, and target column, **Y_cleaned**. To identify the most effective predictive model, a range of algorithms were evaluated on a reduced subset of the data. This assessment included advanced supervised models such as standard Gradient Boosting Decision Tree (GBDT), Histogram-based Gradient Boosting Decision Tree (hGBDT, referred to as HGB for the purposes of this paper), and Random Forest (RF), highlighted for their robust capabilities.

Gradient Boosting Regression is particularly effective at improving prediction accuracy through iterative refinement of models by computing their residuals (errors), making it well-suited for capturing complex, non-linear relationships. Residuals represent the difference between the actual and predicted values:

$$r_i = y_i - \hat{y}_i$$

where r_i is the residual, y_i is the actual value, and $\hat{y}_i$ is the predicted value. The goal is to approximate the optimal predictive function F^* by minimising some loss function $L(y, F(x))$ over the joint distribution of all (y, x) values, expressed as:

$$F^* = \arg\min_F E_{y,x}[L(y, F(x))]$$

where $L(y, F(x))$ quantifies the overall error of the model based on these residuals (e.g. mean squared error). This gradual decrease in error is the key to cumulative improvements in predictive accuracy (Friedman, 2001). However, an iterative approach such as this can be computationally intensive. The Histogram variant further optimises performance by efficiently handling large datasets through the discretisation of continuous features, reducing the number of unique values by categorising nearby data points into *bins*, thereby accelerating computations (Guryanov, 2019).

On the other hand, Random Forest (RF) typically operates through a process known as bootstrap aggregation (also referred to as *bagging*), where multiple subsets of the data are randomly sampled with replacement to train individual decision trees. Each tree $T_b(x)$ is trained on a different bootstrap sample, with the final prediction being made by averaging (in regression, or majority voting in classification) the predictions of all trees in the forest, expressed as:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$$

where B is the number of trees in the forest. This enhances the model's resistance to overfitting and enables it to capture complex interactions between features (Lee, Ullah and Wang, 2019; Breiman, 2001). Additionally, Random Forest can provide valuable insights into feature importance, helping to identify the key factors driving the model's predictions. However, it should be noted that this method also demands higher computational resources (Mohapatra, Shreya and Chinmay, 2020). Other models, including Linear Regression, Ridge, Lasso, SVR, and ElasticNet, were additionally tested during the model selection sequence. However, these models exhibited significantly lower performance metrics with less predictive power on the synthetic dataset.

Hyperparameter tuning was then instrumental in the search for the most favourable model performance. Model training employed scikit-learn's RandomizedSearchCV as a hyperparameter optimisation technique, which randomly samples a specified number of parameter combinations from a given search space, rather than exhaustively evaluating all possibilities. This was applied to a subset of the data, allowing for exploration of a diverse range of potential configurations. By tailoring this search to each specific model, their most effective parameters could be found, enabling a fairer assessment of model capabilities. Refer to Figure 4.1 below for an illustration of this process. This assessment was based on their R-squared scores across training and testing datasets, with 5-fold cross-validation employed to ensure consistency and reliability in performance assessments. Among which, HGB was evident as the superior choice, exhibiting high accuracy and stability across a number of metrics. It displayed a training R-squared of 0.9882, testing R-squared of 0.9359 , and a mean cross-validation of 0.9480. In contrast, the standard Gradient Boosting method, despite having achieved a near-perfect training R-squared of 0.9999, suffered from overfitting, demonstrated through its lower testing score of 0.7556. The Random Forest model performed even less effectively, with training and testing values of 0.6377 and 0.4822 respectively. A detailed table of model performance is available in Appendix C.1.

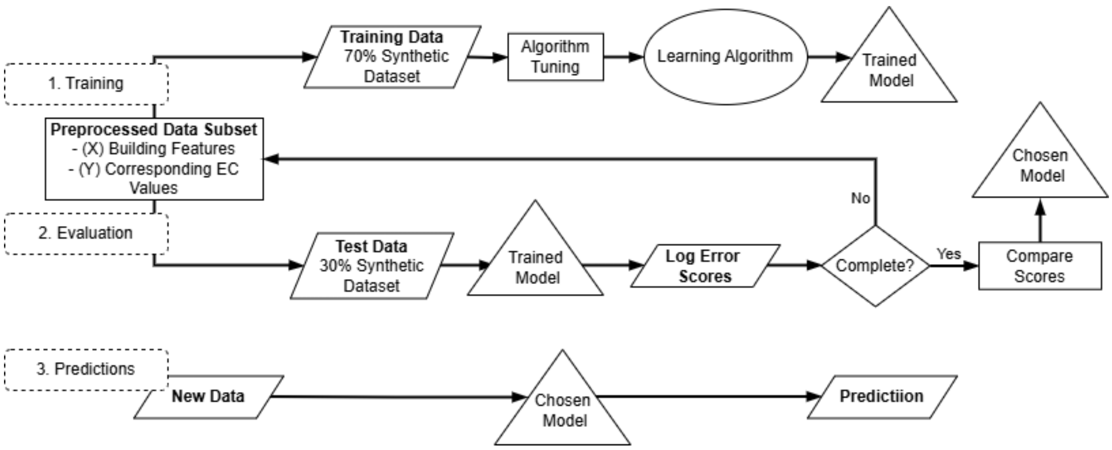

Figure 4.1: Illustration of model training process.

HGB was subsequently chosen as the final model due to its performance balance, with further reasons such as its ability to generate predictions regardless of incomplete data inputs, as well as significantly faster training times. This was then trained on the entire dataset to ensure comprehensive learning and further re-evaluated to confirm its generalisation capabilities on unseen data (see Appendix C.2). Additionally, the chosen model, along with all pertinent metadata – such as feature names, encoders, and performance logs, was stored to enable future integration into a final software.

4.2 Integrated Proof-of-Concept System

For prototyping purposes, the model is hosted on Hugging Spaces, a platform well known for its scalability and accessibility, particularly in deploying and managing machine learning tools (see Hugging Face (2024) for more information).

The full implementation is initiated through an API call from the client, which, in this study, is a user-accessible website (see the prototype client interface in Appendix B). The API is developed using Flask (2010) REST, a lightweight framework that facilitates RESTful services within Python environments. Upon receiving a request, it triggers the backend pipeline with the provided input. The server processes this input to compute a carbon value, expressed in kgCO2/m^2. This computation involves comprehensive data preprocessing, feature extraction, and a final generation of the prediction. The predicted values are then compiled and prepared for transmission back to the client (see Figure 4.2). To facilitate this investigation, the website is deployed through Netlify (n.d.), a platform that supports continuous deployment and hosting for modern web projects. However, the system is designed to support various frontend environments with minimal modifications.

4.3 Backend Pipeline

The backend pipeline is constructed to process user inputs, extract relevant features, and subsequently generate predictions related to early-stage carbon impacts of building designs. This pipeline integrates multiple stages of natural language processing (NLP) and machine learning, intended to transform textual design descriptions into actionable data. The process consists of several critical steps including text preprocessing, feature extraction and matching, conflict resolutions, and final prediction generation (see Figure 4.3). These are interconnected stages, working together to ensure predictions that are accurate and reliable, with an input/output that can be seamlessly integrated into most client interfaces.

4.3.1 Text Preprocessing

The sequence initiates with the receipt of user input text, typically provided by the application in a structured format such as JSON. This input serves as the primary data source for the pipeline. It must then be pre-processed as a means of normalising the input to ensure consistency and to reduce variabilities that might arise from differences in user formatting, structure, or content. The first step here is tokenisation, where the text is separated into *tokens*, in this case at the granularity level of individual words. This segmentation aids in the following analyses by dividing continuous streams of text into more manageable units. Tokenisation further enables the identification and extraction of meaningful patterns in the text, for example recognising that a term such as “New York” is a single entity rather than two words, helping to handle ambiguities in word determination. The text then undergoes the removal of stop words, where commonly used words such as “the” and “and” are eliminated from the text due to their minimal contribution to semantic meanings. This step reduces noise, enhancing computational efficiency in later stages.

Lemmatisation then transforms each token into their base or root forms, known as the *lemma* (e.g. “running” to “run”). Unlike stemming — a similar method which simply cuts off word endings — lemmatisation is sensitive to contexts, allowing it to determine a words root form more accurately. For effective lemmatisation, Parts-Of-Speech (POS) tagging is a necessary step, as a means of identifying parts of speech such as nouns or adjectives, for each token. This step filters out irrelevant words to focus only on those that add meaning to the text. Additionally, a synonym

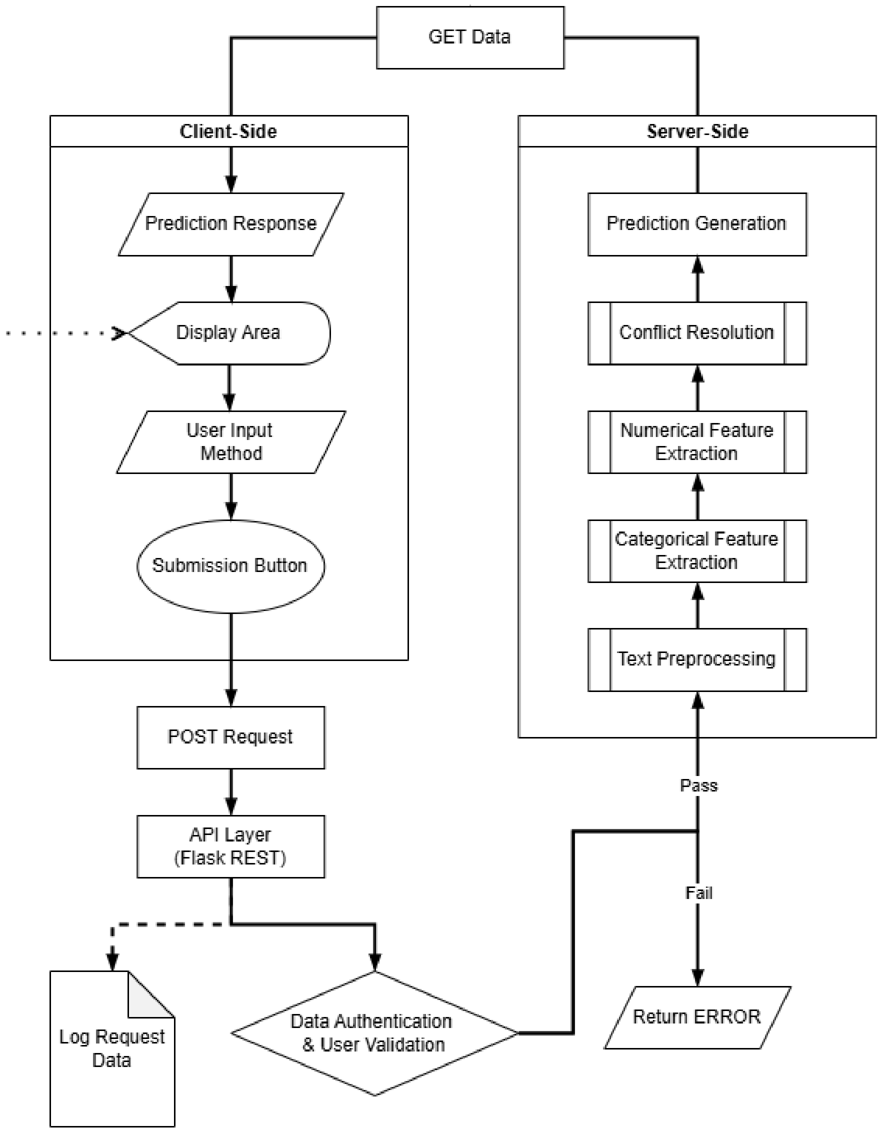

Figure 4.2: Integrated client-server pipeline for ML prediction.

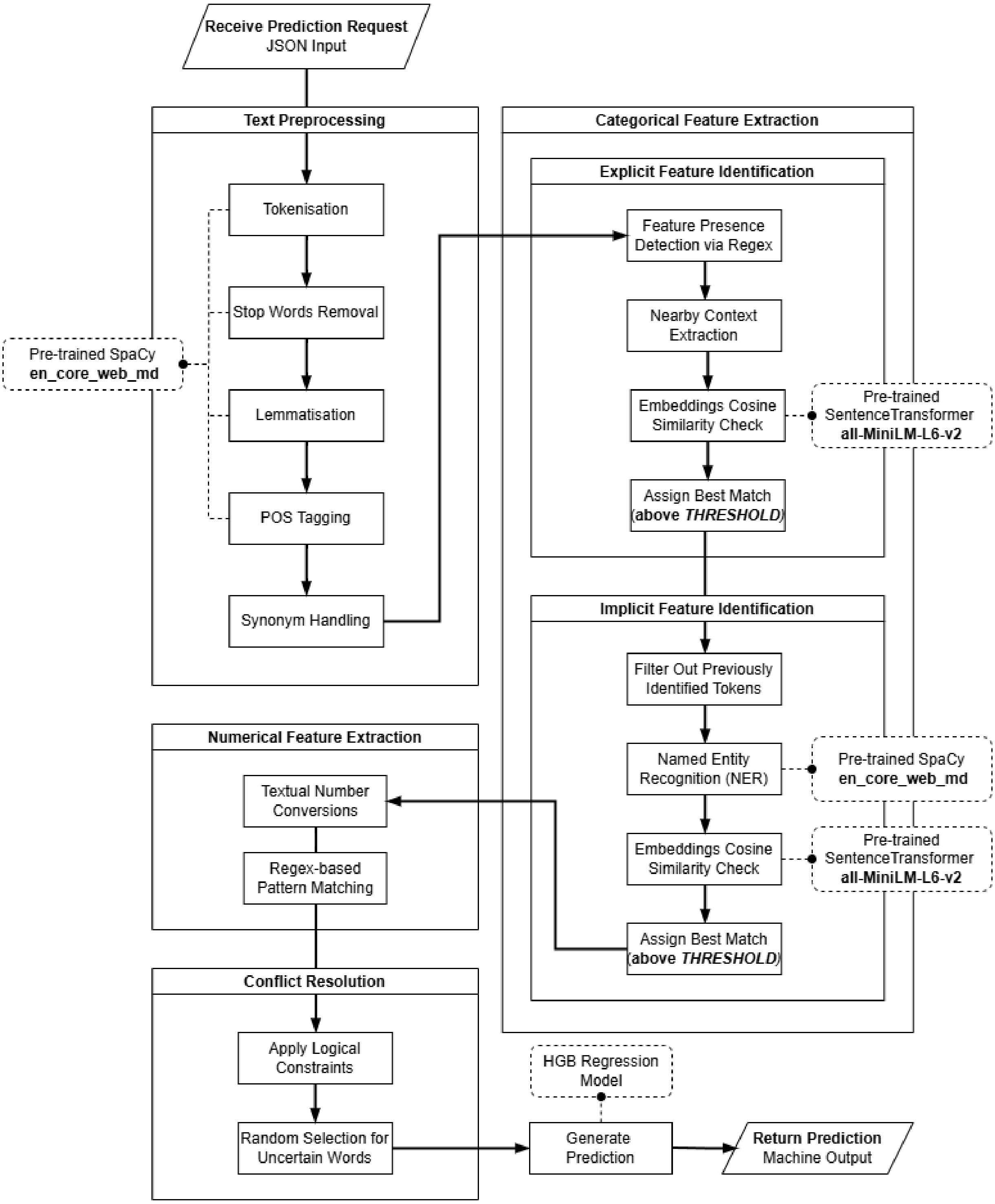

Figure 4.3: Flowchart of backend pipeline with associated models.

dictionary is employed to manage synonyms, ensuring consistency in the final representation of different expressions of the same feature. These preprocessing steps are facilitated through the use of spaCy pre-trained text models, and collectively produce a cleaned, standardised version of the user input, preparing the text for effective feature extraction. To illustrate, consider the sentence:

"A steel-framed concrete building, with large glass windows and a deep-piling foundation, designed to withstand strong winds and earthquakes, supported by roof trusses."

This might be reduced to:

"steel concrete building large glass window deep pile design strong wind earthquake roof truss"

Alternatively, depending on the exhaustiveness of the synonym dictionary, it might be rendered as:

"steel iron alloy metal concrete building large curtain glazing glass window pane deep pile pile caps substructure piles foundation design strong wind earthquake roofing roof truss"

4.3.2 Categorical Feature Extraction

Following preprocessing, the text is then processed with the support of two pretrained NLP models to extract categorical feature values. This process might be delineated into two distinct phases: explicit feature extraction and implicit feature extraction. Explicit features pertain to information that is readily apparent and clearly discernible within the text, e.g.

"The building has *concrete raft* foundations"

In this instance, the term "*raft*" may be simply plucked from the text, with the accompanying value "*concrete*". Implicit is slightly more subtle, requiring a more nuanced approach. A user might not directly articulate the feature name in a similar manner to how they have been predefined, or could also introduce spelling errors. For example,

"The *structure* will be supported by a *strong base* made of *reinforced conrete*."

Here, the user evidently intended to refer to "reinforced concrete" but made a spelling mistake ("conrete" instead of "concrete"). Additionally, the phrases "strong base" and "structure" do not explicitly denote a specific feature like "foundation," but rather imply its presence.

Explicit feature extraction begins with word matching, where the input text is scanned to identify occurrences of predefined features. This involves cleaning the feature names by converting them to lower case and removing irrelevant terms such as "material". A regex pattern is then used to locate these cleaned feature names within the input text. Regex extraction functions as a *specialised engine* for identifying entities that follow a syntactical structure, making it ideal for the explicit extraction task. They tend to excel at isolating desired text snippets within unstructured data, and efficiently capture relevant entities that follow predictable patterns (Bartoli et al., 2016). When a match is found, it strongly indicates the presence of a specific feature explicitly mentioned within the text.

Upon identification of a feature match, the surrounding words (within a proximity window of five words) are retrieved to provide the necessary context. This is crucial for accurately understanding the exact usage of the feature, thereby facilitating a precise determination of its corresponding value. The extracted context once again undergoes preprocessing steps of tokenisation and lemmatisation and, along with the predefined feature values, are converted into vector embeddings using a pretrained SentenceTransformer model for subsequent similarity calculations. These embeddings refer to numerical representations of the text that capture semantic meaning of the words or phrases, enabling a more accurate similarity comparison in a mathematical sense.

Implicit feature extraction functions in a similar manner, identifying features as well as any potential corresponding values in the given context, converting them into vector embeddings for computing. The key distinction in this approach lies in the initial identification of features, which is achieved through Named Entity Extraction (NER). NER is an NLP technique that detects and classifies specific terms within a text into predefined categories, helping to capture information by recognising entities that might imply a certain category or feature, even if not explicitly named. This method allows the system to consider a broader range of potential terms by avoiding restricting the extraction process to simply exact matches of predefined values. The entities identified through NER are treated as *candidate features*; as although they may not directly correspond to any feature names, they could still be contextually relevant. For instance, the term “timber” might imply various manners of wood, such as Glulam or CLT. These candidates, as with the exact matches, are passed to the SentenceTransformer model for conversion into embeddings.

SentenceTransformer is specifically chosen for these tasks due to its ability to generate vector representations of considerable semantic significance through its embeddings, which are essential for capturing nuanced relationships between texts. Its compatibility with cosine similarity calculations, coupled with its efficient performance and versatility, further justify its selection. The cosine similarity between these embeddings is computed to assess the degree to which the matched feature aligns, in context, with the possible predefined values. The value that earns the highest similarity score is determined to be the best semantic match, provided it exceeds an established likelihood threshold of 70%. This threshold is set to achieve balance between precision and recall, ensuring that only items of relevant similarity are considered matches, while minimising false positives.

These semantic matches are confirmed through cosine similarities, a widely adopted metric in information retrieval. It quantifies the similarity score between two vectors (in our case, the embeddings of words or phrases). The approach has use cases in fields such as search engines and relevance feedback (Rocchio and Salton, 1965), image assessments/retrieval (Sadbhawna et al., 2022), and extractive summarisation (Salton et al., 1997). The formula for calculating the cosine similarity between two vectors A and B is represented as:

$$CosineSimilarity = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|}$$

where $\mathbf{A} \cdot \mathbf{B}$ denotes the dot product of the vectors, and $|\mathbf{A}|$ and $|\mathbf{B}|$ are their respective magnitudes. Despite its utility, the cosine similarity method used alone has a notable limitation: it may underestimate the similarity of certain different terms that have similar meanings. This limitation arises due to traditional cosine similarity only accounting for the angle between vectors, with no room for semantic relations (Zhou et al., 2022; Steck, Ekanadham and Kallus, 2024). Therefore, it is necessary to complement this method with semantic approaches, such as employing a context window or utilising semantically-aware vectors, amidst other researched methodologies (Navigli and Martelli, 2019; Apallius De Vos et al., 2021).

Finally, each feature is assigned the confirmed match, which is saved to a dictionary. This dictionary collates all the identified features and their corresponding matched values. Any tokens related to these identified features are then filtered out from the text, leaving the remaining text for further numerical processing.

4.3.3 Numerical Feature Extraction

The extraction of quantitative data is then crucial for capturing values like dimensions, quantities, or other numeric attributes that might be mentioned within the input. This method utilises a regex pattern to discern numbers within the text, irrespective of trailing information such as '*sqm*' (square meters), '*kg*' (kilograms), or '*ft*' (feet). This ensures the accurate capture of all numerical values, whether standalone or followed by a unit. The text is also tokenised, with any word representing a number in textual form (e.g. "five" instead of "5") further converted into its numerical counterpart, enabling the system to effectively process both numeric and written number formats.

Once potential numerical values have been identified, the function systematically examines each word, determining whether it aligns with any predefined numerical feature names by utilising sentence embeddings. To enhance the scope of its search, this method similarly leverages a synonym dictionary for a more comprehensive identification process. When a keyword is found, a window of three words surrounding it is examined to locate any associated numerical values. If a match is found using the regex pattern, the value is extracted and added to a dictionary along with its corresponding feature. Upon assigning all relevant values, the function consolidates them for each feature, assigning *None* to any without a value.

Additionally, a special rule is applied to handle the particular scenario of "Storeys Below Ground". In instances where the text mentions "*a basement*" without specifying a numerical value, the function automatically assigns the value of 1 to this feature, reflecting the presence of at least one below-ground level. The extracted data is then combined into the feature-value dictionary.

4.3.4 Final Steps

To maintain practicality and validity of the predictions, effective conflict resolution is essential. Logical rules are employed to handle specific conflicts that may emerge in the assigned features or that may not be feasible in real-world constructions. For example, determining the appropriate type of joisted floors used in predictions is dependent on whether the user's building is deemed to be Residential or Non-residential. Features are also dynamically adjusted based on contextual conditions. For instance, if "Piles" are absent, related components such as "Pile caps" or "Capping beams" are also automatically set to *None* to avoid inconsistencies.

In situations where the model cannot decisively choose between two or more features, randomness is introduced to resolve ambiguity. This often occurs when the context is unclear, leading to the assignment of incompatible elements , such as "Raft" and "Pile caps". In such a case, the model employs a controlled random selection process, ensuring that it does not stall or return an error. Instead, it makes a decision that is somewhat contextually appropriate. Importantly, these random choices are still made within the constraints of the predefined logical rules, ensuring the outcomes remain relevant.

Once all features have been processed and any conflicts resolved, they are passed into the HGB regressor model for prediction. The model generates a final prediction with a high degree of accuracy. It is then formatted appropriately to meet the expected output standards, packaged as a response and sent back to the client. This prediction can then be utilised for further processes or displayed as needed.

5. System Evaluation

The ECO system has the potential to offer significant advantages by integrating carbon considerations directly into the early stages of the design process. Evaluations of the system's performance tested its efficacy across several key areas such as robustness to variations in phrasing of descriptions, sensitivity to fundamental input parameters, and accuracy of predictions. Moreover, the real-world relevance of the tool is examined through user testing and system usability studies, providing a comprehensive understanding of its applicability as a tool for advancing sustainable architecture.

5.1 Testing Criteria

The first area of focus was assessing the extraction sensitivity of ECO, to ensure that it correctly extracts relevant features from input text. A detailed analysis was conducted to evaluate the tool's performance in identifying changes in key descriptive parameters such as material type or building dimensions. This analysis was crucial to gain insights into the underlying mechanics of the pipeline, ensuring that it responded appropriately to varying design inputs. Major building variables, including structural materials, external finishes, and numerical specifications such as GIA, were systematically changed. The correctness of the associated feature extraction was observed. The objective was to verify that the ECO tool consistently extracts and applies the correct features, leading to reliable and accurate analysis.

Accuracy was then a critical metric to explore, as a means of ensuring that the software provides reliable data for architects to base their sustainable designs on. To assess this, a dataset comprising real-world architectural designs with known embodied carbon footprints was tested as a benchmark. These designs were obtained from a selection of sustainability benchmark projects, as documented in the LETI Embodied Carbon Case Studies (LETI, n.d.). Inputting summary design descriptions of these structures into ECO allowed for a comparison between the software's predictions and the actual embodied carbon values. Further, plotting these provided a clear illustration of the predictive deviation and the overall fit of the model. The expectation was that ECO would demonstrate a reasonable accuracy, with a predictive trend aligning with calculated embodied carbon.

In addition, the robustness of ECO to variability in design descriptions was another critical aspect of evaluation. Given the diversity of language used in architectural design, it was essential to ensure that the software could generate consistent and accurate predictions regardless of how a building might be described. To test this, multiple textual descriptions of the same key features were created for a set of building designs, varying in terminology and phrasing. These descriptions were then input into the software, and the consistency of the predictions was assessed by analysing the variance in produced embodied values. Key metrics such as the standard deviation and coefficient of variation were used to measure this consistency. This was crucial to confirm that ECO would exhibit minimal variation in prediction across different linguistic contexts.

Finally, the real-world applicability of the tool was evaluated through anonymous user testing and system usability assessments. Testing involved a number of individuals in the design industry, of varying experience levels, using ECO in their design process and providing feedback. Their feedback on the software's usability and their perceived accuracy of its predictions was collected and analysed. Additionally, the System Usability Scale (SUS) was used to quantitatively measure the software's usability and overall user-friendliness. The results of these evaluations are a measure of how well ECO integrates into existing workflows, with high user satisfaction showing that it has potential for effective support in sustainable design decision making.

5.2 Results

5.2.1 Extraction Sensitivity

The sensitivity analysis conducted through variations in building descriptions revealed several insights into the effectiveness of the ECO tool in handling unique cases of textual inputs. The tool generally performed well in identifying primary structural materials such as steel, reinforced concrete, and timber. It also identified external materials accurately, picking up façade elements like curtain walling and timber finishes. However, ECO seems to encounter difficulties when detecting corresponding element counterparts, such as pile caps or joisted floors, particularly when these features are not explicitly mentioned in the text. This also led to failures in the detection of related internal materials such as partitions. Additionally, the tool frequently misidentified roof finishes, assuming ceramic tiles where green roofs were specified.

Regarding general building information, the tool effectively captures numerical features like building perimeter and gross internal area when these are correctly formatted. However, it struggles with large numbers separated by commas, failing to interpret "15,000" as "15000." Further, ECO exhibits issues with semantically understanding certain phrases. For instance, it does not recognise "The project is an 8-storey office" as indicating the number of storeys above ground, with this outcome changing when the wording is altered to “The project is an office building with 8 floors”. This shows that numerical changes, especially in format, can lead to extraction errors, revealing a high sensitivity not only to content but also to how the information is presented. These observations further illustrated ECO's ability to detect data within extensive descriptions, as demonstrated by the average percentages of correctly and incorrectly "found" features across various major keywords (see Figure 5.1).

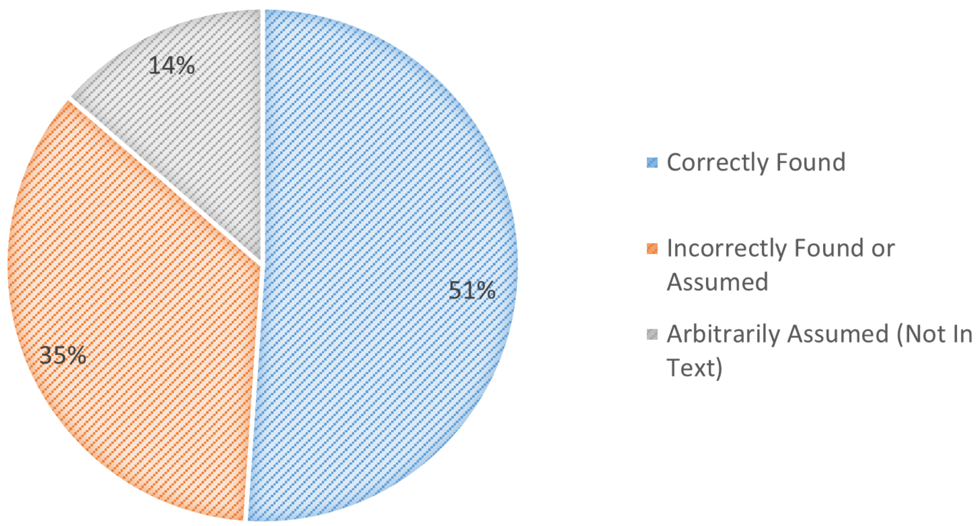

Figure 5.1: Average percentage of correctly and incorrectly "Found" features across tests.

The chart reveals that the tool accurately identified features 51% of the time, suggesting a modest level of accuracy. However, 35% of features were either incorrectly found or falsely assumed, and 14% were arbitrarily assumed by the tool—meaning that these features were neither present in the text directly nor semantically, and should not have been assigned. This distribution highlights that, while the tool achieves a reasonable success rate, there remains a considerable scope for improvement, specifically in minimising incorrect assumptions and enhancing the precision of feature identification. The detailed raw data for these findings can be found in Appendix D.

5.2.2 Accuracy Comparisons

Accuracy was confirmed by inputting summary design descriptions of a sample of 7 LETI case studies into the tool, and comparing the predicted ECs against actual calculated values. The results showed that ECO consistently overestimated carbon values across all cases (see Figure 5.2). For example, ECO predicted an embodied carbon value of 2208 kgCO2e/m^2 for Canal Reach, compared to an actual value of 1178 kgCO2e/m^2. Similarly, for Stephen Taylor Court, ECO predicted 819 kgCO2e/m^2, whereas the actual embodied carbon was 274 kgCO2e/m^2. This pattern of overestimation was observed across all the case studies, with the average standard deviation of residuals found as 573.05 kgCO2e/m^2.

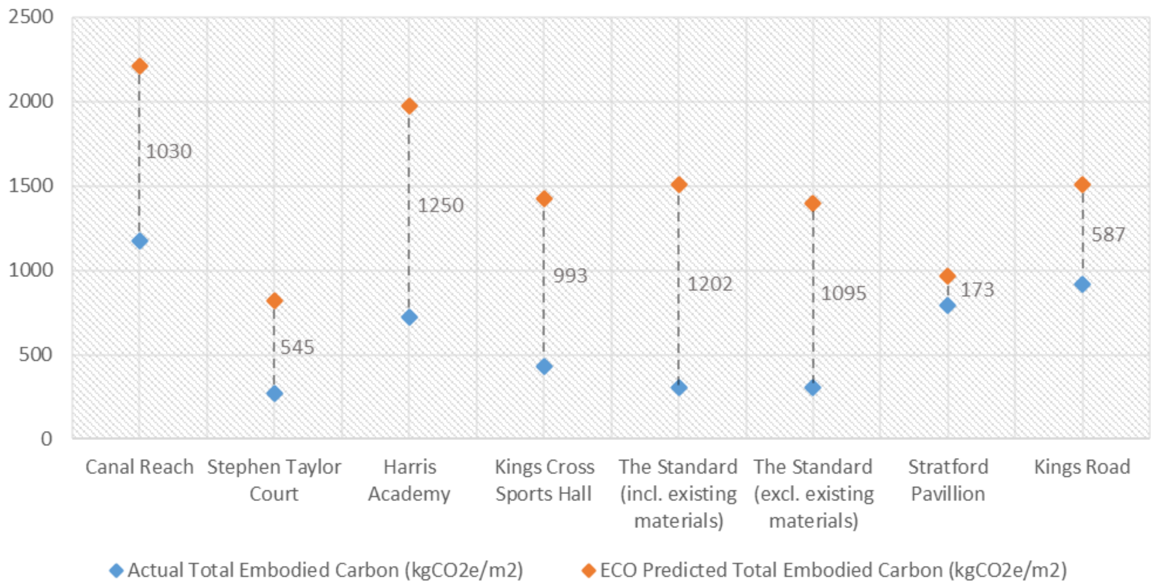

Figure 5.2: Comparison of Actual vs. ECO Predicted Total Embodied Carbon Values (incl. Retrofits)

To further quantify this effect, the standard deviations of both the actual and predicted embodied carbon values were independently calculated. The standard deviation for the actual values was 338.36 kgCO2e/m^2, while the standard deviation for the ECO predictions was 497.69 kgCO2e/m^2. This difference indicates that ECO's predictions exhibit a greater spread and are less consistent compared to the actual embodied carbon values. The higher standard deviation in ECO's predictions suggests that the tool introduces further variability, possibly due to its generalised nature and the absence of detailed input data.

The inflated carbon values predicted by ECO might then be partially attributed to the fact that the software's predictions are derived from high-level descriptions of the buildings, which inherently lack detailed information about smaller design elements that can significantly impact the carbon footprint. This contributes to the increase in overall variability. These elements include factors such as building form, passive ventilation and lighting strategies, use of recycled materials, or the sourcing of local materials. Such aspects, often maximised for sustainability in real-world designs, tend to lower the actual embodied carbon but would not be fully captured in the broad descriptions input into ECO.

Additionally, it could be due to the tool's current limitation in accurately gauging the scale at which each material is used, as ECO does not yet take material masses into account. Its broad generalisations may prevent the minimisation of embodied carbon that would typically occur in more detailed calculations. Further, the inclusion of operational carbon stages (B6 and B7) in ECO's predictions, which is absent from the case studies' calculations, might further contribute to the higher carbon values. However, the extent to which operational carbon impacts the total predicted carbon value remains uncertain. As a consequence, the tool seems to currently produce a conservative estimate, erring on the side of caution. While further study is required to fully understand and minimise the contributors to this overestimation, this may still be considered a positive aspect of ECO, as it avoids giving architects a false sense of security regarding the carbon impact of their designs. An overprojection ensures that designers are aware of the potential maximum impact, prompting them to consider further reductions.

Despite these overestimations, it is important to note that ECO followed a consistent trend that aligns with the relative differences in the actual embodied carbon values across the various designs, as shown through its high Spearman's rank correlation coefficient of 0.7143 (refer to Table 5.1). This indicates a moderately strong positive correlation between the predicted and actual rankings. While not perfect, the coefficient demonstrates that ECO is generally reliable in recognising the correct hierarchy of carbon intensity among the different designs. For instance, buildings with higher actual embodied carbon, such as Canal Reach and Harris Academy, were predicted by ECO to have higher values compared to less carbon-intensive projects like Stephen Taylor Court and The Standard. This consistency in relative ranking suggests that while the software may exaggerate the absolute values, it is correctly identifying the more carbon-intensive designs, which is crucial for comparative analysis of major features in early-stage design.

Case Study	*Actual (kgCO2e/m2)*	*Actual Rank*	*Predicted (kgCO2e/m2)*	*Predicted Rank*	*d*	d^2
Canal Reach	1178	1	2208	1	0	0
Stephen Taylor Court	274	6	819	6	0	0
Harris Academy	725	4	1975	2	2	4
Kings Cross Sports Hall	429	5	1422	4	1	1
Stratford Pavillion	793	3	966	5	2	4
Kings Road	920	2	1507	3	1	1
Rs Value					**0.7143**	

Table 5.1: Comparison of Actual vs Predicted Embodied Carbon Rankings, with Spearman's R_s Value (excl. Retrofits)

Furthermore, the variation in ECO's predicted values mirrored the variation in the actual values. For example, the difference between the predicted embodied carbon for Kings Cross Sports Hall (1422 kgCO2e/m^2) and Stratford Pavilion (966 kgCO2e/m^2) is similar to the difference in their actual values (429 kgCO2e/m^2 and 793 kgCO2e/m^2, respectively). This indicates that ECO responds to changes in design complexity or material usage in a manner that reflects real-world data, even though the absolute predictions are higher. In some cases, however, ECO's predictions were relatively closer to the actual values. For Stratford Pavilion, ECO predicted 966 kgCO2e/m^2, which is closer to the actual value of 793 kgCO2e/m^2, demonstrating that there are certain scenarios where the model's assumptions align more closely with the actual conditions, the predictions can be reasonably accurate. This suggests that the software has the potential to produce accurate predictions in certain contexts, depending on the specific design details and materials used.

The accuracy of ECO in retrofit or renovation projects does seem to present a further challenge. For designs that are made on top of an existing structure, such as the case of The Standard, the tool seems to struggle with accurately predicting the carbon savings associated with the reuse of existing materials. This causes the unusual divergence from the trend displayed in Figure 5.2, as ECO overestimated the embodied carbon significantly. This discrepancy highlights a limitation in ECO's current model, as it fails to recognise the reduced need for new construction materials when considering renovations. Consequently, ECO's predictions for renovation projects may not accurately reflect the lower embodied carbon associated with such projects, suggesting that further refinement is needed to handle the nuances of these scenarios.

Overall, while the ECO software currently exhibits a tendency to overestimate embodied carbon values, its ability to follow the general trend of actual values is a positive indicator. The consistent overestimation, due to the omission of smaller, yet impactful sustainability features, serves as a cautious estimator, ensuring that carbon footprints are not underestimated. Such a trend-following capability means that ECO is effective in providing a comparative analysis of changes in design decisions, which is particularly valuable in the early stages of design when relative differences between options are often more important than absolute accuracy.

5.2.3 Linguistic Robustness

The evaluation of ECO's robustness to linguistic variations revealed some variability in the predicted embodied carbon across different descriptions of the same building designs. For instance, as shown in Figure 5.3, Building 1 had a relatively low standard deviation of 46.57 kgCO2e/m^2, indicating that reworded descriptions — such as "The apartment complex is built using reinforced concrete with brick cladding and features triple-glazed windows" and "A reinforced concrete structure, encased in brick cladding, with energy-efficient triple glazing"— led to only minor variations in predictions. In contrast, Building 4 exhibited a much higher standard deviation of 188.67 kgCO2e/m^2. Descriptions like "The school is built with a timber frame, brick facades, and large aluminium-framed windows" compared to "A timber framed school with brick facades, featuring windows encased in aluminium frames" seemed to cause significant fluctuations in the predicted embodied carbon, highlighting the software's vulnerability to how complex features might be described. A complete table of the synonymous building descriptions used in the analysis can be found in Appendix F.

The general pattern illustrated across all buildings demonstrates that ECO is generally able to produce similar results across different descriptions of the same building. Despite the variations in language and phrasing, the standard deviations relative to the overall predicted values indicate that ECO maintains a level of consistency in its predictions. While the higher standard deviations in some cases suggest that certain descriptions can lead to more significant variations, the overall shows a robust ability to interpret different descriptions and converge on similar embodied carbon estimates within 200 kgCO2e/m^2. This consistency, even in the presence of linguistic variability, is a positive indicator of ECO's reliability. However, the observed variability does

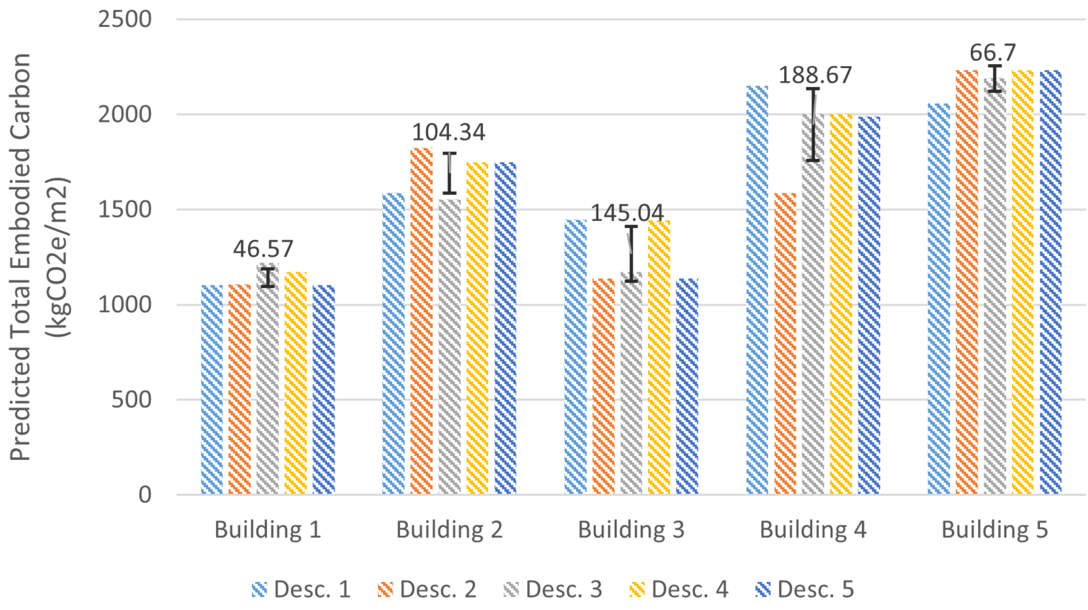

Figure 5.3: Variation in Predicted Embodied Carbon Across Different Descriptions of Building Designs

highlight a potential area for refinement. Particularly, the need to further enhance the natural language processing components to reduce the impact of more complex or nuanced descriptions on prediction outcomes.

5.2.4 Real-World Applicability

The user testing of ECO yielded valuable insights into its practical applicability, usability, and potential to enhance users' awareness of their carbon footprints. A dataset composed of 43 participants, including both students and professionals, provided detailed feedback on various aspects of the tool, such as ease of use, perceived accuracy of predictions, and potential for integration into existing workflows. Most participants were between the ages of 18 and 25, with educational backgrounds ranging from Bachelor's to Master's degrees. Their professional roles in the design industry spanned architects, civil/structural engineers, sustainability consultants, and production designers. 84.2% of individuals marked the influence of sustainability on their daily decisions as moderate to significant, highlighting a need for such tools in their professional activities. Despite this, 53% reported limited proficiency with existing sustainability tools, primarily listing reasons such as lack of training, time constraints, and their perceived complexity of these tools.

The feedback on the usability of ECO was overwhelmingly positive. The majority of participants found it easy to input information, with strong consensus regarding the tool's intuitive user interface and quick feedback response. The prototype achieved a System Usability Scale (SUS) score of 84.75, indicating a high level of user satisfaction (refer to Table 5.2). This score, coupled with its low standard deviation, displays the consistency in positive user experiences. Users widely agreed that the tool was easy to use, with a mean score of 4.89/5, and well-integrated, also with a mean score of 4.89/5, suggesting that the tool's functionality is both accessible and cohesive. This indicates that users consider the tool to be highly usable and effective. Several participants emphasised that ECO significantly increased their awareness of the small changes that make larger carbon impacts, though some suggested that the interface could be further optimised. Despite the tool not being generally perceived as complex or inconsistent, they noted that certain aspects could benefit from more detailed explanations or options for providing nuanced inputs. Also, the varying responses regarding the need for support for the tool and initial learning (seen through a mean score of 2.89/5, with a higher standard deviation) indicate that while most users found the tool accessible, there is room for improvement in onboarding and user guidance.

	Statement	*Mean*	*Standard Deviation*
1.	I think that I would like to use this tool frequently.	4.89	0.31
2.	I found the tool unnecessarily complex.	1.84	0.99
3.	I thought the tool was easy to use.	4.89	0.31
4.	I think that I would need the support of a technical person to be able to use this tool.	2.05	1.19
5.	I found the various functions in this tool were well integrated.	4.89	0.31
6.	I thought there was too much inconsistency in this tool.	2.11	0.72
7.	I would imagine that most people would learn to use this tool very quickly.	4.95	0.22
8.	I found the tool very cumbersome to use.	1.68	0.80
9.	I felt very confident using the tool.	4.84	0.36
10.	I needed to learn a lot of things before I could get going with this tool.	2.89	1.62
	Raw SUS	33.89	5.225
	Final SUS	**84.74**	**13.05**

Table 5.2: Average System Usability Scores and Standard Deviation (Scale 1-5)

In terms of predictive accuracy, most participants perceived ECO's predictions to be slightly higher than their expectations, but still within a reasonable range. This aligns with patterns of overestimation shown in earlier accuracy studies. Regardless, confidence in the tool was generally high, particularly among those with more experience in sustainability. The tool was observed to be largely consistent in identifying major materials and specifications; however, a number of users observed minor discrepancies in more complex or non-standard designs. These discrepancies, while not necessarily undermining the overall utility of the tool, indicate areas where further refinement could enhance accuracy and user trust.

The feedback also included several ideas for future uses, as well as constructive suggestions aimed at improving the tool. Participants highlighted the tool's potential to play a critical role in the early stages of design, particularly in estimating carbon impacts and influencing material choices before decisions are finalised. The tool was seen as a valuable resource for making sustainability considerations more accessible and actionable, especially for users who may not be experts in carbon footprint analysis. A staggering 95% of users further expressed a willingness to integrate ECO into their future projects, particularly for initial feasibility studies and early-stage design iterations, which speaks to its practical utility. Recommendations then included a number of frequently mentioned additional features such as direct integration with CAD software, the ability to break down the full lifecycle carbon impacts, and the automated suggestion of alternative design approaches aimed at reducing carbon emissions. Some users also requested greater transparency in how calculations are performed, which could help build trust in the tool's outputs and support more informed decision-making.

Overall, the ECO tool was well-received, with strong indications that it could effectively support sustainable design practices in real-world applications. Its ease of use, combined with its ability to raise awareness and guide early design decisions, positions it as a valuable asset for designers, engineers, and sustainability professionals. The high SUS score reinforces its viability for integration into professional workflows. However, the feedback also highlights opportunities for further development, particularly in refining feature extraction accuracy, a need consistently observed across all testing phases. Further, the integration of CAD and BIM capabilities, alongside the expansion of the tool's functionality to enhance transparency and provide a deeper understanding of carbon usage within buildings, are mentioned as key improvement opportunities.

5.3 Gaps and Opportunities for Future Research

These evaluations have highlighted several areas where future research and development would significantly enhance the ECO system's capabilities. One critical area worth addressing in future works is the diagnosis of inaccurate predictions within ECO, especially given the complexity of its integrated subsystems. A current challenge lies in the reliable validation of the tool's outputs, which may be a flaw in the approach. To address this, efforts should be made to simplify these subsystems, which would facilitate the diagnosis and correction of inaccuracies. Some subsystems, such as those relying on regex matching, may be particularly brittle, and prone to failure under unexpected input variations. By streamlining the system's architecture, it would enable more precise and efficient analysis of specific subsystems, allowing for a deeper investigation into the root causes of issues.

This simplification would not only enhance the tool's robustness but also creates a stronger foundation for further studies, such as the introduction of a "backward prediction" feature. This feature would allow users to set a desired carbon goal, and the software would work backwards to identify the closest set of design parameters that meet this target, given specific preset materials. Such a functionality could also be extended to provide automated suggestions for design improvements, and might be particularly useful for architects aiming to achieve specific sustainability certifications or meet stringent carbon reduction targets. Further incorporating predefined benchmarks such as RIBA carbon goals would enable direct comparisons, providing actionable insights for designers striving to meet industry standards.

Another significant opportunity lies in further enhancement of the tool through an ability to disaggregate the predicted carbon footprint into more detailed components, such as individual carbon values for substructure, superstructure, and finishes. By calculating the contribution of each element to the overall carbon impact, the tool might offer more granular insights into which design choices are driving emissions. This feature would allow users to pinpoint specific areas where carbon savings could be achieved, leading to more informed and targeted design decisions. Furthermore, a combination of this with an additional pipeline to first predict the material masses for various building typologies could benefit ECO via enhanced analytic capabilities, providing users with higher degrees of granularity in carbon impact assessments. It may also increase accuracy by improving the tool's ability to *understand* the scale at which materials are used, reducing the current limitations of ECO's generalisations.

Moreover, a main area of rework currently within the tool is its extraction system, which has several identified areas of required optimisation. One such area is the current implementation of ECO's NLP system, which treats all terms as equally important when calculating similarity scores, leading to potential inaccuracies in feature extraction. Future enhancements could focus on incorporating weighted similarity calculations that considers the semantic relationships between terms, allowing direct synonyms or closely related hypernyms/hyponyms to contribute more significantly to the final similarity score. This could be achieved by adjusting the cosine similarity score using semantic similarity measures like the Wu and Palmer (1994) method. Additionally, an adaptive similarity threshold for word matching, which adjusts based on the type of semantic relationship, could further improve the system's ability to accurately interpret complex design descriptions. These ideas could also be used to enhance the current conflict resolution method, which randomly selects between equally likely options. An improvement could be the utilisation of semantic information to prioritise the most contextually relevant features, thus increasing the tool's reliability and reducing incorrect assumptions.

Finally, the tool currently overlooks certain smaller features that can have a substantial impact on the building's carbon footprint, such as natural lighting, ventilation, and building orientation. Future research could focus on integrating these features into the carbon prediction model, either through direct input options or by enhancing the NLP model to detect and interpret these aspects from design descriptions. These could reduce the impact of operational carbon (stages B6 to B7) and enable a more comprehensive analysis in design considerations. An approach worth mentioning involves potentially training a specialised text transformer directly on building descriptions and their associated carbon values, rather than relying solely on feature extraction for regression analysis, to measure the effect of various words on final carbon values. This moves beyond ECO's current pipeline limitations by providing the tool with the ability to capture complex relationships and contextual nuances, leading to a more accurate prediction whilst reducing the need for manual feature engineering. The analysis process is streamlined, with the tool now uncovering patterns that the previous method might not have picked up on.

6. Conclusions

The research presented in this thesis demonstrates a potential step forward in integrating sustainable practices into early-stage architectural design through the development and application of the Early-stage Carbon Observer tool. It's ability to predict embodied carbon impact from textual design descriptions provides architects and designers with a practical, data-driven approach to make more informed decisions at stages where minimal quantifiable decisions have been made. ECO's integration of machine learning, particularly through a pipeline of advanced natural language processing techniques and regression models, highlights the potential for AI to reshape the architecture, engineering, and construction industry, acting as a bridge between early design phases and environmental impact analysis.

The tool has several prospective uses. For instance, applications in design meetings presents a strong opportunity. As design teams collaborate and iterate on concepts, an integration of speech recognition approaches with ECO could serve together as a dynamic tool that updates in real-time, providing insights into the carbon implications of various choices. This would empower teams to steer their projects to more sustainable outcomes as they converse with each other. Further, with Building Information Modelling software becoming more prevalent in the industry (Abdelhameed, 2018), an integration of ECO to generate automated carbon reports may become indispensable. The tool could offer detailed analysis and projections on carbon impacts as designs involve, which might subsequently be included in project documentation or shared with clients to demonstrate a commitment to sustainability. Integration of this into architecture and engineering curricula would then have huge benefits, allowing students to experiment with different design choices and see the immediate impact on carbon emissions. ECO would serve as a powerful teaching tool, for designers and engineers at all levels, instilling the importance of sustainable design from the outset of their professional careers.

The system demonstrates promising capabilities in precisely ranking design options by carbon intensity and showing robustness to linguistic variability. However, the results of the system evaluation indicate that, in terms of accuracy, ECO currently tends to overestimate embodied carbon values when compared to actual data from the LETI Embodied Carbon Case Studies. This problem exaggerates a current limitation of the research — the lack of direct, *structured* real-world data — which could otherwise provide more accurate validation and refinement of ECO's predictions. While the conservatism may encourage caution in sustainable designs, future work should aim to identify the issues by investigating the underlying factors contributing to the inflated predictions, as well as the specific subsystems that require refinement, particularly in light of user recommendations for greater transparency in the calculations. The sensitivity analysis also revealed that while ECO is highly effective at identifying primary structural materials and key architectural features, there are opportunities to enhance its detection of more complex or nuanced design elements, such as specific internal materials or non-standard roof finishes. This presents an avenue for further refinement, showing that with targeted improvements, ECO could become even more robust. Enhancing its feature extraction accuracy, particularly for complex or unusual designs, and improving its ability to handle nuanced architectural details could increase its reliability and overall predictive precision. Furthermore, the potential integration of additional functionalities, such as a backward prediction feature and disaggregation of carbon components, presents opportunities for future research and development.

Despite the observed areas of improvement, feedback on real-world applicability was overwhelmingly positive. Earning a high SUS score, ECO was commended for its ease of use and integration into existing workflows. Users appreciated its ability to raise awareness about the carbon impacts of design choices, though they also suggested improvements such as more detailed explanations of the calculations and the ability to handle more detailed or nuanced design inputs. Overall, ECO offers a meaningful contribution to sustainable architecture by providing an innovative solution to the challenge of early-stage carbon assessment. The tool's current capabilities and the insights gained from this research lay a solid foundation for future advancements, ultimately contributing to the global effort to reduce carbon emissions in the built environment.

Number of words until this point, excluding front matter: 10838.

Bibliography

Abdelhameed, W., 2018. Bim in architecture curriculum: a case study. *Architectural science review* [Online], 61, pp.480–491. Available from: https://doi.org/10.1080/00038628.2018.1483888.

AHMM and redboxmedia, 2022. Delivering net zero in use toolkit / allford hall monaghan morris | ahmm [Online]. Available from: https://www.ahmm.co.uk/profile/sustainability/delivering-net-zero-in-use-toolkit/ [Accessed 2024-04-30].

Apallius De Vos, I., Van Den Boogerd, G., Fennema, M. and Correia, A., 2021. Comparing in context: Improving cosine similarity measures with a metric tensor [Online]. Available from: https://aclanthology.org/2021.icon-main.17.pdf [Accessed 2024-08-19].

Arroyo, P., Schöttle, A. and Christensen, R., 2021. The ethical and social dilemma of ai uses in the construction industry. *Proc. 29th annual conference of the international group for lean construction (iglc)* [Online]. Available from: https://doi.org/10.24928/2021/0188.

Bartoli, A., De Lorenzo, A., Medvet, E. and Tarlao, F., 2016. Inference of regular expressions for text extraction from examples. *Ieee transactions on knowledge and data engineering* [Online], 28, pp.1217–1230. Available from: https://doi.org/10.1109/tkde.2016.2515587 [Accessed 2022-01-16].

Breiman, L., 2001. Random forests. *Machine learning* [Online], 45, pp.5–32. Available from: https://doi.org/10.1023/a:1010933404324.

Chen, H., Chiang, R.H.L. and Storey, V.C., 2012. Business intelligence and analytics: from big data to big impact. *Mis quarterly* [Online], 36, pp.1165–1188. Available from: https://doi.org/10.2307/41703503.

Chen, Z., Xiao, F., Guo, F. and Yan, J., 2023. Interpretable machine learning for building energy management: A state-of-the-art review. *Advances in applied energy* [Online], 9, p.100123. Available from: https://doi.org/10.1016/j.adapen.2023.100123.

Donovan, E., 2020. Explaining sustainable architecture. *Iop conference series: Earth and environmental science* [Online], 588, p.032086. Available from: https://doi.org/10.1088/1755-1315/588/3/032086 [Accessed 2022-02-09].

FCBStudios, 2020. Fcbs carbon [Online]. Available from: https://portal.fcbstudios.com/fcbscarbon [Accessed 2024-04-30].

Fenton, S.K., De Rycke, K. and De Laet, L., 2023. Predicting embodied carbon of building structure types through machine learning. Proceedings of the International Association for Shell and Spatial Structures Annual Symposium 2023, IASS, pp.1–11.

Flask, 2010. Welcome to flask — flask documentation (3.0.x) [Online]. Available from: https://flask.palletsprojects.com/en/3.0.x/ [Accessed 2024-08-19].

Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. *The annals of statistics* [Online], 29, pp.1189–1232. Available from: https://doi.org/10.1214/aos/1013203451.

Guryanov, A., 2019. Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees. *Lecture notes in computer science* [Online], 8, pp.39–50. Available from: https://doi.org/10.1007/978-3-030-37334-4_4.

Helm, J.M., Swiergosz, A.M., Haeberle, H.S., Karnuta, J.M., Schaffer, J.L., Krebs, V.E., Spitzer, A.I. and Ramkumar, P.N., 2020. Machine learning and artificial intelligence: Definitions, applications, and future directions. *Current reviews in musculoskeletal medicine* [Online], 13, pp.69–76. Available from: https://doi.org/10.1007/s12178-020-09600-8.

Hugging Face, 2024. Hugging face: Home to the most comprehensive machine learning community [Online]. Accessed: [2024-08-19]. Available from: https://huggingface.co/spaces.

Keitsch, M., 2012. Sustainable architecture, design and housing. *Sustainable development* [Online], 20, pp.141–145. Available from: https://doi.org/10.1002/sd.1530.

Lee, T.H., Ullah, A. and Wang, R., 2019. Bootstrap aggregating and random forest. *Macroeconomic forecasting in the era of big data* [Online], 52, pp.389–429. Available from: https://doi.org/10.1007/978-3-030-31150-6_13.

LETI, n.d. Leti | case studies [Online]. Available from: https://www.leti.uk/case-studies [Accessed 2024-08-26].

Mohammed, M.A., Ahmed, M.A. and Hacimahmud, A.V., 2023. Data-driven sustainability: Leveraging big data and machine learning to build a greener future. *Babylonian journal of artificial intelligence* [Online], 2023, p.17–23. Available from: https://doi.org/10.58496/BJAI/2023/005 [Accessed 2024-03-28].

Mohapatra, N., Shreya, K. and Chinmay, A., 2020. Optimization of the random forest algorithm. *Advances in data science and management* [Online], pp.201–208. Available from: https://doi.org/10.1007/978-981-15-0978-0_19.

Navigli, R. and Martelli, F., 2019. An overview of word and sense similarity. *Natural language engineering* [Online], pp.1–22. Available from: https://doi.org/10.1017/s1351324919000305 [Accessed 2019-09-09].

Neri, E., Coppola, F., Miele, V., Bibbolino, C. and Grassi, R., 2020. Artificial intelligence: Who is responsible for the diagnosis? *La radiologia medica* [Online], 125, p.517–521. Available from: https://doi.org/10.1007/s11547-020-01135-9.

Netlify, n.d. Netlify: All-in-one platform for automating modern web projects. https://www.netlify.com/. Accessed: 2024-08-16.

OpenAI, 2024. Chatgpt [Online]. Accessed: 2024-08-27. Available from: https://www.openai.com/chatgpt.

Passive House Institute, n.d. Passivhaus institut [Online]. Available from: https://passivehouse.com/04_phpp/04_phpp.htm [Accessed 2024-04-30].

Pomponi, F., Anguita, M.L., Lange, M., D'Amico, B. and Hart, E., 2021. Enhancing the practicality of tools to estimate the whole life embodied carbon of building structures via machine learning models. *Frontiers in built environment* [Online], 7. Available from: https://doi.org/10.3389/fbuil.2021.745598 [Accessed 2021-10-26].

Pons-Valladares, O. and Nikolic, J., 2020. Sustainable design, construction, refurbishment and restoration of architecture: A review. *Sustainability* [Online], 12, p.9741. Available from: https://doi.org/10.3390/su12229741.

Rocchio, J.J. and Salton, G., 1965. Information search optimization and interactive retrieval techniques [Online]. [Online]. Available from: https://doi.org/10.1145/1463891.1463926 [Accessed 2023-10-11].

Sadbhawna, Jakhetiya, V., Chaudhary, S., Subudhi, B.N., Lin, W. and Guntuku, S.C., 2022. Perceptually unimportant information reduction and cosine similarity-based quality assessment of 3d-synthesized images. *Ieee transactions on image processing* [Online], 31, pp.2027–2039. Available from: https://doi.org/10.1109/tip.2022.3147981 [Accessed 2023-05-05].

Salton, G., Singhal, A., Mitra, M. and Buckley, C., 1997. Automatic text structuring and summarization. *Information processing management* [Online], 33, pp.193–207. Available from: https://doi.org/10.1016/s0306-4573(96)00062-3 [Accessed 2021-01-28].

Sarker, I.H., 2021. Machine learning: Algorithms, real-world applications and research directions. *Sn computer science* [Online], 2, pp.1–21. Available from: https://doi.org/10.1007/s42979-021-00592-x.

Seyedzadeh, S., Rahimian, F.P., Glesk, I. and Roper, M., 2018. Machine learning for estimation of building energy consumption and performance: a review. *Visualization in engineering* [Online], 6. Available from: https://doi.org/10.1186/s40327-018-0064-7.

Steck, H., Ekanadham, C. and Kallus, N., 2024. Is cosine-similarity of embeddings really about similarity? *arxiv (cornell university)* [Online]. Available from: https://doi.org/10.1145/3589335.3651526 [Accessed 2024-03-23].

Su, S., Zang, Z., Yuan, J., Pan, X. and Shan, M., 2024. Considering critical building materials for embodied carbon emissions in buildings: A machine learning-based prediction model and tool. *Case studies in construction materials* [Online], 20, pp.e02887–e02887. Available from: https://doi.org/10.1016/j.cscm.2024.e02887 [Accessed 2024-08-12].

Tien, P.W., Wei, S., Darkwa, J., Wood, C. and Calautit, J.K., 2022. Machine learning and deep learning methods for enhancing building energy efficiency and indoor environmental quality – a review. *Energy and ai* [Online], 10, p.100198. Available from: https://doi.org/10.1016/j.egyai.2022.100198.

United Nations Environment Programme and Yale Center for Ecosystems + Architecture, 2023. Building materials and the climate: Constructing a new future. *Unep.org* [Online]. Available from: https://doi.org/978-92-807-4064-6 [Accessed 2023-09-17].

University of Bath, 2022. Zebra [Online]. Available from: https://www.zebra-model.org/ [Accessed 2024-04-30].

University of Sheffield, 2020. regenerate - a tool that encourages construction designers to engage with the circular economy [Online]. Available from: https://www.sheffield.ac.uk/civil/news/regenerate-tool-encourages-construction-designers-engage-circular-economy [Accessed 2024-04-30].

Verbs semantics and lexical selection, 1994. [Online], vol. 32. Proceedings of the 32nd annual meeting on Association for Computational Linguistics -, Association for Computational Linguistics. Available from: https://doi.org/10.3115/981732.981751.

Wang, W., Rivard, H. and Zmeureanu, R., 2006. Floor shape optimization for green building design. *Advanced engineering informatics* [Online], 20, pp.363–378. Available from: https://doi.org/10.1016/j.aei.2006.07.001.

Weng, Z., Ramallo-González, A.P. and Coley, D.A., 2014. The practical optimisation of complex architectural forms. *Building simulation* [Online], 8, pp.307–322. Available from: https://doi.org/10.1007/s12273-014-0208-1 [Accessed 2020-12-29].

Xu, Y., Wang, Q., An, Z., Wang, F., Zhang, L., Wu, Y., Dong, F., Qiu, C.W., Liu, X., Qiu, J., Hua, K., Su, W., Xu, H., Han, Y., Cao, X., Liu, E., Fu, C., Yin, Z., Liu, M., Roepman, R., Dietmann, S., Virta, M., Kengara, F., Huang, C., Zhang, Z., Zhang, L., Zhao, T., Dai, J., Yang, J., Lan, L., Luo, M., Huang, T., Liu, Z., Qian, S., An, T., Liu, X., Zhang, B., He, X., Cong, S., Liu, X., Zhang, W., Wang, F., Lu, C., Cai, Z., Lewis, J.P., Tiedje, J.M. and Zhang, J., 2021. Artificial intelligence: a powerful paradigm for scientific research. *The innovation* [Online], 2. Available from: https://doi.org/https://doi.org/10.1016/j.xinn.2021.100179.

Ying, X. and Li, W., 2020. Effect of floor shape optimization on energy consumption for u-shaped office buildings in the hot-summer and cold-winter area of china. *Sustainability* [Online], 12, p.2079. Available from: https://doi.org/10.3390/su12052079 [Accessed 2021-05-19].

Zhou, K., Ethayarajh, K., Card, D. and Jurafsky, D., 2022. Problems with cosine as a measure of embedding similarity for high frequency words. *Proceedings of the 60th annual meeting of the association for computational linguistics* [Online], 2, pp.401–423. Available from: https://aclanthology.org/2022.acl-short.45.pdf.

A. Generated Data

A.1 Data Generation Constraints

Category	*Attribute*	*Constraints*	*Description*
Typology	Sector	- Housing - Office	Defines the sectors
	Sub-Sector	- Flat/maisonette - Single family house - Multi-family house (< 6 Storeys) - Multi-family house (6 - 15 Storeys) - Multi-family house (> 15 Storeys)	Defines the subsectors available for the Housing sector.
		- Office	Defines the subsectors available for the Office sector.
Building Dimensions	GIA	0 - 20000 m^2	Randomly generated gross internal area within given constraints.
	Perimeter	100 - 5000m	Randomly generated building perimeter within given constraints.
	Footprint	100 - 10000m^2	Randomly generated building footprint within given constraints.
	Width	10 - 200m	Randomly generated building width within given constraints.
	Floor Height	2.3 - 6m	Randomly generated floor-to-floor heights within given constraints.
	Storeys (Above Ground)	1 - 60 storeys	Randomly generated storeys above ground within given constraints.
	Storeys (Below Ground)	0 - 5 storeys	Randomly generated storeys below ground within given constraints.
Materiality	Building Elements	*See detailed elements below.*	Defines available building elements such as "Piles", "Pile Caps", "Capping beams", etc.
	Element Options	*See detailed options below.*	Defines available material options for each building element.
Logical Constraints	"Raft"	Cannot be selected if "Capping beams" or "Pile caps" are present.	Conditional constraint due to incompatible elements.
	"Pile caps" "Capping beams"	Cannot be selected if "Raft" is present.	Conditional constraint due to incompatible element.
	"Joisted floors"	Cannot be selected if "Floor slab" is present.	Conditional constraint due to mutually exclusive element.
	"Floor slab"	Cannot be selected if "Joisted floors" present.	Conditional constraint due to mutually exclusive element.
	"Basement walls"	Cannot be selected if storeys below is zero.	Option is only available when basement floors is greater than zero, otherwise unnecessary.

Table A.1: Constraints and Their Descriptions

A.2 Material Options

Building Element	***Material Options***
Piles	- Reinforced Concrete - Steel
Pile caps	- Reinforced Concrete
Capping beams	- RC 32/40 (200kg/m3 reinforcement) - Foamglass (domestic only)
Raft	- RC 32/40 (150kg/m3 reinforcement)
Basement walls	- RC 32/40 (125kg/m3 reinforcement)
Lowest floor slab	- RC 32/40 (150kg/m3 reinforcement) - Beam And Block
Ground insulation	- EPS - XPS - Glass mineral wool
Core structure	- CLT - Precast RC 32/40 (100kg/m3 reinforcement) - RC 32/40 (100kg/m3 reinforcement)
Columns	- Glulam - Iron (existing buildings) - Precast RC 32/40 (300kg/m3 reinforcement) - RC 32/40 (300kg/m3 reinforcement) - Steel
Beams	- Glulam - Iron (existing buildings) - Precast RC 32/40 (250kg/m3 reinforcement) - RC 32/40 (250kg/m3 reinforcement) - Steel
Secondary beams	- Glulam - Iron (existing buildings) - Precast RC 32/40 - RC 32/40 (250kg/m3 reinforcement) - Steel
Floor slab	- CLT - Precast RC 32/40 (100kg/m3 reinforcement) - RC 32/40 (100kg/m3 reinforcement) - Steel Concrete Composite
Joisted floors	- JJI Engineered Joists + OSB topper - Timber Joists + OSB topper (Office) - Timber Joists + OSB topper (Domestic)
Roof	- CLT - Precast RC 32/40 (100kg/m3 reinforcement) - RC 32/40 (100kg/m3 reinforcement) - Steel Concrete Composite - Timber Cassette - Timber Pitch Roof

continues on next page

Roof insulation	- Cellulose, loose fill - EPS - Expanded Perlite - Expanded Vermiculite - Glass Mineral Wool - PIR - Rockwool - Sheeps Wool - Vacuum Insulation - Woodfibre - XPS
Roof finishes	- Aluminium - Asphalt (Mastic) - Asphalt (Polymer modified) - Bitumous Sheet - Ceramic Tile - Fibre Cement Tile - Green Roof - Roofing Membrane (PVC) - Slate Tile - Zinc Standing Seam
Facade	- Blockwork with Brick - Blockwork with Render - Blockwork with Timber - Curtain Walling - Load Bearing Precast Concrete Panel - Load Bearing Precast Concrete with Brick Slips - Party Wall Blockwork - Party Wall Brick - Party Wall Timber Cassette - SFS with Aluminium Cladding - SFS with Brick - SFS with Ceramic Tiles - SFS with Granite - SFS with Limestone - SFS with Zinc Cladding - Solid Brick, single leaf - Timber Cassette Panel with Larch Cladding - Timber Cassette Panel with Lime Render - Timber SIPs with Brick
Wall insulation	- Cellulose, loose fill - EPS - Expanded Perlite - Expanded Vermiculite - Glass Mineral Wool - PIR - Rockwool - Sheeps Wool - Vacuum Insulation - Woodfibre - XPS

continues on next page

Glazing	- Triple Glazing - Double Glazing - Single Glazing
Window frames	- Al/Timber Composite - Aluminium - Steel (single glazed) - Solid Softwood Timber Frame - uPVC
Partitions	- CLT - Plasterboard + Steel Studs - Plasterboard + Timber Studs - Plywood + Timber Studs - Blockwork
Ceilings	- Exposed Soffit - Plasterboard - Steel Grid System - Steel Tile - Steel Tile with 18mm Acoustic Pad - Suspended Plasterboard
Floors	- 70mm Screed - Carpet - Earthenware Tile - Raised Floor - Solid Timber Floorboards - Stoneware Tile -Terrazzo -Vinyl
Services	- High - Medium - Low

Table A.2: Building Element and Material Options

A.3 Assumptions

Material	***Building Element***	***Assumption***
JJI Engineered Joists	Joisted floors	OSB topper
Timber Joists	Joisted floors	OSB topper
Precast Concrete	Beams	32/40 (250kg/m3 reinforcement)
	Columns	32/40 (300kg/m3 reinforcement)
	Core structure	32/40 (100kg/m3 reinforcement)
	Floor slab	32/40 (100kg/m3 reinforcement)
	Roof	32/40 (100kg/m3 reinforcement)
	Secondary beams	32/40 (250kg/m3 reinforcement)
Reinforced Concrete	Basement walls	32/40 (125kg/m3 reinforcement)
	Beams	32/40 (250kg/m3 reinforcement)
	Capping beams	32/40 (200kg/m3 reinforcement)
	Columns	32/40 (300kg/m3 reinforcement)
	Core structure	32/40 (100kg/m3 reinforcement)
	Floor slab	32/40 (100kg/m3 reinforcement)
	Lowest floor slab	32/40 (150kg/m3 reinforcement)
	Pile caps	32/40 (200kg/m3 reinforcement)
	Piles	32/40 (50kg/m3 reinforcement)
	Raft	32/40 (150kg/m3 reinforcement)
	Roof	32/40 (100kg/m3 reinforcement)
	Secondary beams	32/40 (250kg/m3 reinforcement)
Steel Tile	Ceilings	18mm acoustic pad
Screed	Floors	70mm thickness

Table A.3: Processed Material Assumptions

B. ECO Prototype Client Interface

B.1 Input Field

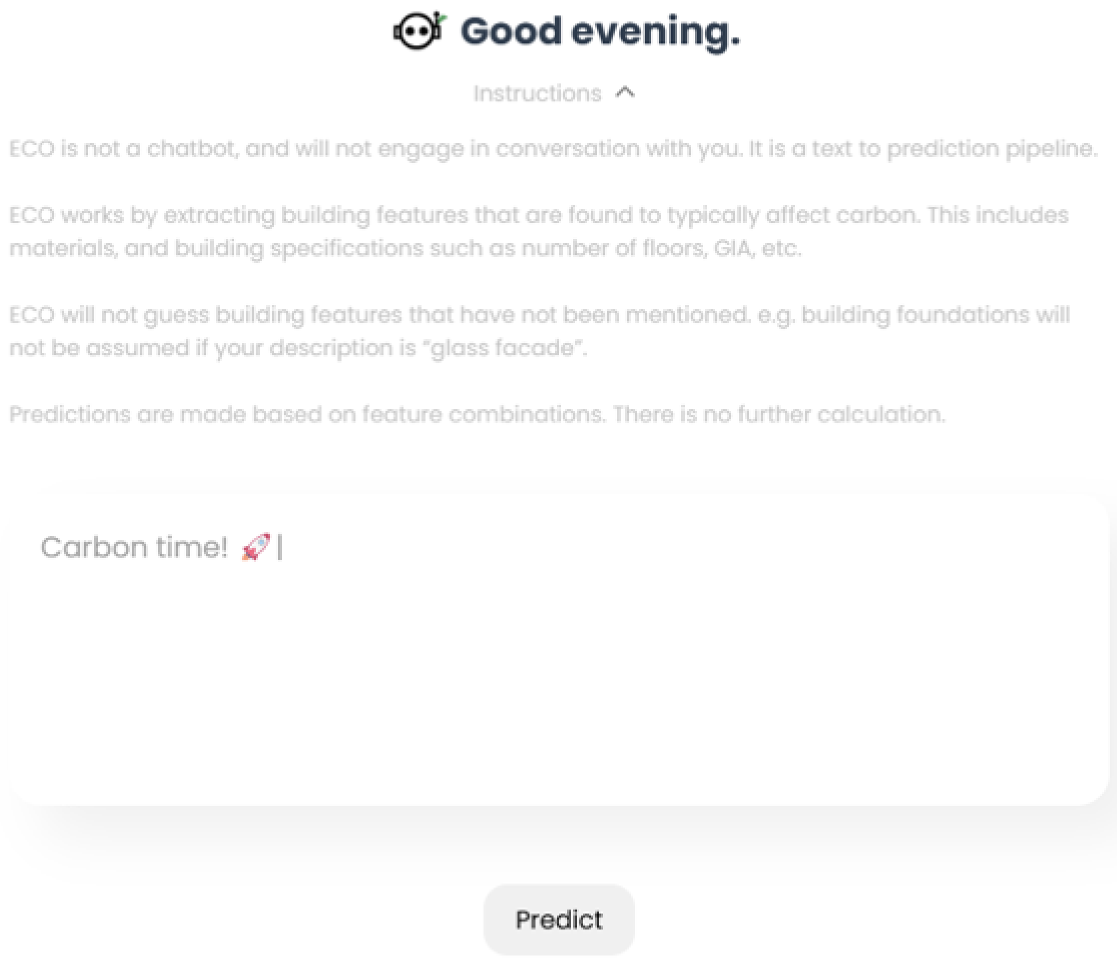

Figure B.1: ECO Prototype Input Field with User Instructions.

B.2 Input Loading

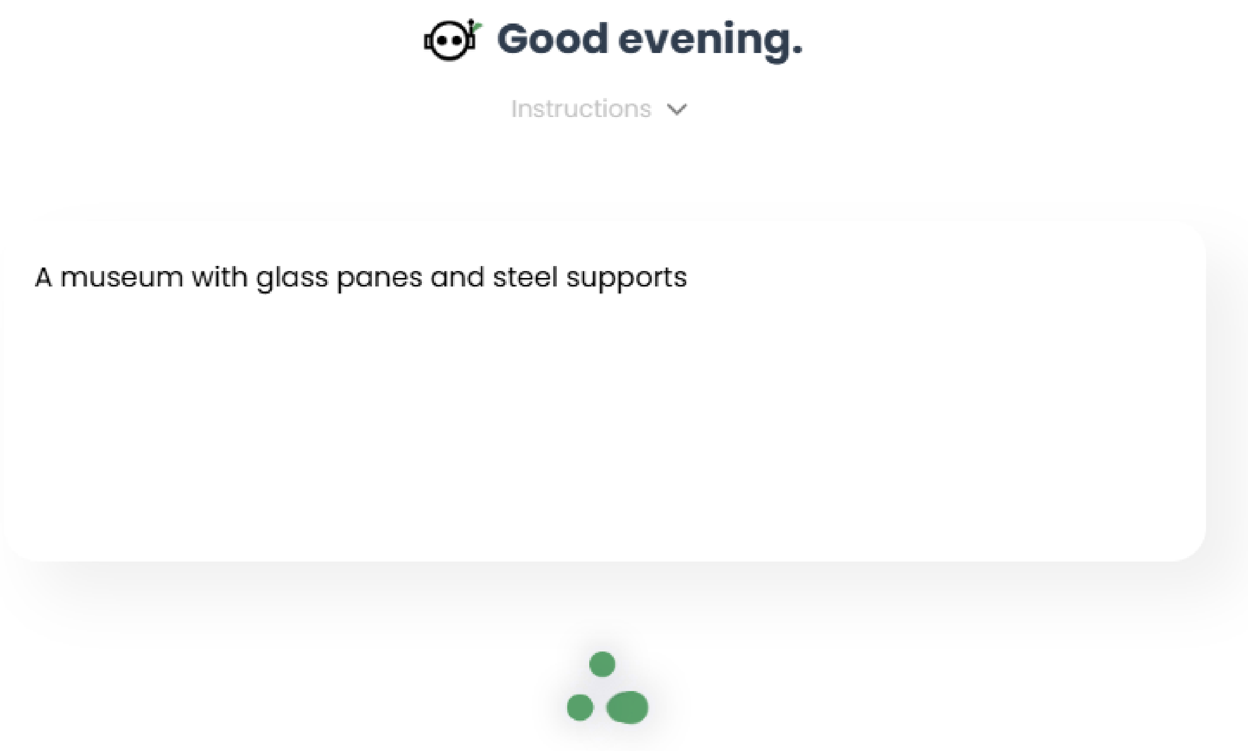

Figure B.2: ECO Prototype Loading Animation.

B.3 Output Display

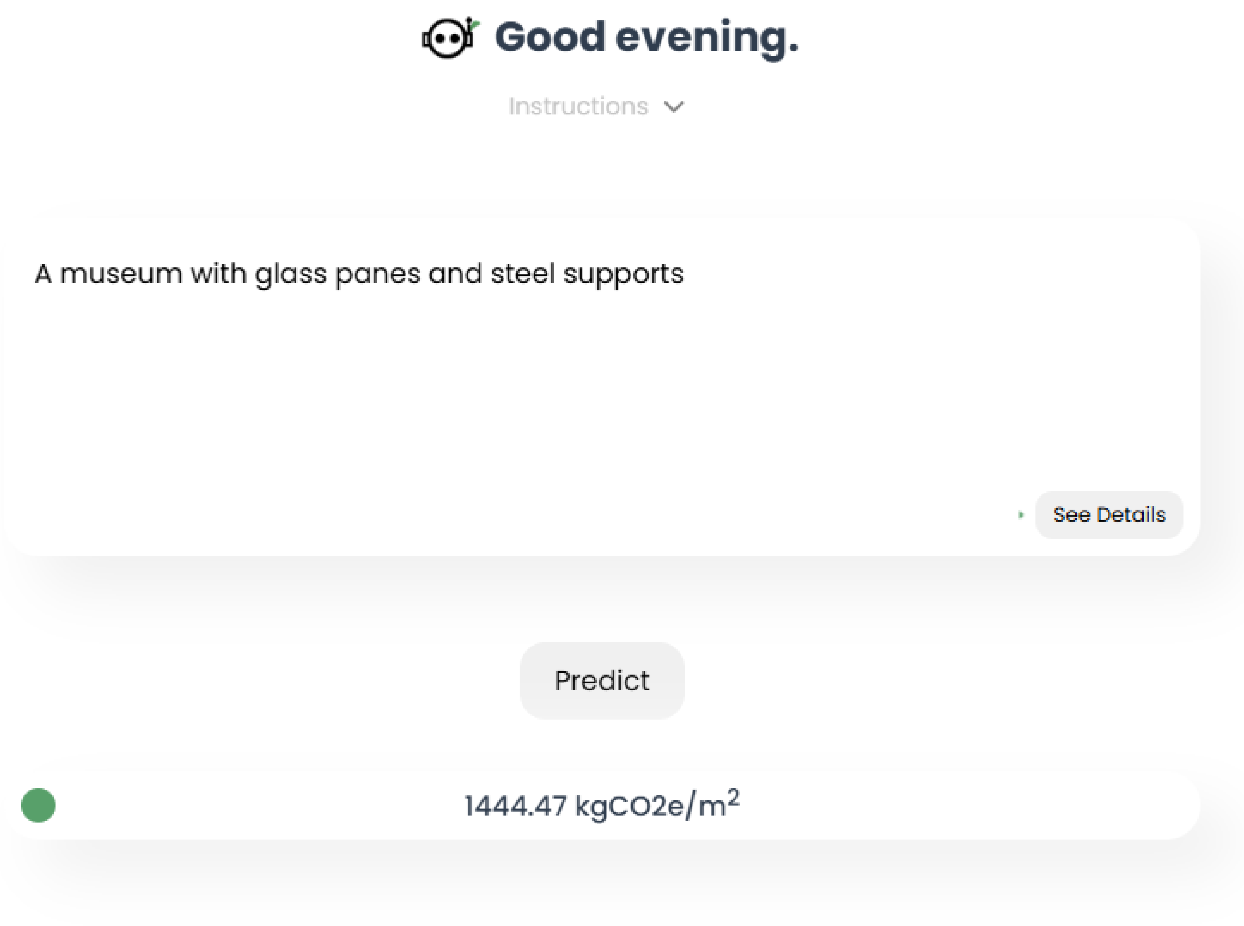

Figure B.3: ECO Prototype Prediction Output.

B.4 Detail Dashboard

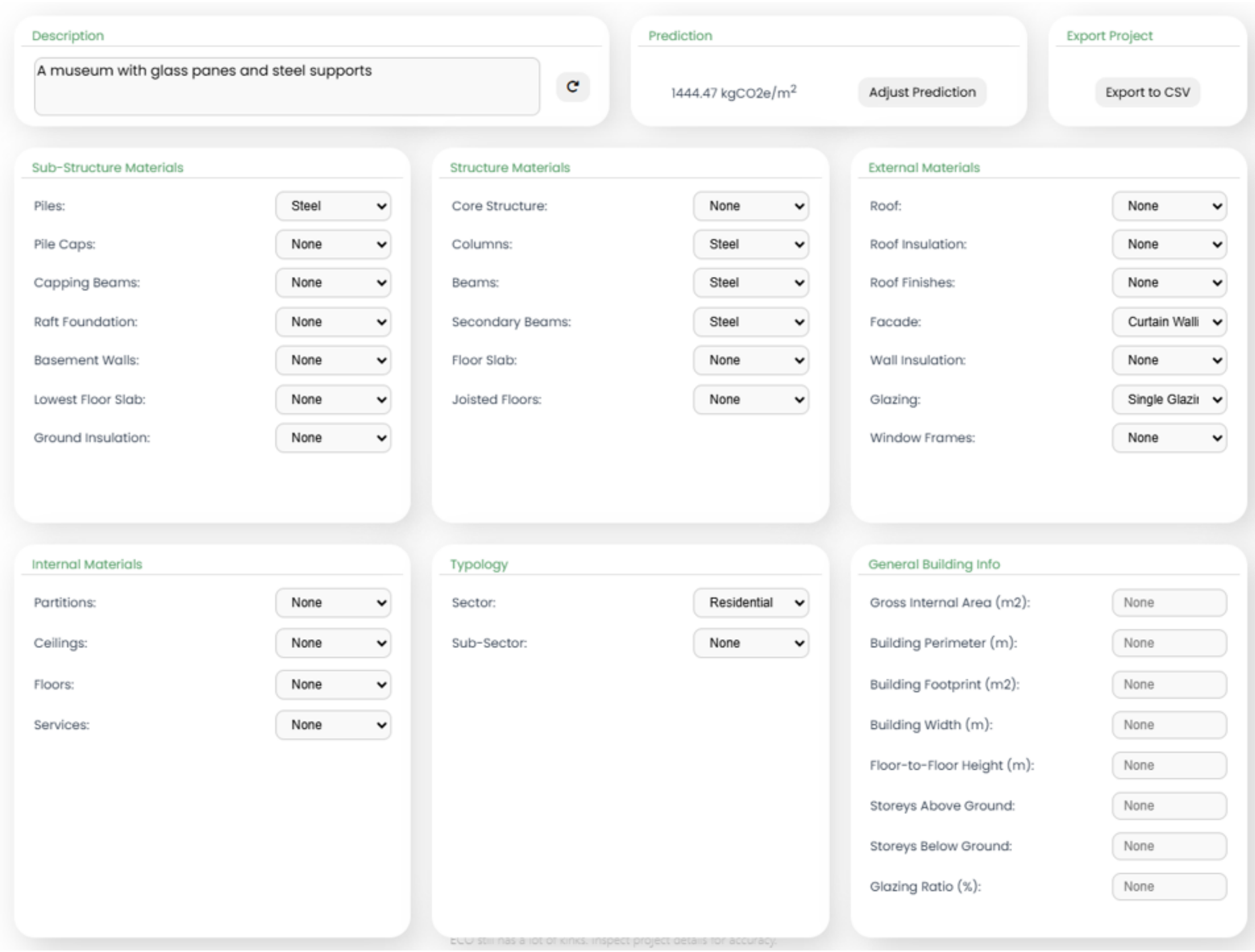

Figure B.4: ECO Prototype Details Dashboard.

C. Model Evaluation

C.1 Trial Model Performance(s)

Model	***Category***	***Score***
Gradient Boosting	Training R-Squared	0.9999
	Testing R-Squared	0.7556
	Individual Cross-Validations	0.7196, 0.7701. 0.7027, 0.7503, 0.7529
	Mean Cross-Validation	0.7337
Histogram-based Gradient Boosting	Training R-Squared	0.9882
	Validation R-Squared	0.9359
	Individual Cross-Validations	0.9304, 0.9335, 0.9308, 0.9289, 0.9277
	Mean Cross-Validation	0.9303
Random Forest	Training R-Squared	0.6377
	Testing R-Squared	0.5527
	Individual Cross-Validations	0.5181, 0.5261, 0.4697, 0.4856, 0.4760
	Mean Cross-Validation	0.4951
Linear Regression	Training R-Squared	0.4562
	Testing R-Squared	0.4807
	Individual Cross-Validations	0.4120, 0.3496, 0.2686, 0.4341, 0.3531
	Mean Cross-Validation	0.3643
Ridge	Training R-Squared	0.4562
	Testing R-Squared	0.4802
	Individual Cross-Validations	0.4115, 0.3516, 0.2718, 0.4333, 0.3531
	Mean Cross-Validation	0.3642
Lasso	Training R-Squared	0.4541
	Testing R-Squared	0.4788

continues on next page

Lasso	Individual Cross-Validations	0.4164, 0.3673, 0.2902, 0.4352, 0.3637
	Mean Cross-Validation	0.3646
Elastic Net	Training R-Squared	0.2063
	Testing R-Squared	0.2020
	Individual Cross-Validations	0.1622, 0.1880, 0.1844, 0.1783, 0.1861
	Mean Cross-Validation	0.1798
Support Vector Regression (SVR)	Training R-Squared	0.0396
	Testing R-Squared	0.0340
	Individual Cross-Validations	0.0136, 0.0130, 0.0241, 0.0191, 0.0199
	Mean Cross-Validation	0.0179

Table C.1: Trialled Model Performances (4s.f.)

C.2 Final Model Performance

Model	***Category***	***Score***
Histogram-based Gradient Boosting	Training R-Squared	0.9687
	Testing R-Squared	0.9479
	Individual Cross-Validation	0.9484, 0.9485, 0.9483, 0.9478, 0.9471
	Mean Cross-Validation	0.9480

Table C.2: Final Model Performance (4s.f.)

D. Extraction Sensitivity Raw Results

D.1 Baseline Case

Description:

" The project is an 8 storey office building with 2 levels of underground parking. It features a steel frame structure with a deep pile foundation. The façade includes 40% curtain wall glazing and brick cladding on the sides. The roof is flat with a green roof system. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced. "

Embodied Carbon Prediction:

1756.26 kgCO2e/m²

Extracted Features

	Feature	**Value**	**Correct?**	**Notes**
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material, material assumed from text
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*		
	Ground Insulation	*None*		
Structure Materials	Core Structure	*None*		
	Columns	Steel	Y	Steel frame structure specified.
	Beams	Steel	Y	Steel frame structure specified.
	Secondary Beams	Steel	Y	Steel frame structure specified.
	Floor Slab	*None*		
	Joisted Floors	*None*		
External Materials	Roof	Metal Deck	N/A	No roof structure specified, assumption seems to be made from "steel".
	Roof Insulation	*None*		
	Roof Finishes	Ceramic tile	N	Should be "Green Roof".
	Facade	Curtain Walling	Y	
	Wall Insulation	*None*		
	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	N	Brick mentioned externally, not blockwork internally.
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".

continues on next page

Typology	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15	N	Should be 15,000m2. Seems to ignore commas in numbers.
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		
	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	2	N	Should be 8. Seems to confuse storeys below and above ground.
	Storeys Below Ground	2	Y	
	Glazing Ratio (%)	40	Y	

Table D.1: Values found for baseline description

D.2 Structural Material Description Change(s)

Baseline:

Steel frame.

Variation 1:

Changed to reinforced concrete frame.

" The project is an 8 storey office building with 2 levels of underground parking. It features a reinforced concrete structure with a deep pile foundation. The façade includes 40% curtain wall glazing and brick cladding on the sides. The roof is flat with a green roof system. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced. "

Embodied Carbon Prediction:

1274.45 kgCO2e/m2

Extracted Features

	Feature	**Value**	**Correct?**	**Notes**
Sub-Structure Materials	Piles	Reinforced Concrete	Y	Element specified but no material specified, material assumed from text.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	Reinforced Concrete	N/A	Element not specified, assumption made from structural frame.
	Basement Walls	*None*		
	Lowest Floor Slab	Reinforced Concrete	N/A	Element not specified, assumption made from structural frame.
	Ground Insulation	*None*		
Structure Materials	Core Structure	Reinforced Concrete	Y	Reinforced concrete structure specified.
	Columns	Reinforced Concrete	Y	Reinforced concrete structure specified.

continues on next page

Structure Materials	Beams	Reinforced Concrete	Y	Reinforced concrete structure specified.
	Secondary Beams	Reinforced Concrete	Y	Reinforced concrete structure specified.
	Floor Slab	Reinforced Concrete	N/A	Element not specified, assumption made from structural frame.
	Joisted Floors	*None*		
External Materials	Roof	Metal Deck	N/A	No roof structure specified, assumption made.
	Roof Insulation	*None*		
	Roof Finishes	Ceramic tile	N	Should be "Green Roof".
	Facade	Curtain Walling	Y	
	Wall Insulation	*None*		
	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	N	Brick mentioned externally, not blockwork internally.
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15	N	Should be 15,000m2. Seems to ignore commas in numbers.
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		
	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	2	N	Should be 8. Seems to confuse storeys below and above ground.
	Storeys Below Ground	2	Y	
	Glazing Ratio (%)	40	Y	

Table D.2: Values found for description with structural material changed (1).

Variation 2:

Changed to timber frame.

" The project is an 8 storey office building with 2 levels of underground parking. It features a timber structure with a deep pile foundation. The façade includes 40% curtain wall glazing and brick cladding on the sides. The roof is flat with a green roof system. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced. "

Embodied Carbon Prediction:

1242.73 kgCO2e/m2

Extracted Features

	Feature	**Value**	**Correct?**	**Notes**
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material specified, assumption made.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*	N/A	
	Ground Insulation	*None*		
Structure Materials	Core Structure	CLT	Y	Timber structure specified.
	Columns	Glulam	Y	Timber structure specified.
	Beams	Glulam	Y	Timber structure specified.
	Secondary Beams	Glulam	Y	Timber structure specified.
	Floor Slab	CLT	N/A	Timber structure specified.
	Joisted Floors	*None*		
External Materials	Roof	Metal Deck	N/A	No roof structure specified, assumption made.
	Roof Insulation	*None*		
	Roof Finishes	Ceramic tile	N	Should be "Green Roof".
	Facade	Curtain Walling	Y	
	Wall Insulation	*None*		
	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	N	Brick mentioned externally, not blockwork internally.
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15	N	Should be 15,000m2. Seems to ignore commas in numbers.
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		

continues on next page

General Building Info	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	2	N	Should be 8. Seems to confuse storeys below and above ground.
	Storeys Below Ground	2	Y	
	Glazing Ratio (%)	40	Y	

Table D.3: Values found for description with structural material changed (2).

D.3 External Material Description Change(s)

Baseline:

Curtain wall glazing with brick cladding and green roof.

Variation 1:

Changed to timber finish with asphalt roof.

" *The project is an 8 storey office building with 2 levels of underground parking. It features a steel frame structure with a deep pile foundation. The façade is a timber finish, with a flat asphalt roof. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced.* "

Embodied Carbon Prediction:

1401.65 kgCO2e/m²

Extracted Features

	Feature	**Value**	**Correct?**	**Notes**
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*		
	Ground Insulation	*None*		
Structure Materials	Core Structure	CLT	N/A	Correct material selected, but specified use is for finishes.
	Columns	Steel	Y	Steel frame structure specified.
	Beams	Steel	Y	Steel frame structure specified.
	Secondary Beams	Steel	Y	Steel frame structure specified.
	Floor Slab	*None*		
	Joisted Floors	Timber Joists	N	Correct material selected, but specified use is for finishes.
External Materials	Roof	CLT	Y	
	Roof Insulation	*None*		
	Roof Finishes	Green Roof	N	No green roof specified. Only timber assumptions should be made.
	Facade	Party Wall Brick	Y	
	Wall Insulation	*None*		

continues on next page

External Materials	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	Y	
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15	N	Should be 15,000m2. Seems to ignore commas in numbers.
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		
	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	2	N	Should be 8. Seems to confuse storeys below and above ground.
	Storeys Below Ground	2	Y	
	Glazing Ratio (%)	40	Y	

Table D.4: Values found for description with external material changed (1).

Variation 2:

Changed to masonry finish with timber roof.

" *The project is an 8 storey office building with 2 levels of underground parking. It features a steel frame structure with a deep pile foundation. The façade is a masonry finish, with a flat timber roof. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced.* "

Embodied Carbon Prediction:

1534.21 kgCO2e/m²

Extracted Features

	Feature	Value	Correct?	Notes
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*		
	Ground Insulation	*None*		
Structure Materials	Core Structure	CLT	N/A	Correct material selected, but specified use is for roof.
Structure Materials	Columns	Steel	Y	Steel frame structure specified.
	Beams	Steel	Y	Steel frame structure specified.

continues on next page

Structure Materials	Secondary Beams	Steel	Y	Steel frame structure specified.
	Floor Slab	*None*		
	Joisted Floors	Timber Joists	N	Correct material selected, but specified use is for roof.
External Materials	Roof	CLT	Y	
	Roof Insulation	*None*		
	Roof Finishes	Asphalt (Mastic)	Y	
	Facade	Blockwork with Timber	Y	
	Wall Insulation	*None*		
External Materials	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	CLT	Y	
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15	N	Should be 15,000m2. Seems to ignore commas in numbers.
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		
	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	2	N	Should be 8. Seems to confuse storeys below and above ground.
	Storeys Below Ground	2	Y	
	Glazing Ratio (%)	40	Y	

Table D.5: Values found for description with external material changed (2).

D.4 Numerical Specification Change

Variation 1:

Changed to "15,000" to "15000" and remove floors below ground. Changed glazing to 80%.

" *The project is an 8 storey office building. It features a steel frame structure with a deep pile foundation. The façade includes 80% curtain wall glazing and brick cladding on the sides. The roof is flat with a green roof system. It has a gross internal area of 15000 square meters, with a floor height of 3.5 meters and a building perimeter of 200 meters. It is highly serviced.* "

Embodied Carbon Prediction:

1747.28 kgCO2e/m²

Extracted Features

	Feature	Value	Correct?	Notes
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*		
	Ground Insulation	*None*		
Structure Materials	Core Structure	*None*		
	Columns	Steel	Y	Steel frame structure specified.
	Beams	Steel	Y	Steel frame structure specified.
	Secondary Beams	Steel	Y	Steel frame structure specified.
	Floor Slab	*None*		
	Joisted Floors	*None*		
External Materials	Roof	Metal Deck	N/A	No roof structure specified, assumption seems to be made from "steel".
	Roof Insulation	*None*		
	Roof Finishes	Ceramic tile	N	Should be "Green Roof".
	Facade	Curtain Walling	Y	
	Wall Insulation	*None*		
	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	N	Brick mentioned externally, not blockwork internally.
	Ceilings	*None*		
	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
General Building Info	Gross Internal Area (m2)	15000	Y	
	Building Perimeter (m)	200	Y	
	Building Footprint (m2)	*None*		

continues on next page

General Building Info	Building Width (m)	*None*		
	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	*None*	N	Should be 8. Is not recognising storeys above ground
	Storeys Below Ground	*None*		
	Glazing Ratio (%)	80	Y	

Table D.6: Values found for description with numerical specification changed (1).

Variation 2:

Removed storeys below ground and changed "storey" wording to "floors". Changed perimeter to 500m.

" *The project is office building with 8 floors. It features a steel frame structure with a deep pile foundation. The façade includes 40% curtain wall glazing and brick cladding on the sides. The roof is flat with a green roof system. It has a gross internal area of 15,000 square meters, with a floor height of 3.5 meters and a building perimeter of 500 meters. It is highly serviced.* "

Embodied Carbon Prediction:

1746.71 kgCO2e/m²

Extracted Features

	Feature	**Value**	**Correct?**	**Notes**
Sub-Structure Materials	Piles	Steel	Y	Element specified but no material.
	Pile Caps	*None*		
	Capping Beams	*None*		
	Raft Foundation	*None*		
	Basement Walls	*None*		
	Lowest Floor Slab	*None*		
	Ground Insulation	*None*		
Structure Materials	Core Structure	*None*		
	Columns	Steel	Y	Steel frame structure specified.
	Beams	Steel	Y	Steel frame structure specified.
	Secondary Beams	Steel	Y	Steel frame structure specified.
	Floor Slab	*None*		
	Joisted Floors	*None*		
External Materials	Roof	Metal Deck	N/A	No roof structure specified, assumption seems to be made from "steel".
	Roof Insulation	*None*		
	Roof Finishes	Ceramic tile	N	Should be "Green Roof".
	Facade	Curtain Walling	Y	
	Wall Insulation	*None*		
	Glazing	Single Glazing	N/A	No glazing type specified, passable assumption made from curtain walling.
	Window Frames	*None*		
Internal Materials	Partitions	Blockwork	N	Brick mentioned externally, not blockwork internally.
	Ceilings	*None*		

continues on next page

	Floors	Raised floor	N	Not mentioned in baseline description.
	Services	*None*	N	Not found.
Typology	Sector	Residential	N	Should be "Non-residential".
	Sub-Sector	*None*	N	Should be "Office"
	Gross Internal Area (m2)	15000	Y	
	Building Perimeter (m)	500	Y	
	Building Footprint (m2)	*None*		
	Building Width (m)	*None*		
General Building Info	Floor-to-Floor Height (m)	3.5	Y	
	Storeys Above Ground	8	Y	
	Storeys Below Ground	*None*		
	Glazing Ratio (%)	40	Y	

Table D.7: Values found for description with numerical specification changed (2).

D.5 Total Feature Correctness

From a total of 34 possible features:

Variation	**Correctly Found**	**Incorrectly Found or Assumed**	**Arbitrarily Assumed**	**Not In Text**
Baseline Case	9	8	2	15
Structural Variation 1	10	8	5	11
Structural Variation 2	10	7	4	13
External Variation 1	11	7	2	14
External Variation 2	12	6	2	14
Numerical Variation 1	9	7	2	16
Numerical Variation 2	10	6	2	16

Table D.8: Correctly and Incorrectly "Found" features across tests.

E. Accuracy Analysis Raw Results

E.1 Case Study Summary Descriptions

Case Study	Summary Description
Canal Reach	This is a 54921m2 GIA commercial office building in the UK. The structure is a new build with 12 storeys, made of a post-tensioned concrete frame structure. The building features an aluminium unitized curtain wall façade, designed with reduced metal content for lower carbon. It is a high service building.
Stephen Taylor Court	The project features a Cross-Laminated Timber (CLT) frame, which contributed to reduced substructure requirements, with the use of a low EC raft foundation. Each building is predominantly three storeys, with brick exteriors and triple glazed windows. It is a low serviced building.
Harris Academy	This is an education building with 4 storeys and a gross internal area of 10625. It uses a hybrid structure of CLT and Glulam, and steel framing, with concrete cores. It has a minimal raft foundation, with copper and aluminium, as well as brick cladding for the external finish. It then has timber cladding internally and timber batten ceiling finishes.
Kings Cross Sports Hall	This is a school sports hall, constructed of mass timber with CLT walls, CLT roofs, and glulam beams. The structure has a dark zinc metal façade and spans a gia of 2032m2 over 2 storeys.
The Standard (incl. existing materials)	This project was an adaptive reuse and retrofit of the former Camden Town Hall. It incorporated the existing precast concrete façade, adding 3 new storeys with a lightweight steel frame. Materials used included a stainless steel cladding, as well as a timber cladding on the ground floor. The new steel columns are integrated into the existing concrete frame. In total, the building has a gia of 17317 m2.
The Standard (excl. existing materials)	This project was an adaptive reuse and retrofit of the former Camden Town Hall. It added 3 new storeys with a lightweight steel frame. Materials used included a stainless steel cladding, as well as a timber cladding on the ground floor. In total, the building has a gia of 17317 m2.
Stratford Pavillion	The pavillion is a 1500m2 gia retail building with a lightweight timber structure, incorporating CLT panels and Glulam beams. The building has a concrete core, ground floor slab, and concrete basement. Some secondary steel beams are used to support cantilever limitations.
Kings Road	This project is the demolition of an existing office, and development of new mixed use structure. It is proposed to be four storeys with a concrete frame, CLT floor flabs, and some recycled steel components. The gia is 17177 m2.

Table E.1: Summary Descriptions for LETI Case Studies.

The building descriptions used for accuracy analysis were summarised from LETI (n.d.) case studies using OpenAI's (2024) ChatGPT 4.

E.2 Case Study Comparisons

Case Study	**Actual Total Embodied Carbon (kgCO2e/m2)**	**ECO Predicted Total Embodied Carbon (kgCO2e/m2)**
Canal Reach	1178	2208
Stephen Taylor Court	274	819
Harris Academy	725	1975
Kings Cross Sports Hall	429	1422
The Standard (incl. existing materials)	304	1506
The Standard (excl. existing materials)	304	1399
Stratford Pavillion	793	966
Kings Road	920	1507

Table E.2: Comparison of ECO Predictions vs Actual Case Study ECs (incl Retrofits).

F. Linguistic Robustness Raw Results

F.1 Case Study Comparisons

Building ID	Description ID	Description	Predicted Embodied Carbon (kgCO/m²)
1	1	The apartment complex is built using reinforced concrete with brick cladding and features triple-glazed windows.	1105.26
	2	A structure with concrete reinforcement, brick exterior walls, and windows outfitted with triple glazing.	1107.11
	3	This residential building employs a reinforced concrete framework, finished with brick facades, and equipped with triple-pane glass windows.	1219.58
	4	The structure of this building includes a reinforced concrete base, brick as the primary exterior material, and high-efficiency triple-glazed windows.	1171.94
	5	A reinforced concrete structure, encased in brick cladding, with energy-efficient triple glazing.	1105.06
2	1	The office tower utilises a steel frame with glass curtain walls and a concrete core for stability.	1587.84
	2	A steel-structured high-rise with expansive glass facades and a central concrete core.	1823.74
	3	This commercial building is made of steel, with a glass panel exterior, reinforced by a concrete core.	1551.92
	4	The skyscraper is built with a steel framework, encased in a glass curtain wall, and stabilized by a concrete core.	1749.51
	5	A steel-framed office tower, wrapped in glass panels, with a structural concrete core for support.	1749.51
3	1	The warehouse is constructed with a prefabricated steel frame, insulated metal panels, and a concrete raft foundation.	1447.15
	2	A prefabricated steel-framed warehouse with metal cladding and a reinforced concrete base.	1139.03
	3	This industrial building features a steel structural frame, insulated metal sheeting, and a durable concrete floor.	1171.89
	4	The structure includes a steel framework, metal exterior panels, and a solid concrete foundation.	1442.91
	5	A steel-framed warehouse with metal paneling for walls and a heavy-duty concrete slab as the foundation.	1139.04
4	1	The school is built with a timber frame, brick facades, and large aluminum-framed windows.	2149.14
	2	A timber structure educational building with brick exteriors and aluminum window frames.	1587.3
	3	This facility uses a wooden frame, complemented by brick outer walls and aluminum-framed glazing.	2001.81

continues on next page

4	4	The building's construction includes a wooden structure, brick facing, and large windows with aluminum frames.	2001.81
	5	A timber framed school with brick facades, featuring windows encased in aluminum frames.	1988.82
5	1	The mixed-use building has a concrete core, steel framing, and a combination of glass and aluminum panel exteriors.	2059.52
	2	A concrete core structure with steel supports and a mix of glass and aluminum cladding on the outside.	2231.7
	3	This development features a central concrete shaft, steel framework, and exterior walls made of glass and aluminum panels.	2192.55
	4	The building includes a reinforced concrete core, steel structural components, and a glass and aluminum facade.	2231.7
	5	A combination of concrete and steel supports the structure, with the exterior finished in glass and aluminum paneling.	2231.7

Table F.1: Predicted Embodied Carbon (kgCO/m^2) for Different Description Phrasing of Building Designs.

The building descriptions used for linguistic robustness tests were re-phrased using OpenAI's (2024) ChatGPT 4.

G. Model Training Code

G.1 File: data_scraper.vb

```
Sub AutomateBuildingVariants()
    Application.Calculation = xlCalculationAutomatic

    Dim wb As Workbook
    Dim inputSheetProject As Worksheet, inputSheetEmbodied As Worksheet
    Dim outputSheet As Worksheet
    Dim sheetName As String
    Dim sheetIndex As Integer
    Dim sector As Variant, subSector As Variant, gia As Double
    Dim perimeter As Double, footprint As Double, width As Double, height As
        Double
    Dim storeysAbove As Integer, storeysBelow As Integer, glazingRatio As
        Double
    Dim rowCounter As Long, sheetCounter As Integer
    Dim buildingElements As Variant, materialOptions As Object
    Dim currentMaterials As Variant

    Dim iterationCount As Long
    Dim userLimit As Long
    userLimit = InputBox("Enter how many data points you want.", "Set Limit",
        1000) ' Default 1000 values

    ' Setup workbook And sheets
    Set wb = ThisWorkbook
    Set inputSheetProject = wb.Sheets("0. INPUT Project Details")
    Set inputSheetEmbodied = wb.Sheets("2. INPUT Embodied Carbon")

    ' Create unique sheet name
    sheetIndex = 1
    sheetName = "Results " & sheetIndex
    While SheetExists(sheetName, wb)
        sheetIndex = sheetIndex + 1
        sheetName = "Results " & sheetIndex
    Wend

    ' Add New sheet With unique name
    Set outputSheet = wb.Sheets.Add(After:=wb.Sheets(wb.Sheets.Count))
    outputSheet.Name = sheetName

    ' Initialize building elements And their corresponding material options
    Set materialOptions = CreateObject("Scripting.Dictionary")
    buildingElements = Array("Piles", "Pile caps", "Capping beams", "Raft",
        "Basement walls", "Lowest floor slab", _
    "Ground insulation", "Core structure", "Columns", "Beams", "Secondary
        beams", "Floor slab", _
```

```
    "Joisted floors", "Roof", "Roof insulation", "Roof finishes", "Facade", "Wall insulation", _
    "Glazing", "Window frames", "Partitions", "Ceilings", "Floors", "Services")

    ' Initialize building elements And their corresponding material options
    Set materialOptions = CreateObject("Scripting.Dictionary")

    ' Add material options For each building element
    materialOptions.Add "Piles", Array("RC 32/40 (50kg/m3 reinforcement)", "Steel", "")
    materialOptions.Add "Pile caps", Array("RC 32/40 (200kg/m3 reinforcement)", "")
    materialOptions.Add "Capping beams", Array("RC 32/40 (200kg/m3 reinforcement)", "Foamglass (domestic only)", "")
    materialOptions.Add "Raft", Array("RC 32/40 (150kg/m3 reinforcement)", "")
    materialOptions.Add "Basement walls", Array("RC 32/40 (125kg/m3 reinforcement)", "")
    materialOptions.Add "Lowest floor slab", Array("RC 32/40 (150kg/m3 reinforcement)", "Beam And Block", "")
    materialOptions.Add "Ground insulation", Array("EPS", "XPS", "Glass mineral wool", "")
    materialOptions.Add "Core structure", Array("CLT", "Precast RC 32/40 (100kg/m3 reinforcement)", "RC 32/40 (100kg/m3 reinforcement)", "")
    materialOptions.Add "Columns", Array("Glulam", "Iron (existing buildings)", "Precast RC 32/40 (300kg/m3 reinforcement)", "RC 32/40 (300kg/m3 reinforcement)", "Steel", "")
    materialOptions.Add "Beams", Array("Glulam", "Iron (existing buildings)", "Precast RC 32/40 (250kg/m3 reinforcement)", "RC 32/40 (250kg/m3 reinforcement)", "Steel", "")
    materialOptions.Add "Secondary beams", Array("Glulam", "Iron (existing buildings)", "Precast RC 32/40 (250kg/m3 reinforcement)", "RC 32/40 (250kg/m3 reinforcement)", "Steel", "")
    materialOptions.Add "Floor slab", Array("CLT", "Precast RC 32/40 (100kg/m3 reinforcement)", "RC 32/40 (100kg/m3 reinforcement)", "Steel Concrete Composite", "")
    materialOptions.Add "Joisted floors", Array("JJI Engineered Joists + OSB topper", "Timber Joists + OSB topper (Domestic)", "Timber Joists + OSB topper (Office)", "")
    materialOptions.Add "Roof", Array("CLT", "Metal Deck", "Precast RC 32/40 (100kg/m3 reinforcement)", "RC 32/40 (100kg/m3 reinforcement)", "Steel Concrete Composite", "Timber Cassette", "Timber Pitch Roof", "")
    materialOptions.Add "Roof insulation", Array("Cellulose, loose fill", "EPS", "Expanded Perlite", "Expanded Vermiculite", "Glass mineral wool", "PIR", "Rockwool", "Sheeps wool", "Vacuum Insulation", "Woodfibre", "XPS", "")
    materialOptions.Add "Roof finishes", Array("Aluminium", "Asphalt (Mastic)", "Asphalt (Polymer modified)", "Bitumous Sheet", "Ceramic tile", "Fibre cement tile", "Green Roof", "Roofing membrane (PVC)", "Slate tile", "Zinc Standing Seam", "")
```

```
    materialOptions.Add "Facade", Array("Blockwork With Brick", "Blockwork
        With render", "Blockwork With Timber", "Curtain Walling", "Load
        Bearing Precast Concrete Panel", "Load Bearing Precast Concrete With
        Brick Slips", "Party Wall Blockwork", "Party Wall Brick", "Party Wall
        Timber Cassette", "SFS With Aluminium Cladding", "SFS With Brick",
        "SFS With Ceramic Tiles", "SFS With Granite", "SFS With Limestone",
        "SFS With Zinc Cladding", "Solid Brick, single leaf", "Timber Cassette
        Panel With brick", "Timber Cassette Panel With Cement Render", "Timber
        Cassette Panel With Larch Cladding", "Timber Cassette Panel With Lime
        Render", "Timber SIPs With Brick", "")
    materialOptions.Add "Wall insulation", Array("Cellulose, loose fill",
        "EPS", "Expanded Perlite", "Expanded Vermiculite", "Glass mineral
        wool", "PIR", "Rockwool", "Sheeps wool", "Vacuum Insulation",
        "Woodfibre", "XPS", "")
    materialOptions.Add "Glazing", Array("Triple Glazing", "Double Glazing",
        "Single Glazing", "")
    materialOptions.Add "Window frames", Array("Al/Timber Composite",
        "Aluminium", "Steel (single glazed)", "Solid softwood timber frame",
        "uPVC", "")
    materialOptions.Add "Partitions", Array("CLT", "Plasterboard + Steel
        Studs", "Plasterboard + Timber Studs", "Plywood + Timber Studs",
        "Blockwork", "")
    materialOptions.Add "Ceilings", Array("Exposed Soffit", "Plasterboard",
        "Steel grid system", "Steel tile", "Steel tile With 18mm acoustic
        pad", "Suspended plasterboard", "")
    materialOptions.Add "Floors", Array("70mm screed", "Carpet", "Earthenware
        tile", "Raised floor", "Solid timber floorboards", "Stoneware tile",
        "Terrazzo", "Vinyl", "")
    materialOptions.Add "Services", Array("Low", "Medium", "High", "")

    ' Initialize sector options And Sub-sectors
    Dim sectorOptions As Variant
    Dim allSubSectors As Object
    Set allSubSectors = CreateObject("Scripting.Dictionary")
    sectorOptions = Array("Housing", "Office")
    allSubSectors.Add "Housing", Array("Flat/maisonette", "Single family
        house", "Multi-family (< 6 storeys)", "Multi-family (6 - 15 storeys)",
        "Multi-family (> 15 storeys)")
    allSubSectors.Add "Office", Array("Office")

    ' Prepare header For the first Results Sheet
    Call PrepareResultsSheetHeader(outputSheet, buildingElements)
    ' Counter initialization
    rowCounter = 2
    sheetCounter = 1
    startRow = 29

    ' Random selection process
    Do While iterationCount < userLimit
        ' Constraint dims
        Dim hasPiles As Boolean
```

```
        Dim hasCappingbeams As Boolean
        Dim hasPilecaps As Boolean
        Dim hasFloorSlab As Boolean

        hasPiles = False
        hasCappingbeams = False
        hasPilecaps = False
        hasFloorSlab = False

        For Each sector In sectorOptions
            For Each subSector In allSubSectors(sector)
                gia = Int((20000 + 1) * Rnd)
                perimeter = Int((5000 - 100 + 1) * Rnd + 100)
                footprint = Int((10000 - 100 + 1) * Rnd + 100)
                width = Int((200 - 10 + 1) * Rnd + 10)
                height = Int((6 - 2.3 + 1) * Rnd * 10) / 10 + 2.3
                storeysAbove = Int((60 - 1 + 1) * Rnd + 1)
                storeysBelow = Int((5 - 0 + 1) * Rnd)
                glazingRatio = Int((80 - 10 + 1) * Rnd + 10)

                ' Initialize current materials array
                ReDim currentMaterials(UBound(buildingElements))

                ' Recursive Call To process all material combinations
                ProcessMaterials 0, buildingElements, materialOptions, currentMaterials, _
                outputSheet, rowCounter, sector, subSector, gia, _
                perimeter, footprint, width, height, storeysAbove, _
                storeysBelow, glazingRatio, startRow, hasPiles, hasCappingbeams, hasPilecaps, _
                hasFloorSlab

                iterationCount = iterationCount + 1
                If iterationCount >= userLimit Then
                 Exit Do
                End If

                If rowCounter Mod 60 = 1 Then
                    Application.Goto outputSheet.Cells(rowCounter - 1, 1), True
                End If

            Next subSector
            If iterationCount >= userLimit Then
             Exit Do
            End If
        Next sector
    Loop
    MsgBox "Automation complete!"
End Sub
' Recursive Function To handle all material combinations
```

```
Sub ProcessMaterials(Byval elementIndex As Integer, Byref buildingElements As Variant, Byref materialOptions As Object, _
    Byref currentMaterials As Variant, Byref outputSheet As Worksheet, Byref rowCounter As Long, _
    Byval sector As Variant, Byval subSector As Variant, Byval gia As Double, _
    Byval perimeter As Double, Byval footprint As Double, Byval width As Double, _
    Byval height As Double, Byval storeysAbove As Integer, Byval storeysBelow As Integer, _
    Byval glazingRatio As Double, Byval startRow As Integer, Byval hasPiles As Boolean, _
    Byval hasCappingbeams As Boolean, Byval hasPilecaps As Boolean, Byval hasFloorSlab As Boolean)

    If elementIndex > UBound(buildingElements) Then
        ' All elements have materials assigned, output the results
        RecordResults currentMaterials, outputSheet, rowCounter, sector, subSector, gia, _
        perimeter, footprint, width, height, storeysAbove, storeysBelow, glazingRatio
        rowCounter = rowCounter + 1
     Exit Sub
    End If

    Dim element As String
    element = buildingElements(elementIndex)
    Dim materials As Variant
    materials = materialOptions(element)

    Randomize ' Initialize random number generator
    Dim i As Integer
    i = Int((UBound(materials) - LBound(materials) + 1) * Rnd + LBound(materials))

    ' Skip logic - If realistic building conditions aren't met
    Select Case element
     Case "Raft"
        If hasCappingbeams Or hasPilecaps Then
            ProcessMaterials elementIndex + 1, buildingElements, materialOptions, currentMaterials, _
            outputSheet, rowCounter, sector, subSector, gia, perimeter, footprint, width, _
            height, storeysAbove, storeysBelow, glazingRatio, startRow, hasPiles, hasCappingbeams, _
            hasPilecaps, hasFloorSlab
         Exit Sub
        End If

     Case "Pile caps", "Capping beams"
        If Not hasPiles Then
```

```
            ProcessMaterials elementIndex + 1, buildingElements,
                materialOptions, currentMaterials, _
            outputSheet, rowCounter, sector, subSector, gia, perimeter,
                footprint, width, _
            height, storeysAbove, storeysBelow, glazingRatio, startRow,
                hasPiles, hasCappingbeams, _
            hasPilecaps, hasFloorSlab
         Exit Sub
        End If

    Case "Basement walls"
        If storeysBelow = 0 Then
            ProcessMaterials elementIndex + 1, buildingElements,
                materialOptions, currentMaterials, _
            outputSheet, rowCounter, sector, subSector, gia, perimeter,
                footprint, width, _
            height, storeysAbove, storeysBelow, glazingRatio, startRow,
                hasPiles, hasCappingbeams, _
            hasPilecaps, hasFloorSlab
         Exit Sub
        End If

    Case "Joisted floors"
        If hasFloorSlab Then
            ProcessMaterials elementIndex + 1, buildingElements,
                materialOptions, currentMaterials, _
            outputSheet, rowCounter, sector, subSector, gia, perimeter,
                footprint, width, _
            height, storeysAbove, storeysBelow, glazingRatio, startRow,
                hasPiles, hasCappingbeams, _
            hasPilecaps, hasFloorSlab
         Exit Sub
        End If
    End Select

    currentMaterials(elementIndex) = materials(i) ' Assign And log the
        selected material

    If materials(i) <> "" Then
        Select Case element
         Case "Piles"
            hasPiles = True

         Case "Pile caps"
            hasPilecaps = True

         Case "Capping beams"
            hasCappingbeams = True

         Case "Floor slab"
            hasFloorSlab = True
```

```
            End Select
        End If


        ' Set the material For the current building element in the input sheet
        ThisWorkbook.Sheets("2. INPUT Embodied Carbon").Cells(startRow +
            elementIndex, 3).Value = materials(i)

        ' Recursively process the Next element With the Next index
        ProcessMaterials elementIndex + 1, buildingElements, materialOptions,
            currentMaterials, _
        outputSheet, rowCounter, sector, subSector, gia, perimeter, footprint,
            width, _
        height, storeysAbove, storeysBelow, glazingRatio, startRow, hasPiles,
            hasCappingbeams, _
        hasPilecaps, hasFloorSlab
End Sub
' Function To record results
Sub RecordResults(Byref currentMaterials As Variant, Byref outputSheet As
    Worksheet, Byval rowCounter As Long, _
        Byval sector As Variant, Byval subSector As Variant, Byval gia As Double,
            _
        Byval perimeter As Double, Byval footprint As Double, Byval width As
            Double, _
        Byval height As Double, Byval storeysAbove As Integer, Byval storeysBelow
            As Integer, _
        Byval glazingRatio As Double)

        Dim wb As Workbook
        Set wb = ThisWorkbook
        embodiedCarbon = wb.Sheets("5. OUTPUT Machine").Cells(19, 2).Value

        outputSheet.Cells(rowCounter, 1).Value = sector
        outputSheet.Cells(rowCounter, 2).Value = subSector
        outputSheet.Cells(rowCounter, 3).Value = gia
        outputSheet.Cells(rowCounter, 4).Value = perimeter
        outputSheet.Cells(rowCounter, 5).Value = footprint
        outputSheet.Cells(rowCounter, 6).Value = width
        outputSheet.Cells(rowCounter, 7).Value = height
        outputSheet.Cells(rowCounter, 8).Value = storeysAbove
        outputSheet.Cells(rowCounter, 9).Value = storeysBelow
        outputSheet.Cells(rowCounter, 10).Value = glazingRatio

        ' Output materials
        Dim colIdx As Integer
        colIdx = 11
        Dim idx As Integer
        For idx = LBound(currentMaterials) To UBound(currentMaterials)
            outputSheet.Cells(rowCounter, colIdx).Value = currentMaterials(idx)
            colIdx = colIdx + 1
        Next idx
```

```

    outputSheet.Cells(rowCounter, colIdx).Value = embodiedCarbon
End Sub

Function SheetExists(sheetName As String, wb As Workbook) As Boolean
    Dim tmpSheet As Worksheet
    On Error Resume Next
    Set tmpSheet = wb.Sheets(sheetName)
    On Error Goto 0
        SheetExists = Not tmpSheet Is Nothing
End Function

Private Sub PrepareResultsSheetHeader(sheet As Worksheet, buildingElements As
   Variant)
    Dim col As Integer
    sheet.Cells(1, 1).Value = "Sector"
    sheet.Cells(1, 2).Value = "Sub-Sector"
    sheet.Cells(1, 3).Value = "GIA (m2)"
    sheet.Cells(1, 4).Value = "Building Perimeter"
    sheet.Cells(1, 5).Value = "Building Footprint"
    sheet.Cells(1, 6).Value = "Building Width"
    sheet.Cells(1, 7).Value = "Floor-To-Floor Height"
    sheet.Cells(1, 8).Value = "No. Storeys Ground & Above"
    sheet.Cells(1, 9).Value = "No. Storeys Below Ground"
    sheet.Cells(1, 10).Value = "Glazing Ratio"

    col = 11
    Dim i As Integer
    For Each element In buildingElements
        Debug.Print element
        sheet.Cells(1, col).Value = element & " Material"
        col = col + 1
    Next element

    sheet.Cells(1, col).Value = "Embodied Carbon (kgCO2e/m2)"
End Sub
```

G.2 File: model_train_validate.py

```
"""
This script is designed to load a dataset, preprocess it, train multiple
    machine learning models,
evaluate their performance using cross-validation, and save the models along
    with relevant metadata.
The script limits the dataset size for model training, tunes the models, and
    logs their performance metrics.
"""

import os
import joblib
from datetime import datetime
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score
from sklearn.base import clone
from model_utils import tune_model, load_datasets, prepare_datasets,
    save_model_and_data

# Define the base directory and model paths
current_dir = os.path.dirname(os.path.abspath(__file__))
model_dir = os.path.join(current_dir, "../src/model")
os.makedirs(model_dir, exist_ok=True)

df = load_datasets()

# Save unique values from the dataset before preprocessing for later use
unique_values = {col: df[col].dropna().unique().tolist() for col in
    df.columns}
joblib.dump(unique_values, os.path.join(model_dir, "unique_values.pkl"))

X_cleaned, y_cleaned, cleaned_label_encoders = prepare_datasets(df)

# Save feature names for later use
joblib.dump(X_cleaned.columns.tolist(), os.path.join(model_dir,
    "features.pkl"))

# Save label encoders for later use
joblib.dump(cleaned_label_encoders, os.path.join(model_dir,
    "label_encoders.pkl"))

# Limiter for the number of data points to train
LIMITER = 150000  # Modify this value as needed

# Ensure LIMITER does not exceed the available data points
LIMITER = min(LIMITER, X_cleaned.shape[0])

X_cleaned_limited = X_cleaned.iloc[:LIMITER]
y_cleaned_limited = y_cleaned.iloc[:LIMITER]

```

```
# Tune models and store the best estimators
model_cleaned = tune_model(X_cleaned_limited, y_cleaned_limited)

# Initialize performance logs
performance_logs = []

for model_name, model in model_cleaned.items():
    full_model_name = f"synthetic_{model_name}"

    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(
        X_cleaned_limited, y_cleaned_limited, test_size=0.3, random_state=42
    )

    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    r_squared_train = r2_score(y_train, y_train_pred)

    print(f"R-squared for {full_model_name} on training set: {r_squared_train}")
    performance_logs.append(f"{full_model_name}: Training R-squared: {r_squared_train}")

    # Evaluate the model on the test set
    y_test_pred = model.predict(X_test)
    r_squared_test = r2_score(y_test, y_test_pred)
    print(f"R-squared for {full_model_name} on testing set: {r_squared_test}")
    performance_logs.append(f"{full_model_name}: Testing R-squared: {r_squared_test}")

    # Clone the model to perform cross-validation without affecting the original model
    model_cv = clone(model)
    if hasattr(model_cv, "verbose"):
        model_cv.verbose = 0

    # Calculate cross-validation scores
    try:
        cv_scores = cross_val_score(
            model_cv, X_train, y_train, cv=5, scoring="r2", verbose=0
        )
        cv_mean = cv_scores.mean()
        print(f"Cross-validation scores for {full_model_name}: {cv_scores}")
        print(f"Mean cross-validation score for {full_model_name}: {cv_mean}")

        performance_logs.append(
            f"{full_model_name}: Cross-validation scores: {cv_scores}"
        )
        performance_logs.append(
            f"{full_model_name}: Mean cross-validation score: {cv_mean}\n"
        )
```

```
    except Exception as e:
        print(f"Error in cross-validation for {full_model_name}: {e}")
        performance_logs.append(f"{full_model_name}: Cross-validation error: {e}\n")

    # Save the model and associated data
    save_model_and_data(model, full_model_name, model_dir, performance_logs)

# Save performance logs to a text file with date and time
log_dir = os.path.join(current_dir, "../data/logs")
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
performance_file_path = os.path.join(log_dir,
    f"performance_models_{timestamp}.txt")
with open(performance_file_path, "w") as f:
    for line in performance_logs:
        f.write(line + "\n")

print(f"Performance data saved to {performance_file_path}")
```

G.3 File: model_utils.py

```
import pandas as pd
import os
import joblib
from sklearn.ensemble import (
    HistGradientBoostingRegressor,
    GradientBoostingRegressor,
    RandomForestRegressor,
    StackingRegressor,
)
from sklearn.svm import SVR
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.stats import uniform, randint
import numpy as np

# Define parameter distributions for hyperparameter tuning
param_dist_gb = {
    "regressor__n_estimators": randint(50, 5000),
    "regressor__learning_rate": uniform(0.001, 0.2),
    "regressor__max_depth": randint(3, 10),
    "regressor__min_samples_split": randint(2, 20),
    "regressor__min_samples_leaf": randint(1, 20),
    "regressor__subsample": uniform(0.5, 0.5),
}

param_dist_hgb = {
    "regressor__max_iter": randint(50, 5000),
    "regressor__learning_rate": uniform(0.001, 0.2),
    "regressor__max_depth": randint(3, 10),
    "regressor__min_samples_leaf": randint(1, 20),
    "regressor__l2_regularization": uniform(0, 1),
    "regressor__max_bins": randint(10, 255),
    "regressor__max_leaf_nodes": randint(10, 255),
}

param_dist_rf = {
    "regressor__n_estimators": randint(50, 5000),
    "regressor__max_depth": randint(3, 20),
    "regressor__min_samples_split": randint(2, 20),
    "regressor__min_samples_leaf": randint(1, 20),
}

param_dist_lr = {"regressor__fit_intercept": [True, False]}

models = {
    "HistGradientBoosting": (
        HistGradientBoostingRegressor(random_state=42, verbose=1),
        param_dist_hgb,
```

```
    ),
    "GradientBoosting": (
        GradientBoostingRegressor(random_state=42, verbose=1),
        param_dist_gb,
    ),
    "RandomForest": (
        RandomForestRegressor(random_state=42, verbose=1),
        param_dist_rf,
    ),
}


# Function to perform hyperparameter tuning
def tune_model(X, y):
    best_estimators = {}
    for model_name, (model, param_dist) in models.items():
        pipeline = Pipeline([("scaler", StandardScaler()), ("regressor",
            model)])

        param_space_size = np.prod(
            [len(v) if hasattr(v, "__len__") else 1 for v in
                param_dist.values()]
        )
        n_iter = min(50, param_space_size)

        random_search = RandomizedSearchCV(
            estimator=pipeline,
            param_distributions=param_dist,
            n_iter=n_iter,
            cv=5,
            n_jobs=-1,
            scoring="r2",
            random_state=42,
            verbose=1,
        )
        random_search.fit(X, y)
        best_estimators[model_name] = random_search.best_estimator_
        print(f"Best parameters for {model_name}:
            {random_search.best_params_}")
    return best_estimators


# Function to load datasets
def load_datasets():
    current_dir = os.path.dirname(os.path.abspath(__file__))
    import_dir = os.path.join(current_dir, "../data/processed/inspect")

    synthetic_PATH = os.path.join(import_dir, "cleaned_synthetic.csv")

    synthetic_df = pd.read_csv(synthetic_PATH)

```

```python
    return synthetic_df


# Function to encode categorical features and return label encoders
def encode_categorical(df):
    le_dict = {}
    for col in df.select_dtypes(include=["object"]).columns:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))
        le_dict[col] = le
    return df, le_dict


# Function to prepare datasets for model training
def prepare_datasets(synthetic_df, target_column="Embodied Carbon (kgCO2e/m2)"):
    synthetic_df, synthetic_df_encoders = encode_categorical(synthetic_df)

    X_synthetic = synthetic_df.drop(columns=[target_column])
    y_synthetic = synthetic_df[target_column]

    return (X_synthetic, y_synthetic, synthetic_df_encoders)


# Function to save models and R-squared data
def save_model_and_data(model, model_name, model_dir, performance_logs):
    joblib_file = os.path.join(model_dir, f"{model_name}.pkl")
    joblib.dump(model, joblib_file)
    print(f"{model_name} saved to {joblib_file}")


# Function to load valid models
def load_valid_models(model_dir, model_files):
    valid_models = []
    for model_file in model_files:
        model_path = os.path.join(model_dir, model_file)
        if os.path.exists(model_path):
            model = joblib.load(model_path)
            if isinstance(model, Pipeline):  # Check if the loaded object is a Pipeline
                valid_models.append(model)
            else:
                print(f"Error: {model_file} is not a valid Pipeline object")
        else:
            print(f"Model file {model_file} not found.")
    return valid_models


# Function to align input features with training features
def align_features(input_df, training_columns):
    input_df = pd.get_dummies(input_df)
```

```
    aligned_df = pd.DataFrame(columns=training_columns)
    for col in training_columns:
        if col in input_df.columns:
            aligned_df[col] = input_df[col]
        else:
            aligned_df[col] = 0
    return aligned_df


# Function to create stacking ensemble
def create_stacking_ensemble(models, final_estimator):
    estimators = [(name, model) for name, model in models.items()]
    stacking_regressor = StackingRegressor(
        estimators=estimators, final_estimator=final_estimator, n_jobs=-1
    )
    return stacking_regressor
```

H. Implementation Code

H.1 Github Link

https://github.com/dfoshidero/Early-Stage-Carbon-Observer$_{E}CO$

H.2 File: app.py

```
import time
import numpy as np
from flask import Flask, request, jsonify
from flask_cors import CORS
import os
import psutil
import gc
import logging
import multiprocessing
from model_predictor import load_resources, predict as model_predict
from feature_extractor import extract, initialize_resources


def create_app():
    app = Flask(__name__)
    CORS(app)
    app.logger.setLevel(logging.INFO)

    model, features, label_encoders, _ = load_resources()
    model_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)),
        "model")
    initialize_resources(model_dir)

    def predict(
        SECTOR,
        SUBSECTOR,
        GIA,
        PERIMETER,
        FOOTPRINT,
        WIDTH,
        HEIGHT,
        ABOVE_GROUND,
        BELOW_GROUND,
        GLAZING_RATIO,
        PILES,
        PILE_CAPS,
        CAPPING_BEAMS,
        RAFT,
        BASEMENT_WALLS,
        LOWEST_FLOOR_SLAB,
        GROUND_INSULATION,
```

```
        CORE_STRUCTURE,
        COLUMNS,
        BEAMS,
        SECONDARY_BEAMS,
        FLOOR_SLAB,
        JOISTED_FLOORS,
        ROOF,
        ROOF_INSULATION,
        ROOF_FINISHES,
        FACADE,
        WALL_INSULATION,
        GLAZING,
        WINDOW_FRAMES,
        PARTITIONS,
        CEILINGS,
        FLOORS,
        SERVICES,
    ):
        user_input = {
            "Sector": [None if SECTOR == "None" else SECTOR],
            "Sub-Sector": [None if SUBSECTOR == "None" else SUBSECTOR],
            "Gross Internal Area (m2)": [None if GIA == "None" else GIA],
            "Building Perimeter (m)": [None if PERIMETER == "None" else PERIMETER],
            "Building Footprint (m2)": [None if FOOTPRINT == "None" else FOOTPRINT],
            "Building Width (m)": [None if WIDTH == "None" else WIDTH],
            "Floor-to-Floor Height (m)": [None if HEIGHT == "None" else HEIGHT],
            "Storeys Above Ground": [None if ABOVE_GROUND == "None" else ABOVE_GROUND],
            "Storeys Below Ground": [None if BELOW_GROUND == "None" else BELOW_GROUND],
            "Glazing Ratio (%)": [None if GLAZING_RATIO == "None" else GLAZING_RATIO],
            "Piles Material": [None if PILES == "None" else PILES],
            "Pile Caps Material": [None if PILE_CAPS == "None" else PILE_CAPS],
            "Capping Beams Material": [
                None if CAPPING_BEAMS == "None" else CAPPING_BEAMS
            ],
            "Raft Foundation Material": [None if RAFT == "None" else RAFT],
            "Basement Walls Material": [
                None if BASEMENT_WALLS == "None" else BASEMENT_WALLS
            ],
            "Lowest Floor Slab Material": [
                None if LOWEST_FLOOR_SLAB == "None" else LOWEST_FLOOR_SLAB
            ],
            "Ground Insulation Material": [
                None if GROUND_INSULATION == "None" else GROUND_INSULATION
            ],
```

```
            "Core Structure Material": [
                None if CORE_STRUCTURE == "None" else CORE_STRUCTURE
            ],
            "Columns Material": [None if COLUMNS == "None" else COLUMNS],
            "Beams Material": [None if BEAMS == "None" else BEAMS],
            "Secondary Beams Material": [
                None if SECONDARY_BEAMS == "None" else SECONDARY_BEAMS
            ],
            "Floor Slab Material": [None if FLOOR_SLAB == "None" else
               FLOOR_SLAB],
            "Joisted Floors Material": [
                None if JOISTED_FLOORS == "None" else JOISTED_FLOORS
            ],
            "Roof Material": [None if ROOF == "None" else ROOF],
            "Roof Insulation Material": [
                None if ROOF_INSULATION == "None" else ROOF_INSULATION
            ],
            "Roof Finishes Material": [
                None if ROOF_FINISHES == "None" else ROOF_FINISHES
            ],
            "Facade Material": [None if FACADE == "None" else FACADE],
            "Wall Insulation Material": [
                None if WALL_INSULATION == "None" else WALL_INSULATION
            ],
            "Glazing Material": [None if GLAZING == "None" else GLAZING],
            "Window Frames Material": [
                None if WINDOW_FRAMES == "None" else WINDOW_FRAMES
            ],
            "Partitions Material": [None if PARTITIONS == "None" else
               PARTITIONS],
            "Ceilings Material": [None if CEILINGS == "None" else CEILINGS],
            "Floors Material": [None if FLOORS == "None" else FLOORS],
            "Services": [None if SERVICES == "None" else SERVICES],
        }

        prediction = model_predict(user_input, model, features,
            label_encoders)

        prediction_list = (
            prediction.tolist() if isinstance(prediction, np.ndarray) else
               prediction
        )

        log_memory_usage("During Prediction")

        return prediction_list

    def log_memory_usage(phase):
        process = psutil.Process(os.getpid())
        memory_info = process.memory_info()
        gc.collect()
```

```
        app.logger.info(
            f"[{phase}] Memory Usage: RSS={memory_info.rss / (1024 * 1024):.2f} MB, VMS={memory_info.vms / (1024 * 1024):.2f} MB"
        )

    def log_free_memory():
        memory_info = psutil.virtual_memory()
        free_memory = memory_info.free / (1024 * 1024)  # Convert bytes to MB
        available_memory = memory_info.available / (1024 * 1024)  # Convert bytes to MB

        app.logger.info(f"Total Free Memory: {free_memory:.2f} MB")
        app.logger.info(f"Total Available Memory: {available_memory:.2f} MB")

    def process_predict(data):
        prediction = predict(
            data.get("SECTOR"),
            data.get("SUBSECTOR"),
            data.get("GIA"),
            data.get("PERIMETER"),
            data.get("FOOTPRINT"),
            data.get("WIDTH"),
            data.get("HEIGHT"),
            data.get("ABOVE_GROUND"),
            data.get("BELOW_GROUND"),
            data.get("GLAZING_RATIO"),
            data.get("PILES"),
            data.get("PILE_CAPS"),
            data.get("CAPPING_BEAMS"),
            data.get("RAFT"),
            data.get("BASEMENT_WALLS"),
            data.get("LOWEST_FLOOR_SLAB"),
            data.get("GROUND_INSULATION"),
            data.get("CORE_STRUCTURE"),
            data.get("COLUMNS"),
            data.get("BEAMS"),
            data.get("SECONDARY_BEAMS"),
            data.get("FLOOR_SLAB"),
            data.get("JOISTED_FLOORS"),
            data.get("ROOF"),
            data.get("ROOF_INSULATION"),
            data.get("ROOF_FINISHES"),
            data.get("FACADE"),
            data.get("WALL_INSULATION"),
            data.get("GLAZING"),
            data.get("WINDOW_FRAMES"),
            data.get("PARTITIONS"),
            data.get("CEILINGS"),
            data.get("FLOORS"),
            data.get("SERVICES"),
        )
```

```
        prediction_list = (
            prediction.tolist() if isinstance(prediction, np.ndarray) else prediction
        )
        return prediction_list

    def process_extract(text):
        extracted_values = extract(text)
        for key, value in extracted_values.items():
            if isinstance(value, np.ndarray):
                extracted_values[key] = value.tolist()
        log_memory_usage("During Extraction")
        return extracted_values

    def process_extract_predict(text):
        extracted_values = process_extract(text)

        formatted_values = {
            "SECTOR": extracted_values.get("Sector"),
            "SUBSECTOR": extracted_values.get("Sub-Sector"),
            "GIA": extracted_values.get("Gross Internal Area (m2)"),
            "PERIMETER": extracted_values.get("Building Perimeter (m)"),
            "FOOTPRINT": extracted_values.get("Building Footprint (m2)"),
            "WIDTH": extracted_values.get("Building Width (m)"),
            "HEIGHT": extracted_values.get("Floor-to-Floor Height (m)"),
            "ABOVE_GROUND": extracted_values.get("Storeys Above Ground"),
            "BELOW_GROUND": extracted_values.get("Storeys Below Ground"),
            "GLAZING_RATIO": extracted_values.get("Glazing Ratio (%)"),
            "PILES": extracted_values.get("Piles Material"),
            "PILE_CAPS": extracted_values.get("Pile Caps Material"),
            "CAPPING_BEAMS": extracted_values.get("Capping Beams Material"),
            "RAFT": extracted_values.get("Raft Foundation Material"),
            "BASEMENT_WALLS": extracted_values.get("Basement Walls Material"),
            "LOWEST_FLOOR_SLAB": extracted_values.get("Lowest Floor Slab Material"),
            "GROUND_INSULATION": extracted_values.get("Ground Insulation Material"),
            "CORE_STRUCTURE": extracted_values.get("Core Structure Material"),
            "COLUMNS": extracted_values.get("Columns Material"),
            "BEAMS": extracted_values.get("Beams Material"),
            "SECONDARY_BEAMS": extracted_values.get("Secondary Beams Material"),
            "FLOOR_SLAB": extracted_values.get("Floor Slab Material"),
            "JOISTED_FLOORS": extracted_values.get("Joisted Floors Material"),
            "ROOF": extracted_values.get("Roof Material"),
            "ROOF_INSULATION": extracted_values.get("Roof Insulation Material"),
            "ROOF_FINISHES": extracted_values.get("Roof Finishes Material"),
            "FACADE": extracted_values.get("Facade Material"),
            "WALL_INSULATION": extracted_values.get("Wall Insulation Material"),
```

```
            "GLAZING": extracted_values.get("Glazing Material"),
            "WINDOW_FRAMES": extracted_values.get("Window Frames Material"),
            "PARTITIONS": extracted_values.get("Partitions Material"),
            "CEILINGS": extracted_values.get("Ceilings Material"),
            "FLOORS": extracted_values.get("Floors Material"),
            "SERVICES": extracted_values.get("Services"),
        }

        prediction = predict(**formatted_values)
        prediction_list = (
            prediction.tolist() if isinstance(prediction, np.ndarray) else prediction
        )
        return prediction_list

    # Function to run target function in a subprocess
    def target_func(queue, func, args):
        result = func(*args)
        queue.put(result)

    # Wrapper functions for subprocess execution
    def subprocess_wrapper(func, *args):
        log_memory_usage("Before Process")
        result_queue = multiprocessing.Queue()
        p = multiprocessing.Process(target=target_func, args=(result_queue, func, args))
        p.start()
        p.join()
        result = result_queue.get()
        p.terminate()
        log_memory_usage("After Process")
        return result

    @app.route("/predict", methods=["POST"])
    def predict_route():
        app.logger.info("#################### PREDICTION CALLED ####################")
        start_time = time.time()
        data = request.get_json()
        prediction = subprocess_wrapper(process_predict, data)
        elapsed_time = time.time() - start_time
        log_free_memory()
        app.logger.info(f"Total time for /predict route: {elapsed_time:.2f} seconds...")
        return jsonify(prediction)

    @app.route("/extract", methods=["POST"])
    def extract_route():
        app.logger.info("#################### EXTRACTION CALLED ####################")
        start_time = time.time()
```

```
        text = request.get_json().get("text")
        extracted_values = subprocess_wrapper(process_extract, text)
        elapsed_time = time.time() - start_time
        log_free_memory()
        app.logger.info(f"Total time for /extract route: {elapsed_time:.2f}
            seconds...")
        return jsonify(extracted_values)

    @app.route("/extract_predict", methods=["POST"])
    def extract_predict_route():
        app.logger.info(
            "#################### FULL PIPELINE CALLED ####################"
        )
        start_time = time.time()
        text = request.get_json().get("text")
        result = subprocess_wrapper(process_extract_predict, text)
        elapsed_time = time.time() - start_time
        log_free_memory()
        app.logger.info(
            f"Total time for /extract_predict route: {elapsed_time:.2f}
                seconds..."
        )
        return jsonify(result)

    return app
```

H.3 File: feature_extractor.py

```python
import os
import re
import json
import joblib
import spacy
import random
import numpy as np

from word2number import w2n
from collections import Counter
from sentence_transformers import SentenceTransformer, util

# Initialize variables for models and other resources
_nlp_model = None
_sentence_transformer_model = None
_stop_words = None
_unique_values = None
_synonym_dict = None
_numerical_keywords = None


def initialize_resources(model_dir):
    global _nlp_model, _sentence_transformer_model, _stop_words,
        _unique_values

    if _nlp_model is None:
        _nlp_model = spacy.load("en_core_web_md", disable=["parser"])

    if _sentence_transformer_model is None:
        _sentence_transformer_model = SentenceTransformer("all-MiniLM-L6-v2")

    if _stop_words is None:
        _stop_words = spacy.lang.en.stop_words.STOP_WORDS

    if _unique_values is None:
        _unique_values = load_unique_values(model_dir)


numerical_features = [
    "Gross Internal Area (m2)",
    "Building Perimeter (m)",
    "Building Footprint (m2)",
    "Building Width (m)",
    "Floor-to-Floor Height (m)",
    "Storeys Above Ground",
    "Storeys Below Ground",
    "Glazing Ratio (%)",
]

```

```python
SIMILARITY_THRESHOLD = 0.7  # Define a similarity threshold


def load_json(json_path, cache):
    if cache is None:
        with open(json_path, "r") as f:
            cache = json.load(f)
    return cache


def load_unique_values(model_dir):
    path_unq_vals = os.path.join(model_dir, "unique_values.pkl")
    unique_values = joblib.load(path_unq_vals)
    return unique_values


def get_related_terms(word):
    related_terms = set()
    for key, synonyms in _synonym_dict.items():
        if word.lower() == key.lower() or word.lower() in map(str.lower,
            synonyms):
            related_terms.add(key)
            related_terms.update(synonyms)
    return related_terms


def preprocess_text(text):
    doc = _nlp_model(text)
    processed_tokens = []

    for token in doc:
        if token.text.lower() not in _stop_words:
            lemma = token.lemma_
            related_terms = get_related_terms(lemma)
            if related_terms:
                processed_tokens.extend(related_terms)
            else:
                processed_tokens.append(lemma)

    return processed_tokens


def filter_pos_tags(tokens):
    doc = _nlp_model(" ".join(tokens))
    filtered_tokens = [token.text for token in doc if token.pos_ in {"NOUN",
        "ADJ"}]
    return filtered_tokens


def find_nearest_word(text, target_word, window_size=5):
    words = text.split()
```

```
    if target_word in words:
        target_idx = words.index(target_word)
        start_idx = max(0, target_idx - window_size)
        end_idx = min(len(words), target_idx + window_size + 1)
        return words[start_idx:end_idx]
    return []


def apply_building_logic(features):
    # Extract features for easier reference
    sector = features.get("Sector")
    sub_sector = features.get("Sub-Sector")
    storeys_below = features.get("Storeys Below Ground", 0)
    timber_joists = features.get("Joisted Floors Material")

    if storeys_below == 0:
        features["Basement Walls Material"] = None

    if sector == "Residential" and timber_joists:
        features["Joisted Floors"] = "Timber Joists (Domestic)"
    elif sector == "Non-residential" and timber_joists:
        features["Joisted Floors"] = "Timber Joists (Office)"

    if sector == "Residential" and sub_sector == "Non-residential":
        features["Sub-Sector"] = None
    elif sector == "Non-residential":
        features["Sub-Sector"] = "Non-residential"

    return features


def random_choice_conflicting_features(features, input_text):
    input_text_lower = input_text.lower()

    has_piles = features.get("Piles") is not None
    if not has_piles:
        features["Pile Caps Material"] = None
        features["Capping Beams Material"] = None

    # Choose "Raft" or "Pile Caps"/"Capping Beams" based on input text
    if "raft" in input_text_lower:
        features["Pile Caps Material"] = None
        features["Capping Beams Material"] = None
    elif "pile caps" in input_text_lower or "capping beams" in input_text_lower:
        features["Raft Material"] = None
    elif features.get("Raft Material") and (
        features.get("Pile Caps Material") or features.get("Capping Beams Material")
    ):
        if random.choice([True, False]):
```

```
                features["Pile Caps Material"] = None
                features["Capping Beams Material"] = None
            else:
                features["Raft Material"] = None

    # Choose "Joisted Floors" or "Floor Slab" based on input text
    if "joists" in input_text_lower:
        features["Floor Slab Material"] = None
    elif "slab" in input_text_lower:
        features["Joisted Floors Material"] = None
    elif features.get("Joisted Floors Material") and features.get(
        "Floor Slab Material"
    ):
        if random.choice([True, False]):
            features["Floor Slab Material"] = None
        else:
            features["Joisted Floors Material"] = None

    return features


def extract_feature_values(
    input_text,
    numerical_features,
    threshold=SIMILARITY_THRESHOLD,
):
    nlp = _nlp_model
    model = _sentence_transformer_model
    doc = nlp(input_text)
    explicit_features, filtered_text = extract_explicit_features(
        input_text, model, numerical_features
    )
    doc_filtered = nlp(filtered_text)
    ner_entities = [ent.text for ent in doc_filtered.ents]

    preprocessed_tokens = preprocess_text(filtered_text)
    filtered_tokens = filter_pos_tags(preprocessed_tokens)

    candidates = set(ner_entities + filtered_tokens)

    feature_matches = explicit_features.copy()
    matched_features = set(explicit_features.keys())

    # Process general cases for remaining features
    for feature, values in _unique_values.items():
        if (
            feature in numerical_features
            or feature == "Embodied Carbon (kgCO2e/m2)"
            or feature in feature_matches
        ):
            continue
```

```

        unique_embeddings = model.encode(values)
        candidate_embeddings = model.encode(list(candidates))

        best_match = None
        highest_score = float("-inf")

        for candidate, candidate_embedding in zip(candidates,
            candidate_embeddings):
            similarities = util.pytorch_cos_sim(candidate_embedding,
                unique_embeddings)
            max_similarity = similarities.max().item()
            if max_similarity > highest_score:
                highest_score = max_similarity
                best_match = values[similarities.argmax().item()]

        if highest_score >= threshold:
            feature_matches[feature] = best_match
        else:
            feature_matches[feature] = None

    # Apply the building logic rules
    feature_matches = apply_building_logic(feature_matches)

    # Randomly choose between conflicting features
    feature_matches = random_choice_conflicting_features(feature_matches,
        input_text)

    return feature_matches


def extract_numerical_feature(text, label, feature_keywords):
    pattern = re.compile(
        r"(\b\d+\.?\d*(?:sqm|sqft|km|m|cm|mm|in|ft|yd|mg|g|kg|lb|oz|liters|ml|gal|kw|hp)?
        re.IGNORECASE,
    )
    feature_numbers = {feature: [] for feature in feature_keywords.keys()}

    words = text.split()
    converted_text = []
    for word in words:
        try:
            number = w2n.word_to_num(word)
            converted_text.append(str(number))
        except ValueError:
            converted_text.append(word)
    updated_text = " ".join(converted_text)
    words = updated_text.split()

    for i, word in enumerate(words):
        for feature, keywords in feature_keywords.items():
```

```
            if any(kw in word.lower() for kw in keywords):
                window = words[max(i - 3, 0) : min(i + 4, len(words))]
                for w in window:
                    match = pattern.match(w)
                    if match:
                        # Extract the numerical value
                        num_str = match.group(1)
                        # Remove any non-numeric characters for conversion
                        num_val = re.sub(r"[^\d.]", "", num_str)
                        feature_numbers[feature].append(float(num_val))

    for feature in feature_numbers:
        if feature_numbers[feature]:
            feature_numbers[feature] = max(
                set(feature_numbers[feature]), key=feature_numbers[feature].count
            )
        else:
            feature_numbers[feature] = "None"

    # Special rule: Set "Storeys Below Ground" to 1 if "a basement" is mentioned
    if "a basement" in text.lower():
        if feature_numbers["Storeys Below Ground"] == "None":
            feature_numbers["Storeys Below Ground"] = 1

    return feature_numbers


def extract_explicit_features(
    input_text,
    model,
    numerical_features,
    threshold=SIMILARITY_THRESHOLD,
):
    explicit_features = {}
    word_count = Counter(input_text.lower().split())
    context_count = Counter()

    for feature in _unique_values.keys():
        if feature in numerical_features or feature == "Embodied Carbon (kgCO2e/m2)":
            continue

        feature_cleaned = feature.lower().replace(" material", "")
        pattern = rf"\b{feature_cleaned}\b"
        matches = re.finditer(pattern, input_text, re.IGNORECASE)

        for match in matches:
            nearby_words = find_nearest_word(input_text, match.group(), window_size=5)
```

```
            preprocessed_tokens = preprocess_text(" ".join(nearby_words))
            filtered_tokens = filter_pos_tags(preprocessed_tokens)

            if filtered_tokens:
                unique_embeddings = model.encode(_unique_values[feature])
                candidate_embeddings = model.encode(filtered_tokens)

                best_match = None
                highest_score = float("-inf")
                original_word = None

                for candidate, candidate_embedding in zip(
                    filtered_tokens, candidate_embeddings
                ):
                    similarities = util.pytorch_cos_sim(
                        candidate_embedding, unique_embeddings
                    )
                    max_similarity = similarities.max().item()
                    if max_similarity > highest_score:
                        highest_score = max_similarity
                        best_match = _unique_values[feature][
                            similarities.argmax().item()
                        ]
                        original_word = candidate

                if highest_score >= threshold:
                    explicit_features[feature] = best_match
                    context_count.update([original_word.lower()])
                    break

    filtered_words = [
        word
        for word in input_text.split()
        if context_count[word.lower()] < word_count[word.lower()]
    ]
    filtered_text = " ".join(filtered_words)
    return explicit_features, filtered_text


def extract(input_text):
    global _synonym_dict, _numerical_keywords
    current_dir = os.path.dirname(os.path.abspath(__file__))
    synonyms_path = os.path.join(current_dir, "config/synonyms.json")
    numerical_keywords_path = os.path.join(
        current_dir, "config/numerical_keywords.json"
    )

    _synonym_dict = load_json(synonyms_path, _synonym_dict)
    _numerical_keywords = load_json(numerical_keywords_path,
        _numerical_keywords)

```

```
    feature_values = extract_feature_values(
        input_text,
        numerical_features,
        SIMILARITY_THRESHOLD,
    )

    for feature in numerical_features:
        numerical_values = extract_numerical_feature(
            input_text, feature, _numerical_keywords
        )
        feature_values[feature] = numerical_values[feature]

    return feature_values
```

H.4 File: model_predictor.py

```
import joblib
import os
import pandas as pd
import numpy as np
import gc


def load_resources():
    """
    Load the necessary resources.
    :return: tuple of loaded resources
    """
    current_dir = os.path.dirname(os.path.abspath(__file__))
    model_dir = os.path.join(current_dir, "model")

    features_filepath = os.path.join(model_dir, "features.pkl")
    label_encoders_filepath = os.path.join(model_dir, "label_encoders.pkl")
    synthetic_model_filepath = os.path.join(
        model_dir, "synthetic_HistGradientBoosting.pkl"
    )
    unique_values_filepath = os.path.join(model_dir, "unique_values.pkl")

    with open(synthetic_model_filepath, "rb") as f:
        model = joblib.load(f)
    with open(features_filepath, "rb") as f:
        features = joblib.load(f)
    with open(label_encoders_filepath, "rb") as f:
        label_encoders = joblib.load(f)
    with open(unique_values_filepath, "rb") as f:
        unique_values = joblib.load(f)

    return model, features, label_encoders, unique_values


def apply_label_encoding(user_input, label_encoders):
    """
    Apply label encoding to the user input using the provided label encoders.

    :param user_input: dictionary with user inputs
    :param label_encoders: dictionary with label encoders
    :return: DataFrame with label encoded features
    """
    encoded_input = {}
    for feature, values in user_input.items():
        if feature in label_encoders:
            encoder = label_encoders[feature]
            encoded_values = []
            for value in values:
                if value in encoder.classes_:
```

```
                    encoded_values.append(encoder.transform([value])[0])
                elif "Other" in encoder.classes_:
                    encoded_values.append(encoder.transform(["Other"])[0])
                else:
                    new_classes = np.append(encoder.classes_, "Unknown")
                    encoder.classes_ = new_classes
                    encoded_values.append(encoder.transform(["Unknown"])[0])
            encoded_input[feature] = encoded_values
        else:
            encoded_input[feature] = values
    return pd.DataFrame(encoded_input)


def preprocess_input(user_input, features, label_encoders):
    """
    Preprocess user input using the provided label encoders.

    :param user_input: dictionary with user inputs
    :param features: list of feature names used during training
    :param label_encoders: dictionary with label encoders
    :return: preprocessed input DataFrame
    """
    input_df = apply_label_encoding(user_input, label_encoders)
    aligned_df = align_features(input_df, features)

    if aligned_df.empty:
        raise ValueError(
            "Aligned DataFrame is empty. Check if input features match
                training features."
        )

    # Clear input DataFrame to free memory
    del input_df
    gc.collect()

    return aligned_df


def predict(user_input, model, features, label_encoders):
    """
    Predict using the model.

    :param user_input: dictionary with user inputs
    :param model: trained model
    :param features: list of feature names used during training
    :param label_encoders: dictionary with label encoders
    :return: prediction result
    """
    preprocessed_input = preprocess_input(user_input, features,
        label_encoders)
    prediction = model.predict(preprocessed_input)
```

```

    # Clear intermediate data
    del preprocessed_input
    gc.collect()

    return prediction


def align_features(input_df, training_columns):
    """
    Align input features with training features.
    :param input_df: DataFrame with user inputs
    :param training_columns: List of feature names used during training
    :return: DataFrame with aligned features
    """
    aligned_df = pd.DataFrame(columns=training_columns)
    for col in training_columns:
        if col in input_df.columns:
            aligned_df[col] = input_df[col]
        else:
            aligned_df[col] = np.nan  # Keep missing values as NaN

    # Clear input DataFrame to free memory
    del input_df
    gc.collect()

    return aligned_df


def validate_user_input(user_input, unique_values):
    """
    Validate user input against unique values.

    :param user_input: dictionary with user inputs
    :param unique_values: dictionary with unique values for each feature
    :return: None, raises ValueError if validation fails
    """
    for feature, values in user_input.items():
        if feature in unique_values:
            for value in values:
                if value not in unique_values[feature]:
                    raise ValueError(
                        f"Value for {feature} can only be
                           {unique_values[feature]}."
                    )

    # Clear user_input and unique_values to free memory
    del user_input, unique_values
    gc.collect()
```

I. (Relevant) Frontend Code

I.1 File: api_utils.py

```
// utils.jsx
import axios from "axios";

// Base URL for the API
const BASE_URL = "https://*************************************";

/**
 * Function to call the extract_predict API.
 * @param {string} text - The input text.
 * @returns {Promise<number>} - The numerical prediction.
 */
export const extractPredict = async (text) => {
  try {
    const response = await axios.post(`${BASE_URL}/extract_predict`, { text:
       text });
    return response.data;
  } catch (error) {
    console.error("Error in extractPredict:", error);
    throw error;
  }
};

/**
 * Function to call the extract API.
 * @param {string} text - The input text.
 * @returns {Promise<Object>} - The extracted features.
 */
export const extract = async (text) => {
  try {
    const response = await axios.post(`${BASE_URL}/extract`, { text: text });
    return response.data;
  } catch (error) {
    console.error("Error in extract:", error);
    throw error;
  }
};

/**
 * Function to call the predict API.
 * @param {Object} data - The extracted features.
 * @returns {Promise<number>} - The numerical prediction.
 */
export const predict = async (data) => {
  try {
    const formattedData = {
```

```
      SECTOR: data["Sector"],
      SUBSECTOR: data["Sub-Sector"],
      GIA: data["Gross Internal Area (m2)"],
      PERIMETER: data["Building Perimeter (m)"],
      FOOTPRINT: data["Building Footprint (m2)"],
      WIDTH: data["Building Width (m)"],
      HEIGHT: data["Floor-to-Floor Height (m)"],
      ABOVE_GROUND: data["Storeys Above Ground"],
      BELOW_GROUND: data["Storeys Below Ground"],
      GLAZING_RATIO: data["Glazing Ratio (%)"],
      PILES: data["Piles Material"],
      PILE_CAPS: data["Pile Caps Material"],
      CAPPING_BEAMS: data["Capping Beams Material"],
      RAFT: data["Raft Foundation Material"],
      BASEMENT_WALLS: data["Basement Walls Material"],
      LOWEST_FLOOR_SLAB: data["Lowest Floor Slab Material"],
      GROUND_INSULATION: data["Ground Insulation Material"],
      CORE_STRUCTURE: data["Core Structure Material"],
      COLUMNS: data["Columns Material"],
      BEAMS: data["Beams Material"],
      SECONDARY_BEAMS: data["Secondary Beams Material"],
      FLOOR_SLAB: data["Floor Slab Material"],
      JOISTED_FLOORS: data["Joisted Floors Material"],
      ROOF: data["Roof Material"],
      ROOF_INSULATION: data["Roof Insulation Material"],
      ROOF_FINISHES: data["Roof Finishes Material"],
      FACADE: data["Facade Material"],
      WALL_INSULATION: data["Wall Insulation Material"],
      GLAZING: data["Glazing Material"],
      WINDOW_FRAMES: data["Window Frames Material"],
      PARTITIONS: data["Partitions Material"],
      CEILINGS: data["Ceilings Material"],
      FLOORS: data["Floors Material"],
      SERVICES: data["Services"]
    };

    const response = await axios.post(`${BASE_URL}/predict`, formattedData);
    return response.data;
  } catch (error) {
    console.error("Error in predict:", error);
    throw error;
  }
};
```