Generative Artificial Intelligence and Optimisation Framework for Sustainable Concrete Mixture Design

A Project Report Submitted
for Partial Fulfilment of the Requirements for the Award of the Degree of
Master of Technology

by

Teiboklang Chyne

M.Tech. (Robotics and Artificial Intelligence)
Roll Number: 234156019

Under the Supervision of
Dr. Biranchi Panda

Center for Intelligent Cyber Physical Systems
Indian Institute of Technology Guwahati
Guwahati-781039, India

May 2025

Dedicated to,

My beloved family, without whose endless love and support, I could not achieve this.

Declaration of Authorship

I, TEIBOKLANG CHYNE, declare that this thesis titled, '*Generative Artificial Intelligence and Optimisation Framework for Sustainable Concrete Mixture Design*', and the work presented in it are my own, under the guidance of my supervisor. I confirm that:

- This work was done wholly while in candidature for Master of Technology in ROBOTICS AND ARTIFICIAL INTELLIGENCE at Indian Institute of Technology Guwahati.
- This work, wholly or partially, has not been submitted to any other institute or university for any degree or diploma.
- I have abided by the norms and guidelines given in the Ethical Code of Conduct of the institute.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have used materials from the work of others, I have given due credit to them by citing them in the text of the report and giving details in the references. With the exception of such citations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: ________________________________

Acknowledgements

I am honoured to have this opportunity to convey my sincere gratitude to my supervisor, Dr. BIRANCHI PANDA, for his precious supervision, constant motivation, and caring mentorship throughout the tenure of this research. Without his immense knowledge and unparalleled vision, this thesis would not have taken this shape.

Place: Guwahati **TEIBOKLANG CHYNE**

Date: **(234156019)**

Center for Intelligent Cyber Physical Systems

Indian Institute of Technology Guwahati

CERTIFICATE

Date:

This is to certify that the dissertation report entitled **'Generative Artificial Intelligence and Optimisation Framework for Sustainable Concrete Mixture Design'**, submitted by **Mr. Teiboklang Chyne** to Indian Institute of Technology Guwahati, is a record of bonafide project work carried out by him under my supervision and guidance, and is worthy of consideration for the award of the degree of *Master of Technology* in *Robotics and Artificial Intelligence* of the Institute.

Place: Guwahati

Date:

Dr. Biranchi Panda

(Supervisor)

Abstract

The integration of 3D concrete printing is transforming the construction industry by enabling innovative structural designs. However, optimizing mix design formulation remains a significant challenge due to its time-consuming and material-intensive nature. This thesis employs machine learning techniques, including the CatBoost algorithm, Conditional Variational Autoencoder (CVAE), and Non-dominated Sorting Genetic Algorithm (NSGA-II), to address these challenges. A dataset of 785 samples with 15 features is analyzed to predict optimal mix designs. The Firefly optimization algorithm is used to fine-tune CatBoost's hyperparameters, enhancing predictive performance. Shapley Additive Explanations (SHAP) analysis is then applied to interpret model predictions and identify key factors influencing mix design. The optimized CatBoost model, with carefully selected hyperparameters such as learning rate, depth, and iterations, reliably predicts concrete performance, ensuring both strength and printability. Additionally, CVAE and NSGA-II are integrated to generate and optimize new mix designs, leveraging the CatBoost model for evaluation. This approach streamlines the design process, reducing material waste and computational time. Future work will focus on validating these predictions through physical 3D printing and testing, particularly in underwater environments, to advance sustainable construction materials.

Keywords: 3D Concrete Printing, Machine Learning, CatBoost Algorithm, Firefly Optimization Algorithm, SHAP Analysis, Conditional Variational Autoencoder, Non-dominated Sorting Genetic Algorithm

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The integration of artificial intelligence (AI) into material science is transforming various industries, with construction being a key beneficiary of this convergence. Among emerging technologies, 3D concrete printing—a form of additive manufacturing—is reshaping traditional construction methods. This technology enables the creation of complex and customized structures that are often difficult, if not impossible, to achieve with conventional methods. In addition to unlocking new possibilities in architectural design, 3D concrete printing offers significant advantages in terms of material efficiency, automation, and cost-effectiveness. Unlike traditional construction processes, which frequently involve substantial material wastage and extended project timelines, 3D concrete printing is highly efficient, requiring fewer resources and less time to complete complex structures. This shift has the potential to enhance sustainability in construction, reducing environmental impact by lowering both material consumption and associated emissions.

Despite its advantages, 3D concrete printing poses unique challenges, especially in the formulation of concrete mix designs suitable for printing. Unlike conventional concrete used in casting, 3D-printable concrete must fulfill multiple criteria to ensure successful application: buildability, extrudability, and interlayer bonding are all crucial for producing structurally sound, durable printed forms. Achieving the right balance among these properties is complex, as each mix component, from cementitious materials and aggregates to additives and water content, directly influences the printability and performance of the final structure. Traditionally, formulating an optimal mix design involves an extensive trial-and-error approach, with researchers iteratively adjusting proportions to achieve desired characteristics. This process is time-consuming and resource-intensive, underscoring the need for predictive modeling approaches that can streamline mix design optimization by identifying the ideal combination of materials without the drawbacks of manual testing.

There are multiple approaches to overcoming these challenges. This paper outlines the steps taken to address them. This research leverages CatBoost, a powerful machine learning algorithm known for its ability to handle categorical data and capture complex, nonlinear relationships within data, to predict optimal mix designs for 3D concrete printing. The dataset analyzed includes various mix constituents, environmental factors, and target concrete properties. To further refine the model's predictive accuracy, the Firefly optimization algorithm (FOA) was employed to fine-tune CatBoost's hyperparameters. This hybrid approach enhances the model's prediction accuracy compared to using CatBoost without hyperparameter optimization.

As part of the model evaluation, a regression slope analysis was performed on the train and test predictions for both the CatBoost model and the Firefly-optimized CatBoost (FOA-CatBoost) model. This analysis provided insight into the model's ability to generalize across different data subsets, with a slope closer to 1 indicating a stronger alignment between the predicted and actual values for both training and testing sets. The results of the regression slope analysis also served as an important validation metric, affirming that the FOA-CatBoost achieved improved generalization and accuracy over the baseline CatBoost model. Furthermore, the performance of the FOA-CatBoost model was compared with studies on data-driven forecasting of the compressive strength of UHPC in the literature, demonstrating performance on par with or superior to other models.

Further insights into the feature importance and model interpretability were obtained using SHapley Additive exPlanations (SHAP) analysis, which provided a detailed examination of the impact of each feature on the model's predictions. This analysis helps to understand how individual mix constituents and environmental factors influence the predicted properties, offering deeper insights for optimizing the mix design.

Through the aforementioned steps, a regression model is developed to predict the compressive strength of concrete based on the quantities of its mix components, with verified performance. However, while such a model is effective for estimation, it alone is insufficient for generating new mixture designs. To address this limitation, a Variational Autoencoder (VAE), a type of neural network used as a generative model, is employed. Once trained, the VAE can generate synthetic data that closely resembles real-world data, making it a valuable tool for creating novel mix designs.

In this study, a Conditional Variational Autoencoder (CVAE), an extension of the VAE, is implemented to facilitate the generation of new concrete mix designs. The CVAE enables the generation of diverse mix compositions conditioned on a specified target compressive strength. By providing the desired compressive strength as a condition, the CVAE produces multiple mix designs with varying component proportions, ensuring that the resulting concrete achieves the specified strength requirement. The training process of the CVAE ensures that the generated mixtures are not only statistically similar to real mix designs but also practically feasible for implementation.

To further enhance the practicality of the generated mixtures, a multi-objective optimization approach is integrated. The Non-dominated Sorting Genetic Algorithm II (NSGA-II), a well-established evolutionary optimization algorithm, is employed to refine the generated mix designs. While the CVAE produces multiple candidate mixtures, the NSGA-II is utilized to identify the most optimal designs by simultaneously minimizing both the cost of the mixture and the embodied carbon associated with its components, all while maintaining the desired compressive strength. This combined approach ensures that the generated mix designs are not only structurally sound but also economically and environmentally sustainable.

The contributions of this research demonstrate that the optimized CatBoost model, supported by regression and SHAP analyses, can effectively predict the performance of 3D-printed concrete. The CVAE has demonstrated its ability to generate mix designs that closely resemble real-world mixtures. The integration of NSGA-II further enhances the framework by optimizing these designs, reducing both cost and embodied carbon while ensuring the desired compressive strength is maintained. This approach significantly reduces reliance on labor-intensive testing and provides a data-driven framework for both predicting the compressive strength of concrete and identifying optimal mix designs. These designs not only meet the required strength criteria for advanced construction projects but also minimize cost and embodied carbon. Future work will focus on validating these predictions through physical 3D printing and testing, particularly in underwater applications, to support the development of sustainable construction materials.

The major contributions of this work are outlined below:

1. **Application to a Unique Dataset:** To the best of current knowledge, the FOA-CatBoost framework has not been previously applied to the dataset used in this study, which comprises 12 mix constituents and 2 environmental parameters. Additionally, this is the first application of a combined CVAE and NSGA-II approach for generating and optimizing mix designs using this dataset. These contributions establish the novelty of both the methodology and the insights derived from the analysis.

2. **Customized Firefly Optimization Strategy:** A modified distance calculation method combining standard scaling and Euclidean distance was introduced to improve the convergence behavior of the Firefly Algorithm during hyperparameter tuning of CatBoost, leading to enhanced model accuracy.

3. **Comparison with Previous Studies:** Rather than comparing against standard machine learning baselines such as XGBoost or Random Forest, this study benchmarks the FOA-CatBoost model against results reported in earlier literature on concrete mix optimization, offering a realistic perspective on its practical relevance.

4. **Feature Importance Analysis:** The application of SHAP (SHapley Additive exPlanations) enables interpretation of the model's predictions, highlighting the most influential mix constituents and environmental conditions that drive the target response.

5. **Improved Prediction Accuracy:** The optimized FOA-CatBoost model demonstrates improved predictive performance, as evidenced by evaluation metrics such as MAE, RMSE, and R^2, underscoring its reliability for data-driven material design.

6. **Exploration of CVAE Architectures:** Multiple Conditional Variational Autoencoder (CVAE) architectures were developed and systematically evaluated based on training/testing loss and convergence behavior. This comparative analysis ensured the selection of a stable and generalizable generative model.

7. **Latent Space Sampling for Mix Design Generation:** The trained CVAE was utilized to sample and generate new concrete mix designs in the latent space conditioned on strength targets, enabling generation of plausible material configurations beyond the training data.

8. **Integration of NSGA-II for Multi-Objective Optimization:** A well-established evolutionary algorithm, NSGA-II, was applied to the CVAE latent space to generate Pareto-optimal mix designs, balancing competing objectives such as compressive strength, material cost, and environmental impact.

9. **Comparison of Sampled vs. Optimized Solutions:** The generated solutions from random sampling in the latent space were compared against those obtained via NSGA-II optimization, demonstrating the effectiveness of evolutionary search in identifying superior designs on the Pareto front.

10. **Generative-Optimization Pipeline for Sustainable Design:** The combination of CVAE-based generative modeling with NSGA-II establishes a pipeline that enables data-driven, sustainable, and multi-objective mix design, bridging predictive modeling with exploratory generation and optimization.

1.1 Objective

This research aims to optimize the mix design formulation for 3D concrete printing using a machine learning algorithm to predict compressive strength of concrete mix designs, enhanced by regression and interpretability analyses. A generative model is developed to produce mix designs for a target compressive strength, followed by a multi-objective optimization algorithm to refine these designs, minimizing both cost and embodied carbon.

By analyzing a dataset comprising 12 mix constituents and 2 environmental factors, this study utilizes CatBoost to improve predictive accuracy while reducing material waste and computational time. The Firefly Optimization Algorithm (FOA) is employed to fine-tune CatBoost's hyperparameters, further enhancing model performance. Additionally, regression slope analysis is conducted on both training and testing predictions for CatBoost and FOA-CatBoost to evaluate generalization, while SHAP analysis provides insights into feature contributions.

A Conditional Variational Autoencoder (CVAE) serves as the generative model for producing new mix designs, and the Non-dominated Sorting Genetic Algorithm II (NSGA-II) is applied to optimize these designs. The ultimate objective is to ensure that the optimized mix designs satisfy essential strength requirements while minimizing cost and embodied carbon, contributing to the development of sustainable construction materials.

1.2 Organization of the Report

This thesis is structured into six main chapters. Chapter one introduces the study, highlighting the role of the CatBoost algorithm and the Firefly Optimization Algorithm (FOA) in predicting the compressive strength of concrete. Additionally, it presents the Conditional Variational Autoencoder (CVAE) as a generative model for mix design generation and the Non-dominated Sorting Genetic Algorithm II (NSGA-II) for multi-objective optimization.

Chapter two provides a comprehensive literature review, covering recent advancements and challenges in predictive modeling and mix design optimization for construction materials.

Chapter three details the materials and methods, including the dataset, the implementation of CatBoost, FOA for hyperparameter tuning, regression analysis comparing baseline CatBoost and FOA-CatBoost, and SHAP analysis for feature interpretation. It also describes the CVAE framework for mix design generation and the NSGA-II optimization approach.

Chapter four presents and discusses the results, including a comparative analysis of the FOA-CatBoost model against data-driven forecasting models for ultra-high-performance concrete (UHPC) in the literature. It also includes a regression slope analysis between training and test sets for both baseline CatBoost and FOA-CatBoost, along with SHAP-based insights into model performance and feature contributions. Furthermore, it evaluates the generative modeling approach by comparing training and validation losses across different CVAE architectures and analyzing learning curves to assess CVAE performance. The characteristics of the generated mix designs and their optimization using NSGA-II are also discussed.

Chapter five summarizes key findings and their implications, followed by recommendations for future research, particularly in sustainable construction applications and presents the conclusion of the thesis.

Chapter 2

Literature Review

To provide a structured and comprehensive overview of the relevant research landscape, the literature review in this study is systematically organized into multiple thematic tables. Each table categorizes the referenced works based on their primary purpose and methodological focus, ensuring clarity in the classification of existing contributions. Specifically, the reviewed literature is grouped under the following categories: (i) Prediction of Concrete Properties, which includes studies focused on estimating mechanical or durability characteristics using machine learning techniques; (ii) Optimization of Concrete Mix Design, encompassing works that apply heuristic or metaheuristic algorithms to achieve optimal mix formulations; (iii) Concrete Mix Generation Using Generative Models, which highlights recent advancements in leveraging deep generative approaches such as CVAEs for creating novel mix designs; and (iv) Foundational Works and Algorithm-Specific Contributions, which presents essential algorithmic developments and theoretical contributions that underpin the predictive and optimization frameworks employed in this research. This organization facilitates a clearer understanding of the progression and integration of different machine learning strategies within the domain of concrete mix design.

To facilitate understanding, a comprehensive list of all abbreviations used in this chapter is provided in Appendix 5.

Table 2.1 presents a compilation of recent studies that employ machine learning techniques to predict key properties of concrete, such as compressive strength, permeability, and workability. These predictive models enhance understanding and performance evaluation of various concrete types.

Table 2.1: Literature on Prediction of Concrete Properties

Author(s)	Year	Objective	Method(s)	Key Contributions
Wu et al. [1]	2024	Predict permeability and compressive strength of pervious concrete	Artificial Neural Networks	Developed ANN models to predict key mechanical and hydraulic properties; demonstrated high accuracy in forecasting permeability and strength
Geng et al. [2]	2024	Predict printability and rheological properties in 3D concrete printing	Random Forest	Developed RF-based predictive models for printability and rheology; demonstrated high accuracy and potential for enhancing 3D printing precision
Huang et al. [3]	2023	Investigate compressive strength of self-compacting concrete with SCMs and recycled aggregate	SVM, KNN, DT, RF, ANN, Gradient Boosting models, and GEP	Applied ML to predict strength of eco-friendly SCC mixes; identified key mix components influencing compressive strength; supported use of recycled materials for sustainable construction
Zou et al. [4]	2023	Analyze the relationship between composition and strength of UHPFRC	AdaBoost, LightGBM, XGBoost, CatBoost and RF	Investigated influence of mix components on UHPFRC strength; used interpretable ML models to enhance understanding of material behavior and support mix optimization
Alyami et al. [5]	2024	Predict compressive strength of 3D printed fiber-reinforced concrete	SVR, DT, RF, GB, and GEP	Applied various ML models to estimate compressive strength; demonstrated effectiveness of data-driven approaches for evaluating 3D printed fiber-reinforced concrete performance

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Elshaarawy et al. [6]	2024	Predict concrete compressive strength and provide user-friendly access	MLR, MNLR, SVR, GEP, ANN, ANFIS, RF, AdaBoost, XGBoost, and CatBoost + Interactive GUI	Developed ML-based predictive models and a graphical user interface; enhanced usability for non-experts and improved prediction accuracy for compressive strength
Khan et al. [7]	2023	Predict and optimize compressive strength of reactive powder concrete	Multilayer stacked model and XGBoost, RF, and KNN	Applied ensemble ML techniques for accurate strength prediction; performed optimization to enhance reactive powder concrete mix design performance
Mustapha et al. [8]	2024	Estimate compressive strength of quaternary blend concrete	Gradient-Boosting Ensemble Methods-LightGBM, XGBoost, and CatBoost	Conducted comparative analysis of gradient-boosting models; identified the most effective ensemble technique for accurate strength prediction of quaternary blends
Katlav et al. [9]	2024	Forecast compressive strength of ultra-high-performance concrete (UHPC)	CatBoost model optimized with various algorithms	Enhanced UHPC strength prediction using optimized CatBoost; demonstrated improved accuracy over traditional ML approaches through comparative optimization techniques
Shi et al. [10]	2024	Predict compressive strength of concrete using optimized ML approach	Random Search + CatBoost (RS-CatBoost)	Proposed RS-CatBoost model for compressive strength prediction; demonstrated improved accuracy and efficiency over traditional models through parameter optimization

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Khodadadi et al. [11]	2024	Predict compressive strength of CFRP-confined circular concrete specimens	Particle Swarm Optimization + CatBoost	Developed a PSO-CatBoost hybrid model for accurate strength prediction; demonstrated superior performance over traditional ML models in confined concrete applications

Table 2.2 summarizes works focused on optimizing concrete mix design through machine learning and metaheuristic algorithms. The methods aim to achieve multi-objective goals including strength, cost-efficiency, and sustainability.

Table 2.2: Literature on Optimization of Concrete Mix Design

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Van Tran et al. [12]	2024	Predict workability of 3D printable concrete with steel slag aggregate	Bayesian Regularization and Evolution Algorithm	Proposed a hybrid ML framework for robust workability prediction; achieved improved generalization and reliability for concrete with alternative aggregates
Malik et al. [13]	2024	Predict anisotropic compressive strength and slump flow in 3D printed concrete	XGBoost, SVM, DTR, GPR, and ANN	Developed predictive models for anisotropic strength and flow; enhanced understanding of directional properties in 3D printed concrete
Huang et al. [14]	2022	Predict compressive strength of ternary concrete containing cement, fly ash, and slag	Firefly Algorithm + RF	Proposed a hybrid FA-RF model for accurate strength prediction; demonstrated superior performance compared to standalone ML methods
Zhang et al. [15]	2021	Automate mix design of lightweight foamed concrete to optimize strength, density, and cost	LSSVR + MOFA	Developed a hybrid MOO framework combining LSSVR and MOFA; achieved high prediction accuracy and generated Pareto-optimal LWC mixtures for early-stage decision making

Author(s)	Year	Objective	Method(s)	Key Contributions
Bui et al. [16]	2018	Predict compressive and tensile strength of high-performance concrete	Modified Firefly Algorithm + ANN	Developed an expert system combining MFA and ANN; improved prediction accuracy and computational efficiency for HPC strength estimation
Mohammadi Golafshani et al. [17]	2024	Predict and optimize compressive strength of recycled aggregate concrete for sustainable mix design	Multiple ML models + Stacking + Multi-objective Water Cycle Algorithm	Achieved high prediction accuracy using stacking; identified key factors via sensitivity analysis; optimized mix for strength, CO2 emissions, and cost using water cycle algorithm; developed user-friendly interface
Li et al. [18]	2023	Predict and optimize compressive strength of sustainable concrete	Squirrel Search Algorithm + XGBoost	Developed SSA-XGBoost hybrid model for compressive strength prediction; achieved high accuracy and enabled optimized sustainable concrete mix design
Zheng et al. [19]	2023	Perform multi-objective optimization of concrete mix design using ML	Bayesian Optimization, 12 Machine Learning models + NSGA-III and C-TAEA	Developed an ML-based framework for optimizing concrete mixes; balanced compressive strength with other mix performance objectives; demonstrated effectiveness in practical mix design scenarios
Golafshani et al. [20]	2024	Develop a low-carbon mix design framework for recycled aggregate concrete with SCMs	XGBoost, LightBoost, and CatBoost + GWO	Proposed an integrated ML and optimization framework; minimized carbon emissions while ensuring mechanical performance of RAC with SCMs

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Li et al. [21]	2024	Predict sintering foundation characteristics of iron ore powder	FOA + CatBoost	Developed FOA-CatBoost model for accurate prediction of sintering properties; demonstrated effectiveness of hybrid approach in handling complex industrial datasets

Table 2.3 lists studies that leverage generative models for creating novel concrete mixtures. These works highlight the integration of deep learning and generative AI for automatic mix formulation, particularly for bioinspired and sustainable designs.

Table 2.3: Literature on Concrete Mix Generation Using Generative Models

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Chiu et al. [22]	2023	Design composite structures with bioinspiration using generative models	CVAE + Genetic Algorithm	Combined CVAE with GA to optimize bioinspired structures; improved material efficiency and performance
Rivera-Perez et al. [23]	2024	Optimize asphalt mix design using deep learning	Autoencoder + DNN	Used unsupervised autoencoders to guide asphalt concrete design; reduced trial-and-error in lab testing
Yu et al. [24]	2023	Use generative AI for engineered cementitious composite design	ANN, and Invertible Neural Network	Applied generative modeling to ECC design; improved mechanical performance and design flexibility
Le Nguyen et al. [25]	2024	Optimize concrete mixtures for cost and carbon using generative AI	BO+ML models, Generative AI + MOO	Developed framework for low-cost, low-carbon mix design; enabled automated generation of optimal concrete formulations
Gao et al. [26]	2024	Generate sustainable concrete mixes using AI	Variational Autoencoder	Demonstrated generative model use in concrete mix design; balanced sustainability and structural performance

Table 2.4 outlines foundational contributions and algorithm-specific innovations that support the theoretical basis of prediction, optimization, and generation methods used in con-

crete mix design. This includes the development of algorithms such as FA, NSGA-II, VAE, and CVAE.

Table 2.4: Foundational Works and Algorithm-Specific Contributions

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Yang et al. [27]	2009	Solve multimodal optimization problems using nature-inspired algorithms	Firefly Algorithm	Introduced the original Firefly Algorithm; demonstrated effectiveness in solving multimodal optimization problems
Fister et al. [28]	2013	Provide a comprehensive review of Firefly Algorithm developments	Literature Review on Firefly Algorithm	Reviewed variations, applications, and performance of FA across domains; identified strengths and future research directions
Deb et al. [29]	2002	Propose an efficient multi-objective optimization algorithm	NSGA-II	Developed fast elitist algorithm for multi-objective problems; became a standard benchmark in evolutionary computation
Kingma et al. [30]	2013	Propose a framework for variational inference in autoencoders	VAE	Introduced VAE, combining probabilistic inference and deep learning; foundational for generative modeling
Sohn et al. [31]	2015	Learn structured outputs with deep generative models	Conditional VAE	Proposed CVAE for structured prediction; demonstrated applications in image and label generation tasks
Burgess et al. [32]	2018	Investigate disentanglement in beta-VAE models	beta-VAE Analysis	Explored role of capacity and constraints in disentanglement; contributed to interpretability in generative modeling
Li et al. [33]	2020	Improve representation learning through progressive disentanglement	Progressive Learning + Disentangled Representation	Proposed hierarchical representation learning method; improved performance and interpretability in generative models

Author(s)	**Year**	**Objective**	**Method(s)**	**Key Contributions**
Boehmke et al. [34]	2020	Guide practical applications of ML using R	Hands-on ML tutorials in R	Offered applied examples and insights into implementing ML algorithms in real-world problems, including preprocessing and modeling

Chapter 3

Materials and Methods

This chapter introduces the dataset, the theoretical foundations of the employed methods, and the methodology utilized in this study. It is organized into four sections: The first section provides a detailed description of the dataset. The second section outlines the regression, optimization, and analysis methods employed for the CatBoost model. The third section presents the generative methods and optimization methods utilized to generate new mixture designs. Finally, the last section discusses the methodology and implementation details.

3.1 Dataset Description

In this study, the dataset used for analysis was sourced from the work of Katlav and Ergen [9], who improved forecasting of the compressive strength of ultra-high-performance concrete (UHPC) using the CatBoost model optimized with different algorithms. The dataset contains 785 rows of data and 15 features, which were used to train and evaluate the models in this research.

The dataset contains several important features relevant to the 3D-printed concrete mix design, categorized into two main groups: input features (independent variables) and an output feature (dependent variable).

3.1.1 Input Features

The input features are the mix constituents and environmental parameters that affect the compressive strength of the printed concrete. These features are as follows:

- **Cement (kg/m^3)**: The amount of cement used in the concrete mix.
- **SF (kg/m^3)**: Silica fume, a fine additive that enhances the concrete's strength.
- **BFS (kg/m^3)**: Blast furnace slag, used as a supplementary cementitious material.

- **FA (kg/m^3)**: Fly ash, a pozzolanic material used to improve the mix's durability and workability.
- **QP (kg/m^3)**: Quarry powder, a filler material in the concrete mix.
- **LSP (kg/m^3)**: Limestone powder, another filler material.
- **NS (kg/m^3)**: Nano-silica, a material used to improve the concrete's mechanical properties.
- **Fiber (kg/m^3)**: Fibers added to enhance tensile strength and reduce cracking.
- **Sand (kg/m^3)**: Fine aggregate used in the mix.
- **Gravel (kg/m^3)**: Coarse aggregate providing structural integrity to the concrete.
- **Water (kg/m^3)**: The water content in the mix, which affects workability and strength.
- **SP (kg/m^3)**: Superplasticizer, a chemical admixture used to improve the flowability of the concrete without adding extra water.
- **T (°C)**: Temperature at the time of mixing and printing, influencing curing and strength development.
- **RH (%)**: Relative humidity during printing, affecting the drying and setting process.
- **Age (days)**: The number of days the printed concrete has been curing.

3.1.2 Output Feature

The output feature is the compressive strength of the printed concrete, measured in megapascals (MPa). This feature is used as the dependent variable in the machine learning model, where predictions are made based on the input features.

- **Compressive Strength f_c (MPa)**: The ultimate compressive strength of the concrete after curing.

3.1.3 Statistical Description of Data

Table 3.1 provides detailed statistical insights into the input and output parameters of the dataset, including several key statistical metrics. The **Parameter** column lists each material component or characteristic measured, such as cement, silica fume (SF), and compressive strength (fc). The **Unit** column provides the units for each parameter, indicating whether measurements are in kilograms per cubic meter (kg/m^3), percentage (%), days, or megapascals

(MPa). The **Mean** column presents the average value of each parameter across all data samples, giving a sense of the central tendency of each variable. Next, the **Standard Deviation (SD)** column shows the degree of variability or spread in the data, indicating how much individual values differ from the mean. The **Minimum (Min)** and **Maximum (Max)** columns display the smallest and largest recorded values for each parameter, respectively, highlighting the range of values within the dataset. **Kurtosis** measures the "tailedness" of each variable's distribution, where higher values indicate more pronounced outliers or extreme values. Finally, the **Skewness** column describes the asymmetry of each variable's data distribution, with positive or negative values indicating a longer tail on the right or left side of the distribution, respectively. Together, these columns provide a comprehensive view of the statistical characteristics of each parameter, aiding in the interpretation of data trends and relationships.

The kurtosis values in the statistical description of the variables provide important insights into the distribution of each parameter. A positive kurtosis value indicates a distribution with heavy tails and a sharper peak compared to a normal distribution, while a negative kurtosis value suggests lighter tails and a flatter peak. The BFS parameter has a kurtosis value of 8.36, which signifies a leptokurtic distribution. This suggests that the data for BFS is concentrated near the mean, with occasional extreme values in the tails. Similarly, the LSP parameter has a kurtosis of 17.69, indicating a very sharp peak with significant outliers, reflecting an extreme leptokurtic distribution. In contrast, parameters such as Cement, SF, and Sand have negative kurtosis values of -0.23, -0.62, and -0.28, respectively. These values indicate platykurtic distributions, which are characterized by data that is more spread out and with fewer extreme outliers compared to a normal distribution. Other parameters, such as Water (1.16), SP (-1.08), and T (135.25), also show varying degrees of kurtosis. The value for T is particularly high, suggesting that the distribution of temperature is highly leptokurtic, with possible extreme fluctuations or skewed data.

The skewness values provide further insights into the asymmetry of the distributions of the variables. A positive skewness indicates that the tail on the right side of the distribution is longer or fatter than the left side, while a negative skewness suggests the opposite, with a longer or fatter left tail. The skewness of the `T` (temperature) parameter is notably high at 11.25, indicating a highly positively skewed distribution. This suggests that the temperature data is concentrated on the lower end, with a few extreme high values driving the skewness. Similarly, the `RH` (relative humidity) parameter exhibits a skewness of -4.37, indicating a significant negative skew. This suggests that the relative humidity values are heavily concentrated on the higher end, with fewer extreme low values. Other parameters, such as `BFS` (skewness of 3.04) and `LSP` (skewness of 3.73), also show positive skewness, implying that these variables have a few high values that are influencing the overall distribution. In contrast, parameters such as `Water` and `SP` exhibit relatively small skewness values (0.54 and -0.18, respectively), suggesting that these variables are more symmetrically distributed, with minimal skew. Parameters like `Fiber` (-1.06) and `Age` (3.79) exhibit a range of skewness, indicating that the data for these parameters may have a moderate degree of asymmetry. The negative skewness of `Fiber` suggests a concentration of values on the higher end of the distribution, while the positive skewness of `Age` indicates that the data is concentrated on the lower end, with a few older samples causing the positive skew.

Table 3.1: The statistical description of variables in the dataset

Parameter	Unit	Mean	Standard Deviation	Min	Max	Kurtosis	Skewness
Cement	kg/m^3	740.44	163.85	270.0	1251.2	-0.23	-0.21
SF	kg/m^3	136.92	105.18	0.0	433.7	-0.62	0.26
BFS	kg/m^3	25.08	74.61	0.0	375.0	8.36	3.04
FA	kg/m^3	26.42	67.22	0.0	356.0	5.12	2.45
QP	kg/m^3	31.60	75.58	0.0	397.0	3.76	2.22
LSP	kg/m^3	33.57	99.35	0.0	772.2	17.69	3.73
NS	kg/m^3	3.52	7.35	0.0	38.0	4.74	2.30
Fiber	kg/m^3	55.03	73.69	0.0	234.0	-1.06	0.80
Sand	kg/m^3	1002.49	280.13	0.0	1502.8	-0.28	-0.36
Gravel	kg/m^3	151.67	354.39	0.0	1195.0	2.14	2.00
Water	kg/m^3	179.13	23.69	90.0	234.0	1.16	0.54
SP	kg/m^3	30.04	13.82	1.1	57.0	-1.08	-0.18
T	°C	23.24	13.89	20.0	200.0	135.25	11.25
RH	%	97.96	7.92	50.0	100.0	18.01	-4.37
Age	days	35.75	49.38	1.0	365.0	19.59	3.79
fc	MPa	122.59	40.17	32.5	220.5	-0.54	0.04

3.1.4 Analysis of Frequency Visualization

The frequency distribution plot in Figure 3.1 provides an overview of the occurrence of different values across the variables. The histograms of various mix constituents in concrete mixtures provide valuable insights into their distribution patterns, dominant values, and potential outliers.
Cement: The histogram indicates a unimodal distribution with a dominant peak around 800 kg/m^3. The distribution exhibits a mild negative skew, suggesting that while most data clusters around 800 kg/m^3, some mixes contain significantly higher cement content. A potential outlier

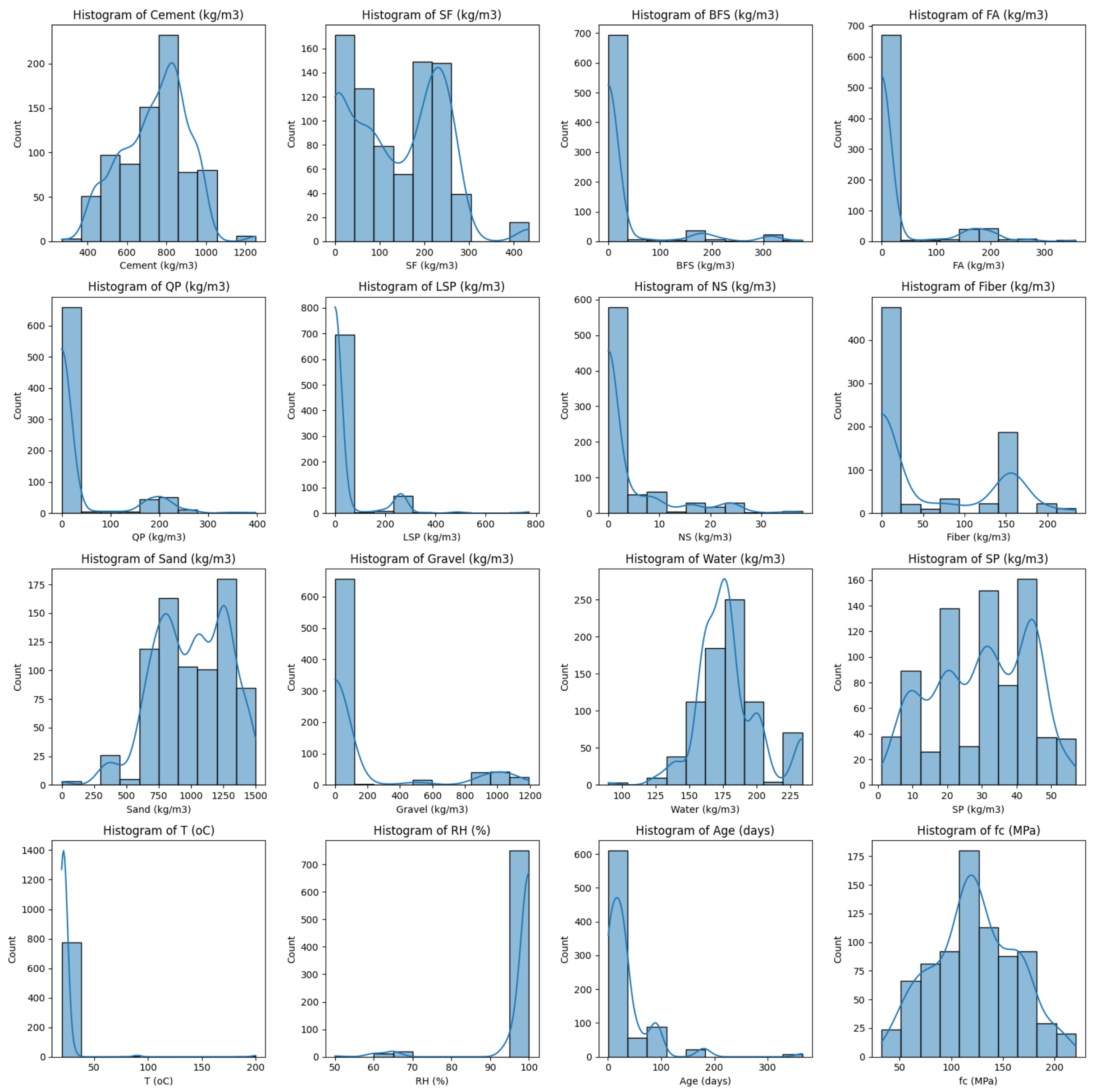


Figure 3.1: Frequency Distribution of Variables

is observed at 1200 kg/m^3. The mode, representing the most frequent value, is clearly at 800 kg/m^3.

Silica Fume (SF): The histogram reveals a bimodal distribution with peaks at 0 kg/m^3 and 200 kg/m^3, suggesting two distinct groups. The dominant peak at 0 kg/m^3 indicates that a large proportion of samples contain little to no SF. A secondary peak at 200 kg/m^3 signifies a substantial subset utilizing SF at this concentration. A noticeable dip between 50-150 kg/m^3 suggests infrequent usage in this range. The distribution is right-skewed, with a potential outlier around 400 kg/m^3.

Blast Furnace Slag (BFS): The histogram displays extreme right skewness, with most data points concentrated at 0 kg/m^3, indicating minimal BFS usage. Small peaks at approximately

150 kg/m^3 and 300 kg/m^3 suggest occasional higher usage. The BFS content ranges from 0 to 350 kg/m^3.

Fly Ash (FA): Similar to BFS, FA exhibits extreme right skewness, with the mode at 0 kg/m^3. A minor peak is observed around 180-200 kg/m^3, indicating occasional higher usage. The FA content ranges from 0 to 350 kg/m^3, and its application appears to be limited to specific mix designs.

Quarry Powder (QP): The histogram indicates a highly skewed right distribution, with a dominant peak at 0 kg/m^3, showing minimal usage. A small peak around 200-250 kg/m^3 suggests some mixes incorporate QP in these quantities. The range extends from 0 to approximately 380 kg/m^3.

Limestone Powder (LSP): The distribution follows a similar pattern to BFS and FA, with extreme right skewness. The dominant mode is at 0 kg/m^3, and a minor peak occurs around 250-300 kg/m^3. The LSP content extends up to 800 kg/m^3.

Nano-Silica (NS): The histogram indicates that NS is sparsely used, with most samples containing little to no NS. Small peaks at 10 kg/m^3 and 20-25 kg/m^3 suggest limited usage in some mixes. The NS content ranges from 0 to 35 kg/m^3.

Fiber: The histogram reveals a bimodal distribution with peaks at 0 kg/m^3 and 150 kg/m^3, indicating two distinct mix groups. The majority of samples contain no fiber, while a significant subset includes fiber at 150 kg/m^3. The fiber content extends up to approximately 220 kg/m^3.

Sand: The distribution is complex, with multiple peaks suggesting potential multimodality. Noticeable increases in frequency occur around 750 kg/m^3, 1000 kg/m^3, and 1250 kg/m^3. A mild left skew is observed, and sand content ranges from 0 to approximately 1500 kg/m^3.

Gravel: The histogram is highly skewed to the right, with the majority of data concentrated at lower values and a long tail extending towards higher values. The mode is at 0 kg/m^3, indicating that a large proportion of samples had minimal gravel content. The frequency of gravel usage declines rapidly as the amount increases, suggesting limited use in large quantities. Small peaks at approximately 600 kg/m^3 and 1000 kg/m^3 indicate that gravel is occasionally used at these levels. The range spans from 0 kg/m^3 to approximately 1200 kg/m^3.

Water: The histogram exhibits a unimodal distribution, with a peak around 175-180 kg/m^3. This suggests that this water content is the most commonly used. The distribution is slightly skewed to the right, with lower frequencies observed below 150 kg/m^3 and a smaller peak around 225 kg/m^3. The range extends from approximately 100 kg/m^3 to 225 kg/m^3, with the highest count concentrated in the 175-180 kg/m^3 range.

Superplasticizer (SP): The histogram displays a multimodal distribution with increasing frequency trends at 10 kg/m^3, 20 kg/m^3, 30 kg/m^3, and 45 kg/m^3, followed by a sharp decline. This trend suggests a preference for certain dosage levels. The range spans from 0 kg/m^3 to 50 kg/m^3, with significant counts at 10, 20, 30, and 45 kg/m^3.

Temperature (T): The histogram is highly skewed to the right, with a dominant peak around 20-30°C. The frequency of higher temperatures drops significantly, indicating that most samples were subjected to relatively low temperatures. A small peak around 100°C suggests specific applications involving elevated temperatures. The temperature range extends from approximately 0°C to 200°C.

Relative Humidity (RH): The histogram reveals a unimodal distribution, with a dominant peak near 100% RH. The frequency of RH values between 70% and 90% is notably low. The range spans from approximately 50% to 100%, with a high concentration at the upper limit.

Age: The histogram is highly skewed to the right, with the majority of data concentrated at lower ages. The mode appears at or below 28 days, indicating that most samples were tested at early ages. Small peaks at 90 days and 180 days suggest specific tests at later ages. The range extends from 0 to approximately 350 days.

Compressive Strength (fc): The histogram exhibits a unimodal distribution, with a dominant peak around 125 MPa. The distribution is slightly right-skewed, with a tail extending toward higher strength values. The range spans from approximately 25 MPa to 225 MPa, with the highest counts concentrated around 125 MPa.

3.1.5 Correlation Analysis

Correlation is a statistical measure that describes the degree to which two variables move in relation to each other. A correlation coefficient ranges from -1 to +1, where values closer to +1 indicate a strong positive correlation, values closer to -1 indicate a strong negative correlation, and values around 0 suggest no significant correlation. In the context of this study, a positive correlation implies that as one feature increases, the other tends to increase as well, while a negative correlation implies the opposite.

The correlation heatmap in Figure 3.2 presents the pairwise correlations among the features in the dataset, with the target variable f_c (compressive strength) highlighted as a key focus of this analysis. The color gradient in the heatmap ranges from blue (indicating negative correlations) to red (indicating positive correlations), with more intense colors corresponding to stronger correlations.

Cement content (kg/m^3) shows a moderate positive correlation with compressive strength (f_c), suggesting that an increase in cement generally contributes to higher compressive strength.

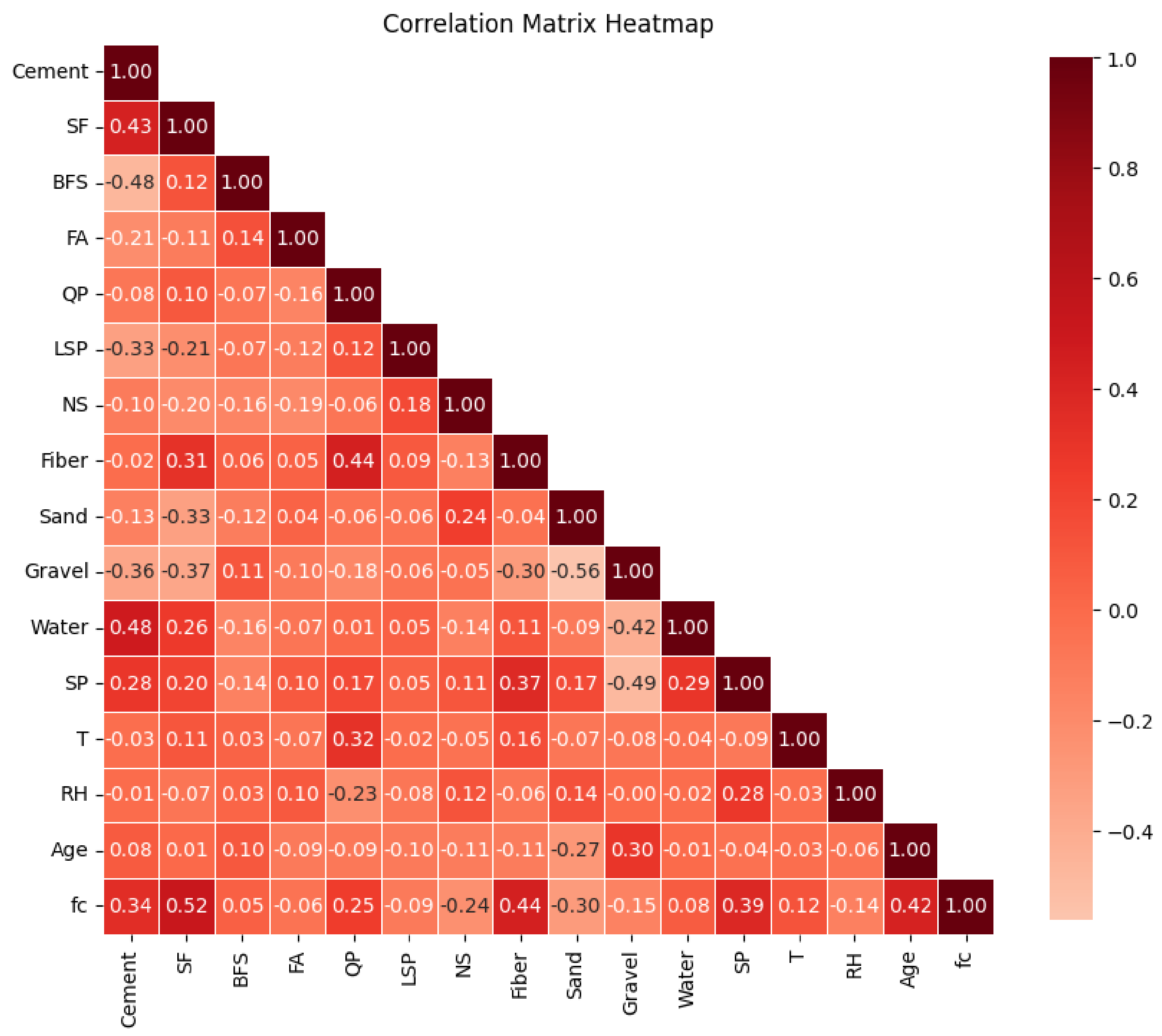


Figure 3.2: Correlation Matrix Heatmap of Variables

Silica fume (SF) exhibits an even stronger positive correlation with f_c, reinforcing its significant role in enhancing compressive strength, likely due to its pozzolanic properties, which improve concrete's microstructure. Fiber content also shows a notable positive correlation with f_c, indicating that the inclusion of fibers in concrete can contribute positively to its compressive strength, potentially by enhancing tensile and flexural properties. Superplasticizer (SP) demonstrates a moderate positive correlation with f_c, suggesting that its presence, by improving workability and reducing water demand, also enhances the compressive strength.

Water content (kg/m^3), interestingly, shows only a weak positive correlation with compressive strength. This minimal correlation may reflect the complex balance between adequate hydration and the detrimental effects of excess water on concrete's structural integrity. Age ($days$) of the concrete exhibits a positive correlation with compressive strength, which is expected, as concrete gains strength over time due to the ongoing hydration process.

Conversely, some features display negative correlations with compressive strength. Notably, sand and gravel content both exhibit negative correlations, suggesting that higher proportions of these aggregates may not positively contribute to compressive strength in this dataset. This may be because excessive aggregate can reduce the cement paste matrix necessary for strength development.

The correlation analysis reveals that cement, silica fume, fiber, superplasticizer, and age are positively associated with compressive strength, whereas sand and gravel content tend to negatively impact it. This analysis offers valuable insights into how various components influence concrete's compressive strength, providing a foundation for selecting and optimizing ingredient ratios in concrete formulations.

3.2 Regression Training, Evaluation, and Optimization Methods

3.2.1 Performance Assessment Metrics

The performance of the models is evaluated through a series of statistical metrics, which provide an objective measure of the models' ability to predict outcomes accurately. Key metrics employed in this evaluation include Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the Coefficient of Determination (R^2).

Root Mean Squared Error (RMSE)

RMSE is a commonly used metric for quantifying the difference between the values predicted by a model and the actual observed values. It is calculated as the square root of the average of the squared differences between predicted and actual values.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.1}$$

where y_i represents the actual values, $\hat{y}_i$ the predicted values, and n the number of observations. A lower RMSE value indicates a better fit of the model to the data.

Mean Absolute Error (MAE)

MAE measures the average magnitude of errors in a set of predictions, without considering their direction. It is the average of the absolute differences between predicted and actual values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.2}$$

Like RMSE, lower MAE values indicate more accurate predictions. MAE provides a straightforward interpretation of the average error in units of the variable of interest.

Coefficient of Determination (R^2)

The R^2 metric provides a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variation explained by the model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.3}$$

where $\bar{y}$ is the mean of the actual values. An R^2 value approaching 1.00 indicates that the model explains most of the variability in the response variable, signifying a high degree of accuracy.

3.2.2 CatBoost Algorithm

The CatBoost algorithm, introduced by Yandex, Russia's largest search engine company, in 2017, and made open-source in April of that year, represents a significant advancement in the field of open-source algorithms. It outperforms both the XGBoost and LightGBM algorithms in terms of performance. The name "CatBoost" is derived from "Category" and "Boosting," is included in the Boosting algorithm family. CatBoost enhances the Gradient Boosted Decision Trees (GBDT) framework by addressing issues such as gradient bias and prediction shift, thereby reducing the risk of overfitting and enhancing both computational accuracy and generalization capabilities (Figure 1). CatBoost is designed specifically for handling datasets containing categorical features. It is known for its efficiency and high accuracy across various types of data, including categorical variables.

Principles of CatBoost

CatBoost operates on the following principles:

1. **Ordered Boosting:** CatBoost builds models iteratively, adding new models to correct the errors of existing ones. This stage-wise process minimizes a loss function L by optimizing the predictions.

2. **Ordered Target Statistic:** Categorical features consist of distinct values, known as categories, which do not have an inherent order or comparability. Consequently, these features are not directly applicable in binary decision tree algorithms. For low cardinality categorical variables, CatBoost uses one-hot encoding. A unique feature of CatBoost is its ability to process high-cardinality categorical variables without extensive preprocessing. A typical approach to handling categorical data involves transforming these categories into numerical representations during preprocessing, whereby each category is replaced with one or more numerical values.

CatBoost employs a sophisticated method to mitigate overfitting and enable the utilization of the entire dataset for training. Consider a dataset $S = \{(X_i, Y_i)\}_{i=1}^{n}$, where each $X_i = (x_{i,1}, \ldots, x_{i,m})$ represents a vector with m dimensions that include both numerical and categorical types, and $Y_i \in \mathbb{R}$ denotes the corresponding label value. For handling categorical features, CatBoost applies a random permutation to the dataset, and for each instance, it calculates the average label value of preceding instances with the same category in the permutation. If the permutation is denoted as $\sigma = (\sigma_1, \ldots, \sigma_n)$, then the categorical feature value $x_{\sigma_p,k}$ is transformed using the formula

$$\hat{x}_{ik} = \frac{\sum_{j=1}^{p-1} \left[x_{\sigma_j,k} = x_{\sigma_p,k}\right] \cdot Y_{\sigma_j} + a \cdot p}{\sum_{j=1}^{p-1} \left[x_{\sigma_j,k} = x_{\sigma_p,k}\right] + a} \tag{3.4}$$

where P is the added prior term, and a is a weighting factor typically greater than 0. For regression problems, the prior term is often the average value of the dataset.

Mathematical Formulation

The goal of CatBoost is to minimize a loss function $L(y, \hat{y})$, where y represents the true label and $\hat{y}$ represents the predicted label. The model prediction can be written as:

$$\hat{y}_i = \sum_{m=1}^{M} f_m(x_i) \tag{3.5}$$

where $\hat{y}_i$ is the predicted value for the i-th instance, f_m is the m-th decision tree, and M is the total number of trees. The update step is defined as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i) \tag{3.6}$$

where η is the learning rate and t is the iteration number. The loss function L can be expressed as:

$$L = \sum_{i=1}^{N} L(y_i, \hat{y}_i) \tag{3.7}$$

where N is the number of instances.

Steps in CatBoost Algorithm

The CatBoost algorithm follows these steps:

1. **Initialization:** Start with initial predictions, typically the mean of the target variable:

$$\hat{y}_i^{(0)} = \frac{1}{N} \sum_{j=1}^{N} y_j \tag{3.8}$$

where $\hat{y}_i^{(0)}$ is the initial prediction for the i-th instance and N is the total number of instances.

2. **Iterative Training:** For each iteration t: - Compute the pseudo-residuals:

$$r_i^{(t)} = -\frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}} \tag{3.9}$$

where $r_i^{(t)}$ is the pseudo-residual for the i-th instance.

- Fit a new decision tree $f_t(x)$ to the pseudo-residuals.

- Update the predictions:

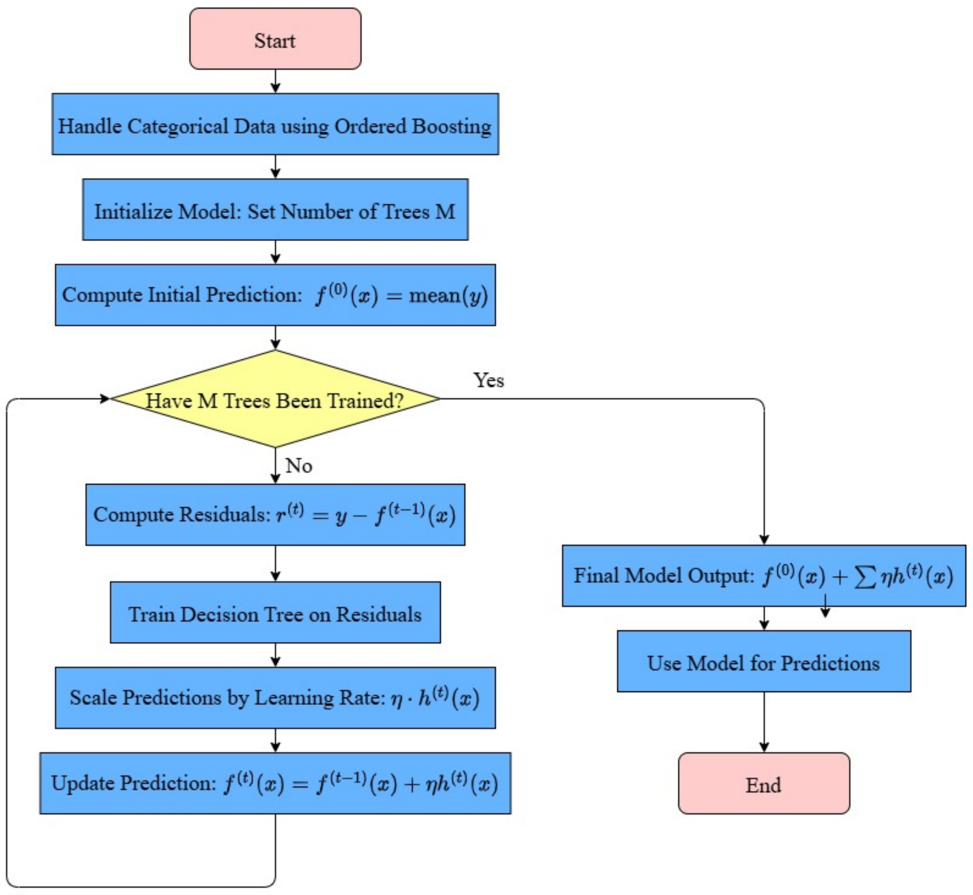
$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i) \tag{3.10}$$

3. **Final Prediction:** After M iterations, the final prediction is given by:

$$\hat{y}_i = \hat{y}_i^{(M)} \tag{3.11}$$

where $\hat{y}_i$ is the final prediction for the i-th instance.

Figure 3.3 illustrates the flowchart of the CatBoost regression algorithm. It highlights the data processing stages, model training, and final model output used to optimize the predictive performance of the algorithm.



Start
Handle Categorical Data using Ordered Boosting
Initialize Model: Set Number of Trees M
Compute Initial Prediction: $f^{(0)}(x) = \text{mean}(y)$
Have M Trees Been Trained?
Yes
No
Compute Residuals: $r^{(t)} = y - f^{(t-1)}(x)$
Train Decision Tree on Residuals
Scale Predictions by Learning Rate: $\eta \cdot h^{(t)}(x)$
Update Prediction: $f^{(t)}(x) = f^{(t-1)}(x) + \eta h^{(t)}(x)$
Final Model Output: $f^{(0)}(x) + \sum \eta h^{(t)}(x)$
Use Model for Predictions
End

Figure 3.3: Flowchart of the CatBoost Regression Algorithm

Advantages of CatBoost

The main advantages of CatBoost include:

1. **Robustness to Overfitting:** CatBoost includes built-in regularization techniques to prevent overfitting.

2. **Efficient Handling of Categorical Features:** CatBoost's native support for categorical variables reduces the need for extensive preprocessing.

3. **High Performance:** CatBoost often achieves state-of-the-art performance on benchmark datasets due to its advanced optimization techniques.

3.2.3 Custom Firefly Optimization Algorithm

The Firefly Optimization Algorithm (FOA) is a nature-inspired optimization technique based on the social behavior of fireflies. It effectively solves complex optimization problems by mimicking the attraction of fireflies toward one another based on their brightness. The brightness of each firefly is related to the quality of the solution it represents, guiding less fit fireflies toward more promising regions in the search space.

Principles of FOA

FOA operates based on the following principles:

1. **Attractiveness** Fireflies are attracted to one another based on their brightness or intensity. In the context of optimization, this brightness is associated with the fitness of the solution, guiding less fit fireflies toward more fit ones.

2. **Movement** Fireflies move toward brighter fireflies, and their movement is influenced by their attractiveness and the distance to other fireflies. This movement can be described mathematically to explore the solution space effectively.

FOA Intensity and Movement Calculation

The intensity of each firefly is calculated using the Mean Squared Error (MSE), which evaluates the performance of a given solution. The intensity I for each firefly j in the population is defined as

$$I_j = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \text{for } j = 1, 2, \ldots, N. \tag{3.12}$$

where n is the number of validation samples, y_i is the true value, and $\hat{y}_i$ is the predicted value for the i-th sample obtained from a CatBoost model fitted with the hyperparameters from the corresponding firefly solution.

The position of each firefly can be represented as X_i, where i denotes the i-th firefly. The movement of a firefly toward another firefly can be expressed as

$$X_i = X_i + \beta_0 e^{-\gamma r_{ij}}(X_j - X_i) + \alpha \tag{3.13}$$

where X_j is the position of the j-th firefly, r_{ij} is the distance between the i-th and j-th fireflies, β_0 is the attractiveness at $r = 0$, γ is the light absorption coefficient, and (α) is a randomization factor.

Due to the absence of a widely accepted standard implementation of the Firefly Optimization Algorithm (FOA), the distance calculation in this study was implemented using a custom approach. The distance between two fireflies is computed by first applying *standard scaling* to each dimension and then calculating the Euclidean distance. Standard scaling is applied to ensure that each dimension contributes equally to the distance calculation, preventing any single feature from disproportionately influencing the result due to differences in scale. For fireflies $\mathbf{X}_i$ and $\mathbf{X}_j$, the scaled values are obtained as

$$X_i' = \frac{X_i - \mu}{\sigma}, \quad X_j' = \frac{X_j - \mu}{\sigma} \tag{3.14}$$

where X_i' and X_j' denote the standard scaled values of fireflies $\mathbf{X}_i$ and $\mathbf{X}_j$, while μ and

σ are the mean and standard deviation of each dimension across the population. After scaling, the distance between the fireflies are calculated.

Five distance metrics—Euclidean Distance, Manhattan Distance (L1 norm), Minkowski Distance (generalized form), Cosine Distance, and Chebyshev Distance—were evaluated. A preliminary run of the Firefly Optimization Algorithm over 30 iterations (with all runs initialized using identical populations) indicated that the Euclidean Distance metric yielded the best performance. The Euclidean distance is computed as

$$d_{ij} = \sqrt{\sum_{k=1}^{d} \left(X_i'^{(k)} - X_j'^{(k)} \right)^2} \tag{3.15}$$

where d_{ij} is the Euclidean distance, d is the number of dimensions, and $X_i'^{(k)}$ and $X_j'^{(k)}$ are the scaled values in the k-th dimension for fireflies $\mathbf{X}_i$ and $\mathbf{X}_j$.

Steps in FOA

The Firefly Optimization Algorithm follows these steps:

1. **Initialization** Initialize the positions of fireflies randomly within the search space and calculate their brightness based on the objective function.

2. **Iterative Improvement** For each firefly, update its position by moving toward less bright fireflies. In FOA, typical movement is towards brighter fireflies but in this project, movement is towards dimmer fireflies. This movement ensures that fireflies move towards minimum MSE. This process involves calculating distances, updating positions, and determining brightness in each iteration.

3. **Termination** The algorithm terminates when a stopping criterion is met, such as a maximum number of iterations or a satisfactory solution.

Figure 3.4 illustrates the steps in firefly algorithm.

Suitability of Firefly Optimization Algorithm for Complex Optimization

FOA has been successfully applied in various fields, including engineering design, feature selection, and scheduling problems. Its ability to balance exploration and exploitation makes it suitable for complex optimization tasks.

FOA is a powerful and versatile optimization technique inspired by natural phenomena. Its ability to effectively explore the solution space while converging toward optimal solutions makes it a valuable tool for tackling a wide range of optimization problems.

3.2.4 Regression Slope Analysis

Regression slope analysis is a fundamental statistical technique used to evaluate the relationship between an independent variable and a dependent variable in a linear regression model. This

Algorithm 1 Firefly Algorithm

```
1: Objective function $f(\mathbf{x})$, $\mathbf{x} = (x_1, \dots, x_d)^T$.
2: Generate an initial population of $n$ fireflies $\mathbf{x}_i$ $(i = 1, 2, \dots, n)$.
3: Light intensity $I_i$ at $\mathbf{x}_i$ is determined by $f(\mathbf{x}_i)$.
4: Define light absorption coefficient $\gamma$.
5: while ($t <$ MaxGeneration) do
6:   for $i = 1$ to $n$ do
7:     for $j = 1$ to $n$ do
8:       if $(I_i > I_j)$ then
9:         Vary attractiveness with distance $r$ via $\exp(-\gamma r^2)$.
10:        Move firefly $i$ towards $j$.
11:      end if
12:    end for
13:  end for
14:  Evaluate new solutions and update light intensity.
15: end while
16: Postprocess results and visualization.
```

Figure 3.4: The Firefly Optimization Algorithm

analysis provides insights into the strength and direction of the association, which is crucial for understanding the underlying patterns in the data.

The slope of a regression line, often denoted as β_1, quantifies the change in the dependent variable Y for a one-unit change in the independent variable X. Mathematically, the linear regression model is represented as

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{3.16}$$

where Y is the dependent variable, X is the independent variable, β_0 is the intercept, representing the value of Y when $X = 0$, β_1 is the slope of the regression line, and ϵ is the error term, capturing the deviations of the observed values from the predicted values.

To estimate the slope β_1, the ordinary least squares (OLS) method is employed. The OLS method minimizes the sum of squared residuals, providing the best-fitting line through the data points. The slope β_1 is calculated as

$$\beta_1 = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2} \tag{3.17}$$

where X_i and Y_i are the individual data points, and $\bar{X}$ and $\bar{Y}$ are the means of X and Y, respectively.

Importance of Regression Slope Analysis

The slope in regression slope analysis indicates the extent to which changes in the independent variable can predict changes in the dependent variable. A steeper slope signifies a stronger predictive relationship.

3.2.5 K-Fold Cross-Validation

K-fold cross-validation is a robust method for assessing the performance of machine learning models, particularly in regression tasks. It involves partitioning the dataset into k subsets or folds of equal (or nearly equal) size. In each iteration, one fold is retained as the validation set for testing the model, while the remaining $k-1$ folds are used for training. This process is repeated k times, with each fold serving as the validation set exactly once.

Advantages in Regression

K-fold cross-validation is especially beneficial in regression tasks for several reasons:

- **Reduced Bias**: By ensuring that every data point has an opportunity to be in the training and validation sets, k-fold cross-validation mitigates bias in the model evaluation.
- **Robustness to Overfitting**: Overfitting occurs when the model memorizes the dataset instead of learning the general patterns. K-fold cross-validation helps in detecting overfitting by providing multiple performance metrics across different data splits.
- **Efficient Use of Data**: Particularly in scenarios with limited data, k-fold cross-validation maximizes the use of the available dataset, ensuring that the model is trained on a variety of data points.
- **Generalization Ability**: Averaging the results across the folds provides a more reliable estimate of the model's performance on unseen data, thus improving its generalization ability.

3.2.6 SHAP Analysis for Model Interpretability

In this study, SHAP analysis was employed to interpret the feature contributions within the optimized CatBoost model. SHAP is a model-agnostic tool that provides insights into how each input feature impacts individual predictions by assigning a SHAP value to each feature for each data instance. These SHAP values quantify the contribution of each feature, indicating whether a feature has a positive or negative effect on the model's prediction of compressive strength for a given input.

The SHAP methodology is grounded in cooperative game theory, where SHAP values are calculated based on the contribution each feature makes when acting in "coalition" with other features. This approach allows for a consistent measure of feature impact, making SHAP a preferred method for understanding complex machine learning models like CatBoost.

Using SHAP analysis on the FOA-CatBoost, the study aimed to achieve two key objectives:

1. **Feature Importance Identification**: Determine the features that have the most significant influence on predicting compressive strength. This information is valuable for identifying the mix design parameters and environmental conditions that contribute most to the model's predictions.

2. **Impact Direction and Magnitude**: Analyze the direction (positive or negative) and magnitude of each feature's impact on compressive strength predictions. High SHAP values indicate features with substantial influence, while the direction reveals whether a feature increases or decreases the predicted strength.

The SHAP analysis not only improves model interpretability by highlighting key predictors but also aids in understanding the underlying relationships between material properties and compressive strength. This level of interpretability supports informed decision-making in mix design formulation, aligning with the study's goal of optimizing 3D concrete mix designs.

3.3 Generative Training, Evaluation, and Optimization Methods

3.3.1 Activation Functions

Activation functions play a crucial role in neural networks by introducing non-linearity. The Rectified Linear Unit (ReLU) and Sigmoid functions are commonly used in deep learning architectures, including Conditional Variational Autoencoders (CVAE).

ReLU: The Rectified Linear Unit (ReLU) is defined as:

$$f(x) = \max(0, x) \tag{3.18}$$

ReLU is widely used due to its ability to mitigate the vanishing gradient problem and improve convergence speed. It is applied in the hidden layers of the encoder and decoder to introduce non-linearity while maintaining computational efficiency.

Sigmoid: The Sigmoid activation function is expressed as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.19}$$

The Sigmoid function maps inputs to a range between 0 and 1, making it useful for probabilistic outputs. In the CVAE, it is used in the final output layer of the decoder to generate normalized values.

3.3.2 Loss Function: Kullback-Leibler (KL) Divergence

The Kullback-Leibler (KL) divergence measures the difference between two probability distributions. In a CVAE, it is used to ensure that the learned latent distribution remains close to a standard normal distribution. The KL divergence between two distributions $q(z|x)$ and $p(z)$ is defined as:

$$D_{\text{KL}}(q(z|x)||p(z)) = \sum q(z|x) \log \frac{q(z|x)}{p(z)} \tag{3.20}$$

The total loss function of the CVAE consists of the reconstruction loss and the KL divergence loss. During training, the KL divergence acts as a regularization term in the loss function, preventing the latent space from collapsing and ensuring meaningful latent representations.

3.3.3 Variational Autoencoder (VAE)

A **Variational Autoencoder (VAE)** [30] is a type of generative model that learns to encode and generate data by mapping inputs into a structured *latent space*. Variational Autoencoders (VAEs) have a wide range of applications across different fields due to their ability to learn meaningful latent representations. Some key applications include image generation and reconstruction, anomaly detection, Natural Language Processing (NLP), and music and speech generation. Unlike traditional autoencoders, a VAE models a **probabilistic** distribution over the latent variables.

Consider a dataset $X = \{x_i\}_{i=1}^{N}$, where each x_i represents an independent and identically distributed (i.i.d.) sample of a continuous or discrete random variable x. The data generation process involves an unobserved continuous latent variable z and occurs in two stages: (1) A latent variable z_i is sampled from a prior distribution $p_\theta(z)$. (2) A corresponding observation x_i is drawn from the conditional distribution $p_\theta(x|z)$. The prior $p_\theta(z)$ and the likelihood $p_\theta(x|z)$ are assumed to belong to parametric families of distributions, denoted as $p_\theta(z)$ and $p_\theta(x|z)$, respectively. Additionally, their probability density functions (PDFs) are assumed to be differentiable with respect to both the parameters θ and the latent variable z, except on a set of measure zero.

Variational Inference

The variational inference method of the VAE is specifically used in the computation of $p(z|x_i)$. By applying Bayes' theorem and the probability chain rule, the posterior probability can be expressed as

$$p(z|x_i) = \frac{p(z, x_i)}{p(x_i)} = \frac{p(x_i|z)p(z)}{\int p(x_i|z)p(z)\, dz} \tag{3.21}$$

In principle, the posterior $p(z|x_i)$ could be computed by evaluating the denominator integral, which requires summing over all possible values of the latent variable z. However, in the absence of simplifying assumptions on $p(z|x_i)$ or $p(z)$, this integral is generally intractable. This intractability arises because any approach that directly computes the integral, including enumeration methods, incurs exponential computational complexity.

The variational inference method addresses this issue by re-framing the posterior inference problem as an optimization task. Instead of computing the exact posterior, a recognition model $q_\phi(z|x_i)$ is introduced as an approximation to the true posterior $p_\theta(z|x_i)$. By minimizing the Kullback-Leibler (KL) divergence between $q_\phi(z|x_i)$ and $p_\theta(z|x_i)$, the posterior inference problem can be effectively solved. To enhance computational efficiency, both the parameters ϕ of the recognition model and θ of the generative model are optimized jointly.

Variational Autoencoder: Key Mechanisms

- **Encoder**: The encoder maps an input x to a latent distribution $q_\phi(z|x)$ instead of a single point. It outputs the mean μ and log-variance $\log \sigma^2$, which define a Gaussian distribution over latent variables z.

- **Sampling using Reparameterization Trick:** In practice, the samples z_i are not directly drawn from $q_\phi(z|x_i)$, as q_ϕ can represent a highly complex distribution that is difficult to sample from. Instead of directly sampling $z \sim q_\phi(z|x)$, we use the reparameterization trick by setting

$$z = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{3.22}$$

  This allows gradients to flow through the sampling operation during training.

- **Decoder:** The decoder takes the sampled latent variable z and reconstructs the input x, modeling $p_\theta(x|z)$.

- **Loss Function:** The objective function consists of:

  – **Reconstruction Loss**: Ensures the generated output $\hat{x}_i$ is similar to the input x. The reconstruction loss used in this paper is the Mean Squared Error (MSE)

$$\text{Loss}_1 = \frac{1}{N}\sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2 \tag{3.23}$$

  – **KL Divergence**: Regularizes the latent space by encouraging $q_\phi(z|x)$ to be close to a prior distribution $p(z)$. In this paper, we assume $p(z)$ as $\mathcal{N}(0, I)$.

$$\text{Loss}_2 = D_{KL}\left(q_\phi(z|x)\,||\,p_\theta(z)\right) = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right) \tag{3.24}$$

where J is the number of latent dimensions.

The total loss is given by

$$\text{Loss} = \text{Loss}_1 + \beta \cdot \text{Loss}_2 \tag{3.25}$$

where β is the weighting factor that controls the trade-off between the reconstruction loss and the KL divergence term.

3.3.4 Conditional Variational Autoencoder (CVAE)

A **Conditional Variational Autoencoder (CVAE)** [31] extends the standard VAE by incorporating conditional information into both the encoder and decoder networks. This allows the model to generate data samples conditioned on additional context, making it particularly useful in applications such as image captioning, text-to-image synthesis, structured prediction tasks, and conditional generative tasks.

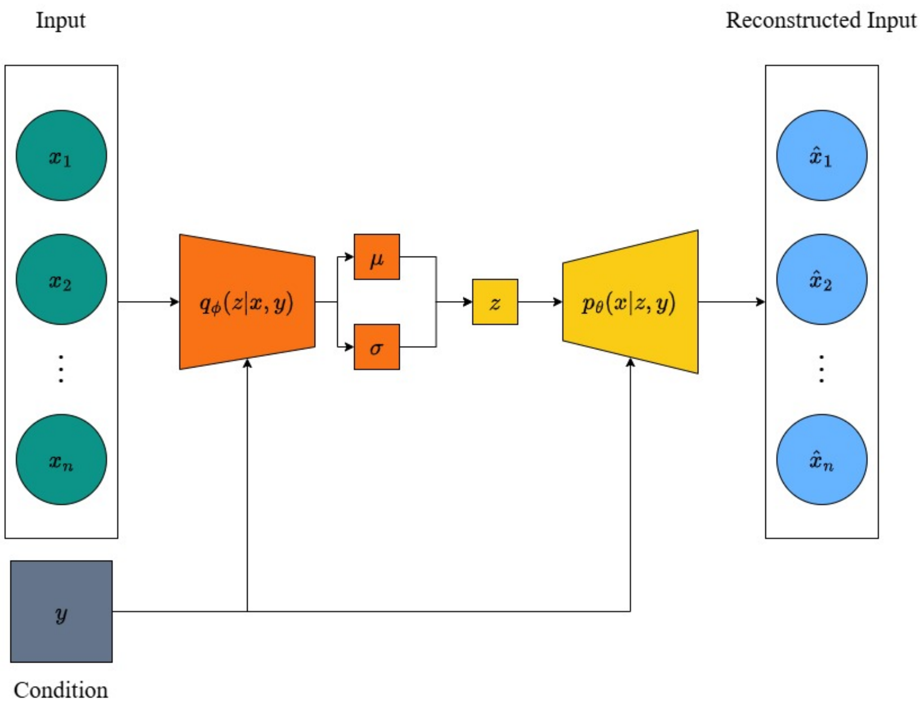


Figure 3.5: Architecture of the Conditional Variational Autoencoder (CVAE)

The inference process in CVAE requires computing the posterior distribution $p(z|x_i, y_i)$, where y_i is the conditioning variable. Using Bayes' theorem, the posterior probability is expressed as

$$p(z|x_i, y_i) = \frac{p(z, x_i|y_i)}{p(x_i|y_i)} = \frac{p(x_i|z, y_i)p(z|y_i)}{\int p(x_i|z, y_i)p(z|y_i)\, dz}. \tag{3.26}$$

Similar to the VAE, this integral is intractable due to the high-dimensional latent space. To address this, an approximate posterior distribution $q_\phi(z|x_i, y_i)$ is introduced, optimizing the variational lower bound by minimizing the Kullback-Leibler (KL) divergence between $q_\phi(z|x_i, y_i)$ and $p_\theta(z|x_i, y_i)$.

Fig 3.5 shows the architecture of CVAE. In CVAE, the encoder maps an input x and conditioning variable y to a latent distribution $q_\phi(z|x, y)$. It outputs the mean μ and log-variance $\log \sigma^2$, defining a Gaussian distribution over latent variables z. Sampling is also performed using reparameterization trick. The decoder reconstructs the input x conditioned on both z and y, modeling $p_\theta(x|z, y)$.

The reconstruction loss for CVAE is

$$\text{Loss}_{\text{recon}} = \frac{1}{N} \sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2, \tag{3.27}$$

where x_i is the input sample and $\hat{x}_i$ is the reconstructed output.

The KL divergence loss is

$$\text{Loss}_{\text{KL}} = D_{KL}\left(q_\phi(z|x, y) \,||\, p_\theta(z|y)\right) = -\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right), \tag{3.28}$$

where J is the number of latent dimensions.

To guide the model toward economically and environmentally favorable designs, additional loss terms are introduced:

$$\text{Loss}_{\text{cost}} = \sum_{j=1}^{d} m_j \left(\hat{x}_j - x_j\right), \tag{3.29}$$

$$\text{Loss}_{\text{carbon}} = \sum_{j=1}^{d} e_j \left(\hat{x}_j - x_j\right), \tag{3.30}$$

where $\hat{x}_j$ and x_j represent the j-th reconstructed and original mix component values (after denormalization), and m_j and e_j denote the unit material cost and embedded carbon factor for component j, respectively.

The final objective function is then given by:

$$\text{Loss}_{\text{CVAE}} = \text{Loss}_{\text{recon}} + \beta \cdot \text{Loss}_{\text{KL}} + \lambda_{\text{cost}} \cdot \text{Loss}_{\text{cost}} + \lambda_{\text{carbon}} \cdot \text{Loss}_{\text{carbon}}, \tag{3.31}$$

where β, λ_{cost}, and λ_{carbon} are weighting factors that control the trade-off among reconstruction accuracy, latent regularization, material cost prediction, and embedded carbon prediction.

3.3.5 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II [29] is a widely adopted algorithm for multi-objective optimization, known for its efficiency in sorting solutions based on dominance, estimating solution density, and implementing a straightforward comparison mechanism.

The main steps involved in NSGA-II are as follows:

1. **Population Initialization:** A set of candidate solutions is randomly generated within the predefined constraints and parameter limits of the problem.

2. **Non-dominated Sorting:** The population is categorized into different non-dominated fronts. Solutions in the first front are not dominated by any other, while solutions in subsequent fronts are dominated only by those in earlier ranks.

3. **Crowding Distance Computation:** To promote diversity, a crowding distance value is assigned to individuals within the same front, reflecting their relative proximity to neighboring solutions.

4. **Selection:** A binary tournament selection process is employed, favoring individuals with superior ranking and greater crowding distance to ensure a well-distributed set of solutions.

5. **Genetic Operators:** Variation is introduced using genetic operations, such as simulated binary crossover (SBX) and polynomial mutation, to explore the search space effectively.

6. **Recombination and Next Generation Formation:** The offspring population is merged with the current population, and non-dominated sorting is reapplied. The next generation is formed by selecting individuals from the highest-ranked fronts until the population size constraint is met.

By iteratively refining the population through these steps, NSGA-II effectively identifies a well-distributed set of Pareto-optimal solutions, making it a powerful tool for solving multi-objective optimization problems.

3.3.6 Learning Curve Analysis

The learning curve is a graphical representation of model performance over successive training iterations, plotted as loss or error against the number of epochs. It provides valuable insights into how well a model is learning and generalizing to unseen data. In this paper, the learning curve is used to evaluate and compare the various CVAE architectures.

Interpreting the Learning Curve

The learning curve consists of two primary lines:

- **Training Loss:** This line represents the error or loss computed on the training dataset. It generally decreases over time as the model optimizes its parameters.

- **Validation Loss:** This line shows the error or loss computed on the validation dataset, providing an estimate of how well the model generalizes to unseen data.

Understanding the Gap Between Training and Validation Loss

The gap between the training and validation loss curves is crucial for diagnosing the model's behavior:

- **Overfitting:** If the training loss is significantly lower than the validation loss and the gap remains large, the model is memorizing the training data rather than learning generalizable patterns.

- **Underfitting:** If both training and validation loss remain high, the model is unable to capture the underlying data structure, indicating inadequate learning capacity or insufficient training.

- **Good Generalization:** When both training and validation losses decrease and converge towards a similar value, it indicates that the model is effectively learning and generalizing to new data.

Using Learning Curves for Comparison

Learning curves serve as a valuable tool for comparing different models, architectures, or training strategies. By analyzing the trends in training and validation loss, one can assess which model generalizes better to unseen data.

When comparing multiple models, key aspects to consider include:

- **Rate of Convergence:** A model with a faster decline in loss reaches optimal performance more quickly, making it more efficient for training.

- **Final Loss Values:** Lower final validation loss indicates better generalization, while a lower training loss with high validation loss may suggest overfitting.

- **Stability of the Curves:** A smooth and gradually decreasing loss curve suggests stable learning, whereas erratic fluctuations may indicate improper hyperparameter tuning or data inconsistencies.

By systematically comparing learning curves, informed decisions can be made regarding model selection, hyperparameter tuning, and potential improvements in training methodologies.

3.4 Implementation

This section outlines the implementation steps in the order they were practically performed, detailing the key processes followed in this study.

3.4.1 CatBoost Model Training and FOA-Based Hyperparameter Optimization

This study applies CatBoost for predicting the compressive strength of concrete, optimized using FOA. The process followed for model training and optimization mirrors previous methods used in similar optimization tasks.

3.4.2 Comparison of Various ML Models

The performance of various machine learning models for predicting compressive strength is compared based on three key evaluation metrics: Mean Absolute Error (MAE), R-squared (R2), and Root Mean Squared Error (RMSE). The models to be compared in this study include various traditional regression methods, ensemble techniques, and advanced boosting models. These models are: Support Vector Regression (SVR), Linear Regression, Ridge Regression, Lasso Regression, ElasticNet, K-Nearest Neighbors (KNN), AdaBoost, Decision Tree, Gradient Boosting, Random Forest, LightGBM, XGBoost, and CatBoost.

The models are trained without any hyperparameter optimization to establish a baseline performance for comparison with the CatBoost model.

Prediction with CatBoost

The prediction model was developed using CatBoost, which is a high-performance gradient boosting algorithm. CatBoost was selected for its ability to handle both numerical and categorical data without requiring significant preprocessing. In this case, the target variable was the compressive strength of concrete (f_c) in MPa, which was predicted based on a range of input variables such as cement content, water content, aggregate properties, and environmental factors like temperature and humidity.

CatBoost was trained by initializing the model with default parameters to establish a baseline performance. The model uses gradient boosting over decision trees, which builds successive trees that correct the errors of the previous ones. This results in a highly accurate model that efficiently handles non-linear relationships and interactions between the features.

Hyperparameter Optimization using FOA

To improve the performance of CatBoost, FOA was employed to fine-tune its hyperparameters. FOA is a metaheuristic algorithm that mimics the behavior of fireflies, where each firefly is attracted to other brighter fireflies. In this context, the ”brightness” is the objective function to be minimized, which is the prediction error of the model.

The hyperparameters optimized in this study were:

- **learning_rate**: The rate at which the model learns from the data, which was set in the range of [0.001, 1.0].

- **depth**: The depth of the trees, with values ranging from [1, 16].
- **l2_leaf_reg**: L2 regularization parameter, set in the range of [0.01, 100] to prevent overfitting.
- **bagging_temperature**: Controls the randomness of data sampling for each iteration, with values between [0, 10].
- **subsample**: The fraction of samples used per tree, with a range of [0.1, 1.0].
- **border_count**: The number of bins used for numerical feature quantization, with a range of [1, 255].
- **iterations**: The number of boosting iterations, set between [1, 3000].

The FOA performed the following steps for optimization:

1. **Initialization**: The fireflies were initialized randomly in the hyperparameter space. Each firefly represents a potential solution, which corresponds to a particular set of hyperparameters.
2. **Evaluation**: For each firefly (set of parameters), a CatBoost model was trained, and its performance was evaluated using Mean Squared Error (MSE).
3. **Attraction and Movement**: Fireflies with better performance (lower error) attract others, and the less-performing fireflies move towards the better ones in the hyperparameter space. This process allows the algorithm to explore the search space effectively.
4. **Update and Termination**: The process is repeated for a predefined number of iterations, at which point the best-performing firefly represents the optimal set of hyperparameters.

Table 3.2 shows the parameters applied to the FOA. These parameters were selected based on the guidelines and empirical findings presented in [27, 28]. FOAs with the same parameters were applied 10 times to obtain the best hyperparameters.

3.4.3 Model Evaluation

The optimized CatBoost model was evaluated against the baseline performance of the standard CatBoost model using standard performance metrics: RMSE, MAE, and (R^2). Additionally, a regression slope analysis was conducted for both the baseline and FOA-optimized models on the train and test sets to assess the alignment between predicted and actual values, providing further

Table 3.2: Parameters for the Firefly Algorithm

Parameter	**Value**
Number of fireflies	10
Number of hyperparameters	7
Epoch	100
Alpha (α)	0.1
Beta (β_0)	1.0
Gamma (γ)	1.0

insight into the accuracy improvements achieved through optimization. SHAP analysis was also performed to interpret the impact of individual features on the model's predictions, enabling a deeper understanding of feature contributions. The results demonstrated a significant reduction in prediction error for the FOA-optimized CatBoost model compared to the baseline, alongside an improved regression slope, underscoring the effectiveness of metaheuristic optimization methods in enhancing model performance through fine-tuning. The SHAP analysis further validated the model's interpretability by identifying key features influencing the predictions.

3.4.4 Regression Slope Analysis

Regression analysis was conducted to compare the performance of the basic CatBoost model with the FOA-CatBoost model. In this study, regression analysis is performed for the CatBoost and FOA-Catboost models separately where the slope is determined using the residuals from the predictions of the respective models. The y-axis shows the predicted values and the x-axis shows the actual values. The sign and magnitude of the slope provide insights into whether the relationship is positive or negative and how significant the impact of the independent variable is on the dependent variable. By analyzing the slope alongside other regression diagnostics, the accuracy of the model is assessed.

3.4.5 K-Fold validation with $k = 5$

In this study, a 5-fold cross-validation ($k = 5$) was employed where the dataset was divided into five equal parts. The model is trained and evaluated five times, each time using a different fold as the validation set and the remaining four folds for training. The results are then averaged over the five iterations to provide a comprehensive measure of model performance.

In this study, valuation metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2) are used to assess the model's predictive accuracy during k-fold cross-validation. By averaging these metrics across all folds, a comprehensive understanding of the model's performance is obtained. K-fold cross-validation ensures that the model does not overfit and has better generalization ability.

3.4.6 SHAP analysis

SHAP analysis was applied to the FOA-CatBoost model to achieve two key objectives: identifying the most influential features in predicting compressive strength and analyzing their impact direction and magnitude. By determining feature importance, the analysis provides insights into the mix design parameters and environmental conditions that significantly affect model predictions. Additionally, the direction and magnitude of each feature's influence were examined, where high SHAP values indicate substantial impact, and the sign of the value reveals whether a feature increases or decreases the predicted strength.

3.4.7 CVAE Implementation

The Conditional Variational Autoencoder (CVAE) was implemented to model the relationship between concrete mix parameters and compressive strength. It enables the generation of new mixture designs that align with the distribution of the dataset while conditioning on a target compressive strength. The implementation process consisted of data preprocessing, model architecture design, and training and evaluation.

Data Preprocessing: The input features X and the target variable Y (compressive strength) were separated, and the dataset was split into training and testing sets in an 8:2 ratio. A min-max validation check was conducted to ensure that each feature in the test set remained within the range observed in the training set. This step was necessary to guarantee that the Min-Max scaler, fitted on the training data, could be applied to the test set without producing values outside the $[0, 1]$ range. If an invalid split was detected, the dataset was re-split up to ten times to maintain consistency. Finally, the features were normalized using Min-Max Scaling before further processing.

Model Architecture: The CVAE consists of an encoder-decoder architecture. The encoder maps the input X and condition Y to a latent distribution, characterized by mean μ and log-variance $\log \sigma^2$. The decoder reconstructs X from the sampled latent representation and the condition. The CVAE design consists of an encoder with fully connected layers with the number of neurons in each layer, followed by separate layers for μ and $\log \sigma^2$. The decoder mirrors this structure, generating the reconstructed X with a Sigmoid activation in the final layer to ensure outputs remain within $[0, 1]$. Several CVAE architectures were designed and evaluated as shown in Table 3.3

In the case of multi-layer architectures, the next hidden layer in the encoder was designed with half the neurons of the preceding layer to progressively reduce the latent repre-

Table 3.3: CVAE Encoder and Decoder Design

Number of Hidden Layers	Encoder	Decoder
1	(16)	(16)
1	(32)	(32)
1	(64)	(64)
1	(128)	(128)
1	(256)	(256)
2	(32, 16)	(16, 32)
2	(128, 64)	(64, 128)
2	(256, 128)	(128, 256)
3	(32, 16, 8)	(8, 16, 32)
3	(256, 128, 64)	(64, 128, 256)
5	(32, 16, 8, 4, 3)	(3, 4, 8, 16, 32)

sentation while retaining essential features, thereby enhancing the model's ability to extract hierarchical patterns [33]. The decoder was designed to mirror the encoder to ensure a symmetric reconstruction process as is standard practice [34] .

Training and Optimization: The loss function used in training the Conditional Variational Autoencoder (CVAE) consisted of four components: the mean squared error (MSE) for reconstruction, the Kullback-Leibler divergence (KLD), the material cost prediction loss, and the embedded carbon prediction loss. The KLD term was scaled by a factor of $\beta = 0.001$ to regularize the latent space, promoting a smooth and structured latent distribution that improves the model's generative capabilities. The material cost loss was weighted by $\lambda_{\text{cost}} = 0.1$, while the embedded carbon loss was assigned a smaller weight of $\lambda_{\text{carbon}} = 0.001$, based on empirical testing. These additional terms guide the model to generate economically and environmentally favorable mix designs without significantly compromising reconstruction accuracy.

The model was trained using the Adam optimizer with a learning rate of 10^{-3}, ensuring stable convergence. Training was conducted over 5000 epochs, with a batch size of 32 to balance computational efficiency and gradient updates. During training, the loss was continuously monitored, with both the reconstruction error and latent space regularization assessed at each epoch.

To evaluate model performance, the test set was used at every epoch to track generalization ability. Additionally, the trained model parameters, along with the optimizer state, were saved to enable further analysis, fine-tuning, or potential reuse in downstream tasks.

This implementation ensures a robust CVAE model capable of capturing complex dependencies in concrete mix design data.

3.4.8 Data Collection for Cost and Embodied Carbon Calculation

To facilitate the implementation of NSGA-II in optimizing 3D concrete printing mix designs, data on material costs and embodied carbon were collected from relevant sources.

Material Costs

The cost data for materials were obtained from online sources, based on the average prices per kilogram for the period 2024-2025. The material costs considered correspond to the bulk procurement prices applicable within the Indian market. The collected cost values are presented in Table 3.4.

Table 3.4: Bulk material prices for concrete constituents (INR per kg, 2024–2025)

Material	Price (INR/kg)	Citation
Cement	6.5	[35]
Silica fume	45.7	[36]
Blast furnace slag	2.3	[37]
Fly ash	2.0	[38]
Quartz powder	3.0	[39]
Limestone powder	2.5	[40]
Nano silica	250	[41]
Fiber	135	[42]
Sand	1.6	[43]
Gravel	0.8	[44]
Superplasticizer	240	[45]

Embodied Carbon Data

The embodied carbon values were sourced from the ICE Advanced Database [46], a widely used reference for material carbon footprint assessment, and other sources. The embodied carbon values for selected materials are summarized in Table 3.5.

These datasets serve as essential inputs for the cost and sustainability objectives in the multi-objective optimization framework using NSGA-II.

Table 3.5: Embodied carbon values (kg CO_2 per kg of material) based on global averages

Material	Embodied Carbon (kg CO_2/kg)	Citation
Cement	0.84	[46]
Silica fume (SF)	0.03	[47]
Blast furnace slag (BFS)	0.08	[46]
Fly ash (FA)	0.004	[47]
Quartz powder (QP)	0.023	[48]
Limestone powder (LSP)	0.02	[49]
Nano silica (NS)	5.00	[49]
Fiber	2.50	[50]
Sand (fine aggregate)	0.007	[47]
Gravel (coarse aggregate)	0.016	[47]
Water	0.0002	[48]
Superplasticizer (SP)	0.94	[48]

3.4.9 NSGA-II Optimization Process

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is employed to optimize a concrete mixture design for achieving desired compressive strength while minimizing cost and embodied carbon. For the NSGA-II optimization, the FOA-CatBoost model trained earlier and the best-performing CVAE architecture, featuring encoder and decoder layers of size (128,64) and (64,128) respectively, were utilized. Fig. 3.6 illustrates the framework for optimizing mix designs using NSGA-II. The CVAE decoder generates new mix designs with curing conditions from the latent space. A regression model then evaluates their performance in terms of embodied carbon, cost, and strength. NSGA-II iteratively explores the latent space to optimize these properties. The process consists of several key steps, as outlined below.

Problem Definition

The optimization problem is formulated as a multi-objective problem with two objectives and one constraint. The objectives are

1. Minimize the total material cost.

2. Minimize the total embodied carbon.

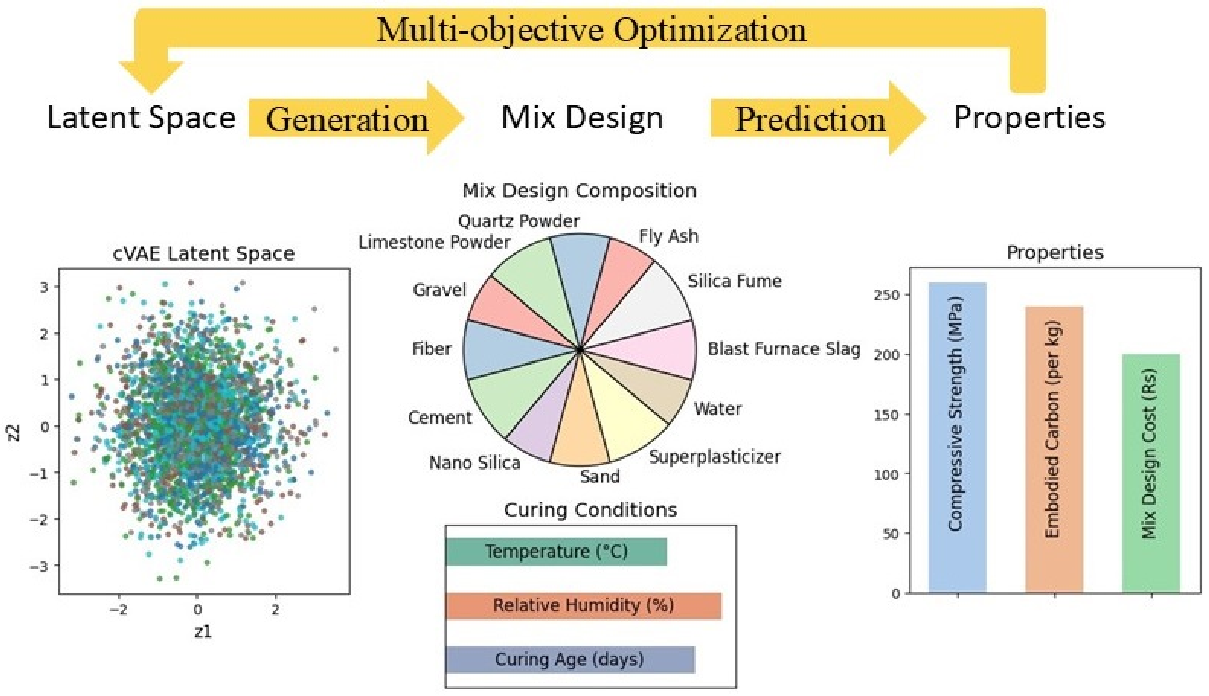


Figure 3.6: Multi-objective optimization plot

and to ensure that the predicted compressive strength of the generated mixture design is within an acceptable range, the following constraint is imposed:

$$|\hat{f}_c - f_c| \leq 1 \text{ MPa} \tag{3.32}$$

where $\hat{f}_c$ represents the predicted compressive strength, and f_c denotes the desired compressive strength.

The decision variables are represented by a latent space encoding, $\mathbf{z}$, which is decoded into material compositions using a trained Conditional Variational Autoencoder (CVAE). The FOA-CatBoost model is employed to predict compressive strength based on the decoded material compositions.

Initialization

An initial population of size $P = 1000$ is generated within the latent space bounds of $[-3, 3]$ for each decision variable. The corresponding decoded material compositions are obtained using the CVAE decoder.

Objective Evaluation

For each individual in the population:

- The decoded material composition is obtained from the CVAE in the normalized form. It is transformed back to the original space using the scaler.

- The FOA-CatBoost model predicts the compressive strength.
- The material cost and embodied carbon are computed as:

$$f_2 = \sum_{i=1}^{n} \text{cost}_i \cdot x_i, \tag{3.33}$$

$$f_3 = \sum_{i=1}^{n} \text{embodied\_carbon}_i \cdot x_i, \tag{3.34}$$

  where x_i, cost_i, and embodied_carbon_i represent the quantity, cost, and embodied carbon of material i.

- The constraint on compressive strength is evaluated to ensure valid solutions.

Selection and Evolution

NSGA-II employs the following steps for evolutionary optimization:

1. Perform non-dominated sorting to classify solutions into Pareto fronts.
2. Compute crowding distance within each front to maintain diversity.
3. Apply tournament selection based on rank and crowding distance.
4. Perform simulated binary crossover and polynomial mutation to generate offspring.
5. Evaluate the offspring population and combine with parents.
6. Select the best P individuals based on non-dominated sorting and crowding distance for the next generation.

Termination and Post-processing

The optimization runs for 300 generations, after which:

- The final Pareto-optimal solutions are identified.
- The hypervolume indicator is computed to rank solutions based on diversity and dominance.
- The best solution is selected as the one with the highest hypervolume contribution.
- The material compositions of the Pareto-optimal solutions are saved for further analysis.

3.4.10 Learning Curve Analysis for CVAE Architectures

To evaluate and compare the performance of different Conditional Variational Autoencoder (CVAE) architectures, learning curves are generated based on training and test loss values recorded during the training process. The procedure follows these steps:

Data Collection

The output files containing the loss values for different CVAE architectures are stored in a designated directory. The filenames encode the architectural details, particularly the encoder and decoder layer sizes.

- The output directory is scanned to identify all relevant files.
- Each file name is parsed to extract the corresponding encoder layer configuration.
- The architectures are sorted based on the complexity of the encoder structure.

Generating Learning Curves

For each CVAE configuration, the training and test loss values are plotted against the number of epochs to visualize model convergence. The process involves:

1. Reading the CSV file corresponding to the CVAE architecture.
2. Extracting the relevant columns: Epoch, Training Loss, and Test Loss.
3. Plotting the training loss and test loss curves using distinct markers and line styles for clarity.
4. Labeling the plot with the architecture details, epoch count, and loss values.
5. Ensuring a well-formatted graph with legends and grid lines for readability.

Interpretation of Learning Curves

The learning curves allow for the assessment of different CVAE configurations. A well-performing architecture exhibits:

- A smooth decline in training and test loss over epochs.
- Minimal overfitting, indicated by a small gap between training and test loss.
- A convergence point where loss stabilizes, signifying effective training.

This comparative analysis assists in selecting the optimal CVAE architecture based on its generalization performance.

Chapter 4

Results and Discussion

This chapter presents the performance evaluation of the CatBoost model before and after hyperparameter optimization with FOA and the SHAP analysis for model interpretation. The primary metrics used to evaluate the performance of the model are MAE, RMSE, and the R^2 score.

4.1 Comparison of ML Models

Table 4.1: Model Comparison Results

Model	**MAE**	R^2	**RMSE**
SVR	27.795783	0.240679	34.483195
Linear Regression	16.618293	0.720116	20.935526
Ridge Regression	16.618252	0.720117	20.935497
Lasso Regression	16.606289	0.720448	20.923120
ElasticNet	16.601269	0.720506	20.920932
KNN	13.587325	0.793887	17.965857
AdaBoost	10.846152	0.885871	13.368785
Decision Tree	6.224994	0.941237	9.592867
Gradient Boosting	7.389356	0.944906	9.288529
Random Forest	5.745649	0.960376	7.877229
LightGBM	5.546900	0.966208	7.274468
XGBoost	4.997384	0.970778	6.764699
CatBoost	4.842734	0.973785	6.407227

The models including Support Vector Regression (SVR), Linear Regression, AdaBoost, etc. are trained without any hyperparameter optimization to establish a baseline performance for comparison with the CatBoost model. The results for each model, as measured by the aforementioned metrics, are summarized in the Table 4.1. The models are sorted in ascending order of R^2 value. The CatBoost model provides the best performance, achieving the lowest MAE and RMSE while delivering the highest R^2 and lowest MAE and RMSE values indicating the best fit to the data among all the ML models employed. Among the models, the linear models exhibited the poorest performance, while the tree-based models demonstrated intermediate results. The boosting models outperformed all others, delivering the best results.

4.2 Baseline Performance of CatBoost Model

Initially, the CatBoost model was trained using default hyperparameters to establish a baseline for performance. The results for this default model are presented in Table 4.3. With an R^2-score of 0.9594, the model could explain approximately 95.94% of the variance in the target variable (compressive strength). The relatively low values of MAE (5.9288) and RMSE (7.9679) indicate that the model's initial predictions were fairly accurate. However, there was potential to enhance its predictive capability by tuning hyperparameters.

4.3 Hyperparameter Tuning

After applying FOA to optimize the CatBoost model's hyperparameters, notable improvements were observed across all metrics. The optimized model, referred to as FOA-CatBoost, was trained with the hyperparameters detailed in Table 4.2.

Table 4.2: FOA-CatBoost Hyperparameters

Hyperparameter	**Value**
Learning Rate	0.2286
Depth	6
L2 Leaf Regularization	35
Bagging Temperature	5.7426
Subsample Ratio	0.6313
Border Count	146
Iterations	2117

The learning rate was optimized to approximately 0.2286, which suggests a moderately aggressive step size during training, facilitating faster convergence while maintaining stability. The tree depth was set to 6, indicating a relatively shallow model designed to prevent overfitting by limiting the complexity of individual decision trees, which helps the model generalize better.

The L2 leaf regularization term was optimized to 35, reflecting a slighthly strong regularization to penalize large weights and reduce overfitting, ensuring that the model captures true patterns rather than noise. The bagging temperature was set to 5.7426, a relatively high value, suggesting that the optimization encouraged more diverse subsets of data in each iteration, promoting better generalization. The subsample ratio of 0.6313 means that approximately 63.13% of the dataset was used for each iteration, introducing randomness to avoid overfitting while retaining enough data for effective training. The border count was optimized to 146, which defines the number of splits or "borders" in the decision trees, allowing the model to capture more complex patterns in the data. Finally, the number of iterations was set to 2117, indicating a relatively high number of boosting rounds, ensuring thorough optimization and improved model performance at the cost of increased training time. These hyperparameters reflect a well-balanced CatBoost model, with regularization, subsampling, and randomization to prevent overfitting, and sufficient complexity to capture the underlying patterns, resulting in a robust and efficient model for the task at hand.

4.4 Performance of Optimized CatBoost Model

Table 4.3: Comparison of Default CatBoost and FOA-CatBoost Model Results

	Training data			**Test data**		
	MAE	R^2	**RSME**	**MAE**	R^2	**RSME**
CatBoost	4.0771	0.9803	5.6401	5.9288	0.9594	7.9679
FOA-CatBoost	1.7785	0.9928	3.4020	3.9006	0.9786	5.5442

Table 4.3 displays the comparison of the performance of CatBoost and FOA-CatBoost in the training and testing sets. FOA-CatBoost performed better in both the training and testing predictions. In the testing set regression, FOA-CatBoost achieved an MAE of 3.9006, indicating a 34.20% improvement over the default model. The R^2-score rose to 0.9786, an enhancement of 2%, demonstrating that the model could explain nearly 97.86% of the variance in compressive strength. Furthermore, the model achieved an RSME score of 5.5442 which is an improvement of 30.42% over the default model.

4.5 Comparison with available studies

Numerous studies in the literature have explored data-driven approaches to forecasting the compressive strength of UHPC. These studies, referenced in works [9], [4], [51], [5], and [7] demonstrate that various machine learning models and datasets can effectively predict the compressive strength of concrete. In this research, the performance of the FOA-CatBoost model,

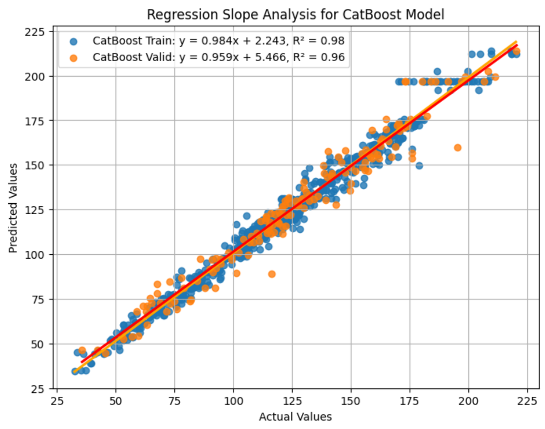
Table 4.4: Comparison of proposed FOA-CatBoost Model results with models reported in available literature

			Test data		
Reference	**Models**	**Dataset size**	**MAE**	R^2	**RSME**
[9]	DMO-CatBoost	785	4.510	0.978	6.150
[4]	GSCV-AdaBoost	400	8.210	0.883	12.484
	GSCV-RF		6.455	0.930	9.686
	GSCV-XGBoost		5.281	0.937	9.167
	GSCV-LightGBM		5.652	0.929	9.770
	GSCV-CatBoost		5.084	0.945	8.567
[51]	GWO-CatBoost	175	Not reported	0.924	3.971
	GWO-AdaBoost		Not reported	0.929	4.101
	GWO-LightGBM		Not reported	0.937	3.606
	GWO-XGBoost		Not reported	0.978	2.129
[5]	SVR	299	10.245	0.840	18.717
	DT		4.644	0.987	6.598
	SVR-Bagging		10.770	0.897	19.007
	SVR-Boosting		9.491	0.961	12.833
	RF		3.989	0.986	7.134
	GB		3.900	0.986	7.210
	GEP		5.690	0.985	7.405
[7]	KNN	810	13.974	0.772	4.345
	XGBoost		5.608	0.954	2.906
	RF		6.247	0.950	2.980
	ETR		5.953	0.950	2.975
	Stacked		5.484	0.957	8.202
This study	FOA-CatBoost	785	3.901	0.979	5.544

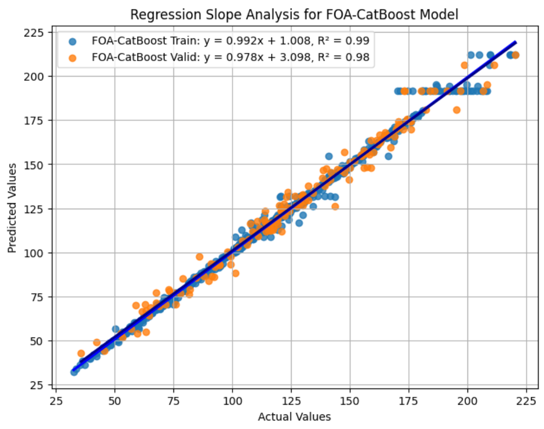
which achieved the highest forecasting accuracy, is compared with previous studies' outcomes in Table 4.4. Upon reviewing Table 4.4, it is evident that the FOA-CatBoost model performs better or at least at par with the methods used in earlier research.

4.6 Regression Slope Analysis

The accuracy of the models is assessed through regression slope analysis, comparing actual values on the x-axis with predicted values on the y-axis as shown in Figure 4.1. On the left, Figure 4.1a shows the performance of the CatBoost model before optimization, while Figure 4.1b on the right demonstrates the improvement in performance after applying Firefly optimization to the CatBoost hyperparameters. This method is commonly employed to evaluate the predictive accuracy of machine learning models. For this study, the CatBoost model and the FOA-CatBoost model were evaluated on both the training and validation datasets.



(a) CatBoost model



(b) FOA-CatBoost model

Figure 4.1: Regression slope analysis of the models

The CatBoost model achieved regression slope values of 0.984 and 0.959 for the training and validation sets, respectively, with R^2 scores of 0.98 and 0.96. These results indicate a strong correlation between predicted and actual values, though the validation slope being slightly lower than the training slope suggests minor overfitting in CatBoost's performance.

On the other hand, the FOA-CatBoost model showed improved alignment between predicted and actual values, with regression slopes of 0.992 and 0.978 for the training and validation sets, respectively, and corresponding R^2 scores of 0.99 and 0.98. The FOA-CatBoost model thus exhibits a closer alignment to the ideal line (slope = 1) in both training and validation, reflecting a more consistent performance across both subsets and reduced overfitting compared to the CatBoost model.

Both models exhibit regression slopes close to 1 on the training and validation sets, which demonstrates a high degree of predictive accuracy. However, the FOA-CatBoost model outperforms the CatBoost model slightly, especially on the validation set, where the slope of 0.978 is nearer to 1 than CatBoost's 0.959. This suggests that the FOA-CatBoost model, optimized with the Firefly Algorithm, achieves greater stability and generalization in estimating the target values, supporting its use as the preferred model for predicting outcomes in this study.

4.7 5-Fold Cross-Validation Results

To evaluate the performance and generalizability of the machine learning model, a 5-fold cross-validation was conducted. This method involves partitioning the dataset into five subsets or "folds" and performing training and validation iteratively, ensuring that each fold serves as a validation set once while the remaining folds constitute the training set. This approach provides a robust estimate of the model's predictive performance by reducing the risk of overfitting.

The accuracies obtained for each fold, expressed in terms of MAE, R^2-score, and RMSE, are presented in Table 4.5.

Table 4.5: 5-Fold Cross-Validation Results

Fold	**MAE**	R^2**-score**	**RMSE**
1	4.1280	0.9792	5.7107
2	4.2752	0.9741	6.2558
3	4.9072	0.9713	7.2520
4	5.4170	0.9543	8.6869
5	3.9006	0.9786	5.5452

The aggregated performance metrics across all folds are summarized as follows:

- **MAE**: 4.5256
- R^2**-score**: 0.9715
- **RMSE**: 6.6891

These results demonstrate the model's strong predictive capabilities, with a high R^2-score indicating that approximately 97.15% of the variance in the target variable is explained by the model. The low MAE and MSE values reflect the model's accuracy and precision in predicting the target outcomes, underscoring its effectiveness in handling the given dataset.

4.8 SHAP Analysis for Predicting Compressive Strength of Concrete

The SHAP analysis is done on the prediction results of FOA-CatBoost as it showed the better performance. Figure 4.2 shows the influence of each feature on the prediction model. The SHAP mean plot provides insights into the most influential features in predicting the compressive strength of concrete. According to this analysis, the curing age of the concrete is the most impactful factor, with a mean SHAP value of approximately 17.41. This finding aligns with our understanding of concrete chemistry, as the compressive strength typically increases over time due to continuous hydration and hardening processes.

Following age, the fiber content is the second most influential feature, with a SHAP value of around 11.45. Fibers are often incorporated into concrete to improve its tensile strength and durability, and their high SHAP value suggests that fiber addition significantly affects compressive strength, likely by enhancing matrix cohesion and crack resistance. Cement content,

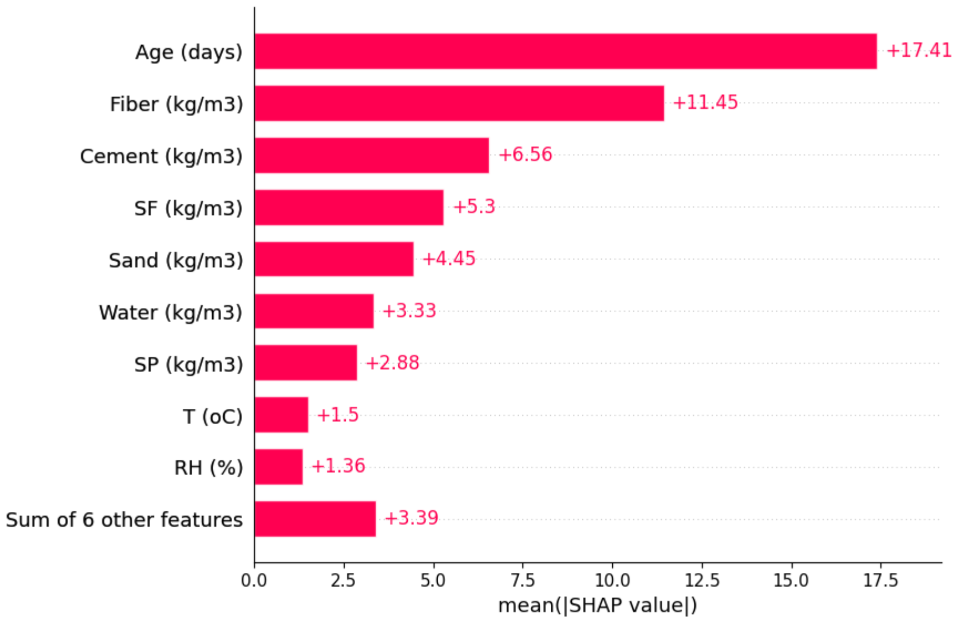


Figure 4.2: Mean Absolute SHAP values plot

with a similar SHAP value of 6.56, is equally crucial since cement serves as the primary binder in concrete, directly contributing to its overall strength. A higher cement content generally correlates with increased compressive strength, so it is logical that cement content would have a substantial impact. Silica fume, with a mean SHAP value of about 5.3, also plays a prominent role. As a supplementary cementitious material, silica fume enhances the strength and durability of concrete by filling voids and improving bond strength. Sand content, showing a SHAP value of 4.45, has a moderate influence on strength as well. As a fine aggregate, sand contributes to the concrete's density and packing efficiency, impacting its structural properties. Water content, with a mean SHAP value of 3.33, also affects compressive strength, though to a lesser degree than cement or superplasticizer. The water-cement ratio is critical, as excess water can weaken concrete by increasing its porosity. Temperature, with a SHAP value of 1.5, has a modest impact on compressive strength as well, as it influences the rate of curing and hydration. The presence of superplasticizer, with a SHAP value of approximately 2.88, also affects compressive strength notably. Superplasticizers improve the workability of the concrete mixture without increasing water content, resulting in a denser and stronger concrete matrix.

Relative humidity (RH), with a SHAP value of 1.36, has a lower impact compared to other features, but it can affect the curing environment and moisture retention, influencing long-term strength. Finally, the remaining six features are summarized with a cumulative SHAP value of 3.39, indicating a smaller but collectively noteworthy effect on compressive strength.

The SHAP analysis reveals that age, fiber content, and key components such as cement, silica fume, cement, sand, water, and superplasticizer are the most critical factors in predicting concrete compressive strength. These findings are consistent with principles of concrete science, where age influences hydration, and high-performance materials like silica fume and fibers play essential roles in enhancing mechanical properties.

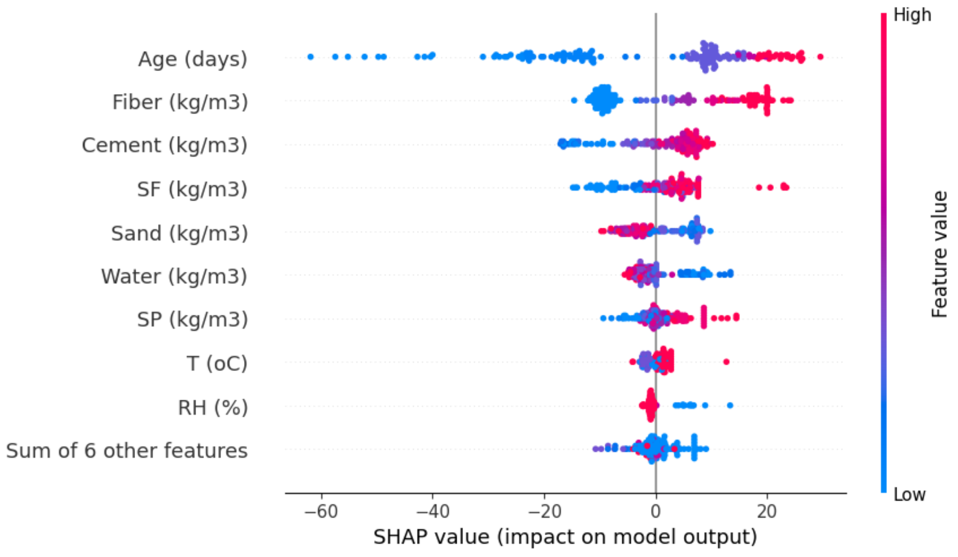


Figure 4.3: SHAP feature impact plot

A SHAP summary plot of each feature's effect on the prediction of compressive strength of concrete is shown in Fig. 4.3. The y-axis lists features in descending order of importance, starting with the most impactful feature at the top. The x-axis represents the SHAP values, which measure the magnitude and direction of each feature's influence on the model's output for compressive strength. Each dot represents a specific data instance, and the color scale—from blue to red—reflects the feature value, with blue indicating lower values and red representing higher values.

The most influential feature, *Age*, shows a strong positive correlation with compressive strength, as indicated by red dots (higher values of age) clustered on the right side (positive

SHAP values). This is consistent with concrete behavior, where increased curing time typically improves compressive strength as the material continues to hydrate and gain solidity over time.

The *Fiber* content also demonstrates a positive impact on compressive strength. Higher values of fiber (indicated by red dots) align with positive SHAP values, suggesting that increasing fiber content enhances the concrete's strength, likely due to improved tensile strength and crack resistance provided by fibers.

For *Cement*, the pattern also aligns with expectations: higher cement content (red dots) is associated with positive SHAP values, indicating that an increase in cement contributes positively to compressive strength. This is due to the cement's role in bonding the aggregate particles and forming a denser matrix.

SF (presumably silica fume) similarly exhibits a positive relationship with compressive strength. Higher values of SF, shown by red dots on the right, indicate a beneficial effect, likely because silica fume improves concrete's microstructure, increasing its strength and durability.

Conversely, the features *Sand* and *Water* show a negative impact on compressive strength, particularly evident in red dots (representing higher values) concentrated on the left side (negative SHAP values). This result aligns with the common observation that excess sand and water in the mix leads to increased porosity, weakening the concrete and reducing its compressive strength. This finding also corresponds to a negative correlation between sand and water content and compressive strength observed in other analyses.

Environmental features, such as *T (°C)* (temperature) and *RH (%)* (relative humidity), exhibit complex relationships. Higher temperatures (red dots) tend to have a positive SHAP impact, potentially due to accelerated curing at warmer temperatures. However, higher humidity (RH) tended to have negative SHAP impact while lower RH tended to have positive SHAP impact.

SHAP analysis highlights how different material and environmental factors contribute variably to concrete's compressive strength. Positive contributors like curing age, cement, fibers, and silica fume increase strength, while higher water content generally reduces it. These insights emphasize the importance of balancing mix constituents and environmental conditions to achieve optimal strength in concrete formulations.

4.9 Comparison of CVAE Architectures Based on Loss Metrics

The performance of different Conditional Variational Autoencoder (CVAE) architectures was evaluated using training and test loss metrics, as presented in Table 4.6. The results indicate a clear trend where increasing the number of hidden layers and neurons generally leads to improved performance, but with some exceptions.

Table 4.6: CVAE Train and Test Loss for Different Architectures

Model No.	**Architecture**	**Train Loss**	**Test Loss**
1	Encoder: (16) Decoder: (16)	0.096568	0.142106
2	Encoder: (32) Decoder: (32)	0.051849	0.082918
3	Encoder: (64) Decoder: (64)	0.041571	0.065377
4	Encoder: (128) Decoder: (128)	0.039315	0.064780
5	Encoder: (256) Decoder: (256)	0.027361	0.046817
6	Encoder: (32,16) Decoder: (16,32)	0.034560	0.074149
7	Encoder: (128,64) Decoder: (64,128)	0.036857	0.085323
8	Encoder: (256,128) Decoder: (128,256)	0.010312	0.044430
9	Encoder: (32,16,8) Decoder: (8,16,32)	0.035585	0.062233
10	Encoder: (256,128,64) Decoder: (64,128,256)	0.011154	0.049881
11	Encoder: (32,16,8,4,3) Decoder: (3,4,8,16,32)	0.509857	0.513288

For single-layer architectures, increasing the number of neurons from 16 to 256 progressively reduces both training and test loss. However, the lowest test loss among single-layer models is observed for the *Encoder: (256) Decoder: (256)* configuration (test loss: 0.046817), after which further increase in width was not explored. The relatively poor performance of the architecture with only 16 neurons can be attributed to the dataset's input dimension of 15, where a transformation from 15 to 16 dimensions provides limited representational power. Notably, only single-layer architectures with 64 neurons or more demonstrated substantial improvements, suggesting that a hidden layer should contain at least three times the input dimension to effectively model the data distribution.

In the case of multi-layer architectures, the *Encoder: (256,128) Decoder: (128,256)* configuration (Model No. 8) achieved the lowest overall test loss of 0.044430, making it the best-performing model in this study. Other strong performers include the *Encoder: (256,128,64) Decoder: (64,128,256)* configuration, which achieved a test loss of 0.049881, and the *Encoder: (128,64) Decoder: (64,128)* model, with a test loss of 0.085323. These results suggest that hierarchical encoding with progressively smaller latent representations enhances the model's ability to generalize, especially when appropriate depth and width are balanced.

Conversely, deeper architectures with more layers, such as the *Encoder: (32,16,8,4,3) Decoder: (3,4,8,16,32)* configuration, exhibited significantly higher loss values (train loss: 0.509857, test loss: 0.513288). This indicates that excessive architectural complexity may hinder effective optimization and generalization, possibly due to vanishing gradients or insufficient data for deep supervision.

Overall, the results indicate that the optimal CVAE architecture balances depth and width, with two or three hidden layers being ideal. Hierarchical multi-layer encoders, particularly with wide initial layers, demonstrated superior performance in capturing the underlying data distribution while avoiding overfitting.

4.10 Analysis of Learning Curves Across Autoencoder Architectures

The learning curves of models 1 through 11 in Fig 4.4 exhibit several common characteristics. Both the training loss (blue line) and test loss (red dashed line) decrease rapidly during the first 1000 epochs, indicating that the models quickly capture general patterns in the data. As training progresses, both losses converge to low values, suggesting that the models have effectively learned the data and no longer show significant improvements. The small and consistent gap between training and test loss throughout training indicates minimal overfitting. However, this gap is notably larger in model no. 1 and model no. 11 compared to other models, suggesting that these particular architecture underfits more than others.

The learning curves of models 1 through 5 display smooth trends in both training and test losses, indicating stable training behavior. Compared to models 1 and 2, the configuration of model 3 achieves slightly lower final loss values, implying improved reconstruction capability. Model 4, with encoder and decoder dimensions of 128, demonstrates better performance than all preceding architectures. Model 5, with encoder and decoder dimensions of 256, exhibits slightly worse performance than model 4 in terms of loss values. However, the smaller gap between training and test losses suggests improved generalization compared to previous models.

The learning curve of model 6 indicates worse performance than models 2 to 5. Additionally, models 6 to 11 exhibit instability, as inferred from the spikes in their learning curves. These spikes suggest that training instability arises due to factors such as noise or unstable weight updates, highlighting that deeper models with multiple layers tend to experience training instability.

The learning curve of model 7, corresponding to an autoencoder with encoder dimensions (128, 64) and decoder dimensions (64, 128), exhibits the best performance, achieving the lowest training and test losses along with the smallest gap between them. Among all multi-layer models, model 7 also demonstrates the least instability during training.

The learning curve of model 8, corresponding to an autoencoder with encoder dimensions (256, 128) and decoder dimensions (128, 256), shows better performance in terms of

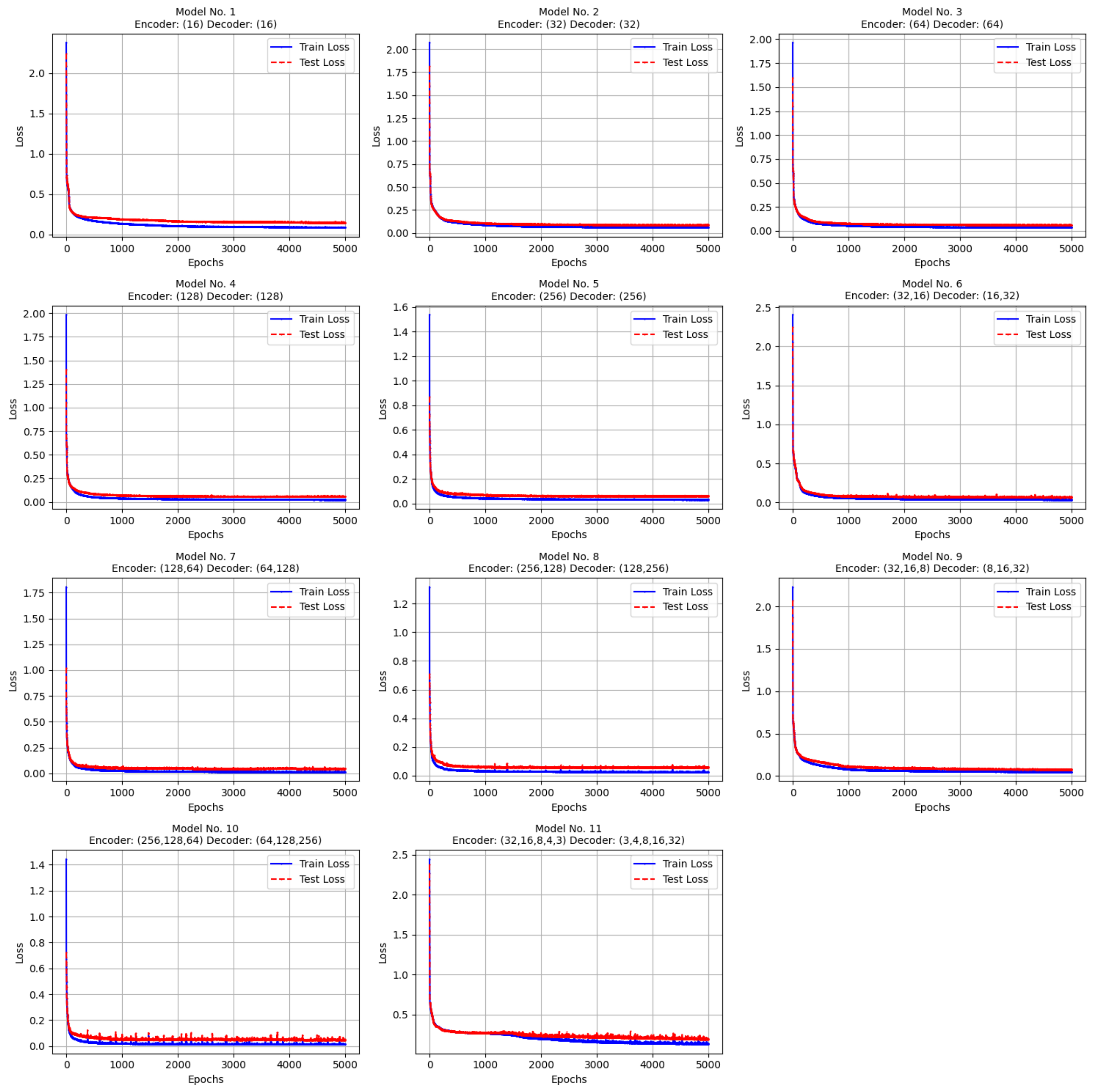


Figure 4.4: Learning curve plots

training and test losses, as well as a reduced gap between them, compared to all single-layer models. However, training instability is higher than in any single-layer model.

The learning curves of models 9 through 11 indicate that adding three or more hidden layers degrades performance, either by increasing training and test losses or by exacerbating training instability. The five-layer model exhibits the highest instability, suggesting that deeper models struggle to effectively capture patterns in the data.

4.11 Mix Design Generation and Optimization

The CVAE, NSGA-II and FOA-CatBoost were integrated to generate and optimize mix designs. Six target compressive strengths were used as condition for the CVAE. Table 4.7 shows the optimized mix designs containing various component weights and curing conditions for six target compressive strengths. Given different strength requirement, the one with the best hypervolume is selected from the Pareto Frontiers. Only mix designs within ±5% of the target compressive strength were considered eligible for selection.

Table 4.7: Optimized Mix Proportions and Environmental Conditions for Target Compressive Strengths

Component	**60 MPa**	**80 MPa**	**100 MPa**	**120 MPa**	**140 MPa**	**160 MPa**
Cement (kg/m^3)	349.36	360.90	325.15	399.30	320.05	336.19
SF (kg/m^3)	10.45	14.88	46.37	0.04	71.13	68.64
BFS (kg/m^3)	66.83	51.73	23.32	0.00	53.00	162.85
FA (kg/m^3)	0.00	0.00	0.00	0.01	0.00	0.00
QP (kg/m^3)	0.00	0.00	0.00	0.00	0.00	0.00
LSP (kg/m^3)	42.98	0.42	91.64	0.00	148.30	217.51
NS (kg/m^3)	0.00	0.00	0.00	1.04	0.00	0.00
Fiber (kg/m^3)	0.00	0.00	0.00	0.00	0.00	0.00
Sand (kg/m^3)	387.04	739.88	676.40	820.85	768.93	695.10
Gravel (kg/m^3)	1113.13	1021.74	1018.94	1064.89	894.18	906.14
Water (kg/m^3)	156.75	161.95	137.66	104.47	139.00	145.10
SP (kg/m^3)	7.16	5.47	9.51	21.44	14.27	14.67
T (°C)	20.07	20.50	20.15	20.11	20.07	20.02
RH (%)	99.53	100.00	96.50	100.00	93.68	94.44
Age (days)	5.00	53.65	36.62	73.19	72.97	130.97
Strength (MPa)	61.97	83.05	101.95	116.22	146.19	152.04
Material Cost (Rs.)	6236.92	6459.94	8694.71	10168.34	11193.99	11598.67
Carbon (kg CO_2/kg)	327.26	334.45	308.22	383.56	311.31	335.03

Table 4.8 highlights the effectiveness of the optimized mix designs in reducing both material cost and embedded carbon across all target compressive strengths. Compared to the mean values from the original dataset, the optimized mixtures demonstrate a substantial reduction in cost, with savings ranging from approximately 50% to over 80%. Similarly, the embedded

Table 4.8: Comparison of Optimized and Original Dataset Mixes for Cost and Embedded Carbon

Target Strength (MPa)	**60**	**80**	**100**	**120**	**140**	**160**
Optimized Cost (Rs.)	6236.92	6459.94	8694.71	10168.34	11193.99	11598.67
Mean Cost (Rs.)	13221.14	24942.65	25241.71	28478.27	32762.81	35642.63
Optimized Carbon	327.26	334.45	308.22	383.56	311.31	335.03
Mean Carbon	556.33	745.38	771.49	831.89	903.52	962.99

carbon is significantly lower in the optimized mixes—often reduced by more than 50%. This confirms that the optimization framework successfully identifies environmentally and economically efficient mix designs, while still meeting the required mechanical performance. For a consistent comparison, the mix designs from the original dataset used to calculate the mean values were limited to those within ±5% of the target compressive strength, in alignment with the selection criteria applied during the optimization process.

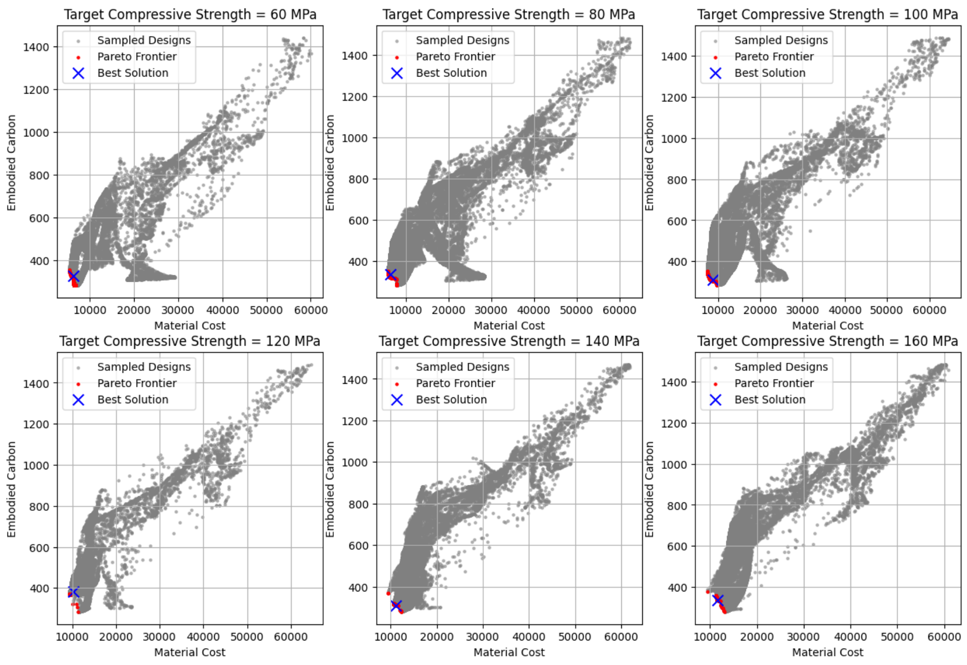


Figure 4.5: Pareto Diagram

Fig 4.5 shows the optimization of cost and embodied carbon is performed for different target compressive strengths. For each target, the Pareto frontiers of cost and embodied carbon are identified. Additionally, the solutions with the highest hypervolume on the Pareto frontiers are selected. The sampled designs represent solutions explored during NSGA-II iterations.

Fig 4.6 illustrates a general inverse relationship between material cost and embedded

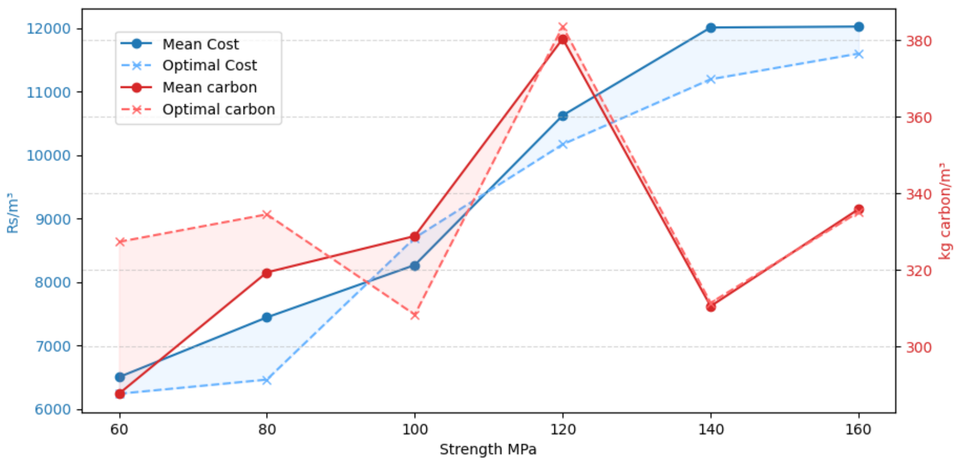


Figure 4.6: Cost Comparison Diagram

carbon—reductions in cost often correspond to increases in carbon emissions, and vice versa. While the optimized mix designs consistently achieve lower material costs compared to the dataset averages, they tend to exhibit slightly higher embedded carbon values at certain strength levels. This indicates that minimizing cost tends to favor mix designs composed of higher quantities of high-carbon materials, whereas minimizing embodied carbon encourages the use of lower-carbon but often more expensive constituents.

4.12 Discussion

The optimized CatBoost model, configured with the hyperparameters outlined in Table 4.2, demonstrated considerable improvements in the testing phase as illustrated in Table 4.3. The decrease in MAE and RMSE, along with the increase in R^2-score, highlights the optimized model's ability to provide more precise predictions with reduced variance when compared to the default configuration.

Fine-tuning critical parameters such as learning rate, depth, and L2 regularization enabled the model to more accurately capture the underlying data patterns while avoiding overfitting in the testing phase.

The 5-fold cross-validation results highlight the model's consistency and reliability across different subsets of data. The slight variations in performance metrics across the folds are typical and indicate the model's robustness in generalizing to unseen data. The overall high R^2-score suggests that the model is well-tuned and capable of providing accurate predictions, making it a valuable tool for the intended application.

Additionally, the SHAP analysis provided insights into feature importance, revealing the primary factors influencing the model's predictions. This interpretability adds value to the predictive model by offering a deeper understanding of how various features impact concrete compressive strength.

In the CVAE architecture designs, single layer architecture shows enough capability to perform the generative tasks. Two-layer architecture show the best performance but at the cost of slight training instability. Three and more layers architectures show increasing instability with worse performance in training and testing losses. This maybe due to deeper models introducing more non-linearity and complex loss surfaces, making it harder for the Adam optimization algorithm to find stable convergence paths.

The integration of FOA optimization and SHAP analysis enhanced the accuracy and interpretability of the CatBoost model for predicting concrete compressive strength. Additionally, the combination of CVAE and NSGA-II yielded optimized mix designs that met the target compressive strength requirements. This approach highlights the effectiveness of model optimization and interpretability techniques, providing valuable tools for predictive modeling and mix design generation.

Chapter 5

Conclusion and Future Work

This study focused on predicting the compressive strength of concrete using the CatBoost algorithm and enhancing its performance through hyperparameter optimization with the Firefly Optimization Algorithm (FOA). Additionally, Conditional Variational Autoencoder (CVAE) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) were employed for mix design generation, ensuring optimized formulations that align with the target compressive strength requirements. The key contributions of this work are summarized as follows:

A comprehensive dataset analysis was conducted to establish a strong foundation for predictive modeling. An initial CatBoost model was developed to predict compressive strength, achieving a baseline R^2-score of 0.95, Mean Absolute Error (MAE) of 5.9288, and Root Mean Square Error (RMSE) of 7.96. To improve predictive performance, FOA was employed to fine-tune key hyperparameters, including learning rate, depth, and L2 regularization. The optimized FOA-CatBoost model demonstrated substantial performance gains, achieving an R^2-score of 0.97, MAE of 3.90, and RMSE of 5.54. This corresponds to a 34.2% reduction in Mean Squared Error (MSE), a 2% increase in the R^2-score, and a 30.41% decrease in RMSE, highlighting the effectiveness of FOA in improving model accuracy.

To ensure model robustness, a regression slope analysis was conducted to compare the baseline and optimized models. Additionally, a 5-fold cross-validation procedure was performed to assess potential overfitting or underfitting. SHapley Additive exPlanations (SHAP) analysis was applied to the optimized FOA-CatBoost model, offering insights into feature importance by quantifying the contribution of each input variable to the prediction of compressive strength. The SHAP analysis identified key material and environmental parameters influencing concrete strength, thereby enhancing model interpretability and reliability.

In addition to predictive modeling, this study explored generative modeling for mix design formulation. Various CVAE architectures were tested to generate feasible concrete mix designs. Training and validation losses were analyzed, and learning curves were examined to evaluate model stability and performance. The CVAE-generated mix designs were further optimized using NSGA-II, which identified optimal formulations balancing multiple objectives, including target compressive strength requirements. The combination of CVAE and NSGA-II proved effective in generating high-quality mix designs, demonstrating the potential of deep generative models and evolutionary algorithms in material design.

This study highlights the significance of data exploration, model optimization, and interpretability in predictive modeling. The integration of CatBoost with FOA for hyperparameter tuning significantly enhanced predictive accuracy, while SHAP analysis provided valuable insights into feature importance. Furthermore, the combination of CVAE and NSGA-II for mix design generation demonstrated an efficient approach to optimizing concrete formulations. Future research could extend this framework to other materials and engineering applications, further validating the utility of machine learning and optimization techniques in complex regression and generative modeling tasks.

While the current study demonstrates the effectiveness of hyperparameter optimization using FOA on the CatBoost model for predicting concrete compressive strength there are several avenues for future research. The mix designs generated have been virtually tested to meet the target compressive strength. However, practical verification would be better.

A key area of future work will involve the practical verification of the predictive model and mix designs through experimental testing. This will be achieved by 3D printing concrete samples based on the predictions of the optimized CatBoost model. By comparing the experimental results to the results obtained in this study, we can further validate the framework in real-world applications.

Through these future efforts, the current research can be extended and applied in practical settings, leading to the development of improved predictive models and innovative material formulations.

Appendix A: List of Abbreviations

Abbreviation	**Definition**
ADA	AdaBoost: An ensemble learning technique based on adaptive boosting.
ANN	Artificial Neural Network: A computing system inspired by the structure of biological neural networks.
ANFIS	Adaptive Neuro-Fuzzy Inference System: A hybrid intelligent system combining neural networks and fuzzy logic.
BO	Bayesian Optimization: A global optimization method based on Bayesian inference.
BR	Bayesian Regularization: A regularization technique used in neural network training.
CatBoost	Categorical Boosting: A gradient boosting algorithm designed for categorical features.
CVAE	Conditional Variational Autoencoder: A variant of VAE that allows conditional generation based on labels.
DNN	Deep Neural Network: A neural network with multiple layers between input and output layers.
DT	Decision Tree: A tree-based machine learning algorithm for classification and regression.
DTR	Decision Tree Regression: A regression model using decision trees.
ECC	Engineered Cementitious Composite: A high-performance fiber-reinforced concrete with strain-hardening behavior.
FA	Firefly Algorithm: A nature-inspired metaheuristic algorithm based on the flashing behavior of fireflies.
FOA	Firefly Optimization Algorithm: A variant of FA used for solving optimization problems.

Abbreviation	**Definition**
GA	Genetic Algorithm: A search heuristic inspired by the process of natural selection.
GB	Gradient Boosting: An ensemble technique using decision trees built sequentially.
GBT	Gradient Boosted Trees: Decision tree ensembles optimized via gradient boosting.
GEP	Gene Expression Programming: An evolutionary algorithm based on genetic programming.
GPR	Gaussian Process Regression: A non-parametric, Bayesian approach to regression.
GWO	Grey Wolf Optimizer: A metaheuristic algorithm inspired by the social behavior of grey wolves.
GUI	Graphical User Interface: A visual interface allowing user interaction with digital systems.
KNN	K-Nearest Neighbors: A non-parametric method used for classification and regression.
LightGBM	Light Gradient Boosting Machine: A fast, efficient gradient boosting framework.
LSSVR	Least Squares Support Vector Regression: A variant of SVR using a least squares loss function.
ML	Machine Learning: A field of artificial intelligence that uses statistical techniques to give computers the ability to learn.
MLR	Multiple Linear Regression: A linear regression model involving multiple predictors.
MNLR	Multinomial Logistic Regression: A regression model used for multi-class classification problems.
MOFA	Multi-objective Firefly Algorithm: An extension of the Firefly Algorithm to handle multiple objectives.
MOO	Multi-Objective Optimization: Optimization involving multiple, often conflicting objectives.
NSGA-II	Non-dominated Sorting Genetic Algorithm II: A popular evolutionary algorithm for multi-objective optimization.

Abbreviation	Definition
PSO	Particle Swarm Optimization: A metaheuristic inspired by the social behavior of birds and fish.
RAC	Recycled Aggregate Concrete: Concrete made using recycled aggregates from demolished structures.
RF	Random Forest: An ensemble learning method using a multitude of decision trees.
RS	Random Search: A parameter search method that samples configurations randomly.
SCM	Supplementary Cementitious Material: Industrial by-products like fly ash or slag used to replace cement in concrete.
SCC	Self-Compacting Concrete: Concrete that can flow and compact under its own weight without vibration.
SSA	Squirrel Search Algorithm: A bio-inspired optimization algorithm based on foraging behavior of squirrels.
SVM	Support Vector Machine: A supervised learning model used for classification and regression.
SVR	Support Vector Regression: A regression model based on SVM principles.
UHPFRC	Ultra-High-Performance Fiber-Reinforced Concrete: A class of concrete with exceptional strength and ductility.
UHPC	Ultra-High-Performance Concrete: High-strength concrete with improved durability and performance.
VAE	Variational Autoencoder: A generative deep learning model for encoding data into a latent space.
XGBoost	Extreme Gradient Boosting: An optimized implementation of gradient boosting decision trees.

Appendix B: Python Code for Comparison of ML Models

In this appendix, the Python code used for comparing various machine learning models as part of the baseline evaluation is presented.

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
↪  AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor

# Step 1: Load dataset
data_filepath = r"Wiley.csv"
dataset = pd.read_csv(data_filepath)

# Clean the dataset by removing unnecessary columns
dataset = dataset.drop(['Index', 'Unnamed: 16'], axis=1)

# Step 2: Prepare the feature matrix (X) and target vector (y)
X_features = dataset.drop('fc (MPa)', axis=1)  # Features (excluding target)
y_target = dataset['fc (MPa)']  # Target variable (compressive strength)

# Split the dataset into training and validation sets (80% train, 20% validation)
X_train, X_valid, y_train, y_valid = train_test_split(X_features, y_target, test_size=0.2,
↪  random_state=42)
```

```python

# Step 3: Define machine learning models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'ElasticNet': ElasticNet(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'AdaBoost': AdaBoostRegressor(),
    'Support Vector Regression (SVR)': SVR(),
    'K-Nearest Neighbors (KNN)': KNeighborsRegressor(),
    'XGBoost': XGBRegressor(verbose=0),
    'LightGBM': LGBMRegressor(verbosity=-1),
    'CatBoost': CatBoostRegressor(verbose=0)
}

# Step 4: Train models, make predictions, and evaluate performance
evaluation_results = []

for model_name, model_instance in models.items():
    # Train the model
    model_instance.fit(X_train, y_train)

    # Predict the target variable for the validation set
    y_pred = model_instance.predict(X_valid)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_valid, y_pred)  # Mean Squared Error
    mae = mean_absolute_error(y_valid, y_pred)  # Mean Absolute Error
    r2 = r2_score(y_valid, y_pred)  # R-squared score
    rmse = np.sqrt(mse)  # Root Mean Squared Error

    # Store the results for each model
    evaluation_results.append({
        'Model': model_name,
        'MAE': mae,
        'R2': r2,
        'RMSE': rmse
    })

```

```python
# Step 5: Display the results in a DataFrame
results_df = pd.DataFrame(evaluation_results)

# Sort the results by R² (higher R² is better)
results_df_sorted = results_df.sort_values(by='R2', ascending=False)

# Print the results
print(results_df_sorted)
```

Appendix C: Python Code for Data Description and Analysis

In this appendix, the Python code used for description and analysis of dataset.

```python
# Data Handling
import pandas as pd
import numpy as np

# Machine Learning & Optimization
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Utility
import math

# Step 1: Load and clean the dataset
data_filepath = r"Wiley.csv"
dataset = pd.read_csv(data_filepath)

# Remove unnecessary columns
dataset_cleaned = dataset.drop(['Index', 'Unnamed: 16'], axis=1)
print(dataset_cleaned.info())

# Step 2: Generate Statistical Description
# Create a copy of the dataset to avoid modifying the original
dataset_copy = dataset_cleaned.copy()

# Generate basic statistics (mean, std, min, max, etc.)
```

```python
stats_description = dataset_copy.describe().T

# Additional statistics: Kurtosis and Skewness
stats_description['Kurtosis'] = dataset_copy.kurtosis()
stats_description['Skewness'] = dataset_copy.skew()

# Display and save the statistical description
print(stats_description)
stats_description.to_csv('stats_description.csv')

# Step 3: Check for unique values in each column
print("Unique values per column:")
print(dataset_copy.nunique())

# Step 4: Plot Histograms for All Columns
# Get the number of numerical columns (features) in the dataset
num_columns = dataset_copy.shape[1]

# Calculate grid size for subplots based on number of columns
rows = math.ceil(num_columns / 4)  # Adjust this number for columns per row
cols = min(num_columns, 4)  # Limit columns per row to 4

# Set dynamic figure size for better visualization
plt.figure(figsize=(cols * 4, rows * 4))

# Loop through the columns to create histograms for each feature
for i, column in enumerate(dataset_copy.columns):
    plt.subplot(rows, cols, i + 1)
    sns.histplot(dataset_copy[column], bins=10, kde=True)
    plt.title(f'Histogram of {column}')

plt.tight_layout()  # Adjust spacing between subplots
plt.show()

# Step 5: Clean up column names
# Rename columns by splitting any space-separated names and taking the first part
dataset_copy.columns = [col.split()[0] for col in dataset_copy.columns]

# Step 6: Calculate Correlation Matrix
correlation_matrix = dataset_copy.corr()

# Create a mask to hide the upper triangle of the correlation matrix
```

```python
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool), k=1)

# Step 7: Plot the Correlation Matrix Heatmap
plt.figure(figsize=(10, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='Reds', fmt='.2f', linewidths=0.5,
            square=True, center=0, mask=mask, cbar_kws={'shrink': 0.65})
plt.title('Correlation Matrix Heatmap')
plt.show()

```

Appendix D: Python Code for Optimizing CatBoost Using Firefly Optimization Algorithm

In this appendix, the Python code used for optimizing CatBoost hyperparameters using Firefly Optimization Algorithm.

```python
import pandas as pd
import numpy as np
import math
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Function to evaluate model performance
def evaluate_model(actual_values, predicted_values):
    """
    Calculate evaluation metrics: MAE, R², and RMSE.
    Args:
    actual_values (array): Actual target values.
    predicted_values (array): Predicted target values.

    Returns:
    dict: Dictionary containing MAE, R², and RMSE.
    """
    mae = mean_absolute_error(actual_values, predicted_values)
    r2 = r2_score(actual_values, predicted_values)
    rmse = math.sqrt(mean_squared_error(actual_values, predicted_values))

    return {"MAE": mae, "R2_score": r2, "RMSE": rmse}

# Step 1: Load and clean the dataset
```

```python
data_filepath = r"Wiley.csv"
dataset = pd.read_csv(data_filepath)

# Remove unnecessary columns
dataset = dataset.drop(['Index', 'Unnamed: 16'], axis=1)

# Step 2: Prepare features and target variable
X = dataset.drop('fc (MPa)', axis=1)  # Features (excluding target)
y = dataset['fc (MPa)']  # Target variable (compressive strength)

# Split into training and validation sets (80% train, 20% validation)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Initialize fireflies for Firefly Algorithm
def initialize_fireflies(population_size, dimension, param_bounds):
    fireflies = np.zeros((population_size, dimension))
    for i in range(population_size):
        for j in range(dimension):
            fireflies[i][j] = np.random.uniform(param_bounds[j][0], param_bounds[j][1])
            # Ensure certain parameters are integers (e.g., depth, iterations)
            if j in [1, 5, 6]:  # Indices for depth, border_count, iterations
                fireflies[i][j] = round(fireflies[i][j])
    return fireflies

# Step 4: Define the objective function (CatBoost model with parameter constraints)
def objective_function(solution):
    # Apply bounds to hyperparameters
    learning_rate = np.clip(solution[0], 0.01, 0.05)
    depth = int(np.clip(solution[1], 4, 10))
    l2_leaf_reg = np.clip(solution[2], 0.01, 100)
    bagging_temperature = np.clip(solution[3], 0, 10)
    subsample = np.clip(solution[4], 0.5, 0.8)
    border_count = int(np.clip(solution[5], 1, 255))
    iterations = int(np.clip(solution[6], 100, 1500))

    # Create the CatBoost model with constrained parameters
    model = CatBoostRegressor(
        learning_rate=learning_rate,
        depth=depth,
        l2_leaf_reg=l2_leaf_reg,
        subsample=subsample,
        border_count=border_count,
        iterations=iterations,
        verbose=0
    )

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Predict on the validation set
    y_pred = model.predict(X_valid)

    # Calculate and return MSE (Objective Value)
    mse = mean_squared_error(y_valid, y_pred)
    return mse

# Step 5: Calculate intensity of fireflies (objective values)
def calculate_intensity(population, objective_func):
    return np.array([objective_func(ind) for ind in population])

# Step 6: Calculate Euclidean distance between two fireflies
def calculate_distance(firefly_i, firefly_j, means, stds):
    firefly_i_scaled = (firefly_i - means) / stds
    firefly_j_scaled = (firefly_j - means) / stds
    return np.linalg.norm(firefly_i_scaled - firefly_j_scaled)

# Step 7: Calculate attractiveness based on distance
def calculate_attractiveness(distance_ij, beta_0, gamma):
    return beta_0 * np.exp(-gamma * distance_ij**2)

# Step 8: Move fireflies based on attractiveness and update their positions
def move_fireflies(population, intensity, max_iter, alpha, beta_0, gamma, param_bounds, means,
↪  stds):
    population_size, dimension = population.shape
    for iter_num in range(max_iter):
        for i in range(population_size):
            for j in range(population_size):
                if intensity[i] > intensity[j]:
                    distance_ij = calculate_distance(population[i], population[j], means,
                    ↪  stds)
                    attractiveness_ij = calculate_attractiveness(distance_ij, beta_0, gamma)
                    random_unit = np.random.rand(dimension)
                    params_scaled = np.array([
                        low + r * (high - low)
                        for r, (low, high) in zip(random_unit, param_bounds)
                    ])
                    population[i] += attractiveness_ij * (population[j] - population[i]) +
                    ↪  alpha * params_scaled

        # Recalculate intensity (objective values)
        intensity = calculate_intensity(population, objective_function)
        save_population_and_intensity_to_csv(population, intensity, iter_num)

    # Return the best solution (lowest intensity)
    best_solution = population[np.argmin(intensity)]
    return best_solution

# Step 9: Save population and intensity to CSV for analysis
def save_population_and_intensity_to_csv(population, intensity, iteration):
    df = pd.DataFrame(population, columns=[f'Param_{i+1}' for i in
    ↪  range(population.shape[1])])
    df['Intensity'] = intensity
    df.to_csv(f'output/fireflies_population_intensity_iteration_{iteration}.csv', index=False)

# Step 10: Firefly Algorithm Parameters and Initialization
param_bounds = [
    [0.01, 0.05],  # Learning rate
    [4, 10],       # Depth
    [0.01, 100],   # L2 leaf regularization
    [0, 10],       # Bagging temperature
    [0.5, 0.8],    # Subsample
    [1, 255],      # Border count
    [100, 1500]    # Iterations
]

population_size = 10
dimension = 7
max_iter = 100
alpha = 0.1
beta_0 = 1.0
gamma = 1.0

means = np.array([(bounds[0] + bounds[1]) / 2 for bounds in param_bounds])
stds = np.array([(bounds[1] - bounds[0]) / 4 for bounds in param_bounds])

# Initialize fireflies
```

```python
fireflies = initialize_fireflies(population_size, dimension, param_bounds)

# Step 11: Calculate initial intensity (objective values) for fireflies
intensity = calculate_intensity(fireflies, objective_function)

# Step 12: Execute the Firefly Algorithm
best_solution = move_fireflies(fireflies, intensity, max_iter, alpha, beta_0, gamma,
↪  param_bounds, means, stds)

# Step 13: Output the best solution and objective value
print("Best Solution:", best_solution)
print("Objective Value:", objective_function(best_solution))

```

Appendix E: Python Code for Analysis of FOA-CatBoost

In this appendix, the Python code used for analysis of FOA-CatBoost.

```python
# Step 1: Import necessary libraries

# Core Libraries
import numpy as np
import pandas as pd

# Machine Learning Models
from catboost import CatBoostRegressor

# Evaluation Metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Data Processing
from sklearn.model_selection import train_test_split

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Statistical Analysis
import shap  # SHAP for feature importance analysis
from scipy.stats import linregress

# Function to calculate performance metrics
def evaluate_model(actual_values, predicted_values):
    """
    Calculate evaluation metrics: MAE, R², and RMSE.
    Args:
    actual_values (array): Actual target values.
    predicted_values (array): Predicted target values.

    Returns:
    dict: Dictionary containing MAE, R², and RMSE.
    """
    mae = mean_absolute_error(actual_values, predicted_values)
    r2 = r2_score(actual_values, predicted_values)
    rmse = np.sqrt(mean_squared_error(actual_values, predicted_values))
    return {"MAE": mae, "R2_score": r2, "RMSE": rmse}

# Step 2: Load and clean dataset
data_filepath = r"Wiley.csv"
dataset = pd.read_csv(data_filepath)

# Remove unnecessary columns (Index and Unnamed: 16)
dataset = dataset.drop(['Index', 'Unnamed: 16'], axis=1)

# Prepare features and target variable
X = dataset.drop('fc (MPa)', axis=1)  # Features (excluding target)
y = dataset['fc (MPa)']  # Target variable (compressive strength)

# Split dataset into training and validation sets (80% train, 20% validation)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create and train an initial CatBoost model with basic parameters
initial_model = CatBoostRegressor(verbose=0)
initial_model.fit(X_train, y_train)

# Predict on the training and validation sets
y_train_pred_initial = initial_model.predict(X_train)
y_valid_pred_initial = initial_model.predict(X_valid)

# Evaluate performance of the initial model
print("Initial Model Performance:")
print("Training Set Performance:", evaluate_model(y_train, y_train_pred_initial))
print("Validation Set Performance:", evaluate_model(y_valid, y_valid_pred_initial))

# Step 4: Create and train the optimized CatBoost model
# Hyperparameter values are obtained by Firefly Optimization
optimized_model = CatBoostRegressor(
    learning_rate=0.2287,
    depth=6,
    l2_leaf_reg=35.0179,
    bagging_temperature=5.7426,
    subsample=0.6313,
    border_count=146,
    iterations=2117,
    verbose=0
)

# Train the optimized model
optimized_model.fit(X_train, y_train)

# Predict on the training and validation sets
y_train_pred_optimized = optimized_model.predict(X_train)
y_valid_pred_optimized = optimized_model.predict(X_valid)

# Evaluate performance of the optimized model
print("Optimized Model Performance:")
print("Training Set Performance:", evaluate_model(y_train, y_train_pred_optimized))
print("Validation Set Performance:", evaluate_model(y_valid, y_valid_pred_optimized))

# Step 5: Linear Regression Analysis for CatBoost on Training and Validation Sets
catboost_slope_train, catboost_intercept_train, _, _, _ = linregress(y_train,
↪  y_train_pred_initial)
catboost_slope_valid, catboost_intercept_valid, _, _, _ = linregress(y_valid,
↪  y_valid_pred_initial)
r2_catboost_train = r2_score(y_train, y_train_pred_initial)
r2_catboost_valid = r2_score(y_valid, y_valid_pred_initial)

# Step 6: Plot the Regression for CatBoost Model
plt.figure(figsize=(8, 6))
sns.regplot(x=y_train, y=y_train_pred_initial, ci=None, line_kws={"color": "orange"},
            label=f"CatBoost Train: y = {catboost_slope_train:.3f}x +
            ↪  {catboost_intercept_train:.3f}, R² = {r2_catboost_train:.2f}")
sns.regplot(x=y_valid, y=y_valid_pred_initial, ci=None, line_kws={"color": "red"},
            label=f"CatBoost Valid: y = {catboost_slope_valid:.3f}x +
            ↪  {catboost_intercept_valid:.3f}, R² = {r2_catboost_valid:.2f}")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Regression Analysis for CatBoost Model")
plt.legend()
plt.grid(True)
plt.show()
```

```

# Step 7: Linear Regression Analysis for Optimized FOA-CatBoost Model
foa_catboost_slope_train, foa_catboost_intercept_train, _, _, _ = linregress(y_train,
↪  y_train_pred_optimized)
foa_catboost_slope_valid, foa_catboost_intercept_valid, _, _, _ = linregress(y_valid,
↪  y_valid_pred_optimized)
r2_foa_catboost_train = r2_score(y_train, y_train_pred_optimized)
r2_foa_catboost_valid = r2_score(y_valid, y_valid_pred_optimized)

# Step 8: Plot the Regression for FOA-CatBoost Model
plt.figure(figsize=(8, 6))
sns.regplot(x=y_train, y=y_train_pred_optimized, ci=None, line_kws={"color": "blue"},
            label=f"FOA-CatBoost Train: y = {foa_catboost_slope_train:.3f}x +
            ↪  {foa_catboost_intercept_train:.3f}, R² = {r2_foa_catboost_train:.2f}")
sns.regplot(x=y_valid, y=y_valid_pred_optimized, ci=None, line_kws={"color": "darkblue"},
            label=f"FOA-CatBoost Valid: y = {foa_catboost_slope_valid:.3f}x +
            ↪  {foa_catboost_intercept_valid:.3f}, R² = {r2_foa_catboost_valid:.2f}")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Regression Analysis for FOA-CatBoost Model")
plt.legend()
plt.grid(True)
plt.show()

# Step 9: SHAP Feature Importance Analysis
shap.initjs()
explainer = shap.Explainer(optimized_model)
shap_values = explainer(X_valid)

# Plot SHAP feature importance
shap.plots.bar(shap_values)
shap.plots.beeswarm(shap_values)

```

Appendix F: Python Code for CVAE Training

This appendix presents the Python code used for training the Conditional Variational Autoencoder (CVAE).

```python
# Data Handling
import pandas as pd
import numpy as np

# Machine Learning & Optimization
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Evaluation Metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Utility
import math
import csv
import os

# Step 1: Load and clean the dataset
data_filepath = r"Wiley.csv"
dataset = pd.read_csv(data_filepath)

```

```python
# Drop unnecessary columns
dataset_cleaned = dataset.drop(['Index', 'Unnamed: 16'], axis=1)
print(dataset_cleaned.info())

# Separate input features (X) and output feature (Y)
X = dataset_cleaned.drop(columns=['fc (MPa)']).values  # All features except target
Y = dataset_cleaned[['fc (MPa)']].values  # Target variable

# Step 2: Check for Min-Max Conditions (ensure proper scaling of test data)
def check_min_max_conditions(train_data, test_data):
    """
    Ensures that the test data is within the min-max range of the training data.
    Args:
        train_data (array): Training data features.
        test_data (array): Test data features.
    Returns:
        bool: True if test data min/max is within train data min/max, else False.
    """
    min_train, max_train = train_data.min(axis=0), train_data.max(axis=0)
    min_test, max_test = test_data.min(axis=0), test_data.max(axis=0)
    return np.all(min_train <= min_test) and np.all(max_train >= max_test)

# Attempt to split the data up to 10 times for valid train-test split
for attempt in range(10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42 +
    ↪  attempt)

    if check_min_max_conditions(X_train, X_test) and check_min_max_conditions(Y_train,
    ↪  Y_test):
        print(f"Valid split found on attempt {attempt + 1}")
        break
else:
    print("Failed to find a valid split after 10 attempts.")

# Step 3: Normalize data using Min-Max Scaler
scaler_X = MinMaxScaler()
scaler_Y = MinMaxScaler()

# Fit and transform training data, transform test data
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)  # Only transform the test data
Y_train_scaled = scaler_Y.fit_transform(Y_train)
```

```
Y_test_scaled = scaler_Y.transform(Y_test)  # Only transform the test data

# Convert to PyTorch tensors for model input
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test_scaled, dtype=torch.float32)

# Step 4: Create DataLoader for batching
batch_size = 32
train_loader = DataLoader(TensorDataset(X_train_tensor, Y_train_tensor),
↪  batch_size=batch_size, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test_tensor, Y_test_tensor), batch_size=batch_size,
↪  shuffle=False)

# Step 5: Define the CVAE model
class CVAE(nn.Module):
    def __init__(self, input_dim, latent_dim, encoder_layers, decoder_layers):
        super(CVAE, self).__init__()

        # Encoder: Learns (z | X)
        encoder_sequential_layers = []
        in_dim = input_dim
        for out_dim in encoder_layers:
            encoder_sequential_layers.append(nn.Linear(in_dim, out_dim))
            encoder_sequential_layers.append(nn.ReLU())
            in_dim = out_dim

        self.encoder = nn.Sequential(*encoder_sequential_layers)

        self.mu = nn.Linear(encoder_layers[-1], latent_dim)
        self.logvar = nn.Linear(encoder_layers[-1], latent_dim)

        # Decoder: Learns (X' | z)
        decoder_sequential_layers = []
        in_dim = latent_dim
        for out_dim in decoder_layers:
            decoder_sequential_layers.append(nn.Linear(in_dim, out_dim))
            decoder_sequential_layers.append(nn.ReLU())
            in_dim = out_dim

        # Final layer outputs to the original input dimension
```

```python
        decoder_sequential_layers.append(nn.Linear(in_dim, input_dim))
        decoder_sequential_layers.append(nn.Sigmoid())  # If output is scaled between 0 and 1

        self.decoder = nn.Sequential(*decoder_sequential_layers)

    def encode(self, x):
        h = self.encoder(x)
        return self.mu(h), self.logvar(h)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std  # Reparameterization trick

    def decode(self, z):
        return self.decoder(z)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

# Step 6: Define the loss function (Reconstruction Loss + KL Divergence + Material Costs +
↪  Embedded Carbon)
def loss_function(recon_x, x, mu, logvar, c1=1, c2=0.001):
    # Reconstruction Loss (MSE)
    MSE = nn.functional.mse_loss(recon_x, x, reduction='sum')

    # KL Divergence Loss
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    # Calculate additional losses (Material Costs and Embodied Carbon Losses)
    material_costs_loss = torch.sum(material_costs * recon_x) - torch.sum(material_costs * x)
    embodied_carbon_loss = torch.sum(embedded_carbon * recon_x) - torch.sum(embedded_carbon *
    ↪  x)

    # Total loss
    total_loss = MSE + c1 * material_costs_loss + c2 * embodied_carbon_loss + KLD
    return total_loss

# Step 7: Train the model and save the results
def train_and_save_model(model, train_loader, test_loader, epochs=5000, lr=1e-3):
    optimizer = optim.Adam(model.parameters(), lr=lr)

    loss_log_file = "training_log.csv"
    with open(loss_log_file, mode="w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["Epoch", "Train Loss", "Test Loss"])

    for epoch in range(epochs):
        model.train()
        train_loss = 0
        for x_batch, y_batch in train_loader:
            optimizer.zero_grad()
            recon_x, mu, logvar = model(x_batch)
            loss = loss_function(recon_x, x_batch, mu, logvar)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        # Evaluation on test set
        model.eval()
        test_loss = 0
        with torch.no_grad():
            for x_batch, y_batch in test_loader:
                recon_x, mu, logvar = model(x_batch)
                loss = loss_function(recon_x, x_batch, mu, logvar)
                test_loss += loss.item()

        # Save losses to CSV
        with open(loss_log_file, mode="a", newline="") as file:
            writer = csv.writer(file)
            writer.writerow([epoch, train_loss / len(train_loader.dataset), test_loss /
            ↪  len(test_loader.dataset)])

    # Save the trained model
    model_path = "cvae_model.pth"
    torch.save(model.state_dict(), model_path)
    print(f"Model saved to {model_path}")

# Initialize the CVAE model and train
input_dim = X.shape[1]
latent_dim = 2  # Size of latent space
encoder_layers = [128, 64, 32]  # Example encoder architecture
```

```
decoder_layers = [64, 128]  # Example decoder architecture

cvae_model = CVAE(input_dim, latent_dim, encoder_layers, decoder_layers)
train_and_save_model(cvae_model, train_loader, test_loader)


```

Appendix G: Python Code for Comparison of CVAE Architectures using Losses

In this appendix, the Python code used for comparison of losses of various CVAE architectures.

```
import os
import pandas as pd

# Folder containing CSV files
folder_path = "output_vae_full_weight"

validation_errors = {"file name":[], "train loss": [], "test loss": []}

# List all output files
files = [f for f in os.listdir(folder_path) if f.startswith("output_file_vae") and
↪  f.endswith(".csv")]
sorted_files = []
for file in files:
    temp = file.split("batch")[0].split("Encoder")[1].split("_")
    temp = [i for i in temp if i!='']
    sorted_files.append((file,temp))

sorted_files = sorted(sorted_files, key = lambda x: (len(x[1]),int(x[1][0])))
# Loop through all files in the folder
for file,encoder_layers in sorted_files:
    file_path = os.path.join(folder_path, file)
    df = pd.read_csv(file_path)
    encoder_design  ="Encoder: (" + ",".join(encoder_layers) +")"
    decoder_design = "Decoder: (" + ",".join(encoder_layers[::-1])+")"
    architecture = encoder_design+" "+decoder_design
    validation_errors["file name"].append(architecture)
    validation_errors['train loss'].append(df.iloc[-1,1])
    validation_errors['test loss'].append(df.iloc[-1,2])

validation_errors_df = pd.DataFrame(validation_errors)
print(validation_errors_df)

```

Appendix H: Python Code for Comparison of CVAE Architectures using Learning Curve

In this appendix, the Python code used for comparison of various CVAE architectures using learning curves.

```python
import os
import pandas as pd
import matplotlib.pyplot as plt
import math

# Data path where output files are stored
data_path = "output_vae_full_weight"

# List all output files
files = [f for f in os.listdir(data_path) if f.startswith("output_file_vae") and
↪  f.endswith(".csv")]
sorted_files = []
for file in files:
    temp = file.split("batch")[0].split("Encoder")[1].split("_")
    temp = [i for i in temp if i!='']
    sorted_files.append((file, temp))

# Sort by the length of the second tuple and then lexicographically
sorted_files = sorted(sorted_files, key=lambda x: (len(x[1]), int(x[1][0])))

# Determine dynamic grid size
num_plots = min(len(sorted_files), 11)  # Limit to 11 plots if more exist
rows = math.ceil(math.sqrt(num_plots))  # Adjust for square layout
cols = math.ceil(num_plots / rows)  # Ensure all plots fit

```

```
fig, axes = plt.subplots(rows, cols, figsize=(15, 15))
axes = axes.flatten()  # Convert to 1D array for easy indexing

for idx, (file, encoder_structure) in enumerate(sorted_files[:num_plots]):
    file_path = os.path.join(data_path, file)
    encoder_design = "Encoder: (" + ",".join(map(str, encoder_structure)) + ")"
    decoder_design = "Decoder: (" + ",".join(map(str, encoder_structure[::-1])) + ")"
    architecture = encoder_design + " " + decoder_design

    try:
        # Load loss log
        df = pd.read_csv(file_path)

        # Ensure required columns exist
        if {"Epoch", "Train Loss", "Test Loss"}.issubset(df.columns):
            axes[idx].plot(df["Epoch"], df["Train Loss"], label="Train Loss", marker="o",
            ↪  linestyle="-", color="blue", markersize=0.5, alpha=0.5)
            axes[idx].plot(df["Epoch"], df["Test Loss"], label="Test Loss", marker=".",
            ↪  linestyle="--", color="red", markersize=0.1, alpha=0.5)

            # Graph labels
            axes[idx].set_xlabel("Epochs")
            axes[idx].set_ylabel("Loss")
            axes[idx].set_title(f"Model No. {idx+1}\n{architecture}", fontsize=10)
            axes[idx].legend()
            axes[idx].grid(True)
        else:
            axes[idx].set_title("Missing Columns")
            axes[idx].axis("off")

    except Exception as e:
        axes[idx].set_title("Error Loading File")
        axes[idx].axis("off")

# Hide any extra unused subplots
for ax in axes[num_plots:]:
    ax.axis("off")

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()

```

Appendix I: Python Code for Mix Design Generation and Optimization

This appendix presents the Python code used for generating and optimizing mix designs.

```python
# PyTorch and Data Handling Libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Scikit-learn for data preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# For data analysis and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# CatBoost model
import catboost

# Multi-Objective Optimization
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.core.problem import Problem
from pymoo.optimize import minimize
from pymoo.visualization.scatter import Scatter
from pymoo.indicators.hv import HV  # Correct import for hypervolume

# Function for checking the validity of min-max scaling for both train and test sets
def min_max_check(train_data, test_data):
    min_train, max_train = train_data.min(axis=0), train_data.max(axis=0)
    min_test, max_test = test_data.min(axis=0), test_data.max(axis=0)
    return np.all(min_train <= min_test) and np.all(max_train >= max_test)

# Load dataset and drop irrelevant columns
data_filepath = r"Wiley.csv"
df = pd.read_csv(data_filepath).drop(['Index', 'Unnamed: 16'], axis=1)

# Separate input features (X) and target variable (Y)
X = df.drop(columns=['fc (MPa)']).values
Y = df[['fc (MPa)']].values

# Try splitting the dataset up to 10 times to ensure valid train-test split
for attempt in range(10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42 +
    ↪  attempt)

    if min_max_check(X_train, X_test) and min_max_check(Y_train, Y_test):
        print(f"Valid split found on attempt {attempt + 1}")
        break
else:
    print("Failed to find a valid split after 10 attempts.")

# Report min-max values for train and test sets
train_min_max_X = pd.DataFrame({'Min': X_train.min(axis=0), 'Max': X_train.max(axis=0)})
test_min_max_X = pd.DataFrame({'Min': X_test.min(axis=0), 'Max': X_test.max(axis=0)})
train_min_max_Y = pd.DataFrame({'Min': Y_train.min(axis=0), 'Max': Y_train.max(axis=0)})
test_min_max_Y = pd.DataFrame({'Min': Y_test.min(axis=0), 'Max': Y_test.max(axis=0)})

print("\nTrain set min-max (X):\n", train_min_max_X)
print("\nTest set min-max (X):\n", test_min_max_X)
print("\nTrain set min-max (Y):\n", train_min_max_Y)
print("\nTest set min-max (Y):\n", test_min_max_Y)

# Initialize MinMaxScaler and scale both training and test sets
scaler_X = MinMaxScaler()
scaler_Y = MinMaxScaler()

# Fit scaler only on training data and apply to both train and test data
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
Y_train_scaled = scaler_Y.fit_transform(Y_train)
Y_test_scaled = scaler_Y.transform(Y_test)
```

```

# Convert the scaled data into PyTorch tensors
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test_scaled, dtype=torch.float32)

# Create DataLoader instances for batch processing
batch_size = 32
train_loader = DataLoader(TensorDataset(X_train_tensor, Y_train_tensor),
↪  batch_size=batch_size, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test_tensor, Y_test_tensor), batch_size=batch_size,
↪  shuffle=False)


# Define the Conditional Variational Autoencoder (CVAE) model
class CVAE(nn.Module):
    def __init__(self, input_dim, cond_dim, latent_dim=2):
        super(CVAE, self).__init__()

        # Encoder: Learns (z | X, Y)
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + cond_dim, 256),  # Concatenate input features with condition
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU()
        )
        self.mu = nn.Linear(128, latent_dim)  # Mean of the latent space
        self.logvar = nn.Linear(128, latent_dim)  # Log variance of the latent space

        # Decoder: Learns (X' | z, Y)
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + cond_dim, 128),  # Concatenate latent vector with condition
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, input_dim),  # Output dimension matches the input dimension
            nn.Sigmoid()  # Sigmoid for [0,1] range output
        )

    def encode(self, x, y):
        """Concatenates the input features and condition, then passes through the encoder."""
        inputs = torch.cat((x, y), dim=1)  # Concatenate input features with condition
        h = self.encoder(inputs)
        return self.mu(h), self.logvar(h)

    def reparameterize(self, mu, logvar):
        """Reparameterization trick to sample from the latent space."""
        std = torch.exp(0.5 * logvar)  # Compute the standard deviation
        eps = torch.randn_like(std)  # Generate random noise
        return mu + eps * std  # Sample from the latent space using reparameterization trick

    def decode(self, z, y):
        """Decodes the latent variable z and condition y back to the original input space."""
        inputs = torch.cat((z, y), dim=1)  # Concatenate latent space with condition
        return self.decoder(inputs)

    def forward(self, x, y):
        """Forward pass through the model: encode, reparameterize, and decode."""
        mu, logvar = self.encode(x, y)  # Get mean and log variance from the encoder
        z = self.reparameterize(mu, logvar)  # Sample from the latent space
        return self.decode(z, y), mu, logvar  # Decode the sample back to input space


# Define the loss function: Reconstruction Loss (MSE) + KL Divergence
def loss_function(recon_x, x, mu, logvar, c1=0.1, c2=0.001):
    """
    Computes the total loss consisting of:
    - MSE (reconstruction error)
    - KL divergence (regularization)
    - Material cost loss
    - Embedded carbon loss
    """

    # 1. Reconstruction loss (Mean Squared Error)
    mse_loss = nn.functional.mse_loss(recon_x, x, reduction='sum')  # MSE loss for
    ↪  reconstruction

    # 2. KL Divergence (Regularization term)
    kl_divergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())  # Standard form
    ↪  of KL divergence

    # 3. Denormalize inputs and reconstructions for material cost and carbon loss calculations
    X_denormalized = torch.tensor(scaler_X.inverse_transform(x.detach().numpy()),
    ↪  dtype=torch.float32)
    recon_X_denormalized = torch.tensor(scaler_X.inverse_transform(recon_x.detach().numpy()),
    ↪  dtype=torch.float32)

    # Convert material cost and embodied carbon to PyTorch tensors
    material_costs_tensor = torch.tensor(material_costs, dtype=torch.float32)
    embedded_carbon_tensor = torch.tensor(embedded_carbon, dtype=torch.float32)

    # 4. Compute additional losses: Material cost and Carbon loss
    material_cost_loss = torch.sum(material_costs_tensor * recon_X_denormalized) -
    ↪  torch.sum(material_costs_tensor * X_denormalized)
    embedded_carbon_loss = torch.sum(embedded_carbon_tensor * recon_X_denormalized) -
    ↪  torch.sum(embedded_carbon_tensor * X_denormalized)

    # 5. Total loss: MSE + KL Divergence + additional losses
    vae_loss = mse_loss + 0.001 * kl_divergence  # Combine MSE and KL divergence

    # 6. Return the total loss with additional terms for material cost and carbon loss
    return vae_loss + c1 * material_cost_loss + c2 * embedded_carbon_loss

class CustomProblem(Problem):
    def __init__(self, params):
        """
        Initializes the custom problem with required parameters for optimization.

        params:
            params (list): A list of parameters containing material costs, carbon values,
                           scalers, compressive strength, CVAE model, CatBoost model,
                           graph points, and input bounds.
        """
        # Unpack parameters
        self.material_costs = params[0]
        self.embedded_carbon = params[1]
        self.scaler_X = params[2]
        self.scaler_Y = params[3]
        self.compressive_strength = params[4]
        self.cvae_model = params[5]
        self.catboost_model = params[6]
        self.graph_points = params[7]
        self.input_bounds = params[8]

        # Initialize lower and upper bounds from input_bounds
        self.lower_bounds = np.array([b[0] for b in self.input_bounds])
        self.upper_bounds = np.array([b[1] for b in self.input_bounds])

        # Initialize the Problem
        super().__init__(
            n_var=2,  # Number of decision variables (latent space)
            n_obj=2,  # Number of objectives (material cost and embedded carbon)
            n_constr=1,  # Number of constraints
            xl=np.array([-3, -3]),  # Lower bounds for latent space
            xu=np.array([3, 3])  # Upper bounds for latent space
        )

    def _evaluate(self, z, out, *args, **kwargs):
        """
        Evaluates the fitness of each solution in the population.

        z (numpy.ndarray): The population of solutions.
        out (dict): A dictionary to store the objective values (F) and constraints (G).
        """
        pop_size = z.shape[0]

        # Normalize the compressive strength for the population
        normalized_strength = self._normalize_compressive_strength(pop_size)

        # Convert solutions (z) and normalized compressive strength (y) to tensors
        z_tensor = torch.tensor(z, dtype=torch.float32)
        y_tensor = torch.tensor(normalized_strength, dtype=torch.float32)

        # Decode the solutions to obtain the feature values
        decoded_output = self._decode_latent_space(z_tensor, y_tensor)

        # Denormalize the decoded feature values
        X_denormalized = self._denormalize_features(decoded_output)

        # Enforce the variable-wise bounds on the solutions
        X_clipped = np.clip(X_denormalized, self.lower_bounds, self.upper_bounds)

        # Compute the objectives: material cost and embedded carbon
        f1, f2 = self._compute_objectives(X_clipped)

        # Store the objective values and constraints
```

```python
        out["F"] = np.column_stack([f1, f2])
        out["G"] = self._compute_constraints(X_clipped)

        # Append the results to graph points for visualization
        self.graph_points.append([f1, f2])

    def _normalize_compressive_strength(self, pop_size):
        """
        Normalizes the compressive strength for the population based on the scaler.

        pop_size (int): The size of the population.
        """
        np_compressive_strength = np.full((pop_size, 1), self.compressive_strength)
        return self.scaler_Y.transform(np_compressive_strength)

    def _decode_latent_space(self, z_tensor, y_tensor):
        """
        Decodes the latent space (z) using the CVAE model and the normalized compressive
        ↪  strength.

        z_tensor (torch.Tensor): The latent space solutions.
        y_tensor (torch.Tensor): The normalized compressive strength values.
        """
        return self.cvae_model.decode(z_tensor, y_tensor)

    def _denormalize_features(self, decoded_output):
        """
        Denormalizes the decoded feature values using the scaler.

        decoded_output (torch.Tensor): The decoded output features.
        """
        return self.scaler_X.inverse_transform(decoded_output.detach().numpy())

    def _compute_objectives(self, X_clipped):
        """
        Computes the objectives: material cost and embedded carbon.

        X_clipped (numpy.ndarray): The decoded and clipped feature values.
        """
        f1 = np.sum(self.material_costs * X_clipped, axis=1)  # Material cost objective
        f2 = np.sum(self.embedded_carbon * X_clipped, axis=1)  # Embedded carbon objective
        return f1, f2

    def _compute_constraints(self, X_clipped):
        """
        Computes the constraints: the difference between predicted and target compressive
        ↪  strength.

        X_clipped (numpy.ndarray): The decoded and clipped feature values.
        """
        g1 = np.abs(self.catboost_model.predict(X_clipped) - self.compressive_strength) - 0.05
        ↪  * self.compressive_strength
        return g1

# Function to calculate Pareto front and sort based on hypervolume
def pareto_sorting(pareto_solutions, comp_strength):
    comp_strength = float(comp_strength)
    # Extracting the objectives (material_cost, embedded_carbon) and compressive strength
    pareto_objectives = pareto_solutions.iloc[:, [-2, -1]].values
    compressive_strengths = pareto_solutions.iloc[:,-3].values

    # Step 1: Find the Pareto front (non-dominated solutions) and filter by compressive
    ↪  strength
    pareto_front = []
    for i in range(len(pareto_objectives)):
        is_dominated = False
        # Check if solution is within 5% of desired compressive strength
        if abs(compressive_strengths[i] - comp_strength) / comp_strength > 0.05:
            continue  # Skip if not within 5% of the target compressive strength

        for j in range(len(pareto_objectives)):
            if i != j:
                # Check if solution i is dominated by solution j
                if np.all(pareto_objectives[j] <= pareto_objectives[i]) and
                ↪  np.any(pareto_objectives[j] < pareto_objectives[i]):
                    is_dominated = True
                    break
        if not is_dominated:
            pareto_front.append(i)

    if not pareto_front:
        print(f"No non-dominated solutions found for compressive strength {comp_strength}
        ↪  MPa")
        return None, None, None, None


    # Get the Pareto front solutions and their corresponding objectives
    pareto_front_solutions = pareto_solutions.iloc[pareto_front]
    pareto_front_objectives = pareto_objectives[pareto_front]

    # Step 2: Calculate hypervolume for each solution in the Pareto front
    ref_point = np.max(pareto_front_objectives, axis=0) + 1  # Reference point (worse than any
    ↪  objective)

    # Create hypervolume indicator using the reference point
    hv_indicator = HV(ref_point=ref_point)

    # Calculate the hypervolume for each solution in the Pareto front
    hv_values = np.array([hv_indicator.do(np.array([f])) for f in pareto_front_objectives])

    # Step 3: Sort solutions by hypervolume in descending order
    sorted_indices = np.argsort(hv_values)[::-1]
    sorted_pareto_front_solutions = pareto_front_solutions.iloc[sorted_indices]

    # Step 4: Select the best solution with the highest hypervolume (considering material cost
    ↪  and carbon)
    best_solution = sorted_pareto_front_solutions.iloc[0]
    best_cost = np.sum(best_solution.iloc[:-3] * material_costs)
    best_carbon = np.sum(best_solution.iloc[:-3] * embedded_carbon)

    # Return the sorted Pareto front and the best solution
    return sorted_pareto_front_solutions, best_solution, best_cost, best_carbon

# Initialize the model, optimizer, and loss function
input_dim = X_train.shape[1]  # Number of input features from the training set
cond_dim = 1  # 'fc (MPa)' is the condition (output variable)
latent_dim = 2  # Size of latent space (can be adjusted)

# Initialize the Conditional Variational Autoencoder (CVAE) model
model = CVAE(input_dim, cond_dim, latent_dim)

# Set optimizer for the model using Adam optimizer
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Load the pre-trained CatBoost model
catboost_model = catboost.CatBoostRegressor()
```

```
catboost_model.load_model("best_catboost_model.cbm")  # Update with the actual path if
↪  necessary

# Define the cost and embodied carbon data (make sure these are correctly sourced and aligned
↪  with your dataset)
material_costs = np.array([6.5, 45.7, 2.3, 2, 3, 2.5, 250, 135, 1.6, 0.8, 0, 240, 0, 0, 0])  #
↪  Cost per kg for different materials
embedded_carbon = np.array([0.84, 0.03, 0.08, 0.004, 0.023, 0.02, 5.0, 2.50, 0.007, 0.016,
↪  0.0002, 0.94, 0, 0, 0])  # Embodied carbon for each material

# Population size for multi-objective optimization (NSGA-II)
pop_size = 1000


epochs = 5000
for epoch in range(epochs):
    model.train()
    for x_batch, y_batch in train_loader:
        optimizer.zero_grad()

        # Forward pass
        recon_x, mu, logvar = model(x_batch, y_batch)

        # Calculate losses
        loss = loss_function(recon_x, x_batch, mu, logvar)

        total_loss = loss.requires_grad_()
        total_loss.backward(retain_graph=True)
        optimizer.step()

    # Evaluation on test set
    model.eval()
    with torch.no_grad():
        for x_batch, y_batch in test_loader:
            recon_x, mu, logvar = model(x_batch, y_batch)
            vae_loss, material_costs_loss, embedded_carbon_loss = loss_function(recon_x,
            ↪  x_batch, mu, logvar)

# Given compressive strengths and subplots
compressive_strengths = [60, 80, 100, 120, 140, 160]
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

```

```python
costs_data = {}

# Define the columns for the results
results_columns = [
    "Cement (kg/m³)", "SF (kg/m³)", "BFS (kg/m³)", "FA (kg/m³)", "QP (kg/m³)",
    "LSP (kg/m³)", "NS (kg/m³)", "Fiber (kg/m³)", "Sand (kg/m³)", "Gravel (kg/m³)",
    "Water (kg/m³)", "SP (kg/m³)", "T (°C)", "RH (%)", "Age (days)", "Strength (MPa)",
    "Material Cost (Rs.)", "Carbon (kg CO2/kg)"
]

# Iterate over compressive strengths
for idx, comp_strength in enumerate(compressive_strengths):
    row, col = divmod(idx, 3)  # Get subplot row and column indices
    ax = axes[row, col]  # Select the correct subplot

    input_bounds = [
    (300, 700),   # Cement (kg/m³)
    (0, 200),     # SF (kg/m³)
    (0, 300),     # BFS (kg/m³)
    (0, 300),     # FA (kg/m³)
    (0, 1000),    # QP (kg/m³)
    (0, 500),     # LSP (kg/m³)
    (500, 1200),  # NS (kg/m³)
    (0, 50),      # Fiber (kg/m³)
    (400, 1000),  # Sand (kg/m³)
    (600, 1200),  # Gravel (kg/m³)
    (150, 250),   # Water (kg/m³)
    (0, 50),      # SP (kg/m³)
    (10, 40),     # T (°C)
    (30, 90),     # RH (%)
    (1, 365)      # Age (days)
    ]

    graphs_points = []
    parameters = [
        material_costs,
        embedded_carbon,
        scaler_X,
        scaler_Y,
        comp_strength,
        model,
        catboost_model,
        graphs_points,
        input_bounds
    ]

    problem = CustomProblem(parameters)
    algorithm = NSGA2(pop_size=pop_size)

    res = minimize(
        problem,
        algorithm,
        ('n_gen', 50),
        seed=1,
        verbose=False,
    )

    feasible_mask = np.all(res.G <= 0, axis=1)
    feasible_solutions = res.X[feasible_mask]
    feasible_objectives = res.F[feasible_mask]

    # Prepare results for the best solution
    np_compressive_strength = np.full((len(feasible_solutions), 1), comp_strength)
    Y_normalized = scaler_Y.transform(np_compressive_strength)
    z_tensor = torch.tensor(feasible_solutions, dtype=torch.float32)
    y_tensor = torch.tensor(Y_normalized, dtype=torch.float32)
    decoded_output = model.decode(z_tensor, y_tensor)
    X_denormalized = scaler_X.inverse_transform(decoded_output.detach().numpy())
    compressive_strength_predictions = catboost_model.predict(X_denormalized)

    # Combine the decoded solutions and the compressive strength predictions
    solutions = np.column_stack((X_denormalized, compressive_strength_predictions))

    # Append the objective values (material_cost and embedded_carbon) to the solutions
    solutions = np.column_stack((solutions, feasible_objectives))
    pareto_solutions = pd.DataFrame(solutions)
    pareto_solutions.to_csv(f"output_vae_full_weight/solutions_c1={c1}, c2={c2},
    ↪  CS={comp_strength}")


    # Prepare results for the best solution
    sorted_pareto_front, best_solution, best_cost, best_carbon =
    ↪  pareto_sorting(pareto_solutions,comp_strength)
    if sorted_pareto_front is None:
        continue  # Skip to the next iteration if no valid solutions were found
    best_solution_values = best_solution.iloc[:-3].values
    best_solution_values = np.append(best_solution_values, [best_solution.iloc[-3], best_cost,
    ↪  best_carbon])

    # Print the best solution in the required format
    print(f"\nBest Solution {comp_strength} MPa:")
    solution_df = pd.DataFrame([best_solution_values], columns=results_columns)
    print(solution_df.to_string(index=False))

    # Plotting data for the current compressive strength
    # Plotting data for the current compressive strength
    graph_points_array = np.array(graphs_points)
    vae_costs = graph_points_array[:, 0, :].flatten()
    vae_carbon = graph_points_array[:, 1, :].flatten()

    points = np.column_stack((vae_costs, vae_carbon))
    points = points[points[:, 0].argsort()]

    pareto_front = []
    current_min_carbon = np.inf
    for cost, carbon in points:
        if carbon < current_min_carbon:
            pareto_front.append([cost, carbon])
            current_min_carbon = carbon

    pareto_front = np.array(pareto_front)

    # Store the results in the correct format for material cost and carbon
    costs_data[str(comp_strength)] = [pareto_solutions.iloc[:, -2].mean(),
    ↪  pareto_solutions.iloc[:, -1].mean()]
    costs_data[str(comp_strength)].append(best_cost)
    costs_data[str(comp_strength)].append(best_carbon)

    ax.scatter(vae_costs, vae_carbon, color="gray", alpha=0.5, s=5, label="Sampled Designs")
    ax.scatter(pareto_front[:, 0], pareto_front[:, 1], color="red", edgecolor="red",
    ↪  label="Pareto Frontier", s=5)
    ax.scatter(best_cost, best_carbon, color="blue", marker="x", s=100, label="Best Solution")

    ax.set_xlabel("Material Cost")
    ax.set_ylabel("Embodied Carbon")
    ax.set_title(f"Target Compressive Strength = {comp_strength} MPa")
```

```
    ax.legend()
    ax.grid(True)

# Extracting data
strength = list(costs_data.keys())  # Strength values
mean_cost = [v[0] for v in costs_data.values()]  # Mean cost
mean_co2 = [v[1] for v in costs_data.values()]  # Mean CO2
optimal_cost = [v[2] for v in costs_data.values()]  # Optimal cost
optimal_co2 = [v[3] for v in costs_data.values()]  # Optimal CO2

fig, ax1 = plt.subplots(figsize=(10, 5))

# First Y-axis (Cost)
ax1.set_xlabel("Strength MPa")
ax1.set_ylabel("Rs/m³", color="#1f77b4")  # Blue
ax1.plot(strength, mean_cost, 'o-', color="#1f77b4", label="Mean Cost")
ax1.plot(strength, optimal_cost, 'x--', color="#66b3ff", label="Optimal Cost")  # Light Blue
ax1.tick_params(axis='y', labelcolor="#1f77b4")

# Second Y-axis (CO2)
ax2 = ax1.twinx()
ax2.set_ylabel("kg carbon/m³", color="#d62728")  # Red
ax2.plot(strength, mean_co2, 'o-', color="#d62728", label="Mean carbon")
ax2.plot(strength, optimal_co2, 'x--', color="#ff6666", label="Optimal carbon")  # Light Red
ax2.tick_params(axis='y', labelcolor="#d62728")

# Fill between shaded regions
ax1.fill_between(strength, mean_cost, optimal_cost, color="#66b3ff", alpha=0.1)  # Light Blue
ax2.fill_between(strength, mean_co2, optimal_co2, color="#ff6666", alpha=0.1)  # Light Red

# Combined Legend
fig.legend(loc="upper left", bbox_to_anchor=(0.15, 0.85))

plt.grid(True, linestyle="--", alpha=0.5)
plt.show()



```

Bibliography

[1] Yinglong Wu et al. "Optimizing pervious concrete with machine learning: Predicting permeability and compressive strength using artificial neural networks". In: *Construction and Building Materials* 443 (2024), p. 137619.

[2] Song-Yuan Geng et al. "Revolutionizing 3D concrete printing: Leveraging RF model for precise printability and rheological prediction". In: *Journal of Building Engineering* 88 (2024), p. 109127.

[3] Peng Huang, Kuangyu Dai, and Xiaohui Yu. "Machine learning approach for investigating compressive strength of self-compacting concrete containing supplementary cementitious materials and recycled aggregate". In: *Journal of Building Engineering* 79 (2023), p. 107904.

[4] DL Zou et al. "Composition-strength relationship study of ultrahigh performance fiber reinforced concrete (UHPFRC) using an interpretable data-driven approach". In: *Construction and Building Materials* 392 (2023), p. 131973.

[5] Mana Alyami et al. "Predictive modeling for compressive strength of 3D printed fiber-reinforced concrete using machine learning algorithms". In: *Case Studies in Construction Materials* 20 (2024), e02728.

[6] Mohamed Kamel Elshaarawy, Mostafa M Alsaadawi, and Abdelrahman Kamal Hamed. "Machine learning and interactive GUI for concrete compressive strength prediction". In: *Scientific Reports* 14.1 (2024), p. 16694.

[7] M Iqbal Khan and Yassir M Abbas. "Intelligent data-driven compressive strength prediction and optimization of reactive powder concrete using multiple ensemble-based machine learning approach". In: *Construction and Building Materials* 404 (2023), p. 133148.

[8] Ismail B Mustapha et al. "Comparative analysis of gradient-boosting ensembles for estimation of compressive strength of quaternary blend concrete". In: *International Journal of Concrete Structures and Materials* 18.1 (2024), p. 20.

[9] Metin Katlav and Faruk Ergen. “Improved forecasting of the compressive strength of ultra-high-performance concrete (UHPC) via the CatBoost model optimized with different algorithms”. In: *Structural Concrete* (2024).

[10] Chuan Shi et al. “Concrete compressive strength prediction model based on RS-Catboost algorithm”. In: *2024 4th International Symposium on Computer Technology and Information Science (ISCTIS)*. 2024, pp. 428–431. DOI: 10.1109/ISCTIS63324.2024.10698947.

[11] Nima Khodadadi et al. “Data-driven PSO-CatBoost machine learning model to predict the compressive strength of CFRP- confined circular concrete specimens”. In: *Thin-Walled Structures* 198 (2024), p. 111763. ISSN: 0263-8231. DOI: https://doi.org/10.1016/j.tws.2024.111763. URL: https://www.sciencedirect.com/science/article/pii/S0263823124002064.

[12] Mien Van Tran et al. “Robust prediction of workability properties for 3D printing with steel slag aggregate using bayesian regularization and evolution algorithm”. In: *Construction and Building Materials* 431 (2024), p. 136470.

[13] Umair Jalil Malik et al. “Advancing mix design prediction in 3D printed concrete: Predicting anisotropic compressive strength and slump flow”. In: *Case Studies in Construction Materials* 21 (2024), e03510.

[14] Jiandong Huang et al. “Predicting the compressive strength of the cement-fly ash–slag ternary concrete using the firefly algorithm (fa) and random forest (rf) hybrid machine-learning method”. In: *Materials* 15.12 (2022), p. 4193.

[15] Junfei Zhang et al. “Automating the mixture design of lightweight foamed concrete using multi-objective firefly algorithm and support vector regression”. In: *Cement and Concrete Composites* 121 (2021), p. 104103. ISSN: 0958-9465. DOI: https://doi.org/10.1016/j.cemconcomp.2021.104103. URL: https://www.sciencedirect.com/science/article/pii/S0958946521001724.

[16] Dac-Khuong Bui et al. “A modified firefly algorithm-artificial neural network expert system for predicting compressive and tensile strength of high-performance concrete”. In: *Construction and Building Materials* 180 (2018), pp. 320–333. ISSN: 0950-0618. DOI: https://doi.org/10.1016/j.conbuildmat.2018.05.201. URL: https://www.sciencedirect.com/science/article/pii/S0950061818312868.

[17] Emadaldin Mohammadi Golafshani et al. “Sustainable mix design of recycled aggregate concrete using artificial intelligence”. In: *Journal of Cleaner Production* 442 (2024),

p. 140994. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2024.140994. URL: https://www.sciencedirect.com/science/article/pii/S0959652624004414.

[18] Enming Li et al. "Compressive strength prediction and optimization design of sustainable concrete based on squirrel search algorithm-extreme gradient boosting technique". In: *Frontiers of Structural and Civil Engineering* 17.9 (2023), pp. 1310–1325.

[19] Wu Zheng et al. "Multi-objective optimization of concrete mix design based on machine learning". In: *Journal of Building Engineering* 76 (2023), p. 107396. ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2023.107396. URL: https://www.sciencedirect.com/science/article/pii/S2352710223015760.

[20] Emadaldin Mohammadi Golafshani et al. "A framework for low-carbon mix design of recycled aggregate concrete with supplementary cementitious materials using machine learning and optimization algorithms". In: *Structures*. Vol. 61. Elsevier. 2024, p. 106143.

[21] Yifan Li et al. "Research on prediction model of iron ore powder sintering foundation characteristics based on FOA-Catboost algorithm". In: *Alexandria Engineering Journal* 86 (2024), pp. 603–615.

[22] Yi-Hung Chiu, Ya-Hsuan Liao, and Jia-Yang Juang. "Designing bioinspired composite structures via genetic algorithm and conditional variational autoencoder". In: *Polymers* 15.2 (2023), p. 281.

[23] José Rivera-Pérez and Imad L Al-Qadi. "Asphalt concrete mix design optimization using autoencoder deep neural networks". In: *Transportation Research Record* 2678.1 (2024), pp. 426–438.

[24] Jie Yu et al. "Generative AI for performance-based design of engineered cementitious composite". In: *Composites Part B: Engineering* 266 (2023), p. 110993.

[25] Khuong Le Nguyen, Minhaz Uddin, and Thong M Pham. "Generative artificial intelligence and optimisation framework for concrete mixture design with low cost and embodied carbon dioxide". In: *Construction and Building Materials* 451 (2024), p. 138836.

[26] Jianhao Gao, Chaofeng Wang, and SH Chu. "Mix design of sustainable concrete using generative models". In: *Journal of Building Engineering* 96 (2024), p. 110618.

[27] Xin-She Yang. "Firefly algorithms for multimodal optimization". In: *International Symposium on Stochastic Algorithms*. Springer, 2009, pp. 169–178.

[28] Iztok Fister et al. "A comprehensive review of firefly algorithms". In: *Swarm and Evolutionary Computation* 13 (2013), pp. 34–46.

[29] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[30] Diederik P Kingma, Max Welling, et al. *Auto-encoding variational bayes*. 2013.

[31] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. “Learning structured output representation using deep conditional generative models”. In: *Advances in neural information processing systems* 28 (2015).

[32] Christopher P Burgess et al. “Understanding disentangling in *beta*-VAE”. In: *arXiv preprint arXiv:1804.03599* (2018).

[33] Zhiyuan Li et al. “Progressive learning and disentanglement of hierarchical representations”. In: *arXiv preprint arXiv:2002.10549* (2020).

[34] Bradley Boehmke and Brandon Greenwell. *Hands-On Machine Learning with R*. Taylor & Francis Group, 2020. Chap. 19. URL: https://bradleyboehmke.github.io/HOML/.

[35] Construction Bazar. *Cement Price Today - OPC/PPC (Per Bag)*. Accessed: 2025-05-17. 2025. URL: https://www.constructionbazaar.com/cement-price.

[36] Indiamart Suppliers. *Silica Fume Bulk Price India*. Accessed: 2025-05-17. 2024. URL: https://www.indiamart.com/proddetail/silica-fume-123456.

[37] Tata Steel Ltd. *GGBS Product Overview and Pricing*. Accessed: 2025-05-17. 2024. URL: https://www.tatasteel.com/products-solutions/ggbs/.

[38] Fly Ash Suppliers India. *Bulk Fly Ash Rate (per ton) for Construction Use*. Accessed: 2025-05-17. 2024. URL: https://www.flyashsupplier.com.

[39] Quartz India Pvt. Ltd. *Quartz Powder High Purity Price*. Accessed: 2025-05-17. 2024. URL: https://www.quartzpowder.com/prices.

[40] Limestone Distributors India. *Limestone Powder Bulk Rate (per ton)*. Accessed: 2025-05-17. 2024. URL: https://www.limestonesupplier.com.

[41] Nanosilica Market Reports. *Nano Silica Price Trends*. Accessed: 2025-05-17. 2024. URL: https://www.marketresearch.com/nano-silica-prices.

[42] Polypropylene Fiber Manufacturers India. *Bulk Rate of Concrete Fibers*. Accessed: 2025-05-17. 2024. URL: https://www.indiamart.com/proddetail/polypropylene-fiber-987654.

[43] India Aggregates Report. *Sand and Fine Aggregate Market Rate*. Accessed: 2025-05-17. 2024. URL: https://www.sandmarket.in/latest-prices.

[44] India Aggregates Report. *Coarse Aggregate (20 mm) Rate Per Ton.* Accessed: 2025-05-17. 2024. URL: https://www.aggregatesupply.in.

[45] Chemical Additive World. *Polycarboxylate Ether Superplasticizer Pricing.* Accessed: 2025-05-17. 2024. URL: https://www.superplasticizersupplier.com.

[46] Circular Ecology. *Inventory of Carbon and Energy (ICE) Database, Version 4.0.* Online Database. Available at https://www.circularecology.com/embodied-carbon-footprint-database.html. 2024.

[47] Nouf Almonayea et al. "Probabilistic Embodied Carbon Assessments for Alkali-Activated Concrete Materials". In: *Sustainability* 17.1 (2025), p. 152. DOI: 10.3390/su17010152. URL: https://doi.org/10.3390/su17010152.

[48] Bo Yang and Xiao-Yong Wang. "Analysis of Hydration Strength and CO2 Emissions of Cement–Quartz Powder Binary Blends Considering the Effects of Water/Binder Ratios and Quartz Contents". In: *Applied Sciences* 15.4 (2025), p. 1923. DOI: 10.3390/app15041923. URL: https://doi.org/10.3390/app15041923.

[49] CarbonCloud. *ClimateHub — Ingredient Emission Data.* Accessed: 2025-05-16. 2025. URL: https://apps.carboncloud.com/climatehub.

[50] Small99. *Embodied Carbon of Common Building Materials.* Accessed: 2025-05-16. 2024. URL: https://www.small99.co.uk/guides/embodied-carbon-of-common-building-materials/.

[51] Ranran Wang et al. "Towards designing durable sculptural elements: Ensemble learning in predicting compressive strength of fiber-reinforced nano-silica modified concrete". In: *Buildings* 14.2 (2024), p. 396.