SUMMARY

In our thesis, we aim to enhance the quantitative analysis of Laser-Induced Breakdown Spectroscopy (LIBS) data for predicting major oxide compositions in geological samples. Our study integrates machine learning techniques and ensemble regression models to tackle challenges such as high dimensionality, multicollinearity, and limited data availability. Key innovations include the use of stacked generalization for improved model performance and an automated optimization framework, which enhances the accuracy and robustness of predictions.

For decades, the National Aeronautics and Space Administration (NASA) has deployed rovers equipped with advanced instruments to analyze the Martian environment. The two most recent rovers, Curiosity and Perseverance, are equipped with the Chemistry and Camera (ChemCam) and SuperCam LIBS instruments, respectively. LIBS is a powerful technique for analyzing the chemical composition of Martian soil, offering valuable insights into the planet's geology and potential for past habitability. This technique involves firing high-powered lasers at soil samples to create plasma, which emits light that is captured by spectrometers aboard the rovers. The resulting spectra can be analyzed using machine learning models to determine the presence and concentration of major oxides, providing insights into Mars' geology. However, predicting oxide compositions from LIBS data presents significant computational challenges due to the data's high dimensionality, non-linearity, and multicollinearity. Additionally, the high cost of data collection often results in small datasets, making it difficult to build accurate and robust models.

Previous research has attempted to improve predictions using regression techniques and dimensionality reduction methods, enhancing both accuracy and interpretability. However, the complex, nonlinear nature of LIBS data remains a significant challenge, necessitating more adaptive and robust machine learning strategies. Our thesis aims to build upon previous work by developing a machine learning pipeline tailored to the unique characteristics of LIBS data, aiming for higher prediction accuracy and robustness.

To achieve these objectives, we systematically explore 12 different machine learning models identified through extensive literature review and the consideration of unconventional approaches. These models fall into three categories: ensemble learning models, linear and regularization models, and neural network models. Additionally, we investigate various preprocessing techniques, including scaling, dimensionality reduction, and data transformation. We developed a k-fold data partitioning algorithm to ensure rigorous evaluation and prevent data leakage. This method involved assigning fold numbers sequentially using a modulo operation for a random-like distribution and handling extreme values by redistributing them evenly across the training folds. Additionally, we managed extreme concentration values by identifying them at specific percentiles and ensuring they were distributed evenly across the training folds, preventing any single fold from being disproportionately influenced. We created a web application that allows users to determine percentile values for handling extreme values and select the target oxide and cross-validation method. The application then visualizes the distribution of extreme values across the folds. Our cross-validation framework systematically evaluated model performance using these partitions, providing robust estimates of accuracy and generalizability. To identify the most effective combinations of models and preprocessing techniques, we developed an automated optimization framework based on Optuna, systematically searching for the optimal configurations for each regression target.

The result is a comprehensive catalog of models and preprocessing techniques for predicting major oxide compositions in LIBS data. This catalog features highly effective configurations for each of the eight major oxides examined in our study. To further enhance performance, we experiment with stacking ensemble methods using the best-performing configurations for each oxide and three different meta-learners, demonstrating improved performance of approximately 24%-34% over baseline Root Mean Squared Error of Prediction (RMSEP).

Our study makes two key contributions. Firstly, it provides a comprehensive catalog of machine learning models and preprocessing techniques for predicting major oxide compositions in LIBS data. This catalog presents highly effective configurations, allowing for a more informed selection of models and preprocessing techniques in future work. Secondly, we contribute directly to the development of Python Hyperspectral Analysis Tool (PyHAT), a Python-based toolset by the United States Geological Survey (USGS) for machine learning and data analysis on hyperspectral data. The integration of our findings into PyHAT enhances its capabilities for the scientific community.

In our experiments, we demonstrate and quantify the effectiveness of various machine learning models in predicting major oxide compositions from LIBS data. We also show that preprocessing techniques such as scaling, dimensionality reduction, and data transformation can significantly enhance model performance. Additionally, we present evidence that the optimal combination of model and preprocessing technique varies significantly depending on the specific oxide being predicted. Systematic evaluation through cross-validation and hyperparameter tuning is essential to fine-tune the models for optimal performance on specific oxides. We therefore propose a stacking ensemble methodology, integrating multiple models and preprocessing steps tailored to the specific characteristics of the data and the oxide being predicted.

Based on the findings of our study, we propose several avenues for future research. This includes exploring quantitative methods for determining optimal percentile values for data partitioning, incorporating supplementary extreme value testing, and investigating methods for augmenting datasets with synthetic data. We also highlight the importance of further experimentation with different base estimators and meta-learners to improve model performance.

LASERGAME: Leveraging Advanced Spectroscopy and Ensemble Regression for Geochemical Analysis and Model Evaluation

CHRISTIAN BAGER BACH HOUMANN, PATRICK FROSTHOLM ØSTERGAARD, and IVIK LAU DALGAS HOSTRUP, Aalborg University, Denmark

This thesis advances the analysis of Laser-Induced Breakdown Spectroscopy (LIBS) data for predicting major oxide compositions in geological samples. By integrating machine learning techniques and ensemble regression models, the study addresses challenges like high dimensionality, multicollinearity, and limited data availability. Key innovations include the use of stacked generalization for improved model performance and an automated hyperparameter optimization framework. The research contributes a comprehensive catalog of models and preprocessing techniques, and integrates findings into the Python Hyperspectral Analysis Tool (PyHAT) by the United States Geological Survey (USGS), enhancing its scientific capabilities. This work aims to establish a robust foundation for future advancements in geochemical analysis and planetary exploration using LIBS data.

Acknowledgements: We would like to thank our supervisors Dr. Daniele Dell'Aglio and Dr. Juan Manuel Rodriguez for their guidance and support throughout this project. We also thank our external supervisor Dr. Jens Frydenvang for his guidance as a domain expert from the ChemCam team, as well as volunteering his time to provide feedback on our work. Furthermore, we extend our gratitude to Dr. Ryan B. Anderson and Dr. Travis S.J. Gabriel for their invaluable discussions regarding calibration and quantification based on LIBS data. We also thank them for the opportunity to contribute to PyHAT.

1 INTRODUCTION

The National Aeronautics and Space Administration (NASA) has been studying the Martian environment for decades through a series of missions, including the Viking missions [26], the Mars Exploration Rover (MER) mission [24, 25], and the Mars Science Laboratory (MSL) mission [23], each building on the knowledge gained from the previous ones. Today, the rovers exploring Mars are equipped with sophisticated instruments for analyzing the chemical composition of Martian soil in search of past life and habitable environments.

Part of this research is facilitated through interpretation of spectral data gathered by LIBS instruments, which fire a high-powered laser at soil samples to create a plasma. The emitted light is captured by spectrometers and analyzed using machine learning models to assess the presence and concentration of certain major oxides, informing NASA's understanding of Mars' geology [7].

However, predicting major oxide compositions from LIBS data still presents significant computational challenges. These include the high dimensionality and non-linearity of the data, compounded by issues of multicollinearity and matrix effects [3]. Such effects can cause the intensity of emission lines from an element to vary independently of that element's concentration, introducing unknown variables that complicate the analysis. Furthermore, the high cost of data collection often results in small datasets, exacerbating the difficulty of building accurate and robust models.

Previous work has aimed to improve the prediction of major oxide compositions from LIBS data by using regression techniques and dimensionality reduction with feature selection. These methods have been used to enhance both the accuracy and interpretability of the prediction models. Tailored approaches have also been developed, where different models are selected based on their performance with specific spectral characteristics [4, 32]. Moreover, models incorporating physical principles have demonstrated improved accuracy by handling residuals that traditional models fail to explain [37]. However, predicting oxide compositions remains challenging due to the complex, nonlinear nature of LIBS data. This underscores the need for continued research into more accurate and robust machine learning strategies to tackle these issues effectively.

This thesis aims to improve upon previous work in the field of LIBS data analysis. Our goal is to develop a machine learning pipeline that is tailored to the unique characteristics of LIBS data, with the goal of achieving higher prediction accuracy and robustness.

To achieve these objectives, we build upon the baseline established in [15] and systematically explore ten different machine learning models. These models were identified and selected through a combination of extensive literature review and the consideration of unconventional approaches not typically covered in the LIBS-based calibration literature. The ten models fall into three categories: Ensemble learning models, linear and regularization models, and neural network models. In addition to model exploration, we also investigate a selection of preprocessing techniques: scaling (including normalization), dimensionality reduction, and data transformation. Specifically, we designed and implemented a framework for experimental analysis using the automated hyperparameter optimization framework Optuna [2]. We then used this framework to determine the most effective combinations of preprocessing methods and models for each regression target. We began by identifying the most promising models from the literature, after which we evaluated various preprocessing techniques to understand their impact on model performance, selecting those that demonstrated the highest impact on improving the performance of each model. Following this, we optimized the chosen models along with various preprocessors,

Authors' address: Christian Bager Bach Houmann, christian@bagerbach.com; Patrick Frostholm Østergaard, ostergaardpatrick@hotmail.com; Ivik Lau Dalgas Hostrup, ivikhostrup94@gmail.com, Aalborg University, Aalborg, Denmark.

using our hyperparameter optimization framework, to ensure optimal performance tailored to the specific data characteristics of each oxide.

As a result, we have developed a comprehensive catalog of models and preprocessing techniques that can be used to predict major oxide compositions in LIBS data. This catalog features configurations of various preprocessing methods and machine learning models for each of the eight major oxides examined in this study, all of which have demonstrated high effectiveness. Additionally, to investigate the potential for further performance enhancement, we used the developed catalog to experiment with stacking ensemble using the best performing configurations for each oxide and three different meta-learners. Though limited in scope, this approach demonstrated improved performance of approximately 24%-34% over baseline Root Mean Squared Error of Prediction (RMSEP).

Our key contributions are as follows:

- We have developed a comprehensive catalog of machine learning models and preprocessing techniques for predicting major oxide compositions in LIBS data. This catalog presents highly effective configurations of several preprocessing methods and machine learning models for each of the eight major oxides examined in this study, allowing for a more informed selection of models and preprocessing techniques in future work.
- We have contributed directly to the development of PyHAT, a Python-based toolset by USGS for machine learning and data analysis on hyperspectral data. Our work has been integrated into the toolset, further enhancing its capabilities for the scientific community.

In the following sections, we provide a comprehensive exploration of our research. Section 2 reviews the existing literature on LIBS data analysis and machine learning models, highlighting previous approaches and their limitations. Section 3 formally defines the problem we address, focusing on the challenges of high dimensionality, multicollinearity, and matrix effects in LIBS data. Section 4 offers background information on the data, as well as the preprocessing techniques and machine learning models that were used. In Section 5, we describe the baseline model used for Martian geological sample analysis, our efforts to replicate it, and the modifications made to improve its performance. This was then used as a baseline to evaluate our proposed stacking ensembles against. Section 6 presents our proposed approach for optimizing pipeline configurations, detailing the selection of models and preprocessing techniques, our approach to data partitioning, validation and testing procedures, and the implementation of the hyperparameter optimization framework. Section 7 presents the design and results of our experiments, as well as the analysis of the results. Our experiments include initial model selection, hyperparameter optimization, and the final evaluation of our proposed stacking ensemble. Section 8 discusses our contribution to PyHAT and how our work has been integrated into the toolset. Finally, Section 9 summarizes our key findings and contributions, while Section 10 discusses potential future research directions and improvements.

Table 1. Table of terminology.

Term	Definition
Sample	A physical specimen of rock, soil, or other material collected for scientific analysis.
Location	The specific point on a sample where a LIBS laser is targeted. There are typically multiple locations per sample.
Target	Refers to the variable that a machine learning model is trained to predict.
Extreme Concentration Values	The concentration values of oxides in the targets that are significantly higher or lower than the majority of the data.

Due to the overlapping nature of terminology used in LIBS data analysis and machine learning, we provide a list of terms in Table 1 to clarify their meaning.

2 RELATED WORK

In addressing the challenge of predicting major oxide compositions from LIBS data, our investigation intersects with a broad spectrum of existing research. Key strategies include the integration of machine learning and deep learning models, the incorporation of domain knowledge, effective preprocessing techniques, and dimensionality reduction methods. These approaches collectively aim to manage the complexities inherent in LIBS data and improve predictive performance. We review existing and relevant work through a thematic taxonomy, highlighting their potential applications in our study.

2.1 Machine Learning Models in LIBS Analysis

Several studies have applied machine learning models to analyze LIBS data, aiming to predict major oxide compositions with high accuracy.

Anderson et al. [4] utilized machine learning models to quantify major oxides on Mars using the SuperCam instrument on the Perseverance rover. Their approach involved extensive preprocessing and normalization of LIBS spectra, followed by the development of multivariate regression models for each oxide. They demonstrated that blending different models, such as Gradient Boosting Regression (GBR) and Partial Least Squares (PLS) for SiO_2, could improve prediction accuracy. Their study serves as a benchmark for model performance on LIBS spectra and offers insights into model selection for similar datasets.

shi [1] compared the performance of Support Vector Regression (SVR) and PLS regression for quantitative analysis of the major elements Si, Ca, Mg, Fe, and Al in sedimentary rock samples using LIBS data. They optimized the SVR model parameters using a genetic algorithm and cross-validation, selecting 20 characteristic emission lines as input features without dimensionality reduction. Their results were evaluated using the Root Mean Squared Error (RMSE) and Relative Standard Deviation (RSD) of predicted versus measured concentrations and showed that the non-linear SVR model significantly outperformed the linear PLS regression model at predicting elemental concentrations. The superior performance of SVR was attributed to its ability to handle non-linearities and matrix effects in the complex geological samples, demonstrating the potential of this machine learning technique for quantitative LIBS analysis in geoscience applications.

El Haddad et al. [11] explored the application of Artificial Neural Network (ANN) for quantitative analysis of soil samples using LIBS, employing a three-layer perceptron ANN architecture to address matrix effects and non-linearities. They demonstrated that ANN is efficient for predicting the concentrations of Al, Ca, Cu, and Fe. Incorporating additional spectral lines from other chemical elements, thereby increasing the amount of data input to the model, was also shown to significantly improve predictive accuracy.

Li et al. [21] developed a method for multi-component quantitative analysis of LIBS data using a deep Convolutional Neural Network (CNN). Using over 1400 spectra from 23 Chinese standard reference materials, the CNN was trained and validated, demonstrating superior performance in regression tasks compared to Back-Propagation Neural Network (BPNN) and PLS regression models. The CNN achieved lower RMSE values and higher prediction accuracy, even without removing the continuum background signal from the data, emphasizing the potential of CNNs for LIBS data analysis.

2.2 Hybrid and Domain-Knowledge-Driven Models

Incorporating domain knowledge into machine learning models can significantly enhance their interpretability and performance.

Song et al. [37] introduced a hybrid model, Dominant Factor (DF)-Kernel Extreme Learning Machine (K-ELM), which integrates domain knowledge-based spectral lines with kernel extreme learning machines. This method was particularly effective across multiple regression tasks, demonstrating improved accuracy and generalizability. The integration of domain-specific insights allowed the model to adhere more closely to the physical principles underlying LIBS quantification, making it highly relevant for applications requiring model interpretability.

Sun et al. [38] applied transfer learning to LIBS spectral data analysis, significantly improving model performance in rock classification on Mars. By leveraging knowledge from one domain to address related problems in another, their approach addressed the physical matrix effect, enhancing the robustness of the models for rock classification.

2.3 Preprocessing and Feature Engineering

Effective preprocessing and feature engineering are critical for enhancing the robustness of LIBS models.

Jeon et al. [19] investigated the effects of various feature-engineering techniques on the robustness of LIBS models for steel classification. They developed a remote LIBS system to classify six steel types, using various feature-engineering and machine learning algorithms, including Support Vector Machine (SVM) and Fully Connected Neural Network (FCNN), to handle different laser energies in test datasets. They found that using intensity ratios, which involve comparing specific spectral line intensities, significantly improved model robustness under varying measurement conditions. This approach effectively filtered out noise and enhanced the model's performance, demonstrating the importance of appropriate feature-engineering method.

Huang et al. [16] conducted a systematic analysis of data preprocessing techniques, emphasizing the need for tailored strategies to enhance model accuracy. Evaluating methods such as feature selection, case selection, scaling, and missing data treatments, they demonstrate that the effectiveness of these techniques varies markedly across different datasets and machine learning algorithms. Their findings underscore the interdependent relationship between preprocessing techniques and model selection, which is crucial for optimizing predictive performance.

2.4 Dimensionality Reduction Techniques

Dimensionality reduction techniques such as Principal Component Analysis (PCA) play a crucial role in managing the high-dimensional nature of LIBS data.

Pořízka et al. [31] conducted a comprehensive review of PCA applied within the context of LIBS. This review highlighted numerous studies that successfully utilized PCA. For instance, Moncayo et al. [28] used PCA to analyze megapixel elemental maps composed of over one million LIBS spectra. The PCA approach effectively separated the contributions of various minerals, including those present in low concentrations, demonstrating its robustness in handling highly diverse samples.

In another example, Pořízka et al. [30] employed PCA to filter outliers and classify samples based on their matrix composition, including elements such as Al, Ca, Na, and Si. This classification was followed by a univariate calibration of copper (Cu) in soil samples, resulting in reduced bias. Additionally, PCA was used to discriminate individual rocks based on their overall elemental composition, effectively addressing matrix effects that can significantly impact the accuracy of analytical results. These studies, among others highlighted by [31], underscore the effectiveness of PCA as a preprocessing technique in LIBS analysis.

Sirven et al. [35] investigated the influence of matrix effects on the performance of quantitative analysis of chromium (Cr) in soil samples using LIBS. PCA was used to classify spectra from two different soils and to detect outliers. It successfully separated spectra from agricultural soil and kaolinite in the plane of the first two components. Furthermore, PCA was used to identify and remove outliers from the dataset, enhancing the accuracy of subsequent analyses using ANNs and PLS regression. This study demonstrated the utility of PCA in managing matrix effects and improving the accuracy of quantitative LIBS analysis.

3 PROBLEM DEFINITION

The objective of this research is to predict major oxide compositions from LIBS data. We aim to enhance the accuracy and robustness of these predictions by developing and validating a computational methodology that addresses the challenges of such quantification of elements in soil samples from LIBS data. This objective presents several significant challenges, including the high dimensionality of spectral data, multicollinearity, matrix effects, and limited data availability.

A fundamental premise of this research posits that by effectively addressing these challenges, the accuracy and robustness of predicting elemental concentrations from LIBS data can be significantly enhanced. This assumption is supported by several key studies in the field. For instance, Anderson et al. [4] demonstrated that preprocessing, normalization, and the use of advanced machine learning models significantly improved the prediction accuracy of major oxides from LIBS data collected by the SuperCam instrument on the Mars 2020 Perseverance rover. Their work highlights the importance of selecting appropriate models and preprocessing techniques to handle high-dimensional spectral data effectively. Similarly, Song et al. [37] showed that incorporating domain knowledge into machine learning models enhances both the interpretability and performance of LIBS quantification. By addressing challenges such as high dimensionality and multicollinearity, their approach improved the accuracy and generalizability of the models across different tasks. The effectiveness of dimensionality reduction techniques in improving model performance was highlighted by Rezaei et al. [32], who demonstrated that methods like PCA can manage noise and computational inefficiency in high-dimensional LIBS data. This supports the notion that reducing data dimensionality can lead to more stable and accurate predictions. Furthermore, Jeon et al. [19] emphasized the importance of feature engineering in enhancing model robustness, particularly under varying measurement conditions. This is crucial for extraterrestrial applications where consistent and reliable predictions are necessary despite the challenges posed by the environment. Lastly, Sun et al. [38] demonstrated the efficacy of transfer learning in overcoming matrix effects and improving model robustness for rock classification on Mars. Their findings suggest that similar improvements can be achieved in oxide quantification by leveraging knowledge from related domains. Studies such as these provide a strong foundation for our assumption that addressing the identified challenges will lead to significant improvements in the accuracy and robustness of predicting elemental concentrations from LIBS data.

3.1 Quantification Based on LIBS Data

LIBS spectral data provides intensity readings across a spectrum of wavelengths in the form of Clean, Calibrated Spectra [3], as described by Wiens et al. [40]. The wavelength intensities are quantified in units of photon/shot/mm^2/sr/nm.

The formal definition of the problem is as follows. In a LIBS dataset, we have:

- **Concentration Tensor** $\mathbf{C}[\chi, o]$: This matrix denotes the chemical concentrations in weight percent for oxides. Each row represents a sample χ, and each column represents an oxide o.
- **Intensity Tensor** $\mathbf{I}[\chi, l, s, \lambda]$: Holds the spectral intensity data, where each entry represents the intensity recorded for a sample χ at location l, for shot s, at wavelength λ. l indicates the location on the sample where the measurement is taken, and λ is the index for wavelengths (specific wavelengths of light measured by the spectrometers).
- **Averaged Intensity Tensor** $\mathbf{A}[\chi, l, \lambda]$: Derived from matrix $\mathbf{I}$ by averaging the intensities across shots to provide a clearer signal for each location and wavelength:

$$A[\chi, l, \lambda] = \frac{1}{|S|} \sum_{s \in S} I[\chi, l, s, \lambda].$$

The primary input to our computational models is the processed LIBS spectral data. Formally, we have:

- **Masked Intensity Tensor** $\mathbf{M}[\chi, l, \lambda]$: This tensor represents the spectral intensity data after applying wavelength-specific masks to the Averaged Intensity Tensor $\mathbf{A}$. It serves as the main input to the models.
- **Feature Vectors** $\mathbf{x} \in \mathbb{R}^N$: These vectors are extracted from the Masked Intensity Tensor $\mathbf{M}$ and represent the processed LIBS signals. Each feature vector corresponds to a sample and contains N dimensions, where N is the number of relevant spectral features.

The outputs of the computational models are the predicted concentrations of major oxides in the samples. These outputs are represented as vectors of estimated oxide concentrations:

- **Estimated Concentration Vectors** $\mathbf{v} \in \mathbb{R}^{n_o}$: Each vector $\mathbf{v}$ contains the predicted concentrations for n_o target oxides. These predictions are derived from the mapping function $\mathcal{F}$ applied to the feature vectors $\mathbf{x}$.

The task of LIBS-based quantification involves fitting the parameters of a mapping function $\mathcal{F} : \mathbb{R}^N \to \mathbb{R}^{n_o}$ to accurately predict oxide concentrations from processed LIBS signals by optimizing these parameters to minimize a loss function.

3.2 Challenges

As mentioned, quantifying chemical compositions from LIBS spectral data involves several significant challenges that must be addressed to ensure accurate and robust predictions.

3.2.1 Data Dimensionality. The large number of dimensions, as seen by having many wavelengths λ in the Intensity Tensor $\mathbf{I}[\chi, l, s, \lambda]$, can lead to challenges such as the inclusion of irrelevant or redundant features.

High-dimensional datasets, like LIBS datasets, may include irrelevant or redundant features that obscure the true signal, complicating the process of accurately estimating the target variables. Effective dimensionality reduction techniques can help ensure the reliability of predictions.

3.2.2 Multicollinearity. The overlapping nature of emission lines from different elements results in high correlation between intensity readings at different wavelengths, making it difficult to extract independent spectral features necessary for accurate quantitative analysis [3].

3.2.3 Matrix Effects. Matrix effects refer to variations in the intensity of emission lines of an element independent of its concentration, arising from the complex interplay of various physical processes within the plasma generated by the LIBS technique. These effects can significantly alter emission intensities, complicating the extraction of accurate and independent spectral features. This makes it challenging to precisely map the processed LIBS signal vector $\mathbf{x} \in \mathbb{R}^N$ to a vector $\mathbf{v} \in \mathbb{R}^{n_o}$ of estimated oxide concentrations [3, 7]. Matrix effects, along with other physical processes, can induce nonlinearity in the mapping function, thereby increasing the complexity of the task[22].

3.2.4 Data Availability. Due to the high cost of data collection, datasets are often small. This limits the number of samples available for evaluation, affecting the generalizability and robustness of the models[15].

3.3 Problem Formulation

The objective of this research is to develop a computational model, denoted as $\mathcal{F} : \mathbb{R}^N \rightarrow \mathbb{R}^{n_o}$, to predict major oxide concentrations in geological samples from processed LIBS spectral data, that maintains accuracy and exhibits robustness against the challenges posed by the high dimensionality of the data, multicollinearity among spectral features, matrix effects, and the limited availability of data.

4 BACKGROUND

In this section, we provide an overview of the data used in this work, preprocessing techniques, and machine learning models used in our proposed pipeline. We outline the various normalization techniques and dimensionality reduction methods, followed by the ensemble learning, linear models, and regularization models used. Finally, we outline stacked generalization.

4.1 Data Overview

Similarly to our previous work (Houmann et al. [15]), we used the publicly available Clean, Calibrated Spectra (CCS) data from NASA's Planetary Data System (PDS) [17]. CCS refers to LIBS data that has been through a series of preprocessing steps such as subtracting the ambient light background, noise removal and removing the electron continuum to derive data that is more suitable for quantitative analysis. A comprehensive description of this preprocessing procedure is available in Wiens et al. [40].

While the CCS data is in a more suitable form for quantitative analysis, it still requires further preprocessing. This includes handling negative values and noise at the edges of the spectrometers, as we will describe in Section 7.1. Additional preprocessing steps will be necessary to further refine the data for subsequent analysis and model training. Table 2 shows an example of the CCS data for a single location of a sample. This corresponds to shots (s) and wavelength (λ) of the Intensity Tensor 2 for this location.

4.2 Preprocessing

In this subsection, we discuss the preprocessing methods used in our machine learning pipeline. We cover the following normalization techniques: Z-Score standardization, Max Absolute scaling, Min-Max normalization, robust scaling, Norm 3, power transformation, and quantile transformation. These techniques are essential for standardizing data, handling different scales, and improving the performance of machine learning models. For the purposes of this discussion, let $\mathbf{x}$ be a feature vector with values $x_1, x_2, \ldots, x_n$.

4.2.1 Z-Score Standardization. Z-Score Standardization, also known as zero-mean normalization, transforms data to have a mean of zero and a standard deviation of one. This technique is useful when the actual minimum and maximum of a feature are unknown or when outliers may significantly skew the distribution. The Z-Score Standardization of a feature vector $\mathbf{x}$ is given by:

$$x_i' = \frac{x_i - \overline{\mathbf{x}}}{\sigma_{\mathbf{x}}},$$

where x_i is the original value, $\overline{\mathbf{x}}$ is the mean of the feature vector $\mathbf{x}$, $\sigma_{\mathbf{x}}$ is the standard deviation of the feature vector $\mathbf{x}$, and x_i' is the normalized feature value. By transforming the data using the Z-score, each value reflects its distance from the mean in terms of standard deviations. Z-Score Standardization is particularly advantageous in scenarios where data features have different units or scales, or when preparing data for algorithms that assume normally distributed inputs [20].

4.2.2 Max Absolute Scaler. Max Absolute Scaling is a normalization technique that scales each feature individually so that the maximum absolute value of each feature is 1. This results in the data being normalized to a range between -1 and 1. The formula for Max Absolute Scaling is given by:

Table 2. Example of CCS data for a single location (from Houmann et al. [15]).

wave	shot1	shot2	⋯	shot49	shot50	median	mean
240.81100	6.4026649e+15	4.0429349e+15	⋯	1.7922483e+15	1.7126615e+15	1.9892956e+15	1.7561699e+15
240.86501	3.8557462e+12	2.2923680e+12	⋯	1.1355429e+12	8.6930379e+11	7.8172542e+11	7.2805052e+11
⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
905.38062	1.8823427e+08	58500403.	⋯	-8449286.6	8710775.0	4.0513312e+09	5.2188327e+09
905.57349	1.9864713e+10	1.2956832e+10	⋯	1.9785415e+10	7.1994239e+09	1.1311150e+10	1.2201224e+10

$$x_i' = \frac{x_i}{\max(|\mathbf{x}|)},$$

where x_i is the original feature value, $\max(|\mathbf{x}|)$ is the maximum absolute value of the feature vector $\mathbf{x}$, and x_i' is the normalized feature value. This scaling method is particularly useful for data that has been centered at zero or is sparse, as Max Absolute Scaling does not alter the mean of the data. Additionally, it preserves the sparsity of the data by ensuring that zero entries remain zero, thereby not introducing any non-zero values [39].

4.2.3 Min-Max Normalization. Min-Max normalization rescales the range of features to a specific range $[a, b]$, where a and b represent the new minimum and maximum values, respectively. The goal is to normalize the range of the data to a specific scale, typically 0 to 1. The Min-Max normalization of a feature vector $\mathbf{x}$ is given by:

$$x_i' = \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}(b - a) + a,$$

where x_i is the original value, $\min(\mathbf{x})$ and $\max(\mathbf{x})$ are the minimum and maximum values of the feature vector $\mathbf{x}$, respectively, and x_i' is the normalized feature value.

This type of normalization is beneficial because it ensures that each feature contributes equally to the analysis, regardless of its original scale [20].

4.2.4 Robust Scaler. The robust scaler is a normalization technique that removes the median and scales the data according to the quantile range. The robust scaler of a feature vector $\mathbf{x}$ is given by:

$$x_i' = \frac{x_i - \mathrm{Q1}(\mathbf{x})}{\mathrm{Q3}(\mathbf{x}) - \mathrm{Q1}(\mathbf{x})},$$

where x_i is the original feature value, $\mathrm{Q1}(\mathbf{x})$ is the first quartile of the feature vector $\mathbf{x}$, and $\mathrm{Q3}(\mathbf{x})$ is the third quartile of the feature vector $\mathbf{x}$. This technique can be advantageous in cases where the data contains outliers, as it relies on the median and quantile range instead of the mean and variance, both of which are sensitive to outliers [39].

4.2.5 Norm 3. As previously mentioned, the Chemistry and Camera (ChemCam) instrument consists of three spectrometers, each producing 2048 channels. For data normalization, we follow the approach taken by the ChemCam team and normalize across individual spectrometers' wavelength ranges, a process known as *Norm 3* [3, 7]. This method ensures that the wavelength intensities captured by each spectrometer are normalized independently, thus preserving the relative intensities within each spectrometer.

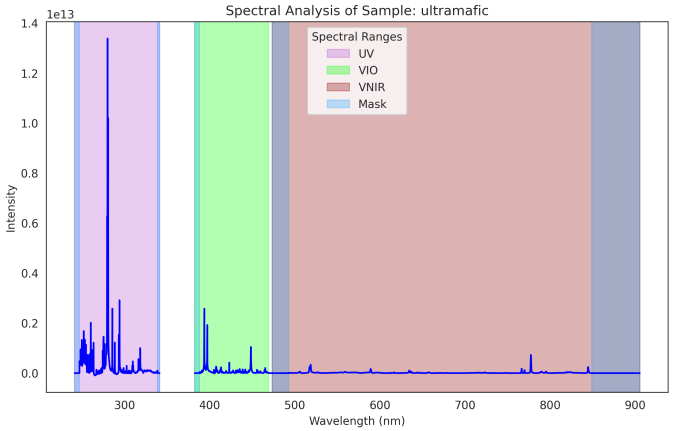

Fig. 1. Spectral plot of the CCS data for the *ultramafic* sample. The wavelengths represent the spectral channels.

Figure 1 shows a spectral plot of the CCS data for the *ultramafic* sample, illustrating the three distinct spectral regions, each captured by one of the three spectrometers. Specifically, one spectrometer captures the Ultraviolet (UV) region, another captures the Violet (VIO) region, and the third captures the Visible and Near-Infrared (VNIR) region.

Let γ represent the spectrometer index, where $\gamma \in \{1, 2, 3\}$, corresponding to the UV, VIO, and VNIR spectrometers, respectively. Then, Norm 3 is formally defined as:

$$\tilde{X}_{i,j}^{(\gamma)} = \frac{X_{i,j}^{(\gamma)}}{\sum_{j=1}^{N} X_{i,j}^{(\gamma)}}, \tag{1}$$

where

- $\tilde{X}_{i,j}^{(\gamma)}$ is the normalized wavelength intensity for the i-th sample in the j-th channel on the γ-th spectrometer,
- $X_{i,j}^{(\gamma)}$ is the original wavelength intensity for the i-th sample in the j-th channel on the γ-th spectrometer, and
- $N = 2048$ is the number of channels in each spectrometer.

This normalization method results in a total of $3N = 6144$ normalized features for each sample, as each of the three spectrometers contributes 2048 channels.

4.2.6 Power Transformation. Power transformations are a class of mathematical functions used to stabilize variance and make data more closely approximate a normal distribution. They are particularly useful in statistical modeling and data analysis to meet the assumptions of linear models.

One of the first influential power transformation techniques is the Box-Cox power transform, introduced by Box and Cox [5] in 1964. This is defined for positive data and is aimed at normalizing data or making it more symmetric. For a feature vector $\mathbf{x}$, the Box-Cox transformation is defined as:

$$\psi^{\mathrm{BC}}(\lambda, \mathbf{x}) = \begin{cases} \frac{\mathbf{x}^{\lambda}-1}{\lambda}, & (\lambda \neq 0) \\ \log(\mathbf{x}), & (\lambda = 0) \end{cases},$$

where λ is the transformation parameter. λ determines the extent and nature of the transformation, where positive values of λ apply a power transformation and $\lambda = 0$ applies a logarithmic transformation.

To overcome the limitations of the Box-Cox transformation, Yeo and Johnson [44] introduced a new family of power transformations that can handle both positive and negative values. The Yeo-Johnson power transformation is defined as:

$$\psi(\lambda, \mathbf{x}) = \begin{cases} \frac{(\mathbf{x}+1)^{\lambda}-1}{\lambda} & (\mathbf{x} \geq 0, \lambda \neq 0) \\ \log(\mathbf{x}+1) & (\mathbf{x} \geq 0, \lambda = 0) \\ -\frac{(-\mathbf{x}+1)^{2-\lambda}-1}{2-\lambda} & (\mathbf{x} < 0, \lambda \neq 2) \\ -\log(-\mathbf{x}+1) & (\mathbf{x} < 0, \lambda = 2) \end{cases}$$

For non-negative values, the Yeo-Johnson transformation simplifies to the Box-Cox transformation, making them equivalent in this context. The key benefit of the Yeo-Johnson transformation is its ability to handle any real number, making it a robust choice for transforming data to achieve approximate normality or symmetry. This property is particularly beneficial for preparing data for statistical analyses and machine learning models that require normally distributed input data.

4.2.7 Quantile Transformer. Quantile transformation is a method that applies a non-linear transformation to map data to a uniform or normal distribution. This process involves mapping the data X to a set of probabilities p using the Cumulative Distribution Function (CDF), which indicates the probability that a random variable will be less than or equal to a specific value in X's original distribution. Subsequently, the quantile function, which is the inverse of the CDF of the desired distribution, is applied to these probabilities p to generate the transformed data. This method forces the data to conform to the specified distribution regardless of the original distribution's form [39].

4.2.8 Principal Component Analysis (PCA). PCA is a dimensionality reduction technique used to reduce the number of features in a dataset while retaining as much information as possible. We provide an overview of PCA in this section based on Jiawei Han [20] and Vasques [39].

PCA works by identifying the directions in which the n-dimensional data varies the most and projects the data onto these k dimensions, where $k \leq n$. This projection results in a lower-dimensional representation of the data. PCA can reveal the underlying structure of the data, which enables interpretation that would not be possible with the original high-dimensional data.

PCA works as follows. First, the input data are normalized, which prevents features with larger scales from dominating the analysis.

Then, the covariance matrix of the normalized data is computed. The covariance matrix captures how each pair of features in the dataset varies together. k orthogonal unit vectors, called *principal components*, are then computed from this covariance matrix. These vectors are perpendicular to each other and capture the directions of maximum variance in the data.

The principal components are then sorted such that the first component captures the most variance, the second component captures the second most variance, and so on. Variance is assumed by PCA to be a measure of information. In other words, the principal components are sorted based on the amount of information they capture.

After computing and sorting the principal components, the data can be projected onto the most informative principal components. This projection results in a lower-dimensional approximation of the original data. The number of principal components to keep is a hyperparameter that can be tuned to balance the trade-off between the amount of information retained and the dimensionality of the data.

4.2.9 Kernel PCA. We provide a brief overview of the Kernel Principal Component Analysis (Kernel-PCA) algorithm based on Schölkopf and Smola [33]. Kernel-PCA is an extension of traditional PCA designed to handle nonlinear relationships among data points. The core idea behind Kernel-PCA is to map data into a higher-dimensional space using a kernel function, a technique known as the kernel trick. This mapping enables linear separation of data points in the higher-dimensional space, even if they are not linearly separable in the original space.

Similar to PCA, as described in Section 4.2.8, the goal of Kernel-PCA is to extract the principal components of the data. Unlike PCA, Kernel-PCA does not compute the covariance matrix of the data directly, as this is often infeasible for high-dimensional datasets. Instead, Kernel-PCA leverages the kernel trick to compute the similarities between data points directly in the original space using a kernel function. This kernel function implicitly computes the dot product in the higher-dimensional feature space without explicitly mapping the data points into that space. That way, Kernel-PCA can

capture nonlinear relationships among data points without explicitly transforming them into a higher-dimensional space.

By using pairwise similarities to construct a kernel matrix, also referred to as a Gram matrix, Kernel-PCA can perform eigenvalue decomposition. This process allows for the extraction of principal components in the feature space, similar to the approach used in regular PCA. However, in Kernel-PCA, the eigenvalue decomposition is performed on the kernel matrix rather than the covariance matrix, resulting in the principal components. These principal components are nonlinear combinations of the original data points, enabling the algorithm to capture complex relationships among data points that are not linearly separable in the original space.

4.3 Linear and Regularization Models

Linear models are a class of models that assume a linear relationship between the input features and the target variable. These models are simple and interpretable, making them a popular choice for regression tasks. Regularization is used to prevent overfitting through the addition of a penalty term to the loss function[18].

In this section, we provide an overview of the linear and regularization models used in this work.

4.3.1 Ordinary Least Squares Regression. We briefly cover the Ordinary Least Squares (OLS) regression algorithm based on James et al. [18] to provide context for the subsequent discussions on linear and regularization models.

OLS regression is a linear regression technique that fits a linear model, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$, to the data by minimizing the Residual Sum of Squares (RSS), where $\hat{y}$ is the predicted target value, $\hat{\beta}_0$ is the intercept, $\hat{\beta}_1$ is the coefficient for the input feature, and x is the input feature. In OLS regression, the objective is to estimate the coefficients that minimize the sum of squared differences between the observed target values and the predicted values, known as the RSS:

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where y_i is the observed target value, $\hat{y}_i$ is the predicted target value, and n is the number of samples in the dataset. To minimize the RSS, OLS regression finds the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the RSS.

4.3.2 Ridge Regression. As an alternative to OLS, one can fit a model with all N features by applying techniques that constrain or regularize the coefficient estimates, effectively shrinking them towards zero. Although it may seem counterintuitive, this constraint can significantly reduce variance in the estimates. One of the most well-known techniques for this purpose are is ridge regression, which we now provide an overview of, based on James et al. [18].

Ridge regression introduces a regularization term to the OLS regression objective function to prevent overfitting and reduce the model's variance. This regularization term is known as the L_2 norm and is defined as the sum of the squared regression coefficients:

$$L_2 = \sum_{j=1}^{p} \beta_j^2,$$

where β_j are the regression coefficients, and p is the number of features. The ridge regression objective function $f_{ridge}(\beta)$ is defined as:

$$f_{ridge}(\beta) = \text{RSS} + \lambda L_2$$

where RSS is the residual sum of squares defined in Section 4.3.1 and λ is the regularization parameter — a hyperparameter that controls the strength of the penalty. Introducing this regularization term causes "shrinkage", which means that the estimated regression coefficients are shrunk towards zero, reducing their variance. This shrinkage usually results in a model with higher bias compared to one fitted with OLS regression because the regularization can cause a worse fit to the training data. However, the model has a lower variance, making it less prone to overfitting and more generalizable to unseen data.

4.3.3 Least Absolute Shrinkage and Selection Operator (LASSO) Regression. A disadvantage of ridge regression is that it includes all features in the final model. While it shrinks the coefficients towards zero, it does not eliminate any by setting them exactly to zero. This may not impact prediction accuracy but makes interpretation harder, especially with many variables. For example, in datasets with numerous features, one might prefer a model that includes only the most important ones. However, ridge regression includes all features, and while increasing the penalty reduces coefficient sizes, it does not set any coefficients to zero[18].

We therefore introduce Least Absolute Selection and Shrinkage Operator (LASSO) regression based on James et al. [18], a regression technique which addresses this issue by adding a different regularization term to the OLS regression objective function. While ridge regression uses the L_2 norm, LASSO uses the L_1 norm, which has the distinct effect of performing feature selection by shrinking some coefficients to exactly zero:

$$L_1 = \sum_{j=1}^{p} |\beta_j|,$$

where β_j are the regression coefficients, and p is the number of features. The LASSO regression objective function $f_{LASSO}(\beta)$ is defined as:

$$f_{LASSO}(\beta) = \text{RSS} + \lambda L_1,$$

where RSS is the residual sum of squares defined in Section 4.3.1 and λ is the regularization parameter. By introducing this L_1 regularization term, LASSO not only penalizes large coefficients but also has the property of setting some coefficients to zero, effectively selecting a simpler model that only includes

the most important features. This makes LASSO particularly useful when dealing with high-dimensional data where feature selection is crucial.

4.3.4 Elastic Net Regression (ENet). Despite LASSO regression's ability to perform feature selection and its effectiveness in settings with multicollinearity, it can struggle when the dataset contains globally highly correlated features. LASSO tends to select only one feature from a group of highly correlated features and ignores the others[45]. When multiple correlated features are important for the prediction, this can lead to suboptimal performance.

To address this, Zou and Hastie [45] introduced the Elastic Net (ENet) regression, a regression method that combines the L_1 and L_2 regularization terms from ridge and LASSO regression. The ENet regression objective function $f_{ENet}(\beta)$ is defined as:

$$f_{ENet}(\beta) = \text{RSS} + \lambda_1 L_1 + \lambda_2 L_2,$$

where RSS is the residual sum of squares defined in Section 4.3.1, and λ_1 and λ_2 are regularization parameters that control the strength of the L_1 and L_2 penalties, respectively. By combining the L_1 and L_2 regularization terms, ENet performs feature selection like LASSO while also shrinking the coefficients of correlated features like ridge regression. This way, ENet can mitigate the limitations of each method when used individually by balancing the trade-offs between the two regularization terms.

4.3.5 Principal Component Regression (PCR) & Partial Least Squares (PLS). In order to understand PLS, it is helpful to first consider Principal Component Regression (PCR), as PLS is an extension of PCR that aims to address some of its limitations. We provide an overview of both regression techniques based on James et al. [18].

PCR extends PCA in the context of regression analysis. First, PCA is performed to identify the M principal components that capture the most variance in the data. These components are then used in a linear regression model to predict the target variable by fitting a linear model via least squares regression. The key intuition behind PCR is that a small number of principal components are sufficient to capture most of the variance in the data, which can then be used to predict the target variable. That is, the directions of the principal components are assumed to be associated with the target variable. If this assumption holds, a least squares regression model fitted to the principal components provides better predictions than a model fitted to the original features because the principal components indeed capture the most important information in the data.

One drawback of PCR is that it does not consider the target variable in the decomposition of the features and therefore assumes that components with larger variance have a stronger correlation with the target than those with smaller variance. The components that capture the most variation may not be the most predictive of the target; some data might be highly variable but irrelevant to the target.

To address this limitation, PLS extends PCR by considering the target variable when identifying the components. PLS uses an iterative method to identify components that maximize the covariance between the features and the target. Similar to PCR, PLS identifies M components that capture the most variance in the data and fits a linear model with least squares regression. However, unlike PCR, PLS uses the target variable to identify the components that are most predictive of the target, resulting in components that not only approximate the data but also relate to the target variable. In essence, PLS aims to find components that are both informative about the data and predictive of the target.

This is an iterative process where the residuals from the previous components are used to calculate the next component. Specifically, the m-th component is derived from the residuals of the previous $m-1$ components. These residuals represent the part of the data that has not been explained by the previous components. The iteration can be repeated M times to identify M components, where M is a tunable hyperparameter. Afterwards, the components are used to predict the target variable by fitting a linear model via least squares regression, just like in PCR.

4.3.6 Support Vector Regression (SVR). SVR is a regression technique that extends the principles of SVM to regression problems. We therefore provide an overview of SVMs based on James et al. [18] before discussing SVRs.

SVM is a supervised learning algorithm used primarily for classification tasks. A core concept in SVM is the *hyperplane*. Generally, a hyperplane is a subspace of one dimension less than its ambient space. This means that in a two-dimensional space, a hyperplane is a line, while in a three-dimensional space, it is a plane, and so on.

SVM is built on the idea of finding the hyperplane that best separates the data points into different classes. This hyperplane is chosen to maximize the margin, which is the distance between the hyperplane and the nearest data point from either class. The instances right on or inside the margin are called *support vectors*, which are used to 'support' the margin and decision boundary.

SVR extends the principles of SVM to regression problems. We use our previous discussion of SVM to introduce SVR based on Drucker et al. [8] and Smola and Schölkopf [36].

SVR aims to fit a function that predicts continuous values rather than finding the hyperplane that best separates data points. Instead of using a hyperplane to separate the data, SVR uses two parallel hyperplanes to define a margin within which the function should lie, often referred to as the ϵ-*tube*, where ϵ is a hyperparameter that defines the width of the tube. The goal is to find a function $f(x)$ that lies within this tube and has the maximum number of data points within the tube. $f(x)$ is typically defined as a linear function of the form:

$$f(x) = \mathbf{w} \cdot \mathbf{x} + b,$$

where:

- $\mathbf{w}$ is the weight vector,

- $\mathbf{x}$ is the input vector, and
- b is the bias term.

The two parallel hyperplanes at a distance ϵ from the hyperplane are defined as:

$$\mathbf{w} \cdot \mathbf{x} + b = f(\mathbf{x}) + \epsilon,$$
$$\mathbf{w} \cdot \mathbf{x} + b = f(\mathbf{x}) - \epsilon.$$

Or, more succinctly:

$$f^+(\mathbf{x}) = f(\mathbf{x}) + \epsilon,$$
$$f^-(\mathbf{x}) = f(\mathbf{x}) - \epsilon,$$

where $f^+(\mathbf{x})$ and $f^-(\mathbf{x})$ are the upper and lower bounds of the ϵ-insensitive tube, respectively.

The optimization problem in SVR is to find the coefficients $\mathbf{w}$ and b that minimize the norm of $\mathbf{w}$ (i.e., keep the regression function as flat as possible) while ensuring that most data points lie within the ϵ margin.

4.4 Ensemble Learning Models

In this section we introduce the concept of ensemble learning and decision trees based on James et al. [18], as they are fundamental aspects of the ensemble learning models we discuss. Following this, we outline Extra Trees Regression (ETR), GBR, Natural Gradient Boosting (NGBoost), and eXtreme Gradient Boosting (XGBoost).

4.4.1 Ensemble Learning. Ensemble learning is a machine learning technique that combines multiple models, referred to as weak learners, to generate more accurate predictions. Individually, the predictive ability of weak learners may be limited, but when combined, they can produce a more precise and robust model. Ensemble learning encompasses several methods, including bagging, boosting, and stacking[18, 29]

In this section, we provide an overview of the ensemble learning methods used in this work. We begin by introducing decision trees, a commonly used weak learner in ensemble methods.

4.4.2 Decision Trees. A decision tree is a supervised learning model that partitions data into subsets based on feature values, creating a tree structure. The goal is to create a tree that predicts the target variable by dividing the data into increasingly homogeneous subsets. Each internal node in the tree represents a decision based on a specific feature, while each leaf node represents a prediction for the target variable. The tree can make predictions for new data points by following a path from the root to a leaf node[18].

4.4.3 Extra Trees Regressor (ETR). We give an overview of Random Forest (RF) based on James et al. [18]. Then, we introduce the ETR model based on Geurts et al. [12].

RF is an ensemble learning method that improves the accuracy and robustness of decision trees by building multiple trees and combining their predictions. Each tree is trained on a random subset of the data using bootstrap sampling, where samples are drawn with replacement, meaning the same sample can be selected multiple times. Bagging, or bootstrap aggregating, involves training each tree on a different bootstrap sample and then aggregating their predictions to form the final output. This introduces variability in the training data available to the models and reduces overfitting, as some data points may appear multiple times while others may be omitted. In addition to bootstrap sampling, random forests introduce an additional layer of randomness by selecting a random subset of features for splitting at each node of the trees. This further decorrelates the trees, enhancing the model's robustness and reducing the risk of overfitting. By aggregating predictions from multiple trees, the model achieves better generalization and robustness.

For a feature vector x, the prediction of a RF model can be represented as an aggregation of the predictions of individual trees:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x),$$

where $f_m(x)$ is the prediction of the m-th tree, and M is the total number of trees. This form of aggregation reduces the overall variance of the model, by averaging the predictions of the individual trees to produce a final prediction.

ETR extends the RF model by introducing additional randomness in the tree-building process, specifically through random feature selection and random split points. While RF uses bootstrap sampling and selects the best split from a random subset of features to create a set of diverse samples, ETR instead selects split points randomly within the chosen features, introducing additional randomness. This process results in even greater variability among the trees, aiming to reduce overfitting and improve the model's robustness. As a trade-off, ETR is less interpretable than a single decision tree, as the added randomness can introduce more bias than RF. However, it often achieves better generalization performance, especially in high-dimensional or noisy datasets.

4.4.4 Gradient Boosting Regression (GBR). In this section, we introduce GBR based on Hastie et al. [14] and Burkov [6].

Gradient Boosting is a machine learning technique used for various tasks, including regression and classification. The fundamental concept involves sequentially adding models to minimize a loss function, where each successive model addresses the errors of the ensemble of preceding models.

This technique utilizes gradient descent to optimize the loss function, allowing for the selection of different loss functions depending on the specific task. The loss function is generally defined as $L(y, \hat{y})$, and measures the discrepancy between the true values y and the predicted values $\hat{y}$. GBR is a specialized application of gradient boosting for regression tasks, where it minimizes a regression-appropriate loss function, such as mean squared error or mean absolute error. Typically, decision trees are used as the base models in each iteration.

The process starts with an initial model $f_0(x)$ that minimizes the loss function over the entire dataset:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$

where L is the chosen loss function, N is the number of samples, y_i are the true values, and γ is a constant that represents the prediction of the initial model.

Then we start the iterative process of adding models to the ensemble. For each iteration m, from 1 to M:

(1) Compute the residuals of the current model. For regression, this could be the squared error loss, $L(y, \hat{y}) = (y - \hat{y})^2$. The residuals $r_i^{(m)}$ for each data point i are calculated as $r_i^{(m)} = y_i - f_{m-1}(x_i)$, where $f_{m-1}(x_i)$ is the prediction of the previous model.
(2) Fit a new decision tree $h_m(x)$ to the residuals. This tree aims to correct the errors of the current ensemble by using the residuals instead of ground truth values. Essentially, $h_m(x)$ tries to predict the residuals $r_i^{(m)}$.
(3) Update the ensemble model by adding the predictions of the new tree $h_m(x)$, multiplied by a learning rate η. The learning rate η controls the contribution of each new tree to the ensemble, preventing overfitting by scaling the updates:

$$f_m(x) = f_{m-1}(x) + \eta h_m(x)$$

This iterative process continues until the maximum number M of trees is combined, resulting in the final model $\hat{f}(x) = f_M(x)$.

4.4.5 Natural Gradient Boosting (NGBoost). NGBoost[9] is a variant of the gradient boosting algorithm that leverages the concept of natural gradients with the goal of improving convergence speed and model performance. In more complex models, the parameter space can be curved and thus non-Euclidean, making the standard gradient descent less effective. Consequently, using the standard gradient descent can lead to slow convergence and suboptimal performance. In such scenarios, the application of natural gradients becomes particularly advantageous.

Natural gradients account for the underlying geometry of the parameter space by using information about its curvature. By incorporating this information, natural gradients can navigate the parameter space more efficiently, leading to faster convergence and better performance. In addition, NGBoost provides its predictions in the form of probability distributions, allowing it to estimate the uncertainty associated with its predictions.

The algorithm starts by initializing a model with a guess for the parameters of the probability distribution, usually starting with something simple like a Gaussian distribution. This initial model prediction represents the probability distribution over the target variable based on the given features.

Then, the algorithm enters an iterative process to refine its predictions. At the start of each iteration, the model computes its current predictions using the existing set of parameters. The algorithm then calculates the negative gradient of the loss function with respect to the current predictions. This involves computing the gradient of the negative log-likelihood, which quantifies the discrepancy between the current predictions and the actual observed data. The negative log-likelihood quantifies how well the model's predicted probability distribution matches the observed data, with lower values indicating better alignment between predictions and observations.

Next, the *Fisher information matrix* is computed. This matrix encodes the curvature of the parameter space at the current parameter values, reflecting how sensitive the likelihood function is to changes in these parameters. For example, if the likelihood function is highly sensitive to changes in a particular parameter, the Fisher information matrix will have a high value for that parameter. Using this information, the model can adjust its parameters more effectively, focusing on the most sensitive parameters to improve performance.

The standard gradient, or residuals, which is derived from the negative log-likelihood, is then transformed using the inverse of the Fisher information matrix to obtain what is known as the natural gradient. Next, a weak learner, typically a decision tree, is fitted to these natural gradients. This step is similar to traditional gradient boosting, where a tree is fitted to the residuals, but in NGBoost, the tree is fitted to the natural gradients instead.

The parameters of the model are then updated using the output from the weak learner. This update process incorporates a learning rate to control the step size, ensuring that the model makes gradual improvements rather than drastic changes.

Using the newly updated parameters, the model recalculates its predictions, refining the probability distribution of the target variable. This iterative process of computing predictions, calculating gradients, fitting weak learners, and updating parameters continues for a predetermined number of iterations or until the model's performance converges.

4.4.6 Stacked Generalization. Stacked generalization, introduced by Wolpert [42], is an ensemble method that combines the predictions of multiple base models, which are trained on the original dataset, as input to a meta-model. This meta-model is trained to make the final prediction. The strategy allows the meta-model to learn the optimal way to combine the predictions of the base models to minimize the generalization error.

Formally, let $\mathbf{X}$ denote the input data and $\mathbf{y}$ the target variable. Initially, N base models $G_1, G_2, \ldots, G_N$ are trained on the dataset $\mathbf{X}$. Each base model generates a set of predictions $\hat{\mathbf{y}}_i = G_i(\mathbf{X})$.

The predictions from the base models are then compiled into a new dataset $\mathbf{Z}$, where each column $\mathbf{z}_i \in \mathbf{Z}$ represents the predictions from the i-th base model:

$$\mathbf{Z} = [\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N]$$

A meta-model F is subsequently trained on this new dataset $\mathbf{Z}$ to predict the target variable $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = F(\mathbf{Z})$$

The final prediction is provided by the meta-model F, which effectively integrates the outputs of the base models to enhance overall performance. The effectiveness of stacked generalization stems from its ability to leverage the unique strengths of different base models while mitigating their individual weaknesses, thereby producing a more accurate and generalizable ensemble model.

5 BASELINE & REPLICA

For analyzing Martian geological samples, the ChemCam team currently uses the Multivariate Oxide Composition (MOC) model [7]. This model integrates PLS and Independent Component Analysis (ICA) via Joint Approximation Diagonalization of Eigen-matrices (JADE) to predict the composition of major oxides.

As shown in figure 2, the input to the MOC model is the CCS data, mentioned in Section 3. This spectral data is collected on Earth in a laboratory setting simulating the Martian environment. The instrument used to collect this data is a LIBS instrument replicating the ChemCam instrument on the Curiosity rover. Both the ChemCam and laboratory instrument consist of three spectrometers, each producing 2048 channels. These spectrometers are used to capture the UV, VIO, and VNIR regions of the spectrum. For each sample, five CCS datasets are collected by firing 50 laser shots at five different locations on the sample and processing the raw spectral readings [40]. Consequently, the CCS data for each sample forms a high-dimensional `Intensity Tensor` I (Tensor 2) with dimensions $5 \times 50 \times 6144$. An entry in this matrix represents the intensity of a specific wavelength in nanometers. Complementing the data is the matrix of the corresponding major oxide concentrations for each sample $C1$, which serves as the target variable for the model. For more details, refer to Section 5 in Houmann et al. [15].

The PLS and JADE phases of the MOC operate in parallel, and their predictions are blended to form the final predictions. Though the MOC model has proven useful, it suffers from limitations in predictive accuracy and robustness. An overview of the MOC model is shown in Figure 2.

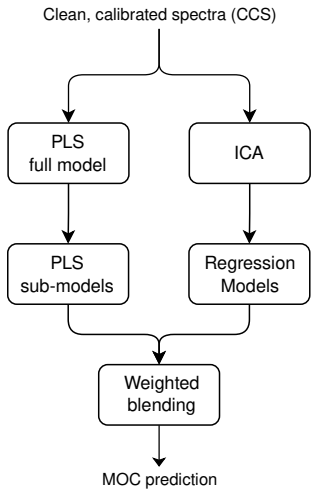

Fig. 2. Overview of the MOC model.

In Houmann et al. [15], we presented our efforts to replicate the MOC model. Based on the insights gained from that work, we have made several modifications to the replica in preparation for this work.

Our replica only utilized a single dataset for the ICA phase, while the original model used all five datasets. This difference was due to the original paper not specifying how the five datasets were used, and so we designed an experiment to determine how to use them in a way that would most closely replicate the original model. We initially assumed that the datasets were aggregated and used as a single dataset. This approach, however, did not align with the original model's results, likely due to the loss of information from the individual datasets. Following this discovery, we modified the replica to instead use the datasets in the same way as in the Partial Least Squares 1 - sub-model (PLS1-SM) phase, which yielded results aligning more closely with the original model.

Furthermore, our initial replica used a random train/test split for training, in contrast to the original model's manual curation to ensure representation of extreme compositions in both sets. This difference stemmed from the original authors' application of domain expertise in their dataset curation — a process we could not directly replicate. Nevertheless, we found that automatically identifying extreme compositions and ensuring that they were present in both the training and testing sets brought us closer to the original model. We chose to pull out the n largest and smallest samples by concentration range, for each oxide, and reserve them for the training set. Then we would do a random split on the remaining dataset, such that the final train/test split would be a 80%/20% split.

With these changes, we created a more accurate replica of the MOC model, which we will use as our baseline for the rest of this paper. We have presented these changes to one of the original authors of Clegg et al. [7], who confirmed that they were reasonable and in line with the original model's implementation.

Table 3 shows the RMSEs of the original models and our replicas after the changes. Figure 3 illustrates the distribution of these RMSEs as a grouped histogram. The results show that the RMSEs of our replicas exhibit similar tendencies to the original models. However, in some cases, our replicas have a lower RMSE than the original models, and in others, they have a higher RMSE. These differences are due to a number of factors.

Firstly, the original models were trained with datasets from 1600mm and 3000mm standoff distances [7], while we only

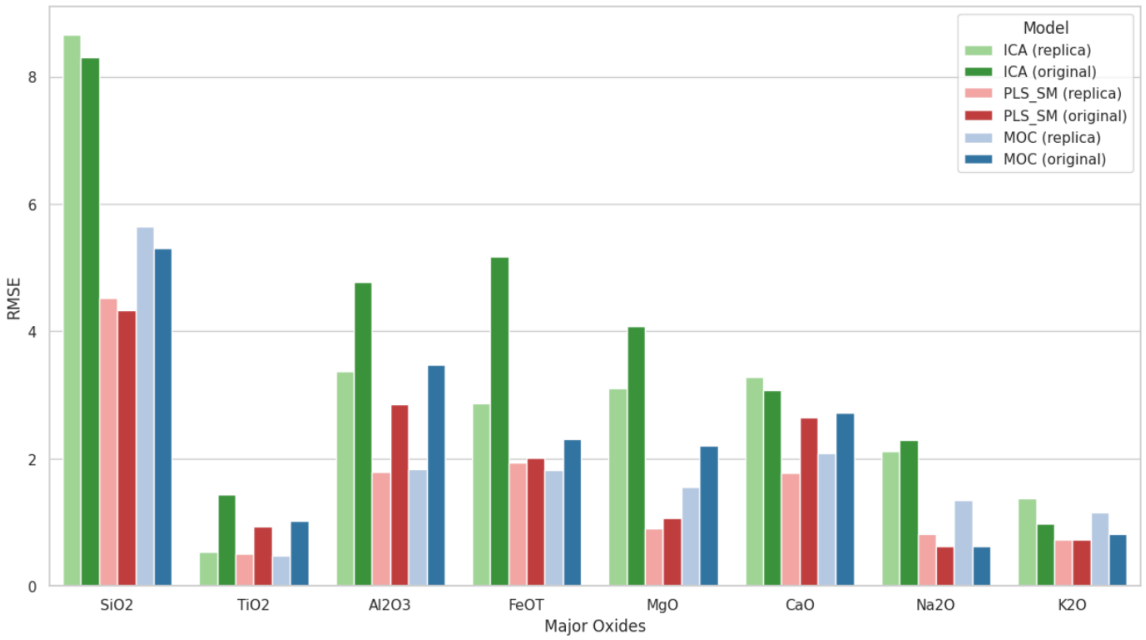

Fig. 3. Grouped histogram of the RMSEs of the original and our replicas of the PLS1-SM, ICA, and MOC models.

had access to the 1600mm dataset for our replicas. Additionally, we automated the outlier removal for the PLS1-SM phase, unlike the original manual process. As mentioned, the original authors manually curated their training and test sets, ensuring a broad elemental range, while we implemented an automatic process for our replicas due to lack of domain expertise. Differences might also stem from varied implementation specifics, such as programming languages and libraries used.

Through a series of comparative experiments, we showed that the model selection was the primary cause of these limitations, and we showed how both ANN and GBR methods could be used to improve the model's predictive accuracy and robustness. This is further underscored by work from the SuperCam team. In 2021, the Perseverance rover landed on Mars, equipped with the SuperCam instrument, which is the successor to the ChemCam instrument. As part of the ongoing work to support the SuperCam instrument, Anderson et al. [4] experimented with various machine learning models to predict the composition of major oxides in geological samples using the SuperCam LIBS calibration dataset. While the team decided to retain PLS for analyzing certain oxides, ICA was entirely discontinued. Instead, models based on GBR, RF, and LASSO were selected for other oxides. This decision reinforces our finding that ICA regression models fall short in accurately predicting the composition of major oxides in geological samples. Consistent with our observations, GBR was also identified as a high-performing model in their analyses.

6 PROPOSED APPROACH

To address the challenges in predicting major oxide compositions from LIBS data, we propose the development of advanced computational models capable of effectively handling the multifaceted challenges we describe in 3.2. These issues complicate the accurate and robust prediction of elemental concentrations, necessitating advanced computational methodologies.

Our approach aims to enhance the prediction accuracy and robustness for major oxides in LIBS data by leveraging specific combinations of machine learning models and preprocessors that are particularly effective at predicting individual oxides. The models will use feature vectors $\mathbf{x} \in \mathbb{R}^N$ derived from the `Masked Intensity Tensor` $\mathbf{M}[\chi, l, \lambda]$ as input, where N is the number of features. The output will be Estimated Concentration Vectors $\mathbf{v} \in \mathbb{R}^{n_o}$.

As highlighted in Section 2, the literature suggests that various models and preprocessing techniques are adept at handling high-dimensional data, multi-collinearity, and matrix effects. The literature also indicates that different machine learning models perform better on some oxides than others. These challenges and model-specific strengths suggests that an optimal approach would involve hybrid methodology, integrating multiple models and preprocessing steps tailored to the specific characteristics of the data. This could include leveraging ensemble learning techniques to combine the predictions of various models, implementing dimensionality reduction techniques like PCA to mitigate high-dimensionality issues, and

Table 3. RMSEs of the original and our replicas of the PLS1-SM, ICA, and MOC models.

Element	PLS1-SM (original)	PLS1-SM (replica)	ICA (original)	ICA (replica)	MOC (original)	MOC (replica)
SiO_2	4.33	4.52	8.31	8.63	5.30	5.61
TiO_2	0.94	0.49	1.44	0.54	1.03	0.61
Al_2O_3	2.85	1.79	4.77	3.18	3.47	2.47
FeO_T	2.01	2.16	5.17	2.87	2.31	1.82
MgO	1.06	0.91	4.08	3.11	2.21	1.56
CaO	2.65	1.73	3.07	3.28	2.72	2.09
Na_2O	0.62	0.80	2.29	1.39	0.62	1.33
K_2O	0.72	0.72	0.98	1.38	0.82	1.91

employing robust preprocessing strategies to address multicollinearity and matrix effects. Furthermore, a systematic evaluation through cross-validation and hyperparameter tuning would be essential to fine-tune the models for the best performance on the specific oxides of interest. The notion of using multiple models per oxide is supported by the advent of models such as the MOC [7] model, which combines the predictions of multiple models using a predetermined weighting for each model's predictions on a per-oxide basis. While this approach improved accuracy compared to individual models, it required manual tuning of the weights for each model. This manual tuning presents limitations, including the analysis required to determine appropriate weights and the risk of suboptimal weighting. Given these limitations, it is reasonable to explore techniques that can automate the weighting process while still leveraging the strengths of multiple models. To fulfill these criteria, we chose to adopt a stacking ensemble approach. Stacking, as described in Section 4.4.6, is a method that utilizes multiple base estimators trained on the same data, whose predictions are then used to train a meta-learner. By combining a diverse set of base models, stacking can correct for the biases of individual models. Since each model focuses on different patterns within the data, stacking mitigates the inherent biases of individual models by estimating and correcting for these biases. Leveraging the strengths of multiple models that each approach the problem differently can lead to better generalization on unseen data. This is achieved by using a meta-learner to discern patterns in the base predictors' outputs[27, 42], with the added benefit of automating and potentially improving upon the manual tuning employed by the MOC model. However, it is crucial to consider the training of the base models to prevent data leakage and overfitting. As emphasized by Witten and Frank [41], if the base models are trained on the same dataset, the meta learner might favor certain base models over others. This can cause the meta learner to be influenced by the same patterns and biases that the base models are susceptible to, leading to overfitting. To mitigate this risk and ensure generalizability, a cross-validation strategy should be employed to ensure that the meta learner's training data accurately reflects the true performance of the base learners.

We adopted an experimental approach to empirically evaluate the potential of various models and preprocessing techniques for use in our stacking ensemble. This ensured our selections were informed by the literature review while allowing for independent assessment and validation.

To systematically address the challenges in predicting major oxide compositions from LIBS data, we have devised an approach that integrates model and preprocessing selection, an experimental framework, evaluation and comparison, and the construction of a stacking ensemble.

Firstly, we conducted a literature review and performed preliminary experiments to select a diverse set of machine learning models and preprocessing techniques. These include ensemble learning models, linear and regularization models, neural network models, scaling methods, dimensionality reduction techniques, and data transformations. This selection process is detailed in Section 6.1.

Next, in Section 6.2, we introduce our validation and testing procedures, delineate our data partitioning and cross-validation strategy, and present our evaluation and comparison metrics, all developed to ensure robust performance assessment and generalizability of the models by addressing challenges such as data leakage and uneven distribution of extreme values.

We present the metrics we use to evaluate the performance of our models in Section 6.2.2. These metrics include the RMSE for accuracy and the sample standard deviation of prediction errors for robustness. By evaluating both cross-validation and test set metrics, we ensure a thorough assessment of the models' generalizability and performance on unseen data.

Next, we implemented an optimization framework using Optuna as a foundation [2]. This framework facilitates automated hyperparameter optimization, allowing us to efficiently explore a vast search space of model and preprocessing configurations. The specifics of this framework are discussed in Section 6.3.

Finally, the top-performing configurations are used to construct a stacking ensemble. This ensemble leverages the strengths of multiple models, with a meta-learner trained to optimize the final predictions. The process of constructing and validating this stacking ensemble is described in Section 7.6.

By following this structured approach, we aim to enhance the prediction accuracy and robustness for major oxides in LIBS data, ultimately leading to more reliable and generalizable models.

6.1 Model and Preprocessing Selection

Choosing the right models and preprocessing techniques for LIBS data analysis is a challenging task.

We had several considerations to guide our selection of preprocessing techniques. Firstly, our review of the literature revealed that there seems to be no consensus on a single, most effective normalization method for LIBS data. Therefore, we included traditional normalization methods in our experiments, such as Z-Score Normalization, Min-Max normalization, and Max Absolute Scaling. This approach allowed us to determine which normalization method was most effective for our dataset. Additionally, dimensionality reduction techniques are considered by the literature to be effective techniques for LIBS data due to its high dimensionality. Specifically, PCA has been widely adopted by the spectroscopic community as an established dimensionality reduction technique [31]. However, Pořízka et al. [31] argue that the assumptions of PCA regarding the linearity of the data are only valid up to a certain point, beyond which they break down. They argue that this non-linearity inherent in the data makes Kernel-PCA a valid candidate for LIBS data. Based on their review of the field, and our own review of the literature, not many have studied the effectiveness of Kernel-PCA in the context of LIBS data. Therefore, we decided to include this in our experiments to further assess its potential. In addition to the non-linearity, Pořízka et al. [31] also argue that the assumptions of normality in the data are not always met in LIBS data. For this reason, we decided to include power transformation and quantile transformation in our experiments, as models such as PCA benefit from a normal distribution of the data. We assume that models such as PLS may also benefit from a more Gaussian-like data distribution, given that the model is partly based on PCA.

While these preprocessing techniques are not an exhaustive list, they represent a diverse set of methods. Techniques such as feature selection were not considered in this study to limit its scope and due to time constraints.

We also had several requirements for the model selection. The selected models for experimentation had to be diverse to ensure sufficient breadth in our results, enabling informed decisions about which models to include in the final stacking ensemble pipeline. Additionally, the models had to be suitable for regression tasks. In the absence of research specific to LIBS data, we selected models that have shown promise in other domains. Our literature review found that a variety of models fit these criteria. For example, Anderson et al. [4] demonstrated that models such as GBR, PLS, LASSO, and RF were each effective at predicting different major oxides from LIBS data. Additionally, Shi et al. [34] showed that SVR outperforms PLS regression in predicting Si, Ca, Mg, Fe, and Al using LIBS data. As a result, we included GBR, PLS, LASSO, RF, and SVR in our experiments.

In the neural network domain, El Haddad et al. [10] demonstrated that their 3-layer ANN achieved a relative prediction error below 20% for Ca, Fe, and Al using LIBS data. Similarly, Yang et al. [43] showed that CNN outperformed methods such as Logistic Regression (LR), SVM, and linear discriminant analysis in correctly classifying twelve different types of rocks based on LIBS data. While this example for CNN involves a classification task, CNN can be adapted for regression by changing the loss function and output layer. Based on these factors, we decided to include ANN and CNN in our experiments to further increase the diversity of our model selection.

To further bolster our selection pool, we included models from the same family as those that showed promise in the literature.

XGBoost and NGBoost both belong to the gradient boosting family, but they approach gradient boosting in distinct ways. XGBoost uses advanced algorithmic optimizations, such as regularization, tree pruning, and parallel processing, to improve performance and prevent overfitting. On the other hand, NGBoost focuses on sophisticated probabilistic loss functions, optimizing the natural gradient to model the entire probability distribution of the target variable, making it well-suited for tasks requiring uncertainty estimation and probabilistic forecasting. Given these differences and the limited studies on their application to LIBS data, we decided to include both in our experiments.

Finally, ridge regression, ENet, ETR, and LASSO were included in various studies and showed promising results, even if they were not the top performers in their respective studies. Therefore, we chose to include these in our experiments to further diversify our model selection.

Table 4 summarizes the preprocessing techniques and models selected for our experimentation.

To tackle the challenge of selecting the optimal preprocessing techniques and models, we have developed a hyperparameter optimization framework, which we describe in Section 6.3.

6.2 Validation and Testing Procedures for Model Evaluation

This section describes the validation and testing procedures our experiments follow. Selecting appropriate testing procedures is crucial for ensuring the validity and reliability of the results. For that reason, we delineate a methodological approach that ensures our models are accurate and generalizable. We begin by outlining our general strategy for model evaluation. Next, we describe the procedure used to partition our dataset into training, validation, and testing sets. Finally, we present the evaluation metrics used to assess the performance of our models.

We have chosen to test and evaluate all our experiments using both cross-validation and a separate test set. Evaluating

Table 4. Overview of preprocessing techniques and models.

Normalization / Scaling:
Z-Score Standardization
Min-Max Normalization
Max Absolute Scaling
Robust Scaling
Norm 3
Transformation Methods:
Power Transformation
Quantile Transformation
Dimensionality Reduction Methods:
Principal Components Analysis
Kernel Principal Components Analysis
Model Types:
Linear and Regularized Models:
Partial Least Squares
Support Vector Regression
Elastic Nets
Least Absolute Shrinkage and Selection Operator
Ridge Regression
Ensemble Models:
Random Forest
Gradient Boost Regression
Extra Trees Regression
XGBoost
Natural Gradient Boosting
Neural Networks:
Artificial Neural Networks
Convolutional Neural Networks

results solely on the test set could lead to models that are overly specialized to the test set. This occurs when searching for the optimal configuration of hyperparameters specifically tailored to the test set. For this reason, it is common to use a validation set to tune hyperparameters and evaluate model performance during experimentation. However, further splitting the training set into a validation set exacerbates the challenges of limited data availability, as described in Section 3.2. Our objective is to develop models that demonstrate high accuracy and robustness, even on entirely unseen data. To achieve this, we employ k-fold cross-validation. This allows us to train our models on $k-1$ folds and evaluate them on the remaining fold, which provides a more robust estimate of model performance, and allows us to use all the data for training. Since our approach includes large-scale optimization, we prefer k-fold cross-validation over Leave-One-Out Cross-Validation (LOOCV). LOOCV uses each individual sample as a single test case, resulting in n iterations, where n is the number of samples. In each iteration, models and preprocessors would need to be refit, making this approach too computationally expensive and time-consuming for the scope of our study.

While we employ conventional techniques like holdout sets and k-fold cross validation, the dataset we use imposes additional challenges to the process.

One of the primary challenges is preventing data leakage. As per concentration matrix $\mathbf{C}$ in Section 3, each target only has one ground truth concentration value per oxide. However, each target is shot at multiple locations, resulting in multiple instances of the same target in the dataset, as shown in Table 6. Although the intensity values vary for each location, they fundamentally represent measurements of the same target. If we were to randomly split the dataset, some locations from a target could end up in the testing set while others remain in the training set. This would cause data leakage, as the testing set would no longer consist solely of unseen targets. To prevent this, we ensure that each target is represented only once in the dataset by grouping data from all locations on a given target.

Furthermore, the limited availability of data poses another significant challenge. The dataset we use consists of 408 samples, which is relatively large by LIBS standards. However, there are only a few samples with concentration values for the oxides in the targets that are either very high or very low compared to the rest of the data, which we refer to as *extreme values*. These infrequent high and low concentration values can be problematic. If such values end up in the test set, the model may be evaluated on data points outside the range it was trained on. This situation can lead to an inaccurate assessment of the model's performance, as it might not handle these uncommon concentration ranges effectively.

When performing a random split of the dataset into multiple folds for cross-validation, as well as for training and testing sets, this small number of extreme values can result in an uneven distribution. The presence or absence of these extreme values in any given fold can heavily influence the model's performance metrics. If extreme values are disproportionately allocated to the testing set, the resulting model may struggle to generalize accurately. This uneven distribution can lead to models that perform well on the majority of the data but fail to predict accurately for these extreme concentration values. Conversely, if the scarce extreme values are disproportionately assigned to the training set, the model may become overly specialized in handling these extreme values, potentially leading to overfitting. This means the model might perform well on the training set, including the extreme values, but fail to generalize effectively to new, unseen data, especially if the test set does not contain similar extreme values. This could result in an inaccurate assessment of the model's performance, as the test set would not adequately challenge the model's ability to predict across the full range of data variability.

Figure 4 illustrates the distributions of various oxide concentrations in our dataset. Across all oxides, there is a general pattern of skewed distributions, with concentrations heavily weighted towards lower values. This is particularly notable in TiO_2, FeO_T, MgO, CaO, and Na_2O. SiO_2 and Al_2O_3 show

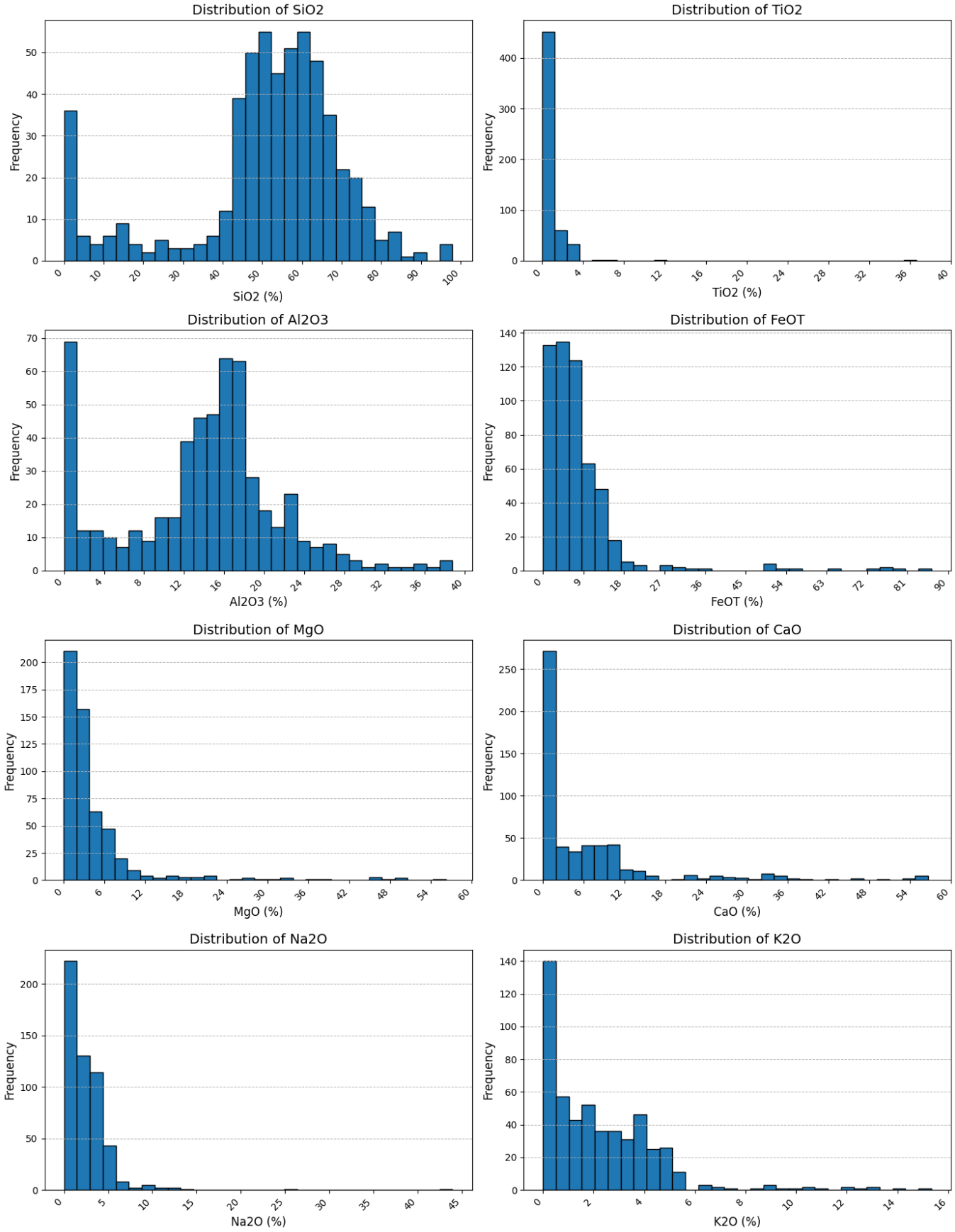

Fig. 4. Distributions of various oxide concentrations in the dataset. The histograms show the frequency of concentration values for SiO_2, TiO_2, Al_2O_3, FeO_T, MgO, CaO, Na_2O, and K_2O.

more variability, with SiO_2 exhibiting a bimodal distribution. These distributions confirm the presence of extreme values across all oxides, which are significantly overrepresented or underrepresented, further complicating the model training process.

This necessitates careful dataset partitioning to ensure that the model training process accounts for these challenges, improving the generalizability and robustness of the models.

In Section 5, we described how we ensured representation of extreme compositions in both the training and testing sets by automatically identifying the n largest and smallest samples by concentration range for each oxide and reserving them for the training set. We then performed a random split on the remaining dataset, resulting in a final train/test split of 80%/20%. In this process, we also employ a rudimentary procedure to prevent data leakage, ensuring that each target was only present once in the training set. The baseline did not employ cross-validation, as our goal was to replicate the MOC model that was presented in Clegg et al. [7]. We note that this procedure is insufficient to support the testing and validation strategy we have laid out above, as it does not support k-fold cross-validation. A random k-fold split of the training data would not account for the uneven distribution of extreme values across the folds, and would furthermore cause data leakage between the folds. Moreover, the procedure failed to consider the concentration of each oxide individually, instead aggregating concentrations across all oxides. This represents a significant limitation, as it attempts to generate a uniform test set for each oxide, thereby neglecting the unique distribution characteristics of individual oxides. Therefore, a more sophisticated procedure is needed to ensure that the data partitioning accounts for these challenges.

6.2.1 Dataset Partitioning. To ensure rigorous evaluation of our models and to address the challenges of data leakage and uneven distribution of extreme values, we have implemented a customized k-fold data partitioning procedure. This approach divides the dataset into k folds, which are used to define cross-validation datasets, as well as a training set and a test set. The procedure ensures that all data points from a given target are only present in one of the k folds, effectively preventing the aforementioned data leakage. Additionally, it ensures that extreme values are handled by redistributing them evenly across the training folds, preventing any single fold from being disproportionately influenced by these values.

The procedure outlined in Algorithm 1 begins by setting a random seed for reproducibility if one is provided (Line 1). This ensures that the results are consistent across different runs of the algorithm. Next, the dataset is processed to remove any duplicate entries based on the group column g and then sorted by the target column t (Line 2). This step ensures that each group is uniquely identified and ordered appropriately. The dataset we illustrate in Table 6 would require a group column g of `"Target"` to group the data by target. The target column t refers to the column with the target variable, which would be the oxide for which we are predicting the concentration, for example, SiO_2. By sorting the dataset by the target column t, we ensure that the data is ordered by the target concentration values in ascending order.

Algorithm 1 Data Partitioning With Extreme Value Handling

Require: Dataset $\mathbf{D}$, group column g, target column t, number of splits k, percentile p, random seed *seed*
Ensure: Training and validation sets for cross-validation $\mathbf{T}_{\text{cv}}$, training set $\mathbf{D}_{\text{train}}$, and test set $\mathbf{D}_{\text{test}}$
1: Set random seed for reproducibility if seed is not None
2: Remove duplicate entries based on g and sort by t
3: Assign fold numbers sequentially from 0 to $k-1$ to unique targets
4: **if** extreme values handling is enabled **then**
5: Identify extreme values at percentiles p and $1-p$
6: Reassign extreme values to folds 0 to $k-2$
7: **end if**
8: Merge fold assignments information into the original dataset
9: Split dataset into test set $\mathbf{D}_{\text{test}}$ (fold $k-1$) and remaining data $\mathbf{D}_{\text{train}}$
10: Create $k-1$ training and validation sets
11: **for** each fold i from 0 to $k-2$ **do**
12: $\mathbf{T}_{\text{train}}[i] \leftarrow \mathbf{D}_{\text{train}} \setminus \text{fold}_i$
13: $\mathbf{T}_{\text{val}}[i] \leftarrow \text{fold}_i$
14: Append $(\mathbf{T}_{\text{train}}[i], \mathbf{T}_{\text{val}}[i])$ to $\mathbf{T}_{\text{cv}}$
15: **end for**
16: Remove fold column from all datasets
17: **return** $\mathbf{T}_{\text{cv}}, \mathbf{D}_{\text{train}}, \mathbf{D}_{\text{test}}$

Fold numbers are then assigned sequentially using a modulo operation to ensure a random-like distribution of the unique targets across the folds (Line 3). This means that, while the assignment process follows a sequence, the resulting distribution of targets is effectively randomized. Fold numbers start in 0 and go up to $k-1$, as implied by the modulo operation. If a percentile p is provided, extreme value handling is enabled, and the algorithm identifies the top and bottom percentiles of the target values (Line 5). These extreme values are then reassigned to the training folds (0 to $k-2$), ensuring they are as evenly distributed as possible across these folds (Line 6).

The fold assignments are then merged into the original dataset, as described in Line 8. Essentially, this step enables the partitioning steps that follow, by ensuring each data item has an associated fold number. Following this, the dataset is divided into a train and test set, as outlined in Line 9. The test set consists of the data points assigned to fold $k-1$, and the remaining folds forms the training set. The training data is further divided into $k-1$ sets for cross-validation. For each fold i where $i \in \{0, 1, \ldots, k-2\}$, we create a cross-validation training set $\mathbf{T}_{\text{train}}[i]$ by excluding the i-th fold from the set of $k-1$ folds, and use the i-th fold as the validation set $\mathbf{T}_{\text{val}}[i]$.

These pairs of training and validation sets are then appended to the list of cross-validation sets $\mathbf{T}_{\text{cv}}$ (Line 10).

Finally, the fold indicator column is removed from all datasets before returning the final partitions (Line 16). The fold indicator column was added to keep track of which data points belong to which folds, which is crucial for ensuring that data points are correctly partitioned into their respective training and test sets during cross-validation. This cleanup step ensures that the fold information does not interfere with subsequent data processing or model training.

The final output of this procedure consists of:

- A set of tuples $\mathbf{T}_{\text{cv}}$, where each tuple contains a training set and a validation set.
- The overall training set $\mathbf{D}_{\text{train}}$, consisting of all the data points not in the test set.
- The test set $\mathbf{D}_{\text{test}}$, distinct from the training set.

The data partitioning procedure does not modify the original dataset; it merely partitions it. For that reason, each of the datasets that are returned has the same structure as shown in Table 6.

Given that the data partitioning procedure aims to distribute extreme concentration values evenly among the training folds while minimizing their presence in the test set, it is crucial to determine an optimal value of p that minimizes the number of extreme values in the test set while still maintaining its general representativeness. By general representativeness, we mean ensuring that the test set reflects the general distribution of the dataset without being skewed by extreme values. This balance is essential for accurately assessing the model's performance on typical data points.

Our method is inspired by the approach described by Anderson et al. [3]. They employed a similar strategy to assess the performance of their PLS model, using k-fold cross-validation and a separate test set. Their process involved dividing the full set of laboratory data into five folds, using four for cross-validation and combining them as the final training set, while the fifth fold served as a test set. For consistency, we also use $k = 5$ for our data partitioning. Given that the k-th fold is used as the test set, having $k = 5$ results in 4 folds for cross-validation.

Additionally, by using $k = 5$ folds, we have effectively chosen an 80%/20% split between the training and testing datasets. In our experience, this ratio maximizes the training set's capacity for effective model learning while ensuring that the testing set is sufficiently representative to provide an accurate assessment of the model's performance on new data. Allocating too much data to the testing set could compromise the comprehensiveness of the training set, undermining the model's ability to generalize effectively due to the limited availability of data.

6.2.2 Evaluation Metrics. To evaluate the performance of these models, we will use the RMSE to measure accuracy and the sample standard deviation of prediction errors to assess robustness. We define accuracy as the ability of a model to predict the composition of major oxides in geological samples, while robustness refers to the stability of these predictions across samples.

The metric used to evaluate the accuracy of the models is the RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}_i - \hat{\mathbf{v}}_i)^2}$$

where $\mathbf{v}_i$ is the vector of actual oxide concentrations for the i-th sample, $\hat{\mathbf{v}}_i$ is the corresponding vector of predicted oxide concentrations, and n is the total number of samples. This measure quantifies the average magnitude of the prediction error across all predicted values.

Robustness is evaluated using the sample standard deviation of prediction errors:

$$\sigma_{error} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(e_i - \bar{e})^2}$$

where $e_i = \mathbf{v}_i - \hat{\mathbf{v}}_i$ and $\bar{e}$ is the mean error. A lower standard deviation indicates a more robust model across different samples.

These metrics are calculated for each fold and averaged across all folds to provide comprehensive indicators of model accuracy and variability. In addition, we also compute the metrics for the test set to provide a measure of the model's performance on unseen data. Therefore, we have the following metrics for each experiment:

(1) **Fold-specific RMSE and Standard Deviation:** For each of the k folds, we calculate both the RMSE and standard deviation, denoted as `rmse_cv_n` and `std_dev_cv_n`, where `n` ranges from 1 to k.

(2) **Average RMSE and Standard Deviation:** The overall cross-validation RMSE (`rmse_cv`) and standard deviation (`std_dev_cv`) are computed as the mean of the fold-specific values. Formally, if `rmse_cv_n` and `std_dev_cv_n` represent the RMSE and standard deviation for the n-th fold respectively, then:

$$\texttt{rmse\_cv} = \frac{1}{k}\sum_{n=1}^{k}\texttt{rmse\_cv\_n}$$

and

$$\texttt{std\_dev\_cv} = \frac{1}{k}\sum_{n=1}^{k}\texttt{std\_dev\_cv\_n}$$

where k is the total number of folds.

(3) **Test Set RMSE and Standard Deviation:** The RMSE and standard deviation are also computed for the test set, denoted as `rmsep` and `std_dev`, to provide a measure of the model's performance on unseen data.

6.2.3 Discussion of Testing and Validation Strategy. Our proposed approach to data partitioning addresses several critical challenges and represents a deliberate trade-off to improve the reliability of our model evaluation. The first challenge concerned data leakage. We believe our method effectively handles this issue without significant trade-offs. The second

challenge we presented is the trade-off between having a representative training set that includes extreme values, thereby avoiding overfitting, and ensuring the test set is sufficiently challenging to accurately assess the model's generalization across the full range of data variability. We will further examine this trade-off below.

Our method for partitioning mitigates the risk of uneven distribution of extreme values, which can disproportionately affect model performance metrics. If extreme values are unevenly distributed between the training and test sets, the evaluation of the model can be heavily skewed, leading to unreliable and misleading performance metrics. By redistributing extreme values evenly across the training folds, we ensure a more balanced and fair assessment of the model's capabilities during cross-validation. However, excluding extreme values from the test set may limit the test set's ability to challenge the model fully. Our method for handling extreme values means that the test set does not include samples outside the range seen in the training set. Although the test set may be less representative of the full range of data, particularly concerning the rare extreme values, the evaluation focuses on the model's ability to generalize from the training data to new data within the typical distribution range. Essentially, this trade-off ensures that the model is evaluated on data points within the range the model was trained on, thereby providing a fairer assessment. However, it is important to recognize the limitation of this approach: excluding extreme values from the test set means we can only confidently assess the model's performance within the test set's range. If predictions fall outside this range, we cannot reliably assert their accuracy, as the model has not been fully evaluated on such data. This could potentially reduce the model's usefulness in scenarios where predictions on extreme values are critical. Furthermore, since our data partitioning method allocates the most extreme values to the training data, the testing data tends to be closer to the mean of the data distribution, making it easier to predict. In practice, this results in lower `rmsep` and `std_dev` values compared to the cross-validation metrics. This further emphasizes the importance of evaluating the model's performance using both the cross-validation metrics (`rmse_cv` and `std_dev_cv`) and the test set metrics (`rmsep` and `std_dev`).

Therefore, although our approach may render the test set less representative of the full dataset, it is a deliberate trade-off aimed at achieving a more accurate and reliable evaluation of the model's generalization performance. By evaluating with both cross-validation and a separate test set, we ensure that the model both generalizes well and performs well under typical conditions. Cross-validation allows us to evaluate the model's performance across the entire dataset, including extreme values, while the test set provides a measure of the model's performance on unseen, typical data. This combination of cross-validation and a separate test set provides a comprehensive assessment of the model's performance, ultimately helping to ensure that the model is both robust and accurate.

In our initial and optimization experiments, we prioritize cross-validation metrics to evaluate the models. This strategy mitigates the risk of overfitting to the test set by avoiding a bias towards lower RMSEP values. Conversely, for the stacking ensemble experiment, we emphasize test set metrics to comprehensively assess the ensemble's performance, while still considering cross-validation metrics. Using cross-validation for initial model selection and tuning experiments aligns with standard machine learning conventions[13]. In the initial experiment, cross-validation metrics serve as thresholds for model selection. During the optimization phase, only cross-validation metrics guide the search for optimal hyperparameters. For the stacking ensemble experiment, both cross-validation and test set metrics are evaluated, with a primary focus on the RMSEP metric. This approach aims to make our final model accurate, robust, and generalizable to unseen data, providing a balanced evaluation through both cross-validation and test set metrics.

6.3 Optimization Framework

One of the primary challenges in developing a stacking ensemble is determining the optimal choice of base estimators. Wolpert [42] highlighted that this can be considered a 'black art' and that the choice usually relies on intelligent guesses. In our case, this problem is further exacerbated by the fact that the optimal choice of base estimator may vary depending on the target oxide. The complexity of the problem is increased because different oxides require different models, and the optimal preprocessing techniques will depend on both the model and the specific oxide being predicted. Due to the challenges highlighted in 3.2, namely high dimensionality, multicollinearity, and matrix effects, it is difficult to determine which configuration is optimal. Selecting the appropriate preprocessing steps for each base estimator is essential, as incorrect preprocessing can significantly degrade performance and undermine the model's effectiveness. Furthermore, choosing the right hyperparameters for each base estimator introduces additional complexity, as these decisions also significantly impact model performance and must be carefully tuned for each specific oxide. Some estimators might require very little tuning to achieve accurate and robust predictions, while others might require extensive tuning, depending on the target oxide. For instance, simpler approaches like ENet and ridge regression may quickly reach their optimal performance with minimal hyperparameter adjustments. However, due to their simplicity, they often fail to capture the complex patterns in the data that more advanced models can, making them less competitive despite their ease of tuning. In contrast, more complex models like CNN or GBR involve both a larger number of hyperparameters and architectural considerations that need fine-tuning to perform well. The extent of tuning required is also influenced by the characteristics of the target oxide, such as its data distribution, noise levels, and feature interactions. These factors can affect how sensitive an estimator is to its hyperparameters. Finally,

hyperparameters cannot be considered in isolation, because depending on the preprocessing steps applied to the data, the optimal hyperparameters may vary. Given these complexities, we need a systematic approach to determine the optimal configuration of hyperparameters and preprocessing steps tailored to each estimator and oxide.

To guide this process we have developed a working assumption. Specifically, we assume that selecting the top-n best pipelines for each oxide, considering different preprocessors and models for each pipeline, will result in the best pipelines for a given oxide in our stacking ensemble. Here, n is a heuristic based on the results and *best* is evaluated in terms of the metrics outlined in Section 6.2.2. Additionally, each permutation will utilize our proposed data partitioning and cross-validation strategy outlined in Section 6.2. Utilizing our proposed data partitioning and cross-validation strategy, along with the aforementioned evaluation metrics, will ensure that the top-n pipelines align with our goals of generalization, robustness, and accuracy outlined in Section 3. This narrows our focus to three key tasks: selecting suitable preprocessors and models, finding the optimal hyperparameters, and devising a guided search strategy to evaluate various permutations and identify the top-n pipelines for each oxide. First, we curated a diverse set of models and preprocessing techniques, as detailed in Section 6.1. Next, we developed an optimization framework to systematically explore and optimize these pipeline configurations, which will be described in the following section.

6.3.1 The Framework. To systematically explore and optimize pipeline configurations, the search process should be guided by an objective function. Based on the evaluation process outlined in Section 6.2, whereby we argue that solely evaluating on the RMSEP may lead to misleading and poor results, we define the objective function we wish to optimize as a multi-objective optimization on minimizing the `rmse_cv` and `std_dev_cv`.

Given these goals, traditional methods like grid search and random search could be used, but they often fall short due to several inherent limitations. Grid search involves exhaustively evaluating all possible combinations of hyperparameters within specified ranges. While thorough, this method quickly becomes computationally prohibitive as the number of hyperparameters increases. The expansion of the search space, driven by the increasing number and finer granularity of hyperparameters, renders the approach impractical.

Random search, on the other hand, selects hyperparameter values at random within predefined ranges. It is generally more efficient than grid search and can cover a broader area of the hyperparameter space. However, random search can miss optimal regions, especially in high-dimensional spaces where the probability of sampling near-optimal configurations by chance is low.

These limitations make the traditional methods unsuitable for our problem and highlight the need for a more sophisticated optimization method. Both grid search and random search could be enhanced using adaptive techniques, such as Bayesian optimization, to greatly improve on these approaches. However, while very feasible, implementing such enhancements and integrating it with our existing tooling would be too time-consuming. The ambition was therefore to find tools that would provide similar or better hyperparameter optimization capabilities, while being easy to integrate with our existing framework. For this reason we chose to use Optuna as the basis for our optimization framework[2].

Optuna provides well-defined abstraction which allowed us to more quickly construct a framework that helped us efficiently explore and optimize pipeline configurations. Optuna provides Bayesian optimization search algorithms, but with additional configurable parameters that allowed us to customize the search process to our specific needs.

Using Optuna as the foundation for our optimization framework, we designed a comprehensive system for LIBS data that handles the entire process, from CCS data to partitioning, cross-validation, and hyperparameter optimization. Using this framework, it is possible to find the best configurations by optimizing the objective function: minimizing the `rmse_cv` and `std_dev_cv`.

The framework we developed can be divided into two main components. A function responsible for running and managing the optimization process, as seen in Algorithm 2(Optimizer), and a function responsible for measuring the objective, as seen in Algorithm 3(Objective).

The purpose of the Optimizer is to perform and facilitate the optimization process, doing so for each oxide and model combination. By managing the optimization process in this way, we obtain the flexibility to evaluate each model separately with different preprocessors and hyperparameters. This means that each model is evaluated fairly against each oxide and that the resulting configurations are optimized specifically for the model and oxide in question. Our assumption is that this approach will best identify the top-n pipelines for use in our stacking ensemble.

To manage the optimization process, the function receives the number of trials to run, a list of models, and a list of oxides, as seen in line 0, and initializes the sampler, as seen in line 1.

The sampler is responsible for managing the search space of the hyperparameters for the optimization process. This means that any hyperparameters being evaluated, for any preprocessor or model, will be managed by this sampler, which allows us to optimize for all hyperparameters at the same time. Optuna provides several options for samplers that have different characteristics and each have their strengths and weaknesses. However, because we require multi-objective optimization, this naturally limits the choice of sampler to those that support this. For our framework, we chose to use the Tree-structured Parzen Estimator (TPE) sampler due to its stated optimization efficiency and its ability to handle all use cases. Additionally, the TPE sampler allows us to control how many

of the trials to be reserved for exploration, which is beneficial when the search space is large[2].

Guiding the optimization process is the Objective function, which evaluates the performance of each trial. In our case we are seeking to minimize the `rmse_cv` and `std_dev_cv`, as mentioned previously.

The role of the Objective function is to provide the metric data to an *optimize* function, seen in Line 4. As we step through each oxide and model in Lines 2 to 3, we call the `optimize` function with the number of trials to run and the Objective function. The number of trials is an important parameter, as it specifies the number of iterations the optimization framework will execute to refine and optimize a given model. In an ideal scenario, this number would be very high to ensure that the optimization process has identified the best possible configuration. However, depending on the number of models in consideration, a high number of trials can quickly become computationally prohibitive with our approach. The `optimize` function uses the number of trials and the objective of minimizing the `rmse_cv` and `std_dev_cv` to manage the optimization process and mediating the metrics returned by the Objective function to the sampler.

This leads us to the Objective function which, as previously mentioned, returns the evaluation metrics for the given trial.

To this function we supply the current model m, the target oxide o, and the sampler.

In Lines 1 to 8, we instantiate the model and the preprocessors with the hyperparameters sampled by the sampler. The sampling process is being guided by the objective function via the returned metrics from the previous trial. In our setup, we always require that a scaler is instantiated to ensure that the data is correctly scaled. However, to measure the impact of data transformation and dimensionality reduction, we allow for these to be initialized as an identity function, which does not modify the data. Whether to instantiate a specific preprocessor or an identity function is determined by the sampler.

Once the preprocessors are instantiated, we construct a pipeline of these to ensure that the data is processed in the order they are defined, seen in Line 9. The order of preprocessing steps is crucial due to their interdependence: scaling standardizes the feature ranges, ensuring that subsequent transformations are applied uniformly. Similarly, dimensionality reduction techniques typically also produce better results if the data has been scaled. However, it may not always be advantageous or yield better results if the data has already been transformed. As such, we allow for the optimization framework to optionally use these if they are deemed to be beneficial.

In Lines 10 to 12, we fetch the data, apply our data partitioning strategy to generate four cross-validation sets, a training set and a test set, and apply the preprocessing to the datasets. This partitioning is applied with respect to the current oxide. The purpose of fetching the data for each trial is to ensure no modifications leak through trials, corrupting the dataset over time. This prevents any form of double preprocessing from occurring, which would lead to potential issues.

As mentioned in Section 6.2, we use both cross-validation and a test set to evaluate the model. This can be seen in Line 13 and Lines 16 to 17, where cross-validation, training, and evaluation are also performed with respect to the current oxide. It is important to note that in practice, the model m is being reinstantiated in each iteration of the cross-validation, and again before the model is trained, so no learned parameters are carried over between them.

Once a trial is complete, the metrics are returned in Line 19 to the `optimize` function in the Optimizer, which then determines the next steps in the optimization process.

In summary, using this framework we are able to systematically explore and optimize preprocessing, model and hyperparameter configurations for each model on a per-oxide basis. This allows us to identify the top-n pipelines for each oxide, which we can then use in our stacking ensemble.

Algorithm 2 Optimizer

Require: Number of Trials N, List of Models M, List of Target Oxides O

Ensure: The optimization process is run for each model and oxide.

```
1:  Initialize: sampler ← Sampler(sampler_params)
2:  for each oxide o in O do
3:      for each model m in M do
4:          optimize(N, lambda()
5:              return objective(
6:                  m,
7:                  o,
8:                  sampler
9:          ))
10:     end for
11: end for
```

7 EXPERIMENTS & RESULTS

This section outlines the experimental design used to identify the top-n models to be used for our stacking ensemble. We begin with a description of the prerequisite data preparation necessary for all experiments, followed by an overview of the hardware and software used. Then we provide a visual and statistical analysis to confirm that our data partitioning method works as intended, and effectively separates the extreme and non-extreme values. Next, we outline the design of our initial experiment in Section 7.4, which provides a preliminary assessment of the models selected in Section 6.1. The results of this initial experiment are then presented and discussed in Section 7.4.1. We then describe the design of our optimization experiment in Section 7.5, which leverages our optimization framework to identify the top-n models. The results are then presented and discussed in Section 7.5.1. Finally, we use the identified models to construct a stacking ensemble, which is then evaluated and compared to the individual models and our baseline in Section 7.6.

Algorithm 3 Objective

Require: Model m, Target Oxide o, Sampler `sampler`
Ensure: Returns $rmse_{cv}$ and std_dev_{cv} for each trial

$hp \leftarrow$ `sample_hyperparameters(sampler)`
$m \leftarrow$ `instantiate_model(`m, hp`)`

$s\_params \leftarrow$ `sample_scaler_params(sampler)`
$s \leftarrow$ `instantiate_scaler(`s_params`)`
$t\_params \leftarrow$ `sample_transformer_params(sampler)`
$t \leftarrow$ `instantiate_transformer(`t_params`)` **or** `Identity`
$dim\_params \leftarrow$ `sample_dim_reduction_params(sampler)`
$dim \leftarrow$ `instantiate_dim_reduction(`dim_params`)` **or** `Identity`

$pipeline \leftarrow [s, t, dim]$

Dataset: $D \leftarrow$ `get_data()`
$T_{cv}, D_{train}, D_{test} \leftarrow$ apply data partitioning to D

$T_{cv}, D_{train}, D_{test} \leftarrow$ apply pipeline to $T_{cv}, D_{train}, D_{test}$

$CV_{metrics} \leftarrow$ `cross_validate(`m, T_{cv}, o`)`
$rmse_{cv} \leftarrow$ `mean(`$CV_{metrics}$`.rmse_values)`
$std\_dev_{cv} \leftarrow$ `std(`$CV_{metrics}$`.rmse_values)`

$m' \leftarrow$ `train(`m, D_{train}, o`)`
$rmsep, std\_dev_{test} \leftarrow$ `evaluate(`m', D_{test}, o`)`

`store_metrics(`t, m, $pipeline$, $rmse_{cv}$, std_dev_{cv}, $rmsep$, std_dev_{test}`)`

return $rmse_{cv}$, std_dev_{cv}

7.1 Data Preparation

The first step in our methodology is to prepare the datasets for model training and evaluation. As mentioned in Section 4.1, the data used in this study was obtained from NASA's PDS and consists of CCS data and major oxide compositions for various samples.

The initial five shots from each sample are excluded because they are usually contaminated by dust covering the sample, which is cleared away by the shock waves produced by the laser [7]. The remaining 45 shots from each location are then averaged, yielding a single spectrum s per location l in the `Averaged Intensity Tensor` (Tensor 3), resulting in a total of five spectra for each sample.

At this stage, the data still contains noise at the edges of the spectrometers. These edges correspond to the boundaries of the three spectrometers, which collectively cover the UV, VIO, and VNIR light spectra. The noisy edge ranges are as follows: 240.811-246.635 nm, 338.457-340.797 nm, 382.138-387.859 nm, 473.184-492.427 nm, and 849-905.574 nm. In addition to being noisy regions, these regions do not contain any useful information related to each of the major oxides. Consequently, these regions are masked by zeroing out the values, rather than removing them, as they represent meaningful variation in the data [7].

Additionally, as a result of the aforementioned preprocessing applied to the raw LIBS data, negative values are present in the CCS data. These negative values are not physically meaningful, since you cannot have negative light intensity [15]. Similar to the noisy edges, these negative values are also masked by zeroing out the values.

We transpose the data so that each row represents a location and each column represents a wavelength feature. Each location is now represented as a vector of wavelengths, with the corresponding average intensity values for each wavelength. These vectors are then concatenated to form a tensor, giving us the full `Averaged Intensity Tensor`.

For each sample, we have a corresponding set of major oxide compositions in weight percentage (wt%). These compositions are used as the target labels for the machine learning models. An excerpt of this data is shown in Table 5. While the *Target*, *Spectrum Name*, and *Sample Names* are part of the dataset, our analysis focuses primarily on the *Sample Names*. The concentrations of the eight oxides SiO_2, TiO_2, Al_2O_3, FeO_T, MnO, MgO, CaO, Na_2O, and K_2O represent the expected values for these oxides in the sample, serving as our ground truth. The *MOC total* is not utilized in this study.

The major oxide weight percentages are appended to the matrix of spectral data, forming the final dataset. This dataset is shown in Table 6. The *Target* column corresponds to the sample name, while the *ID* column contains the unique identifier for each location.

7.2 Experimental Setup

Experiments were conducted on a machine equipped with an Intel Xeon Gold 6242 CPU, featuring 16 cores and 32 threads. The CPU has a base clock speed of 2.80 GHz and a maximum turbo frequency of 3.90 GHz. The system has 64 GB of RAM and runs on Ubuntu 22.04.2 LTS. Models were implemented using Python 3.10.11. The primary libraries used were Scikit-learn 1.4.2, XGBoost 2.0.3, Torch 2.2.2, NumPy 1.26.4, Pandas 2.2.1, Keras 3.2.1 and Optuna 3.6.1. Additionally, all experiments were run using the hyperparameter optimization tool described in Section 6.3.

7.3 Visual and Statistical Analysis of SiO_2 Distribution in Partitioned Data

This section provides a detailed visualization and statistical analysis of the SiO_2 concentration distribution across data partitions, following the customized k-fold data partitioning procedure described in Section 6.2. This analysis is conducted to validate the consistency of our data partitioning method and

Table 5. Excerpt from the composition dataset (from Houmann et al. [15]).

Target	Spectrum Name	Sample Name	SiO_2	TiO_2	Al_2O_3	FeO_T	MnO	MgO	CaO	Na_2O	K_2O	MOC total
AGV2	AGV2	AGV2	59.3	1.05	16.91	6.02	0.099	1.79	5.2	4.19	2.88	97.44
BCR-2	BCR2	BCR2	54.1	2.26	13.5	12.42	0.2	3.59	7.12	3.16	1.79	98.14
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
TB	—	—	60.23	0.93	20.64	11.6387	0.052	1.93	0.000031	1.32	3.87	100.610731
TB2	—	—	60.4	0.93	20.5	11.6536	0.047	1.86	0.2	1.29	3.86	100.7406

Table 6. Excerpt from the final dataset (values have been rounded to two decimal places for brevity).

240.81	···	425.82	425.87	···	905.57	SiO_2	TiO_2	Al_2O_3	FeO_T	MgO	CaO	Na_2O	K_2O	Target	ID
0	···	1.53e+10	1.62e+10	···	0	56.13	0.69	17.69	5.86	3.85	7.07	3.32	1.44	jsc1421	jsc1421_2013_09_12_211002_ccs
0	···	1.28e+10	1.30e+10	···	0	56.13	0.69	17.69	5.86	3.85	7.07	3.32	1.44	jsc1421	jsc1421_2013_09_12_211143_ccs
0	···	1.87e+10	1.83e+10	···	0	56.13	0.69	17.69	5.86	3.85	7.07	3.32	1.44	jsc1421	jsc1421_2013_09_12_210628_ccs
0	···	1.77e+10	1.78e+10	···	0	56.13	0.69	17.69	5.86	3.85	7.07	3.32	1.44	jsc1421	jsc1421_2013_09_12_210415_ccs
0	···	1.75e+10	1.79e+10	···	0	56.13	0.69	17.69	5.86	3.85	7.07	3.32	1.44	jsc1421	jsc1421_2013_09_12_210811_ccs
0	···	5.52e+10	3.74e+10	···	0	57.60	0.78	26.60	2.73	0.70	0.01	0.38	7.10	pg7	pg7_2013_11_07_161903_ccs
0	···	5.09e+10	3.41e+10	···	0	57.60	0.78	26.60	2.73	0.70	0.01	0.38	7.10	pg7	pg7_2013_11_07_162038_ccs
0	···	5.99e+10	3.97e+10	···	0	57.60	0.78	26.60	2.73	0.70	0.01	0.38	7.10	pg7	pg7_2013_11_07_161422_ccs
0	···	5.22e+10	3.47e+10	···	0	57.60	0.78	26.60	2.73	0.70	0.01	0.38	7.10	pg7	pg7_2013_11_07_161735_ccs
0	···	5.29e+10	3.62e+10	···	0	57.60	0.78	26.60	2.73	0.70	0.01	0.38	7.10	pg7	pg7_2013_11_07_161552_ccs
⋮	···	⋮	⋮	···	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

to provide a visual understanding of the resulting data distribution. The analysis focuses on SiO_2 as a representative example. Similar analyses have been conducted for other oxides, and the resulting plots are shown in Appendix A.2.

As discussed in Section 6.2, it is crucial to determine an optimal value of p for the data partitioning algorithm. This value should minimize the number of extreme values in the test set while ensuring the test set remains representative. Our approach is to select the lowest p that excludes extreme values from the test set while maximizing its general representativeness. This approach helps avoid overestimating the model's performance due to extreme values while keeping the test set reflective of the majority of the data distribution. We developed a web-based platform to evaluate the performance of the data partitioning algorithm for different values of p, as shown in Figure A.1 in Appendix A.1. Using the platform, we conducted analyses for various values of p and determined that the optimal value is $p = 5\%$. Furthermore, the 5% threshold has proven effective across all oxides within our dataset, and consequently, it has been uniformly applied throughout this study. This methodology is adaptable and can be employed to determine an optimal value for p tailored to various targets, depending on the specific dataset in use. Finally, we use a $k = 5$ for our data partitioning algorithm. This results in five folds, with four folds used for cross-validation training and one fold designated as the test set. Consequently, the full training set comprises the first four folds.

Figures 5 and 6 illustrate the histograms and Kernel Density Estimation (KDE) curves for SiO_2 concentrations in each training fold, the test set, and their combined distributions. The consistent histograms and KDE curves across different training folds indicate that the data distribution within each fold closely matches the overall distribution, confirming their consistency and representativeness.

Figure 7 contrasts the SiO_2 concentration distribution before and after data partitioning. The left plot shows the original distribution, while the right plot displays the fold-assigned distribution, color-coded by fold. This visualization highlights that the partitioning strategy maintains the overall data distribution while ensuring balanced representation across folds.

To further validate our visual analysis, we can examine quantitative measures such as the means and standard deviations of SiO_2 concentrations across the folds and the overall dataset.

Figure 8 shows that the means and standard deviations of SiO_2 concentrations for each fold, as well as the combined training set, are consistent with those of the full dataset. This quantitative consistency supports the visual evidence that each training fold is representative of the entire dataset. Furthermore, we observe that the standard deviation in the training sets is higher than in the test set, which is expected given the reassignment of extreme values to the training sets.

In conclusion, the visual and statistical analyses presented in this section confirm that our customized k-fold data partitioning procedure effectively maintains balanced and representative distributions across all folds. This consistency is crucial for the robustness and generalizability of our models, as discussed in Section 6.2. The alignment between the visual evidence and the quantitative measures reinforces the reliability of our approach, ensuring that our models are well-equipped to perform accurately on unseen data.

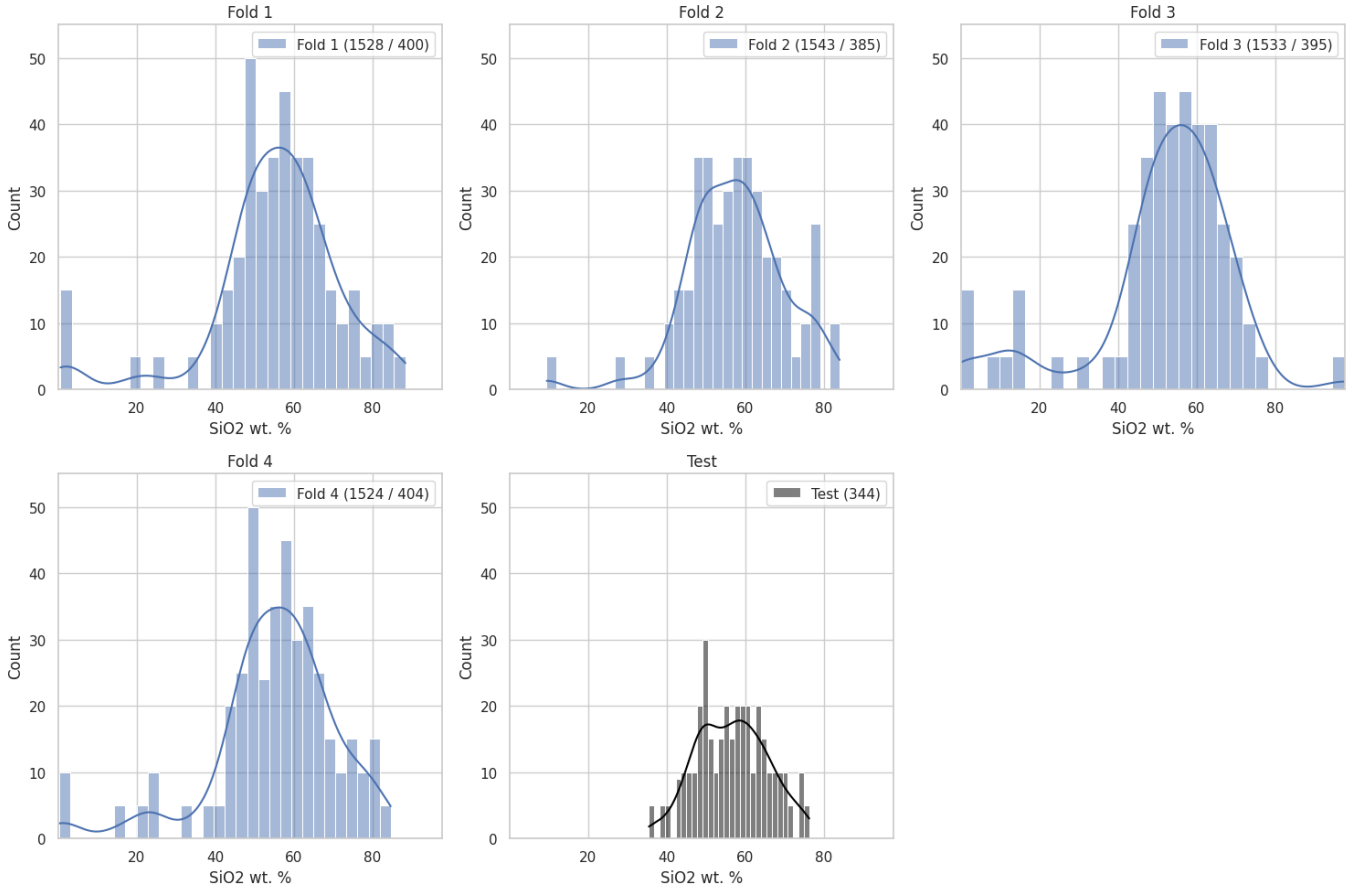

Fig. 5. Histogram and KDE of SiO_2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents SiO_2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

7.4 Initial Experiment

As described in Section 6, we conducted an initial experiment to evaluate the performance of various machine learning models on the prediction of major oxide compositions from our LIBS dataset. These experiments aimed to provide a preliminary assessment of the models' performance, allowing us to identify the most promising models for further evaluation and inclusion in our stacking ensemble. All models were trained on the same preprocessed data using the Norm 3 preprocessing method described in Section 4.2.5. This ensured that the models' performance could be evaluated under consistent and comparable conditions.

Furthermore, all experiments used our data partitioning and were evaluated using our testing and validation strategy, as described in Section 6.2. To ensure as fair of a comparison between models as possible, all models were trained using as many default hyperparameters as possible, and those hyperparameters that did not have default options were selected based on values found in the literature. However, due to the nature of the neural network models' architecture, some extra time was spent on tuning the models to ensure a fair comparison. This included using batch normalization for the CNN model, as early assessments showed that this was necessary to produce reasonable results. Finally, we evaluated each model once per oxide given the selected configuration of hyperparameters. As stated, the goal of this experiment was merely to get an initial indication of the performance of the models.

The hyperparameters used for the models in the initial experiment can be found in the Appendix A.3.

7.4.1 Results for Initial Experiment. Table 8 presents the results of the initial experiment, including the RMSEP, Average Root Mean Squared Error of Cross-Validation Folds (RMSECV), standard deviation, and standard deviation of cross-validation prediction errors for each model across all oxides. The means of each metric are also provided to give an overall indication of the models' performance. Furthermore, we present an overview of these mean values in Figure 9 to facilitate a visual comparison of the models' general performance.

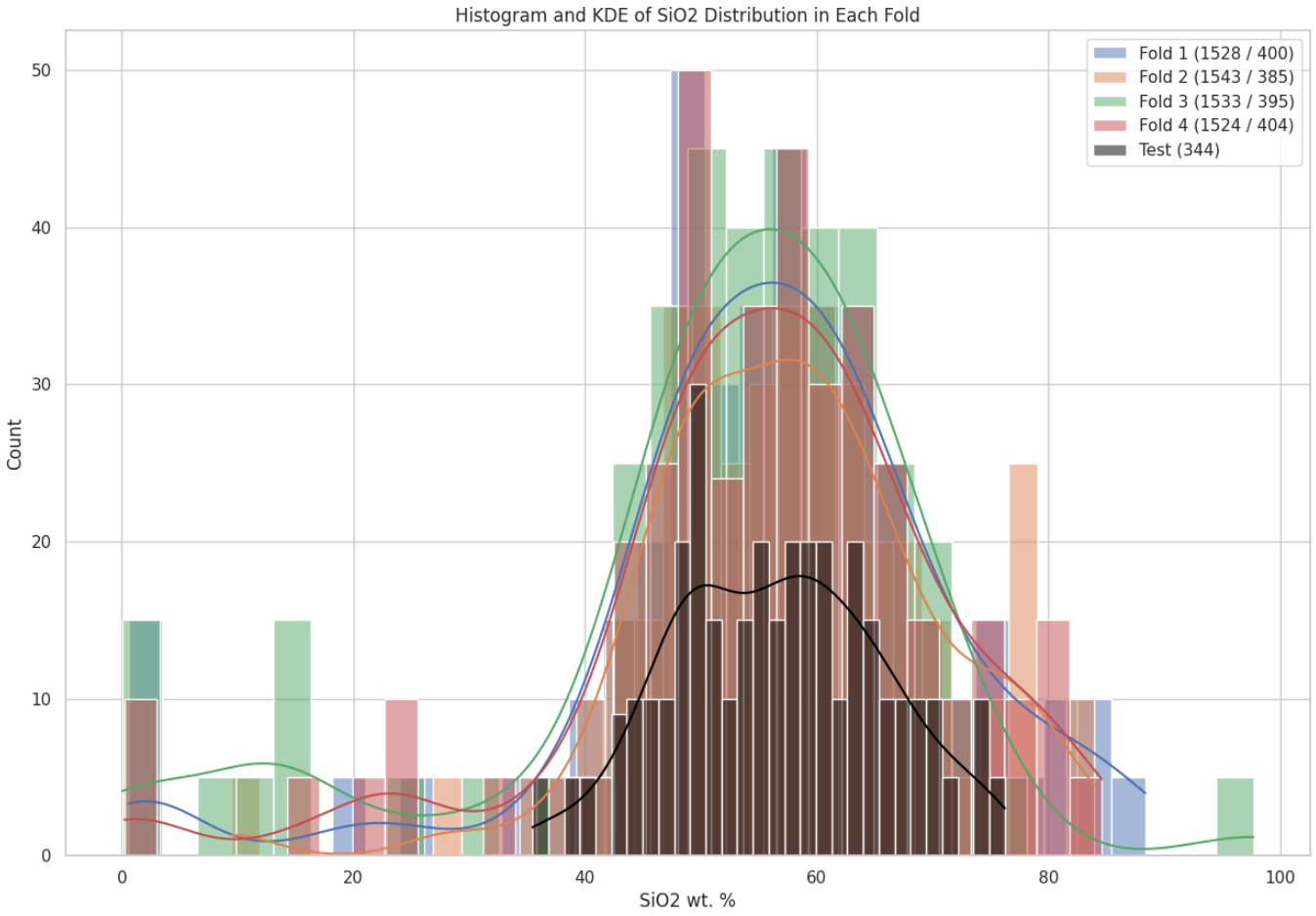

Fig. 6. Combined Histogram and KDE of SiO_2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents SiO_2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

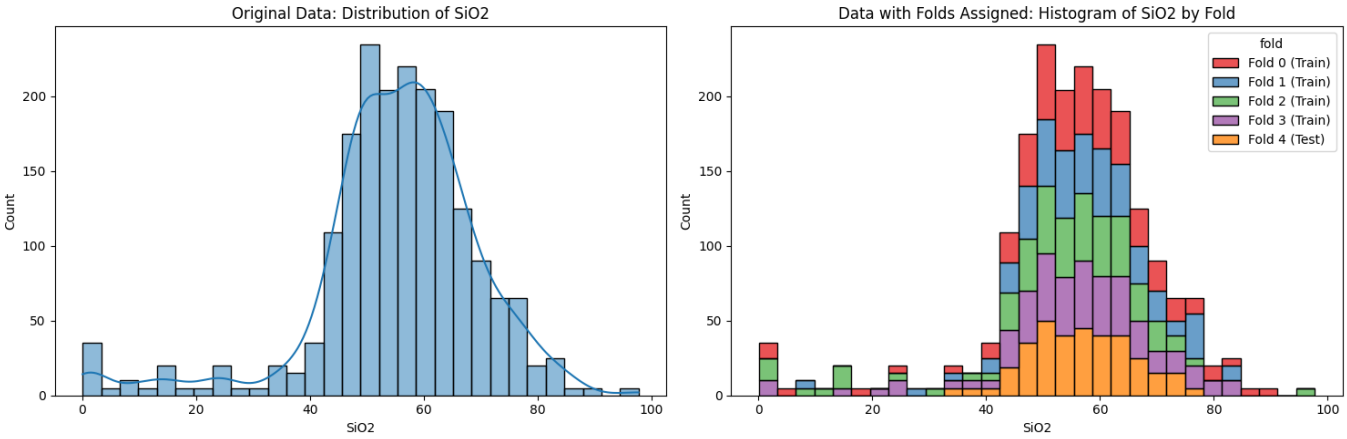

Fig. 7. Distribution of SiO_2 concentrations before and after fold assignment. The left plot shows the original distribution of SiO_2, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

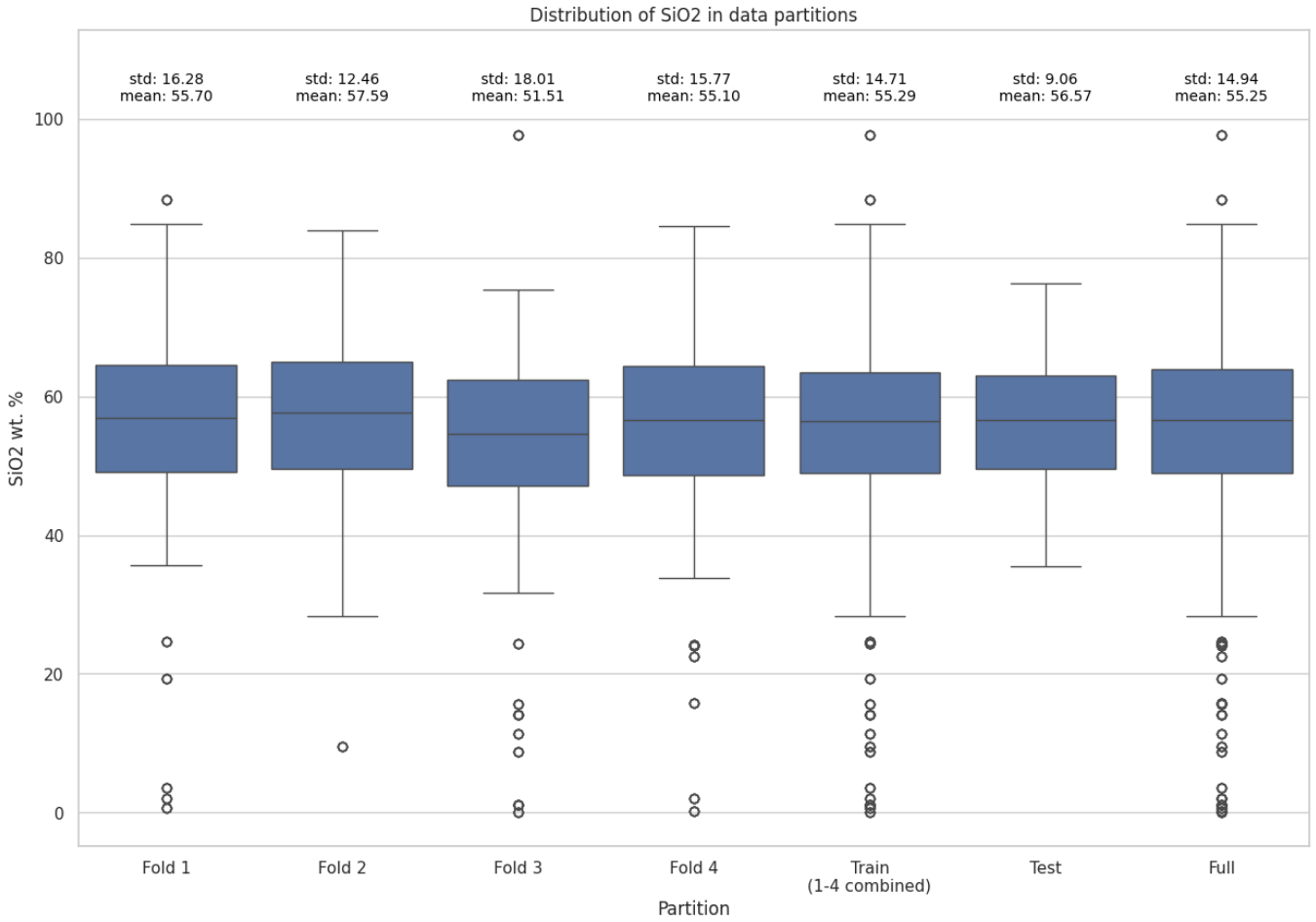

Fig. 8. Distribution of SiO_2 concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

The results indicate that the gradient boosting models, XGBoost, GBR, and NGBoost, consistently perform well across all oxides, with XGBoost generally outperforming the other two gradient boosting models. Interestingly, GBR has the lowest RMSEP, while XGBoost achieves the lowest RMSECV, suggesting that the regularization in XGBoost may improve the model's generalizability. These models exhibit both low mean RMSEP and RMSECV values, indicating high accuracy, as well as low standard deviation values, underscoring their robustness. SVR is also among the top-performing models, with mean RMSEP and RMSECV values close to those of XGBoost and low standard deviation values.

While usually outperformed by gradient boosting models and SVR, the other ensemble models, RF and ETR, also exhibit good performance. The PLS, ridge, LASSO, and ENet models typically seem to perform worse than the other models, with higher mean RMSEP and RMSECV values and higher standard deviation values. We observe that ENet performs between ridge and LASSO in terms of both error and standard deviation, which aligns with expectations since ENet combines the regularization techniques of both models.

The CNN and ANN models perform the worst across all oxides, exhibiting the highest mean RMSEP and RMSECV values, as well as the highest standard deviation values. This poor performance is further highlighted in Table 7, which shows the relative performance of each model compared to the best-performing model, XGBoost. The table also includes the difference in performance relative to the next best model, with XGBoost serving as the baseline for comparison, assigned a relative performance of 100%. From this table, it is evident that the CNN and ANN models experience notable drops in performance compared to the top-performing models. While deep learning models such as these have the theoretical potential to perform well with LIBS data, given their ability to learn complex patterns and relationships, the relatively small size

of our dataset may limit their efficacy. Furthermore, achieving optimal performance with these models necessitates extensive tuning of both their architectures and hyperparameters, which involves exploring a vast space of potential configurations and design choices. Although methods for systematic hyperparameter optimization, as detailed in Section 6.3, could be employed, the associated computational cost would be prohibitively high. Additionally, there are numerous architectural design decisions and advanced techniques that could potentially enhance model performance, but their inclusion would expand the scope of this study beyond feasible limits. For these reasons, we decided to exclude the CNN and ANN models from further experimentation.

Tables 9 and 10 list the best-performing model for each oxide and the frequency with which each model achieves top performance according to various metrics, respectively. These tables are intended to provide an overview of model performance rather than to determine an overall 'winner by majority'. Their purpose is to illustrate the general trends and behavior of different models across various metrics and oxides. Although XGBoost and SVR appear the most frequently in Table 10, this does not imply that they are the best models for every oxide. For example, if one were to only consider the mean of the performance metrics, PLS would be considered among the worst performing models, as shown in Figure 9. However, inspecting Table 9 reveals that PLS exhibits the lowest RMSECV and standard deviation of prediction errors for both MgO and Na_2O. This indicates that PLS is the most accurate and robust model for these oxides, underscoring the importance of evaluating model performance on a per-oxide basis, as discussed in Section 6. Moreover, for some oxides, multiple models perform similarly well, such as XGBoost, GBR, and ridge for CaO. This observation suggests the potential benefit of leveraging the strengths of multiple models, provided they do not make similar types of errors, which warrants further investigation.

To summarize, the initial results indicate that gradient boosting models, particularly XGBoost, demonstrated the most consistent and accurate performance across all oxides. SVR also performed well, with similar accuracy and robustness to the gradient boosting models. In contrast, deep learning models such as CNN and ANN underperformed, likely due to the small dataset size and insufficient tuning of their architectures and hyperparameters. Inspecting the model performances per oxide revealed that the best model varied depending on the oxide, and several models performed well for each oxide. This emphasizes the need for further evaluation of model performances on a per-oxide basis to identify suitable configurations for our stacking ensemble approach, which aims to leverage the strengths of multiple models.

7.5 Optimization Experiment

Using the remaining ten models, we conducted an extended experiment to further refine their performance for each oxide.

Table 7. Relative performance of each model compared to the best performing model, measured by normalized RMSECV and multiplied by 100 for percentage. A higher percentage indicates worse performance. The 'Diff. vs Prev.' column shows the difference in performance compared to the next best model, measured in percentage points.

Model	Relative Performance (%)	Diff. vs Prev.
XGB	100.00	-
SVR	100.85	0.85
GBR	103.07	2.22
NGB	103.94	0.87
RandomForest	104.45	0.51
ExtraTrees	104.84	0.39
Ridge	105.04	0.20
PLS	111.66	6.61
ElasticNet	114.12	2.46
LASSO	114.30	0.19
ANN	127.82	13.52
CNN	143.18	15.36

The goal was to identify which preprocessing techniques and hyperparameters would yield the best performance for each model by doing a thorough search for each configuration. To achieve this, we evaluated multiple permutations of each model with various preprocessors and hyperparameter configurations. Each configuration included a mandatory scaler, while data transformation and dimensionality reduction techniques were optional. The optimization process was conducted using our optimization framework, outlined in Section 6.3

To ensure a fair assessment of each configuration, we needed to balance conducting enough iterations for the optimization to converge with the practical limitations imposed by our time constraints. Therefore, we decided to perform 200 iterations per model for each oxide, resulting in a total of 16,000 iterations across ten models and eight oxides. We deemed this to be a reasonable number of iterations to obtain a reliable indication of the performance of each configuration. As mentioned in Section 6.3, we used the TPE algorithm for the optimization process. For this sampler, we set the number of startup trials to 25%. The number of startup trials determines the number of random samples drawn before the TPE sampler engages. By choosing 25%, we reserve the first quarter of the iterations for exploration. We believed this approach would allow sufficient time for the sampler to explore the search space while still providing enough iterations for refinement.

For the experiment, we defined a range or set of discrete values for each hyperparameter of the models and preprocessors. To determine these ranges, we used a combination of values reported in the literature, our own analysis, and the default values for each hyperparameter as a starting point. Our methodology involved expanding the hyperparameters with value ranges to include reasonable lower and upper extremes. For hyperparameters with a discrete set of possible values, we

Table 8. Initial results for the different models and metrics.

Model	Ridge				LASSO				ENet			
Metric	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV
SiO_2	4.104	5.004	4.108	5.005	4.412	5.431	4.417	5.437	4.412	5.431	4.417	5.437
TiO_2	0.424	0.470	0.413	0.469	0.398	0.556	0.389	0.555	0.398	0.556	0.389	0.555
Al_2O_3	2.322	2.913	2.324	2.888	2.349	3.063	2.352	3.044	2.349	3.063	2.352	3.044
FeOT	2.068	3.173	2.070	3.122	2.236	3.490	2.238	3.440	2.236	3.490	2.238	3.440
MgO	1.150	1.509	1.152	1.492	1.267	1.682	1.249	1.661	1.267	1.682	1.249	1.661
CaO	1.844	1.485	1.833	1.478	1.963	1.554	1.962	1.549	1.963	1.554	1.962	1.549
Na_2O	0.632	1.089	0.633	1.084	0.625	1.114	0.616	1.111	0.588	1.085	0.587	1.082
K_2O	0.651	0.668	0.645	0.668	0.638	0.859	0.629	0.856	0.638	0.859	0.629	0.856
Mean	1.649	2.039	1.647	2.026	1.736	2.219	1.732	2.207	1.731	2.215	1.728	2.203
Model	PLS				SVR				RF			
Metric	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV
SiO_2	4.141	5.701	4.145	5.693	3.552	4.908	3.555	4.908	3.715	5.304	3.699	5.292
TiO_2	0.452	0.531	0.441	0.530	0.461	0.463	0.455	0.462	0.331	0.427	0.321	0.425
Al_2O_3	2.073	3.322	2.061	3.302	1.931	2.700	1.934	2.693	2.076	2.443	2.079	2.433
FeOT	3.222	3.117	3.221	3.114	1.823	2.847	1.814	2.809	2.091	3.091	2.073	3.053
MgO	1.106	1.296	1.103	1.296	0.789	1.426	0.785	1.419	0.911	1.742	0.904	1.731
CaO	1.937	1.813	1.923	1.792	1.626	1.532	1.594	1.508	1.765	1.503	1.754	1.499
Na_2O	0.545	0.908	0.536	0.906	0.742	1.096	0.725	1.086	0.420	1.028	0.421	1.023
K_2O	0.774	0.650	0.772	0.646	0.567	0.690	0.555	0.689	0.524	0.681	0.476	0.676
Mean	1.781	2.167	1.775	2.160	1.436	1.958	1.427	1.947	1.479	2.027	1.466	2.017
Model	NGBoost				GBR				XGBoost			
Metric	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV
SiO_2	4.112	5.071	4.081	5.010	3.576	4.995	3.479	4.922	3.953	4.898	3.926	4.876
TiO_2	0.340	0.433	0.333	0.430	0.474	0.449	0.473	0.446	0.334	0.437	0.328	0.436
Al_2O_3	1.931	2.291	1.933	2.282	1.894	2.518	1.891	2.511	1.912	2.198	1.913	2.193
FeOT	1.588	3.561	1.590	3.530	1.594	3.069	1.596	3.068	1.848	3.020	1.838	3.002
MgO	0.849	1.578	0.845	1.574	0.964	1.766	0.960	1.763	0.905	1.781	0.901	1.771
CaO	1.740	1.610	1.723	1.602	1.768	1.468	1.769	1.468	1.765	1.467	1.749	1.457
Na_2O	0.416	0.921	0.415	0.916	0.481	1.130	0.481	1.123	0.387	1.071	0.387	1.062
K_2O	0.582	0.675	0.545	0.673	0.727	0.609	0.719	0.610	0.547	0.658	0.511	0.657
Mean	1.445	2.017	1.433	2.002	1.435	2.001	1.421	1.989	1.456	1.941	1.444	1.932
Model	ETR				ANN				CNN			
Metric	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV	RMSEP	RMSECV	Std. dev.	Std. dev. CV
SiO_2	3.995	5.230	3.970	5.225	4.664	7.025	4.670	6.981	4.662	6.061	4.626	6.046
TiO_2	0.330	0.439	0.321	0.438	0.436	0.543	0.431	0.540	0.571	0.634	0.565	0.628
Al_2O_3	1.845	2.368	1.847	2.359	2.624	3.049	2.628	3.026	2.482	2.871	2.457	2.854
FeOT	2.144	3.299	2.126	3.257	2.534	3.836	2.497	3.748	2.588	4.584	2.521	4.488
MgO	0.906	1.755	0.895	1.738	1.315	1.818	1.300	1.768	1.292	2.892	1.280	2.857
CaO	1.837	1.515	1.831	1.510	1.799	1.633	1.772	1.634	2.009	2.142	2.008	2.099
Na_2O	0.411	1.031	0.409	1.028	0.539	1.095	0.532	1.091	0.656	1.364	0.657	1.357
K_2O	0.591	0.642	0.540	0.636	0.659	0.850	0.640	0.845	0.783	1.684	0.742	1.657
Mean	1.507	2.035	1.492	2.024	1.821	2.481	1.809	2.454	1.880	2.779	1.857	2.748

Table 9. Lowest metric and corresponding model for each oxide.

Oxide	RMSEP	RMSECV	Std. dev.	Std. dev. CV
SiO_2	3.552 (SVR)	4.898 (XGBoost)	3.479 (GBR)	4.876 (XGBoost)
TiO_2	0.330 (ETR)	0.427 (RF)	0.321 (ETR)	0.425 (RF)
Al_2O_3	1.845 (ETR)	2.198 (XGBoost)	1.847 (ETR)	2.193 (XGBoost)
FeOT	1.588 (NGBoost)	2.847 (SVR)	1.590 (NGBoost)	2.809 (SVR)
MgO	0.789 (SVR)	1.296 (PLS)	0.785 (SVR)	1.296 (PLS)
CaO	1.626 (SVR)	1.467 (XGBoost)	1.594 (SVR)	1.457 (XGBoost)
Na_2O	0.387 (XGBoost)	0.908 (PLS)	0.387 (XGBoost)	0.906 (PLS)
K_2O	0.524 (RF)	0.609 (GBR)	0.476 (RF)	0.610 (GBR)

Table 10. Occurrences of the best model for each oxide.

Model	Occurrences
XGBoost	8
SVR	7
ETR	4
RF	4
PLS	4
GBR	3
NGBoost	2

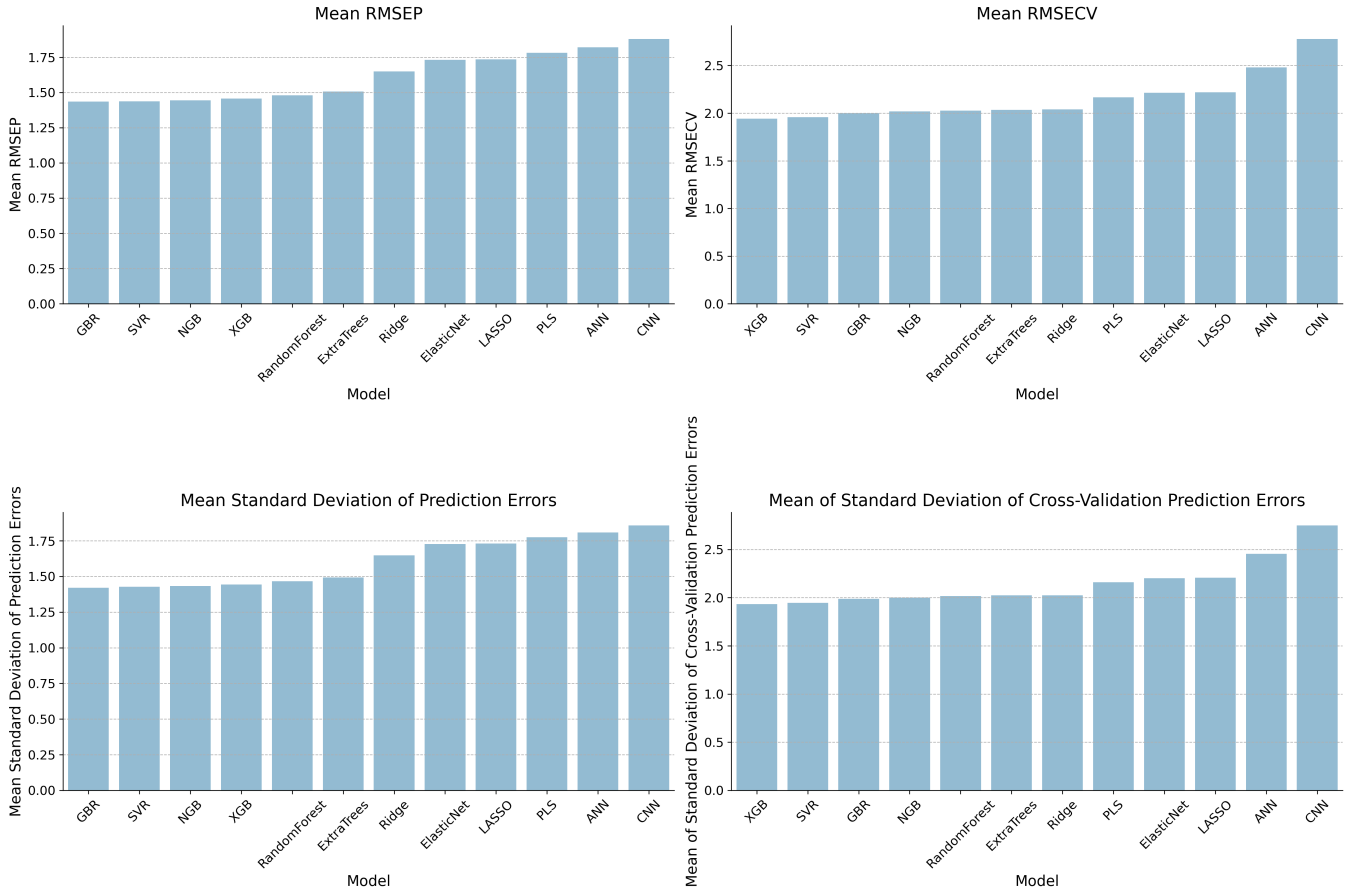

Fig. 9. Mean RMSEP, RMSECV, standard deviation of prediction errors, and standard deviation of cross-validation prediction errors for each model across all oxides.

included all options. As an example, for the PLS model, we used the elbow method to approximate the optimal number of components. Based on this, we defined the lower extreme as 1 and the upper extreme as 30, as we believed that the optimal number of components would be somewhere within this range. A similar approach was used for the preprocessor Kernel-PCA, where we defined the number of components to be between 1 and 100.

A different example is GBR, for which we based the hyperparameters on their default values. The default value for the number of estimators is 100, so we defined this as the lower bound and set 1000 as the upper bound. Given the complexity of the patterns in LIBS data, we believed that the ideal number of weak learners would likely be above 100. Therefore, we considered 100 to be a reasonable lower bound. Determining the upper bound was more challenging, but we considered 1000 to be a reasonable upper bound, as it would allow the model to sufficiently capture the patterns in the data. Given that we allow for a relatively large number of estimators, we wanted to balance this with a relatively low bound for the learning rate. We did this to ensure that the search space included a learning rate capable of scaling with the number of estimators, thereby reducing the likelihood of overfitting. The default value for the learning rate is 0.1, so we defined the lower bound as 10^{-3} and the upper bound as 1. The max depth of each weak learner was set between 3 and 10, allowing for varying levels of complexity. The subsample parameter was set between 0.5 and 1.0, to accommodate random sampling of the data when fitting each weak learner. Finally, the max features parameter was set to either *sqrt* or *log2*. Since this parameter has a discrete set of possible values, we included all options.

Using this approach of considering reasonable lower and upper bounds for each hyperparameter or using all options for discrete hyperparameters, we defined the ranges for each model and preprocessor.

The selected hyperparameter ranges for each model and preprocessor can be found in Table 12 and Table 11, respectively.

7.5.1 Results for Optimization Experiment. In this section, we present and analyze the results of running the optimization experiment that we described in Section 7.5. As mentioned,

Table 11. Optuna preprocessing configuration ranges.

Model	Parameter	Range
PCA	n_components	1 - 50
	whiten	{True, False}
KernelPCA	n_components	1 - 100
	kernel	{linear, poly, rbf, sigmoid, cosine}
	gamma	10^{-3} - 10^{1} (log scale)
	degree	1 - 5
RobustScaler	quantile_range	{25-75, 10-90, 5-95, 35-65, 30-70, 40-60}
	with_centering	{True, False}
StandardScaler	with_mean	{True, False}
	with_std	{True, False}
MinMaxScaler	feature_range	{0,1}, {-1,1}
PowerTransformer	method	yeo-johnson
	standardize	{True, False}
QuantileTransformer	n_quantiles	100 - 1000
	output_distribution	{uniform, normal}
	subsample	10000 - 100000
MaxAbsScaler	–	-
Norm3Scaler	–	-

our primary objective was to identify the optimal configurations for predicting the concentration of various oxides in our dataset. We systematically evaluated a range of machine learning models, preprocessing techniques, and hyperparameter settings to determine the most effective combinations for each oxide.

The results of the experiment were 16.000 trials worth of data on the configurations used, hyperparameters, as well as metrics.

Our data cleaning for this dataset primarily included filtering out failed runs, which was caused by configurations that did not work well together, as well as filtering out extreme error values. We filter out any runs that had an RMSECV above 50. Approaches like SVR could occasionally yield this kind of outlier result in specific configurations. We chose a threshold of 50 to include as many trials that were not clearly outliers.

Our experiment proceeded mostly without encountering any issues. Given the scale of the experiment, some issues were expected. A server issue required re-running some oxides and models. We successfully recovered and re-ran most of these. However, NGBoost for MgO was only partially finished. Given that each of the ten models would undergo 200 trials for each oxide, this resulted in 2000 runs per oxide. The exception is MgO, for which NGBoost ran 143 trials, making the total trials for MgO 1943. After the filtering process, we are left with a total of 15245 trials to analyze.

Since we stored the configurations as well as each hyperparameter value for the trials, we had 100 variables to consider during our analysis. Among these, the primary variables of interest are the metrics and overall configuration variables, namely `Model Type`, `Scaler Type`, `PCA Type`, and `Transformer Type`.

We used this data to identify the best configurations for each oxide, as measured by RMSECV. We began our analysis broadly by examining the usage of preprocessors across trials. Subsequently, we narrowed our focus and reviewed the top 100 trials for each oxide to identify the optimal model, scaler, and transformer for each oxide. Finally, we examined the single best-performing configurations across oxides, showing each of the 10 models with their corresponding best configuration for each oxide.

As described in Section 6.3, our optimization system searches for the best configurations through multi-objective optimization. The optimization process involves adjusting the configuration and hyperparameters of the machine learning model and preprocessing pipeline to minimize the objective. As the sampler conducts initial exploration and subsequently seeks to identify the optimal configuration through exploitation, we expect that the values of variables

Table 12. Optuna model configuration ranges.

Model	Parameter	Range
GBR	n_estimators	100 - 1000
	learning_rate	10^{-3} - 10^{0} (log scale)
	max_depth	3 - 10
	subsample	0.5 - 1.0
	max_features	{sqrt, log2}
SVR	C	10^{-3} - 10^{3} (log scale)
	epsilon	10^{-3} - 10^{1} (log scale)
	kernel	{linear, poly, rbf, sigmoid}
	degree	1 - 5
	gamma	{scale, auto}
	coef0	0 - 10
XGBoost	n_estimators	100 - 1000
	learning_rate	10^{-3} - 10^{0} (log scale)
	max_depth	2 - 15
	subsample	0.3 - 1.0
	colsample_bytree	0.5 - 1.0
	gamma	10^{-3} - 10^{1} (log scale)
	reg_alpha	10^{-3} - 10^{3} (log scale)
	reg_lambda	10^{-3} - 10^{3} (log scale)
ETR	n_estimators	100 - 1000
	max_depth	2 - 15
	min_samples_split	2 - 20
	min_samples_leaf	1 - 25
	max_features	{sqrt, log2}
PLS	n_components	1 - 30
NGBoost	max_depth	2 - 10
	natural_gradient	{True, False}
	n_estimators	50 - 1000
	learning_rate	0.01 - 0.5 (log scale)
	minibatch_frac	0.5 - 1.0
	col_sample	0.5 - 1.0
	tol	10^{-5} - 10^{-3} (log scale)
	validation_fraction	0.1 - 0.5
	early_stopping_rounds	10 - 100
Lasso	alpha	10^{-3} - 10^{3} (log scale)
Ridge	alpha	10^{-3} - 10^{3} (log scale)
ENet	alpha	10^{-3} - 10^{3} (log scale)
	l1_ratio	0 - 1
RF	n_estimators	100 - 300
	max_depth	2 - 15
	min_samples_split	2 - 10
	min_samples_leaf	1 - 10
	max_features	{sqrt, log2}

frequently appearing in the results are most likely to be optimal. Table 13, 14, and 15 display these values for the various preprocessors. The optimization results indicate that configurations with the highest total values across oxides are often the most frequently exploited, suggesting they are most likely to optimize performance. `Norm3Scaler` was used in 5090 trials, and therefore appears to be the most effective scaler. This outcome was expected, as the method was specifically designed for this type of LIBS dataset, as discussed in Section 4.2.5. For dimensionality reduction, we see that `None`, indicating no PCA, was used in 9419 trials, constituting approximately 59% of all trials. This suggests that either no dimensionality reduction is optimal, or a more suitable method should be identified. Neither PCA nor Kernel-PCA appear to be effective for this dataset. `QuantileTransformer` was used in 5710 trials, indicating that it may be the most optimal transformer. We observe that `PowerTransformer` was employed in 5277 trials, while no transformer was used in 4956 trials. Unlike other preprocessing method types, there does not appear to be a clear winner for the transformers.

We chose to examine the top 100 trials for each configuration to provide a clearer understanding of the performance variance among the best configurations. Given the wide range of configurations and their varying sensitivity to tuning, data distributions, and other factors, examining all trials would result in misleading descriptive statistics. By focusing on the top 100 trials, we can more accurately identify which configurations perform well. While a quantitative analysis of the top 10% of trials based on metrics like RMSECV, RMSEP, and other factors was feasible, selecting the top 100 trials for each oxide made it easier to see how many trials were present for each oxide. This approach helps us avoid the 'best of the worst' scenario and ensures that we are analyzing a representative set of good configurations. We present the results in Figure 10, 11, 12, and 13. These figures and their corresponding subplots illustrate the performance of various configuration elements (models, scalers, transformers, PCA techniques) for each oxide, based on their RMSECV. Any elements that do not appear in a subplot were not used in the top 100 trials for the given oxide. It is important to note that the variance in performance is influenced by multiple factors, not solely by the variable depicted in each plot. Factors such as the interaction between different preprocessing techniques, specific hyperparameter settings, and the inherent variability in the data all contribute to the observed performance. Therefore, while the plots offer valuable insights into the effectiveness of individual configuration elements, the overall performance is a result of complex interactions within the entire machine learning pipeline. Therefore, we prioritize the analysis of the top-performing trials and examine a larger sample of these to draw generalizable conclusions about the optimal configurations for each oxide.

From Figure 10, it is evident that SVR, gradient boosting methods, and PLS demonstrate the best performance. Figure 13 confirms our earlier hypothesis that not using any PCA or Kernel-PCA yields the lowest RMSECV values. However, we do observe that either PCA or Kernel-PCA appear in four of the plots, with Kernel-PCA being the most frequently used among them. This indicates that they are indeed used in some top-performing configurations. However, based on the results in Table 14, we did not expect them to be as prevalent as they are, suggesting that while they are not the most frequently used, they can still be highly effective in specific scenarios. Interestingly, Figure 11 shows that, although `Norm3Scaler` is the most frequently used and best-performing scaler, this is not always the case. Min-Max normalization appears to yield better results for SiO_2 and CaO, while robust scaling seems more effective for MgO. For Al_2O_3, Norm 3 scaling exhibits the lowest RMSECV values but a higher mean RMSECV value compared to the other scalers. Finally, Figure 12 reveals another nuanced finding. Power transformations appear to most frequently yield the best results across oxides, while quantile transformation or no transformation show the lowest RMSECV values for the remaining oxides.

These results further reinforce our hypothesis that a tailored configuration is necessary for each oxide, and there is no single configuration that performs well across all oxides.

We conclude our analysis by presenting the best configurations for each oxide in Section A.4. The section shows the single top-performing configurations for each model for each oxide, presented in Tables A.5 through A.12. Similar to the previous plots, we use the RMSECV values to determine the best configurations. Notably, these tables illustrate how certain configurations may exhibit low RMSECV values but relatively high RMSEP values. This observation could suggest that they generalize well to the dataset containing extreme values but struggle with values closer to the mean. For example, the top-performing configuration for SiO_2 consists of PLS with Kernel-PCA and `MinMaxScaler`. This configuration has the lowest RMSECV value of 4.55, but a relatively high RMSEP value of 4.08. The next-best performing configuration for SiO_2 is SVR with `MinMaxScaler`. This configuration has a RMSECV value of 4.59, but a RMSEP value of 3.53. Although the difference in RMSECV is negligible, the RMSEP value for the SVR configuration is much lower than that of the PLS configuration. This indicates that the SVR configuration is likely a better overall predictor for SiO_2 than the PLS configuration.

The analysis of the best configurations for each oxide reveals that certain models and preprocessing techniques consistently outperform others. SVR and PLS models, in particular, frequently appear among the top configurations. The use of transformers such as the Power Transformer and scalers like Norm 3 and Min-Max Scaler are also common among the best configurations.

Finally, we use a combination of these top-performing configurations by selecting the top-n performing configurations per oxide for our stacking ensemble. This approach is further elaborated on in Section 7.6.

7.6 Stacking Ensemble

Given the results of the optimization process, we implemented a stacking ensemble to combine the predictions of the

Table 13. Comparison of different scalers across the eight major oxides.

Oxide	MaxAbsScaler	MinMaxScaler	Norm3Scaler	RobustScaler	StandardScaler
Al_2O_3	336	476	495	453	240
CaO	310	362	681	338	309
FeO_T	498	287	561	339	315
K_2O	258	320	622	430	370
MgO	239	340	646	413	305
Na_2O	316	327	748	320	289
SiO_2	221	421	830	309	219
TiO_2	280	322	507	565	326
Total across oxides	2458	2855	5090	3167	2373

Table 14. Comparison of different PCA types across the eight major oxides.

Oxide	None	KernelPCA	PCA
Al_2O_3	1243	389	368
CaO	1246	374	380
FeO_T	1217	382	401
K_2O	1250	373	377
MgO	1037	543	363
Na_2O	1167	382	451
SiO_2	1096	457	447
TiO_2	1163	468	369
Total across oxides	9419	3368	3156

Table 15. Comparison of different transformers across the eight major oxides.

Oxide	None	PowerTransformer	QuantileTransformer
Al_2O_3	442	649	909
CaO	673	595	732
FeO_T	633	644	723
K_2O	504	822	674
MgO	725	701	517
Na_2O	441	583	976
SiO_2	760	566	674
TiO_2	778	717	505
Total across oxides	4956	5277	5710

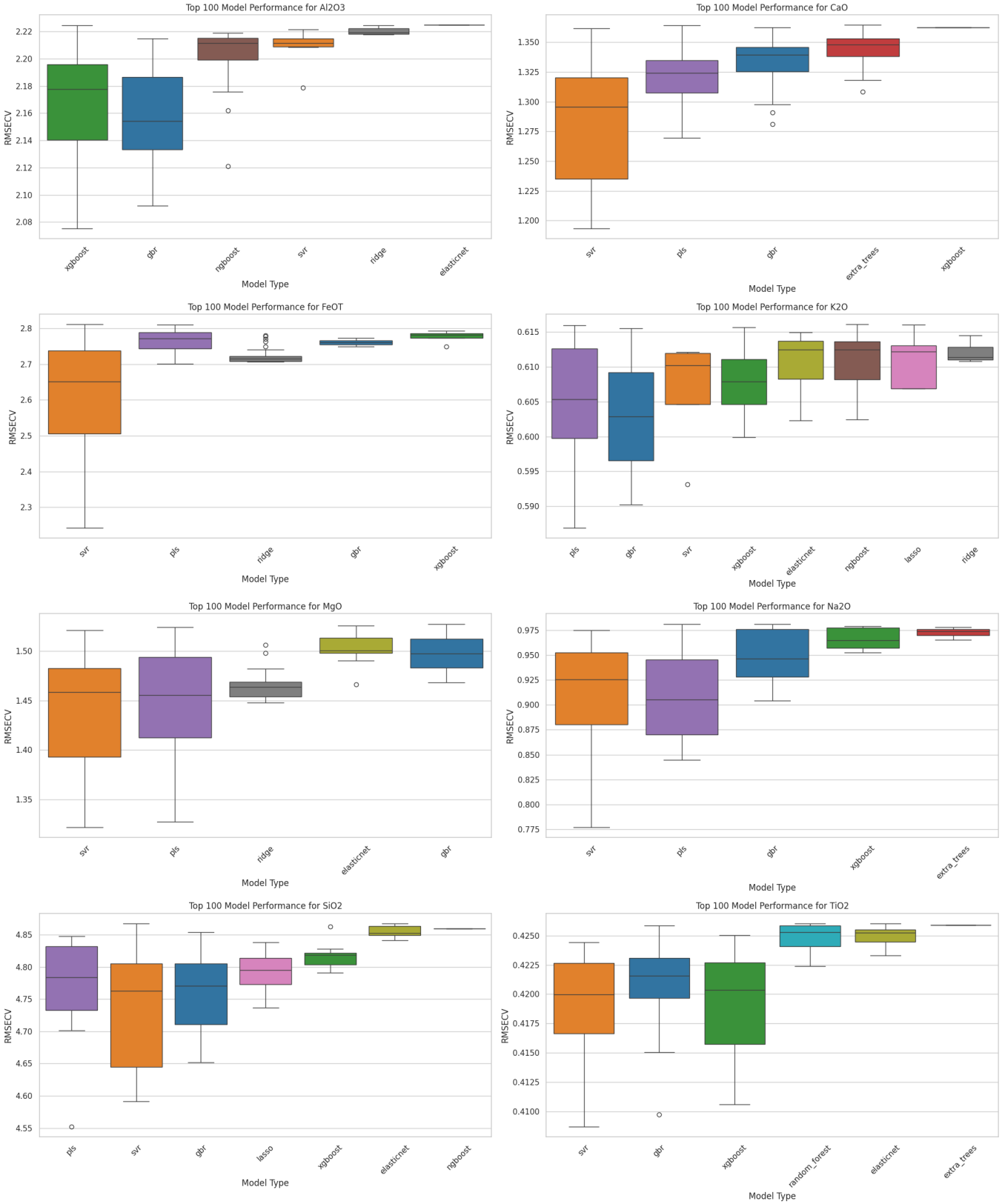

Fig. 10. Top 100 model performance across oxides. The subplots show the distribution of RMSECV values for the top 100 trials for each model type across the eight different oxides. This helps identify the most effective models for each oxide within the top-performing trials.

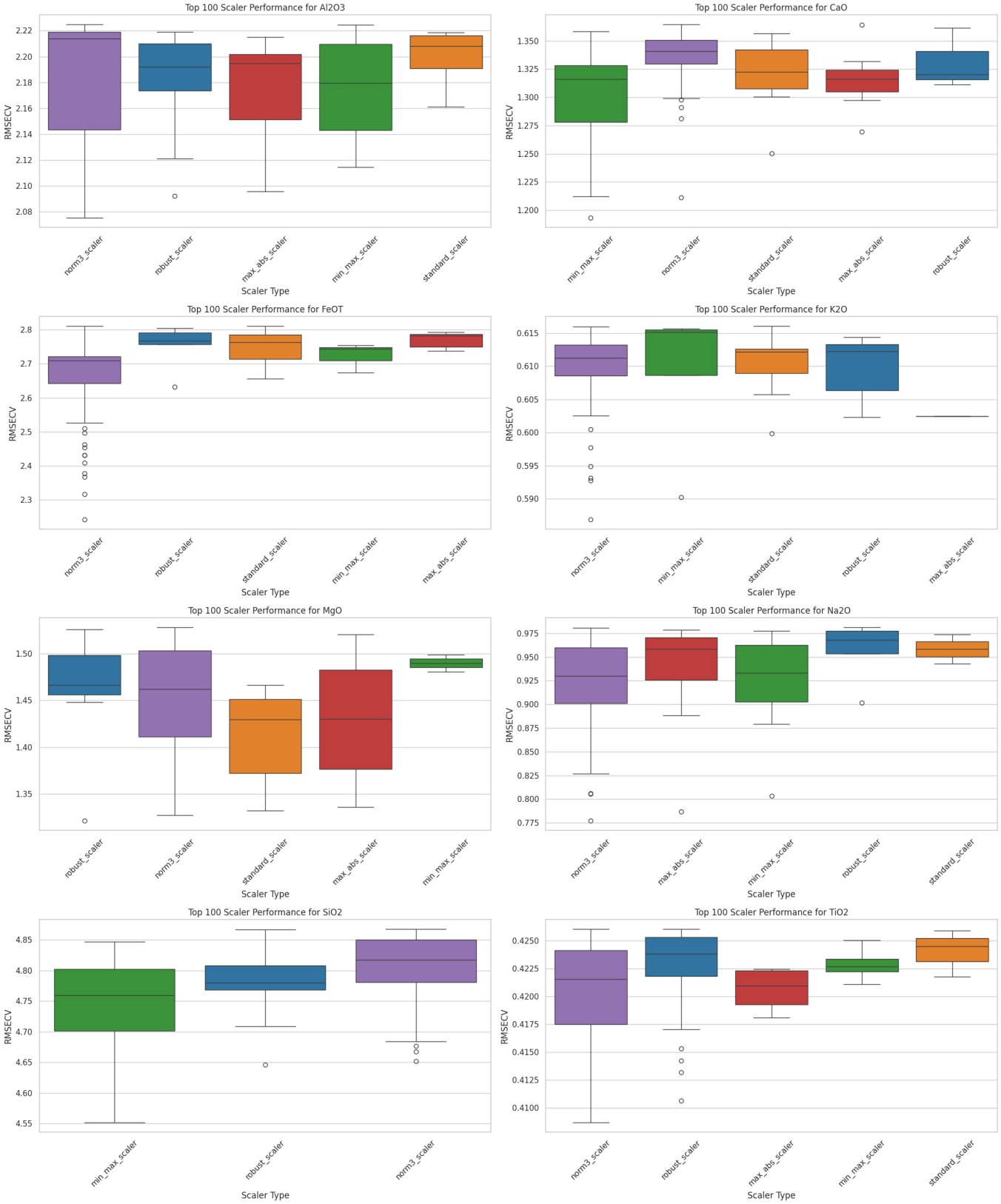

Fig. 11. Top 100 scaler performance across oxides. The subplots illustrate the distribution of RMSECV values for the top 100 trials for each scaler type across the different oxides. This helps pinpoint which scalers perform best within the top-performing trials.

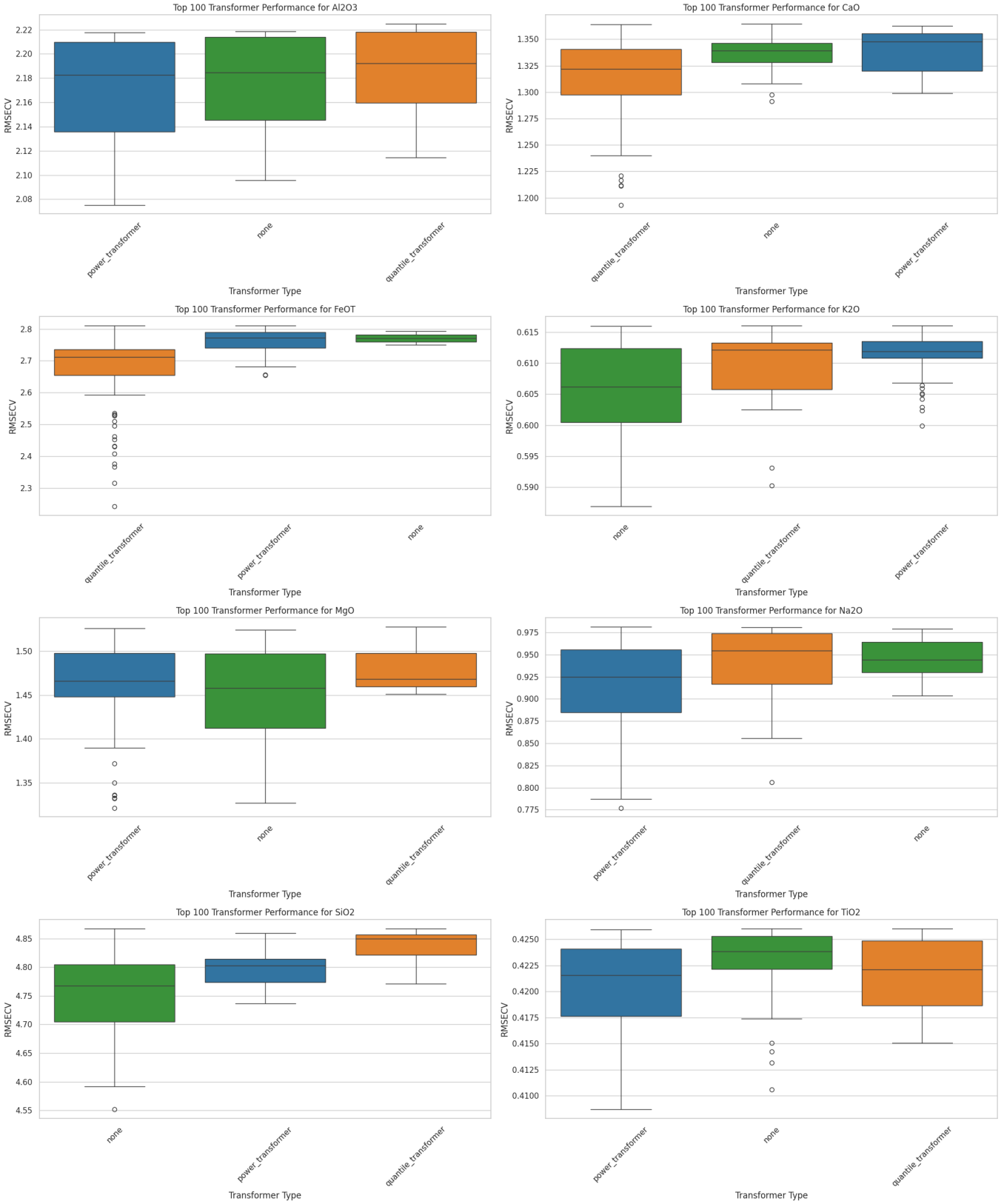

Fig. 12. Top 100 transformer performance across oxides. The subplots display the distribution of RMSECV values for the top 100 trials for each transformer type across the different oxides. This helps determine the effectiveness of each transformer for different oxides within the top-performing trials.

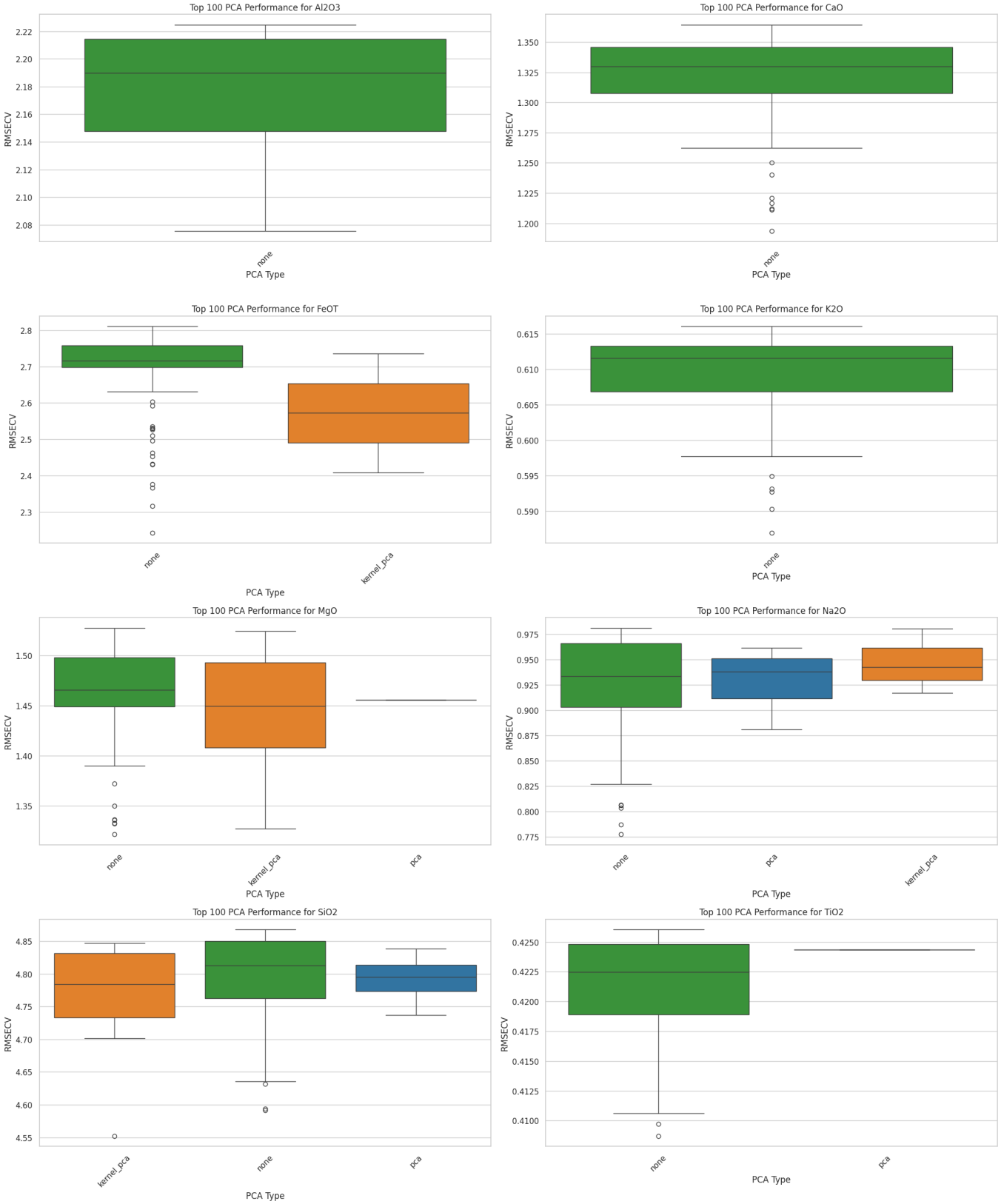

Fig. 13. Top 100 PCA performance across oxides. The subplots present the distribution of RMSECV values for the top 100 trials for each PCA type across the different oxides. This allows us to understand the impact of PCA techniques on model performance for each oxide within the top-performing trials.

top-performing configurations. As with all our experiments, we follow the procedure outlined in Section 6.2 to evaluate the performance of the stacking ensemble.

For each oxide, we first identified the top-performing configurations to use in the stacking ensemble. To identify the top-performing configurations for use in the stacking ensemble, we developed a category-based method that groups models into distinct categories for systematic comparison. This approach helps in selecting models that perform optimally for specific tasks while ensuring diversity within the ensemble. Additionally, we developed and employed a grid search to identify the optimal ensemble configurations, but time constraints prevented us from fully completing this process. Finally, we employed the stacking model and evaluated its performance. We present the results as well as the 1:1 plots in Section 7.6.3.

7.6.1 Category Method & Pipeline Selection. The category-based method organizes models into the following groups:

- Gradient Boosting: GBR, XGBoost, NGBoost
- Tree-Based: ETR, RF
- Linear and Regularized Models: LASSO, Ridge, ENet
- SVM: SVR
- PLS: PLS

We used the top-performing configurations identified in Section 7.5.1 to select configurations for the stacking ensemble.

For each oxide, we filtered the trial data from the optimization experiment to create a dataset containing configuration information and performance metrics. These datasets were sorted by RMSECV. We then selected unique model types for further analysis, based on the categories above. Our process involved selecting a set number of configurations for each category until a maximum number of configurations for the oxide's ensemble was reached.

To ensure diversity and limit scope, we set a maximum of one model per category and three models per stacking ensemble, with one ensemble for each oxide. For each oxide, we developed pipelines to preprocess the data according to the given configuration and to utilize the models with their optimal hyperparameters. These pipelines represent the optimal configurations for each oxide, and are used as base estimators in the stacking ensemble.

7.6.2 Grid Search. The grid search process begins by generating all possible combinations of the selected models. Specifically, we generate combinations with at least two models, up to a configurable maximum number of models.

For each oxide, we constructed a pipeline for each of the top model configurations identified in Section 7.5.1. The evaluation function iterates over each combination of base estimators, constructs a stacking ensemble pipeline, and assesses its performance using cross-validation.

The evaluation function prepares the pipeline with the current combination of base estimators and splits the data into training and testing sets using our data partitioning method. The stacking ensemble is then fitted on the training set, and the meta-features generated from the base estimators are used to train the final estimator.

We compute the evaluation metrics to assess the ensemble's effectiveness. The best combination is identified based on the lowest RMSECV value.

It is possible to vary the meta-learner in this process, adding another variable to tune as part of the search process.

While our grid search implementation was limited to the TiO_2 oxide, the methodology demonstrates the potential for identifying optimal stacking ensembles by systematically evaluating model combinations and their configurations. We chose not to pursue this method further due to the time constraints of this project, but believe it is a promising approach for future work.

7.6.3 Results for Stacking Ensemble. For each major oxide, we ran the stacking ensemble and evaluated its performance. The evaluations were conducted using the same configurations, varying only the meta-learner between runs. We present the results for the ENet meta-learner with `alpha` = 1.0, as well as for the ENet meta-learner with `alpha` = 0.1. Additionally, we present the results for the SVR meta-learner, using the default hyperparameters provided by `scikit-learn`.

The evaluation metrics are shown in Table 16, Table 17, and Table 18. Additionally, we provide 1:1 plots for each ensemble in Figures 14, 15, and 16, showing the actual versus predicted values for each oxide.

For the ENet meta-learner with $\alpha = 1$, the RMSEP values range from 0.470 for Na_2O to 3.588 for SiO_2. The RMSECV values are generally higher, which could initially suggest overfitting. However, considering our testing and validation strategy, this discrepancy is expected. Our method for partitioning ensures that extreme values are included in the training folds but not in the test set, making the test set easier to predict. This results in lower RMSEP values compared to RMSECV values, which is a deliberate trade-off to provide a fairer assessment of the model's generalization performance. The standard deviations of the RMSECV are relatively low, suggesting consistent performance across folds.

When the ENet meta-learner's α is reduced to 0.1, there is a noticeable improvement in the RMSEP for TiO_2, dropping from 0.571 to 0.319. This suggests that reducing the regularization parameter helps in better capturing the variance in the data. The RMSECV values also show a slight improvement, indicating better generalization. However, the standard deviations remain similar, suggesting that the model's consistency across folds is maintained.

The SVR meta-learner shows the best performance for several oxides, particularly SiO_2 and Na_2O, with RMSEP values of 3.473 and 0.369, respectively.

We generally observe that the standard deviation metrics are close to the corresponding RMSE values, indicating low variability in the prediction errors. This suggests robustness of the predictions.

Table 16. Stacking ensemble results using the ENet model as the meta-learner with $\alpha = 1$.

Oxide	RMSEP	Std. Dev.	RMSECV	Std. Dev. CV
SiO_2	3.588	3.582	4.680 ± 0.500	4.670 ± 0.516
TiO_2	0.571	0.565	0.818 ± 0.111	0.814 ± 0.117
Al_2O_3	1.656	1.657	2.211 ± 0.023	2.199 ± 0.226
FeO_T	1.794	1.789	2.792 ± 0.649	2.745 ± 0.646
MgO	0.711	0.711	1.660 ± 0.525	1.635 ± 0.516
CaO	1.636	1.619	1.307 ± 0.205	1.290 ± 0.202
Na_2O	0.470	0.462	1.075 ± 0.706	1.067 ± 0.710
K_2O	0.476	0.462	0.653 ± 0.195	0.652 ± 0.193

Table 17. Stacking ensemble results using the ENet model as the meta-learner with $\alpha = 0.1$.

Oxide	RMSEP	Std. Dev.	RMSECV	Std. Dev. CV
SiO_2	3.598	3.591	4.686 ± 0.489	4.677 ± 0.505
TiO_2	0.319	0.310	0.450 ± 0.083	0.448 ± 0.083
Al_2O_3	1.658	1.660	2.192 ± 0.235	2.180 ± 0.238
FeO_T	1.841	1.840	2.781 ± 0.664	2.731 ± 0.659
MgO	0.768	0.768	1.632 ± 0.531	1.608 ± 0.518
CaO	1.647	1.627	1.310 ± 0.213	1.291 ± 0.211
Na_2O	0.442	0.433	1.093 ± 0.690	1.079 ± 0.692
K_2O	0.494	0.477	0.556 ± 0.155	0.553 ± 0.153

Table 18. Stacking ensemble results using the SVR model as the meta-learner with default hyperparameters.

Oxide	RMSEP	Std. Dev.	RMSECV	Std. Dev. CV
SiO_2	3.473	3.478	5.064 ± 0.932	5.061 ± 0.926
TiO_2	0.340	0.333	0.442 ± 0.087	0.442 ± 0.087
Al_2O_3	1.729	1.732	2.285 ± 0.226	2.274 ± 0.233
FeO_T	1.693	1.681	4.821 ± 1.490	4.768 ± 1.495
MgO	0.819	0.820	2.569 ± 1.274	2.559 ± 1.271
CaO	1.594	1.574	1.475 ± 0.294	1.456 ± 0.304
Na_2O	0.369	0.368	0.978 ± 0.885	0.971 ± 0.887
K_2O	0.511	0.497	0.669 ± 0.199	0.666 ± 0.196

Table 19. Comparison of RMSEP values for the MOC (replica) model and various stacking ensemble models.

Oxide	MOC (replica)	ENet ($\alpha = 1$)	ENet ($\alpha = 0.1$)	SVR
SiO_2	5.61	3.59	3.60	**3.47**
TiO_2	0.61	0.57	**0.32**	0.34
Al_2O_3	2.47	**1.66**	1.66	1.73
FeO_T	1.82	1.79	1.84	**1.69**
MgO	1.56	**0.71**	0.77	0.82
CaO	2.09	**1.64**	1.65	1.59
Na_2O	1.33	0.47	0.44	**0.37**
K_2O	1.91	**0.48**	0.49	0.51

A notable observation from our results is that different meta-learners exhibited varying performance levels across oxides. We observed that the final predictions were strongly affected by the meta-learner, going as far as rendering some predictions nonsensical if the wrong meta-learner was chosen. Specifically, for TiO_2, we observed that predictions remained near-constant values despite varying the combination of model configurations in the TiO_2 ensemble. In fact, this was the reason for our implementation of the grid search process. The consistency of our observations supported our hypothesis: changing only the meta-learner significantly impacts the RMSECV and prediction outcomes. We decided to investigate further and identified potential issues with small value predictions when using an ENet meta-learner. For example, most values for TiO_2 fell between 0 and 2.5, as shown in Figure 4. The regularization term likely dominated the fitting process, leading to underfitting and resulting in nearly constant predictions. This hypothesis was confirmed by adjusting the regularization parameter, `alpha`, in the ENet. Lowering `alpha` produced better outcomes, indicating that regularization adversely affected the predicted values. The 1:1 plot in Figure 14 shows the near-constant predictions for TiO_2 when using a ENet meta-learner, and Figure 15 shows the improved predictions with `alpha` = 0.1. This leads us to conclude that the meta-learner's choice significantly impacts the RMSECV and prediction outcomes.

The stacking approach demonstrated strong improvements in prediction accuracy compared to the baseline described in Section 5, validating the efficacy of our methodology. We measured this improvement using RMSEP, which provides the fairest comparison between the baseline and the stacking approach. As mentioned, RMSEP evaluates the model's performance on the test set. In Section 5, we described how the baseline test set was constructed by sorting extreme concentration values into the training set, and then performing a random split. As noted in Section 6.2, a more sophisticated procedure is required to support the testing and validation strategy in this work. Despite the differences in test set construction, the test sets remained similar in composition[1], which allowed us to use RMSEP as a fair comparison metric. Table 19 compares the RMSEP values of different oxides for the MOC (replica) model with three stacking ensemble models: ENet with $\alpha = 1$, ENet with $\alpha = 0.1$, and SVR. Overall, the stacking ensemble models tend to produce lower RMSEP values compared to the MOC (replica) model. Notably, SiO_2, TiO_2, Na_2O, and K_2O show large improvements across all stacking ensemble models. For instance, the RMSEP for SiO_2 is reduced from 5.61 (MOC (replica)) to around 3.59 (ENet with $\alpha = 1$) and further to 3.47 (SVR). Similarly, TiO_2 shows a reduction from 0.61 (MOC (replica)) to 0.32 (ENet with $\alpha = 0.1$). The improvements are consistent across most oxides, with ENet and SVR models both outperforming the MOC (replica) model. This shows that the ensemble approach,

[1]The analysis of this can be found on our GitHub repository: https://github.com/chhoumann/thesis-chemcam

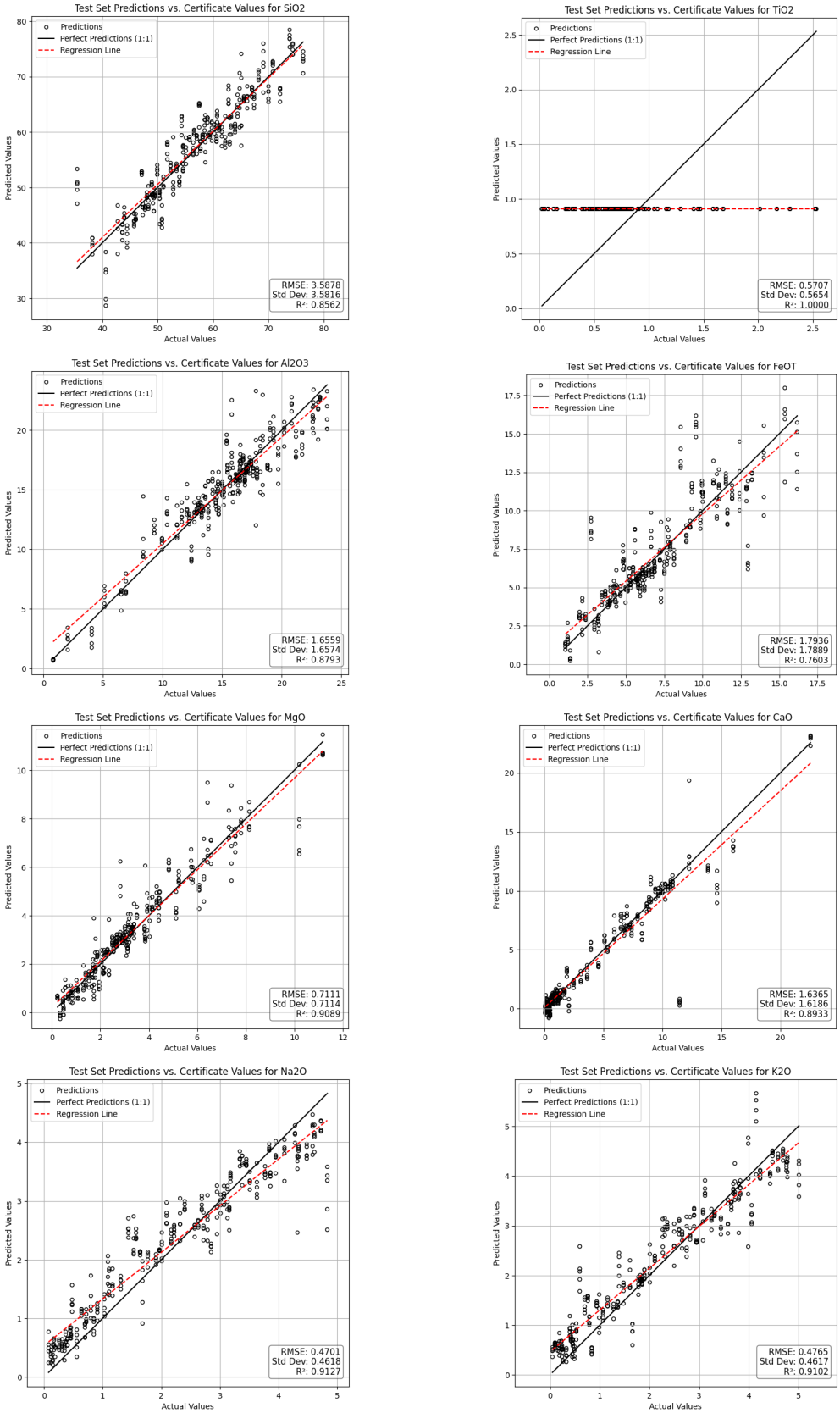

Fig. 14. One-to-one plots for the stacking ensemble model with the ENet as the meta-learner with $\alpha = 1$

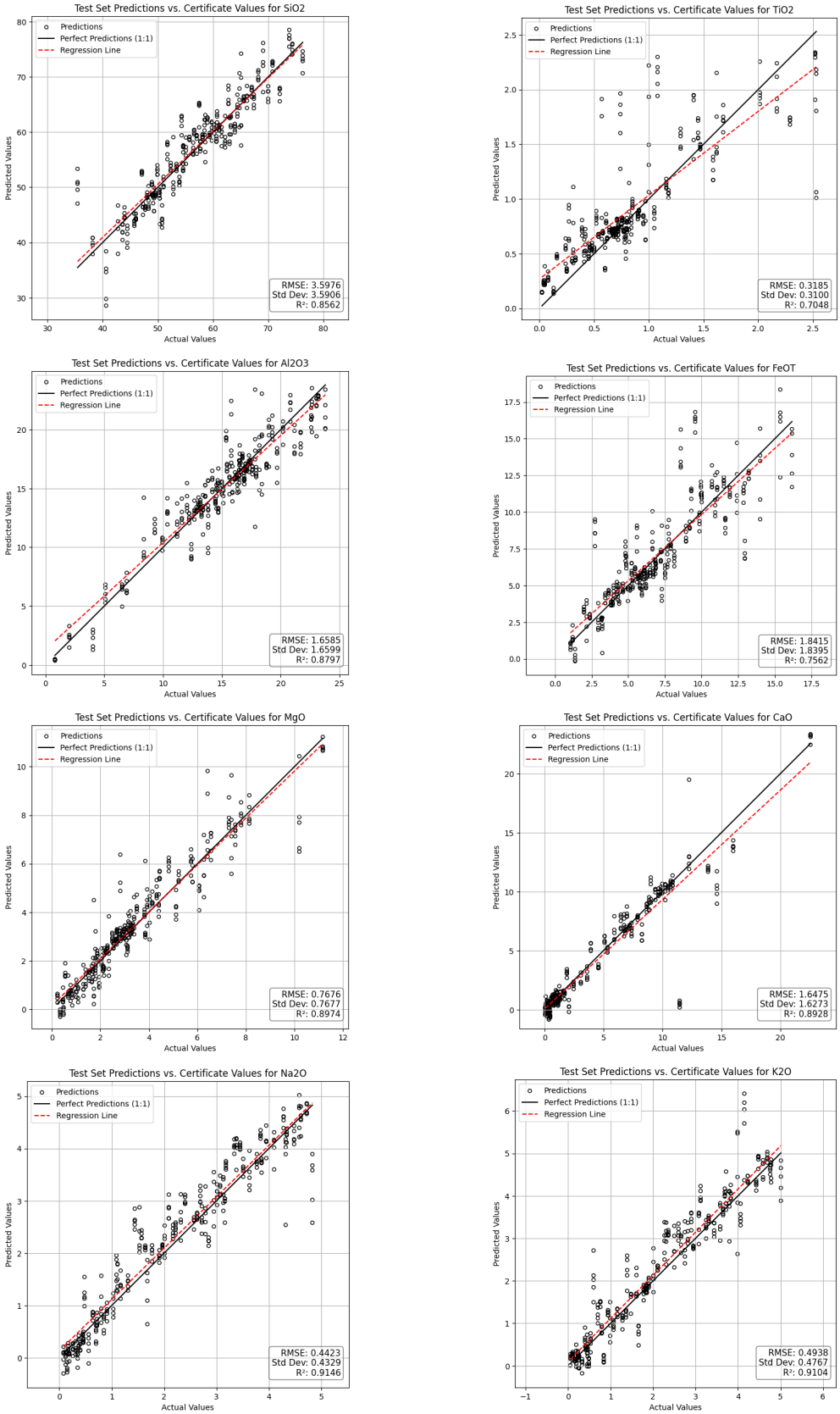

Fig. 15. One-to-one plots for the stacking ensemble model with the ENet as the meta-learner with $\alpha = 0.1$.

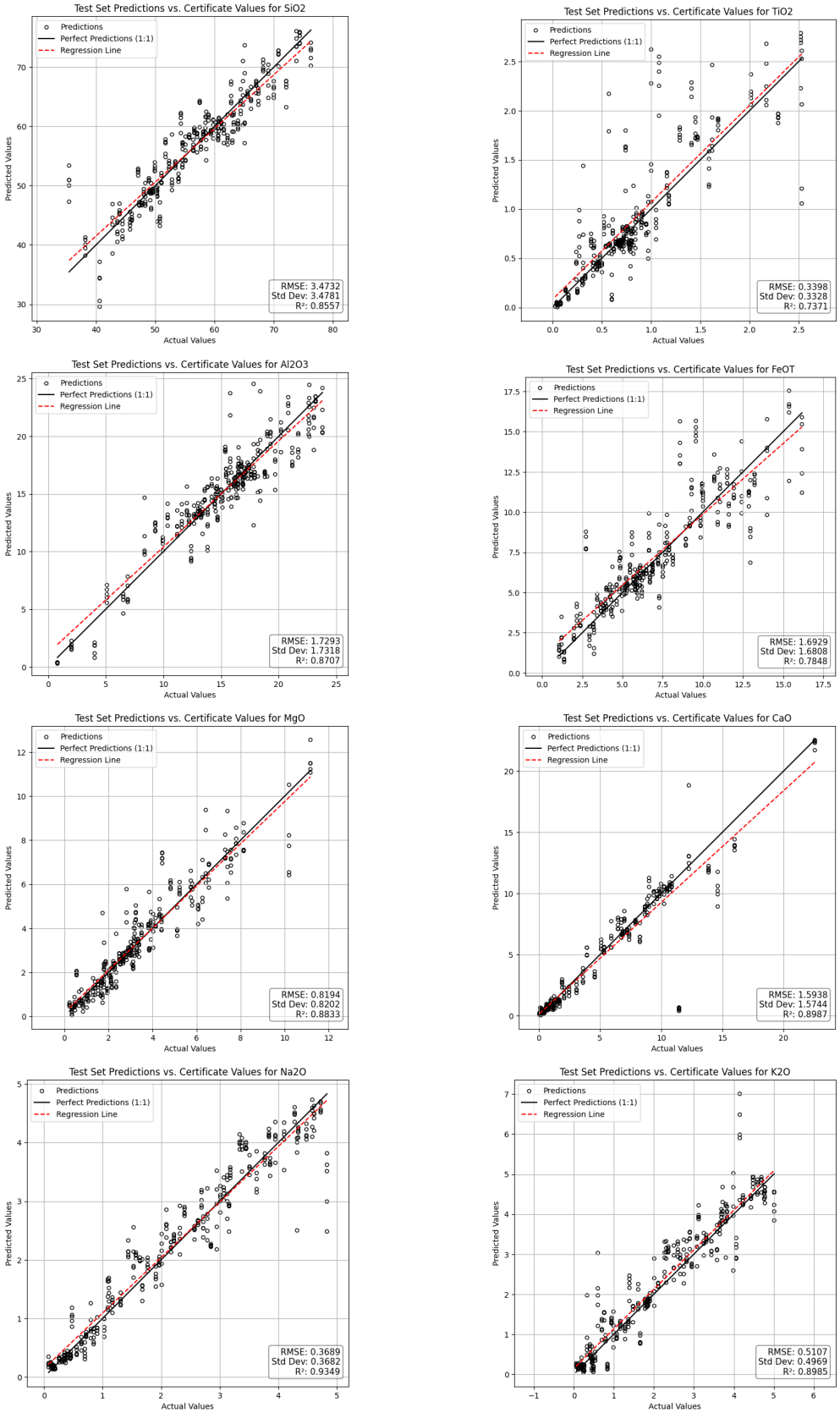

Fig. 16. One-to-one plots for the stacking ensemble model with the SVR as the meta-learner

particularly with these meta-learners, enhances prediction accuracy for the oxides we tested.

The results presented indicate a strong performance from the stacking ensemble approach. However, it is important to note that some evaluation metrics are worse in the stacking approach than in certain individual configurations. We believe that further tuning, particularly of the meta-learner's hyperparameters, could substantially improve these results.

8 PYHAT CONTRIBUTION

As part of our work, we have made several contributions to PyHAT. We describe these contributions here.

PyHAT offers a user-friendly interface designed for performing machine learning and data analysis tasks specifically for hyperspectral data. Our collaboration was initiated through a series of discussions with two members from USGS that are responsible for PyHAT, wherein we identified mutual challenges and opportunities for integrating our solutions into the tool.

We implemented an outlier detection method in PyHAT that uses the Mahalanobis distance and the chi-squared test. This statistical approach identifies outliers without relying on qualitative assessments. The process involves computing leverage, which measures a sample's influence, and spectral residuals, which are the differences between observed and predicted values, for each sample using a PLS model. The process involves calculating leverage and spectral residuals for each sample using a PLS model. Leverage measures a sample's influence, and spectral residuals represent the differences between observed and predicted values.

These metrics are combined into a two-dimensional dataset, and the Mahalanobis distance for each sample is calculated. Samples are classified as outliers if their Mahalanobis distance exceeds a chi-squared critical value at a confidence level based on the threshold. Outliers are then excluded, and the model is retrained iteratively until no further performance improvement is observed. We developed this method as a part of our work on the MOC model replica presented in Houmann et al. [15], where it served as an automated version of the approach presented by Anderson et al. [3].

This method was integrated into PyHAT's library and Graphical User Interface (GUI), allowing users to configure the chi-squared threshold, number of PLS components, and maximum iterations. Users can select their dataset and regression target, configure the method, and run it through the GUI.

This contribution also included the development of a GUI component for the existing PyHAT GUI to configure and visualize the outlier removal process. This included utilities to select a threshold, select a given oxide for which to perform outlier removal, and a logging mechanism to display the number of outliers removed at each iteration in the GUI.

We also contributed by resolving a critical issue in the JADE implementation within PyHAT. The fix provided the ability to properly identify which of the original data points has the highest correlation with each independent component produced by the JADE algorithm. The correlation scores produced by this functionality can be used in a regression context, where a linear model learns the coefficients that best fit the relationship between the independent components and the original data points.

Finally, we made some contributions to improve the performance of various processes in PyHAT. At the time of writing, all contributions have been demonstrated to work as intended to the two USGS members responsible for managing PyHAT and are undergoing final review.

In recognition of our efforts, we received a formal letter of acknowledgment from the USGS. This letter acknowledges our contributions to PyHAT and highlights the positive impact our work has had on enhancing the tool's functionality and the field of chemometrics. The acknowledgment letter is presented in Appendix A.5.

9 CONCLUSION

This thesis set out to advance the analysis of LIBS data for predicting major oxide compositions in geological samples. By integrating sophisticated machine learning techniques and ensemble regression models, we aimed to tackle the substantial challenges posed by the high-dimensional, nonlinear nature of LIBS data.

Our research confronted and addressed critical challenges, including the complexities of high dimensionality, non-linearity, multicollinearity, and the limited availability of data. These issues traditionally hinder the accurate prediction of major oxides from spectral data, necessitating the development of robust and adaptive computational methodologies.

Throughout our study, we systematically explored a diverse range of machine learning models, categorized into ensemble learning models, linear and regularization models, and neural network models. Using the developed evaluation framework, we identified the strengths and limitations of each model in relation to predicting major oxides within the context of LIBS data analysis.

Normalization and transformation techniques played a crucial role in our approach. We investigated and employed various methods such as Z-Score standardization, Max Absolute scaling, Min-Max normalization, robust scaling, Norm 3, power transformation, and quantile transformation. These techniques proved vital for standardizing the data, managing different scales, and ultimately enhancing the performance of our models.

Dimensionality reduction techniques such as PCA and Kernel-PCA showed potential in managing the high dimensionality of the spectral data; however, their efficacy was not conclusively demonstrated.

One of the key innovations in our approach was the use of stacked generalization. Such an approach has seen limited use in the field LIBS data analysis and our work demonstrated its potential in this context. This ensemble method combined the

predictions of multiple base models, each trained on the same data, to form a meta-learner. By leveraging the strengths of various models and mitigating their individual weaknesses, this technique significantly improved generalization on unseen data.

We also designed and implemented a framework, using the automated hyperparameter optimization tool Optuna as its foundation. This framework allowed us to identify the most effective combinations of preprocessing methods and models tailored to the specific characteristics of each oxide, ensuring highly effective performance.

Finally, we designed and implemented a data partitioning method that addresses the challenges of data leakage and uneven distribution of extreme values, ensuring robust and reliable model evaluation.

The outcome of our work is a comprehensive catalog of machine learning models and preprocessing techniques for predicting major oxide compositions in LIBS data. This catalog, featuring highly effective configurations, provides a resource for future research and model and preprocessor selection.

Moreover, our contributions extend beyond this thesis. We integrated our findings into the PyHAT library developed by the USGS, thereby enhancing its capabilities for the scientific community.

In conclusion, by addressing the inherent challenges and developing a robust computational framework, this thesis has laid groundwork for future advancements in geochemical analysis and planetary exploration using LIBS data.

10 FUTURE WORK

The findings of this study present several opportunities for future research.

Firstly, regarding our data partitioning algorithm detailed in Section 6.2.1, we observed the significance of identifying the optimal percentile value p. This value is crucial for minimizing extreme values in the test set while preserving its overall representativeness. Future work should explore quantitative methods for determining this optimal value.

Another potential improvement to the validation and testing approach we delineate is incorporating supplementary extreme value testing after the primary evaluation. This type of testing could be conducted using a small, separate subset of extreme values to assess the model's performance in these critical scenarios. For example, this might involve slightly reducing the percentile value p and using the extreme values that fall within this reduced range to evaluate the model's effectiveness.

Tackling the challenges of limited data availability has proven important, as we mention throughout our report. The small dataset size inherently restricts the number of extreme values present. These extreme values are crucial for enhancing the model's generalizability, as they represent the most challenging cases to predict. Future research could investigate methods for augmenting the dataset with synthetic data, including extreme values, to provide the model with more exposure to these cases during training. This is a particularly challenging task, as it requires the production of synthetic data for a physics-based process. We contemplate that some approximation may be sufficient and could be used, for instance, as part of a transfer-learning project as initial training material.

Future work should also consider further experimentation with the choices of base estimators and meta-learners. Our study demonstrated that various model and preprocessor configurations perform well. However, identifying the optimal configurations and meta-learner for a specific oxide remains a challenging task. In this study, we used a simple grouping method to ensure diversity in our base estimator selection, choosing from the top-performing configurations. We also experimented briefly with a grid search approach, which could be examined further. However, using a variation of the optimization framework we presented is likely to provide better trade-offs, as we have discussed.

REFERENCES

[1] Quantitative analysis of sedimentary rocks using laser-induced breakdown spectroscopy: comparison of support vector regression and partial least squares regression chemometric methods. 30. ISSN 0267-9477, 1364-5544. doi: 10.1039/C5JA00255A. URL https://xlink.rsc.org/?DOI=C5JA00255A.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[3] Ryan B. Anderson, Samuel M. Clegg, Jens Frydenvang, Roger C. Wiens, Scott McLennan, Richard V. Morris, Bethany Ehlmann, and M. Darby Dyar. Improved accuracy in quantitative laser-induced breakdown spectroscopy using sub-models. 129:49–57, . ISSN 0584-8547. doi: 10.1016/j.sab.2016.12.002. URL https://www.sciencedirect.com/science/article/pii/S0584854716303925.

[4] Ryan B. Anderson, Olivier Forni, Agnes Cousin, Roger C. Wiens, Samuel M. Clegg, Jens Frydenvang, Travis S. J. Gabriel, Ann Ollila, Susanne Schröder, Olivier Beyssac, Erin Gibbons, David S. Vogt, Elise Clavé, Jose-Antonio Manrique, Carey Legett, Paolo Pilleri, Raymond T. Newell, Joseph Sarrao, Sylvestre Maurice, Gorka Arana, Karim Benzerara, Pernelle Bernardi, Sylvain Bernard, Bruno Bousquet, Adrian J. Brown, César Alvarez-Llamas, Baptiste Chide, Edward Cloutis, Jade Comellas, Stephanie Connell, Erwin Dehouck, Dorothea M. Delapp, Ari Essunfeld, Cecile Fabre, Thierry Fouchet, Cristina Garcia-Florentino, Laura García-Gómez, Patrick Gasda, Olivier Gasnault, Elisabeth M. Hausrath, Nina L. Lanza, Javier Laserna, Jeremie Lasue, Guillermo Lopez, Juan Manuel Madariaga, Lucia Mandon, Nicolas Mangold, Pierre-Yves Meslin, Anthony E. Nelson, Horton Newsom, Adriana L. Reyes-Newell, Scott Robinson, Fernando Rull, Shiv Sharma, Justin I. Simon, Pablo Sobron, Imanol Torre Fernandez, Arya Udry, Dawn Venhaus, Scott M. McLennan, Richard V. Morris, and Bethany Ehlmann. Post-landing major element quantification using SuperCam laser induced breakdown spectroscopy. 188:106347, . ISSN 0584-8547. doi: 10.1016/j.sab.2021.106347. URL https://www.sciencedirect.com/science/article/pii/S0584854721003049.

[5] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 00359246. URL http://www.jstor.org/stable/2984418.

[6] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov. ISBN 978-1-77700-547-4.

[7] Samuel M. Clegg, Roger C. Wiens, Ryan Anderson, Olivier Forni, Jens Frydenvang, Jeremie Lasue, Agnes Cousin, Valérie Payré, Tommy Boucher, M. Darby Dyar, Scott M. McLennan, Richard V. Morris, Trevor G. Graff, Stanley A. Mertzman, Bethany L. Ehlmann, Ines Belgacem, Horton Newsom, Ben C. Clark, Noureddine Melikechi, Alissa Mezzacappa, Rhonda E. McInroy, Ronald Martinez, Patrick Gasda, Olivier Gasnault, and Sylvestre Maurice. Recalibration of the Mars Science Laboratory ChemCam instrument with an expanded geochemical database. 129:64–85. ISSN 0584-8547. doi: 10.1016/j.sab.2016.12.003. URL https://www.sciencedirect.

com/science/article/pii/S0584854716303913.

[8] Harris Drucker, Christopher J C Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support Vector Regression Machines.

[9] Tony Duan, Anand Avati, Daisy Yi Ding, Khanh K. Thai, Sanjay Basu, Andrew Y. Ng, and Alejandro Schuler. NGBoost: Natural Gradient Boosting for Probabilistic Prediction, June 2020. URL http://arxiv.org/abs/1910.03225. arXiv:1910.03225 [cs, stat].

[10] J. El Haddad, M. Villot-Kadri, A. Ismaël, G. Gallou, K. Michel, D. Bruyère, V. Laperche, L. Canioni, and B. Bousquet. Artificial neural network for on-site quantitative analysis of soils using laser induced breakdown spectroscopy. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 79-80:51–57, 2013. ISSN 0584-8547. doi: https://doi.org/10.1016/j.sab.2012.11.007. URL https://www.sciencedirect.com/science/article/pii/S0584854712003679.

[11] J. El Haddad, M. Villot-Kadri, A. Ismaël, G. Gallou, K. Michel, D. Bruyère, V. Laperche, L. Canioni, and B. Bousquet. Artificial neural network for on-site quantitative analysis of soils using laser induced breakdown spectroscopy. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 79-80:51–57, January 2013. ISSN 0584-8547. doi: 10.1016/j.sab.2012.11.007. URL https://www.sciencedirect.com/science/article/pii/S0584854712003679.

[12] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, April 2006. ISSN 1573-0565. doi: 10.1007/s10994-006-6226-1. URL https://doi.org/10.1007/s10994-006-6226-1.

[13] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, third edition edition. ISBN 978-1-09-812597-4.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, second edition, 2009. ISBN 978-0-387-84857-0.

[15] Christian Bager Bach Houmann, Patrick Frostholm Østergaard, and Ivik Lau Dalgas Hostrup. Identifying limitations in the ChemCam multivariate oxide composition model for elemental quantification in martian geological samples. URL https://vbn-aau-dk.zorac.aub.aau.dk/ws/files/659161040/p9_report_31_01_24.pdf.

[16] Jianglin Huang, Yan-Fu Li, and Min Xie. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Inf. Softw. Technol.*, 67:108–127, 2015. URL https://api.semanticscholar.org/CorpusID:19650667.

[17] Washington University in St. Louis. Pds geoscience node. URL https://pds-geosciences.wustl.edu/. Last Accessed: 2023-09-01.

[18] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in Python*. Springer, 2023.

[19] Gookseon Jeon, Hohyun Keum, Hyunkeun Lee, Kyunghwan Oh, and Janghee Choi. Effects of feature engineering on the robustness of laser-induced breakdown spectroscopy for industrial steel classification. 212:106857. ISSN 0584-8547. doi: 10.1016/j.sab.2024.106857. URL https://www.sciencedirect.com/science/article/pii/S0584854724000016.

[20] Micheline Kamber Jiawei Han, Jian Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2nd edition edition, 2011. ISBN 0123814804.

[21] Lu-Ning Li, Xiang-Feng Liu, Wei-Ming Xu, Jian-Yu Wang, and Rong Shu. A laser-induced breakdown spectroscopy multi-component quantitative analytical method based on a deep convolutional neural network. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 169:105850, July 2020. ISSN 0584-8547. doi: 10.1016/j.sab.2020.105850. URL https://www.sciencedirect.com/science/article/pii/S0584854719304938.

[22] Hao Liu, Kai Han, Weiqiang Yang, and Minsun Chen. Recent advances in machine learning methodologies for LIBS quantitative analysis. doi: 10.5772/intechopen.1004414.

[23] mars.nasa.gov. Mars science laboratory | missions, . URL https://mars.nasa.gov/mars-exploration/missions/mars-science-laboratory. Last Accessed: 2024-03-12.

[24] mars.nasa.gov. Mars observer | missions, . URL https://mars.nasa.gov/mars-exploration/missions/mars-observer. Last Accessed: 2024-01-23.

[25] mars.nasa.gov. Mars exploration rovers | missions, . URL https://mars.nasa.gov/mars-exploration/missions/mars-exploration-rovers.

[26] mars.nasa.gov. Viking 1 & 2 | missions, . URL https://mars.nasa.gov/mars-exploration/missions/viking-1-2. Last Accessed: 2024-01-23.

[27] Ibomoiye Domor Mienye and Yanxia Sun. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access*, 10:99129–99149, 2022. doi: 10.1109/ACCESS.2022.3207287.

[28] Samuel Moncayo, Ludovic Duponchel, Niloofar Mousavipak, Gérard Panczer, Florian Trichard, Bruno Bousquet, Frédéric Pelascini, and Vincent Motto-Ros. Exploration of megapixel hyperspectral libs images using principal component analysis. *J. Anal. At. Spectrom.*, 33:210–220, 2018. doi: 10.1039/C7JA00398F. URL http://dx.doi.org/10.1039/C7JA00398F.

[29] Bohdan Pavlyshenko. Using stacking approaches for machine learning models. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pages 255–258, 2018. doi: 10.1109/DSMP.2018.8478522.

[30] P. Pořízka, A. Demidov, J. Kaiser, J. Keivanian, I. Gornushkin, U. Panne, and J. Riedel. Laser-induced breakdown spectroscopy for in situ qualitative and quantitative analysis of mineral ores. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 101:155–163, 2014. ISSN 0584-8547. doi: https://doi.org/10.1016/j.sab.2014.08.027. URL https://www.sciencedirect.com/science/article/pii/S058485471400202X.

[31] Pavel Pořízka, Jakub Klus, Erik Képeš, David Prochazka, David W. Hahn, and Jozef Kaiser. On the utilization of principal component analysis in laser-induced breakdown spectroscopy data analysis, a review. *Spectrochimica Acta Part B: Atomic Spectroscopy*, 148:65–82, 2018. ISSN 0584-8547. doi: https://doi.org/10.1016/j.sab.2018.05.030. URL https://www.sciencedirect.com/science/article/pii/S0584854718301526.

[32] Mohsen Rezaei, Fatemeh Rezaei, and Parvin Karimi. Using various machine learning algorithms for quantitative analysis in LIBS technique.

[33] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 06 2018. ISBN 9780262256933. doi: 10.7551/mitpress/4175.001.0001. URL https://doi.org/10.7551/mitpress/4175.001.0001.

[34] Qi Shi, Guanghui Niu, Qingyu Lin, Taoran Xu, Fengjun Li, and Y. Duan. Quantitative analysis of sedimentary rocks using laser-induced breakdown spectroscopy: comparison of support vector regression and partial least squares regression chemometric methods. *Journal of Analytical Atomic Spectrometry*, 30:2384–2393, 2015. doi: 10.1039/C5JA00255A.

[35] J.-B. Sirven, B. Bousquet, L. Canioni, and L. Sarger. Laser-induced breakdown spectroscopy of composite samples: Comparison of advanced chemometrics methods. *Analytical Chemistry*, 78(5):1462–1469, 2006. doi: 10.1021/ac051721p. URL https://doi.org/10.1021/ac051721p. PMID: 16503595.

[36] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004. ISSN 1573-1375. doi: 10.1023/B:STCO.0000035301.49549.88. URL https://doi.org/10.1023/B:STCO.0000035301.49549.88.

[37] Weiran Song, Zongyu Hou, Weilun Gu, Muhammad Sher Afgan, Jiacheng Cui, Hui Wang, Yun Wang, and Zhe Wang. Incorporating domain knowledge into machine learning for laser-induced breakdown spectroscopy quantification. 195:106490. ISSN 0584-8547. doi: 10.1016/j.sab.2022.106490. URL https://www.sciencedirect.com/science/article/pii/S0584854722001343.

[38] Chen Sun, Weijie Xu, Yongqi Tan, Yuqing Zhang, Zengqi Yue, Long Zou, Sahar Shabbir, Mengting Wu, Fengye Chen, and Jin Yu. From machine learning to transfer learning in laser-induced breakdown spectroscopy analysis of rocks for Mars exploration. 11(1):21379. ISSN 2045-2322. doi: 10.1038/s41598-021-00647-2. URL https://www.nature.com/articles/s41598-021-00647-2.

[39] Xavier Vasques. *Machine Learning Theory and Applications: Hands-on Use Cases with Python on Classical and Quantum Machines*. Wiley, Hoboken, New Jersey, 2024. ISBN 9781394220618.

[40] R. C. Wiens, S. Maurice, J. Lasue, O. Forni, R. B. Anderson, S. Clegg, S. Bender, D. Blaney, B. L. Barraclough, A. Cousin, L. Deflores, D. Delapp, M. D. Dyar, C. Fabre, O. Gasnault, N. Lanza, J. Mazoyer, N. Melikechi, P. Y. Meslin, H. Newsom, A. Ollila, R. Perez, R. L. Tokar, and D. Vaniman. Pre-flight calibration and initial data processing for the ChemCam laser-induced breakdown spectroscopy instrument on the Mars Science Laboratory rover. 82:1–27. ISSN 0584-8547. doi: 10.1016/j.sab.2013.02.003. URL https://www.sciencedirect.com/science/article/pii/S0584854713000505.

[41] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition* (*Morgan Kaufmann Series in Data Management Systems*). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120884070.

[42] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[43] Fan Yang, Weiming Xu, Zhicheng Cui, Xiangfeng Liu, Xuesen Xu, Liangchen Jia, Yuwei Chen, Rong Shu, and Luning Li. Convolutional Neural Network Chemometrics for Rock Identification Based on Laser-Induced Breakdown Spectroscopy Data in Tianwen-1 Pre-Flight Experiments. 14(21):5343. ISSN 2072-4292. doi: 10.3390/rs14215343. URL https://www.mdpi.com/2072-4292/14/21/5343.

[44] InKwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. ISSN 0006-3444.

[45] Hui Zou and Trevor Hastie. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, April 2005. ISSN 1369-7412, 1467-9868. doi: 10.1111/j.1467-9868.2005.00503.x. URL https://academic.oup.com/jrsssb/article/67/2/301/7109482.

A APPENDIX

A.1 Web-Based Platform for Data Partitioning Evaluation

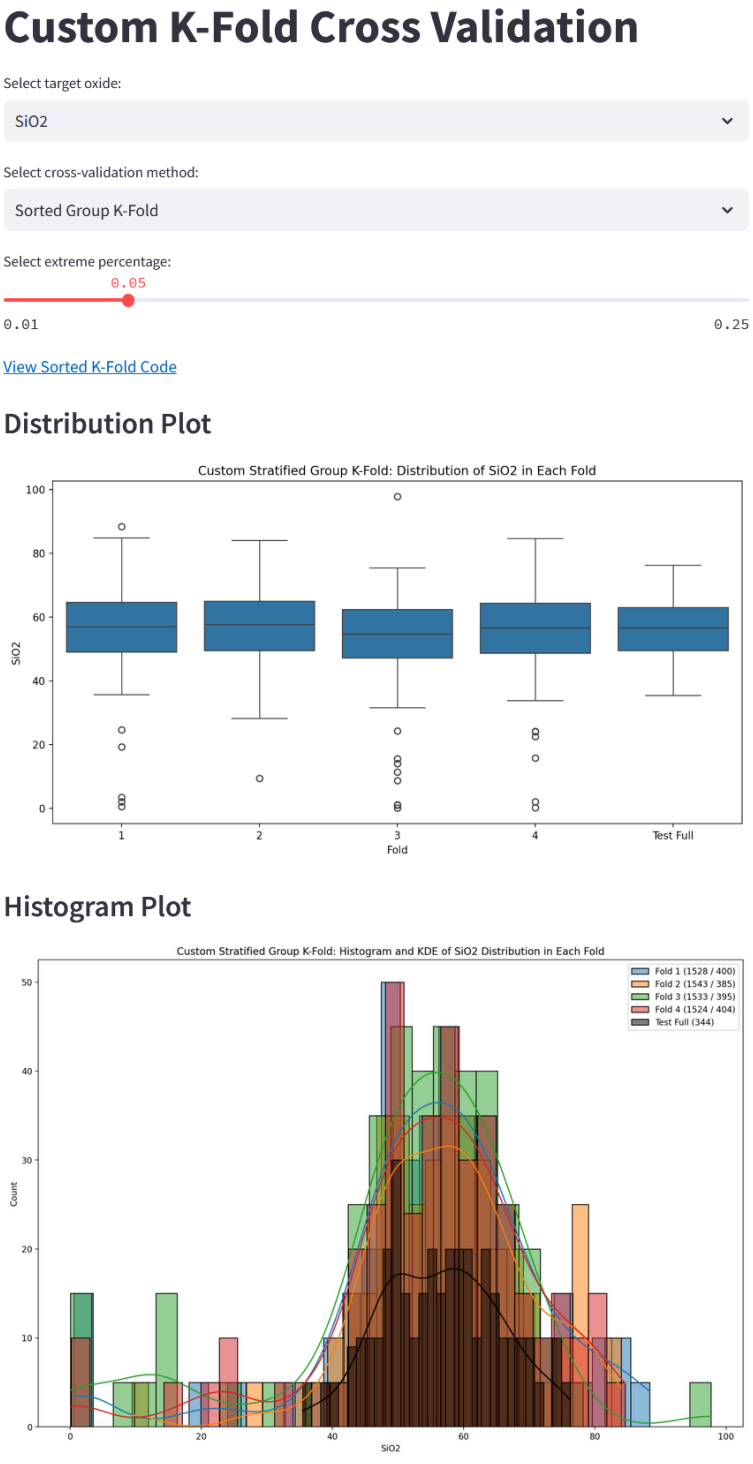

Fig. A.1. Web-based platform used to determine the optimal value of p for the data partitioning algorithm.

A.2 Cross-Validation Fold Plots for Major Oxides

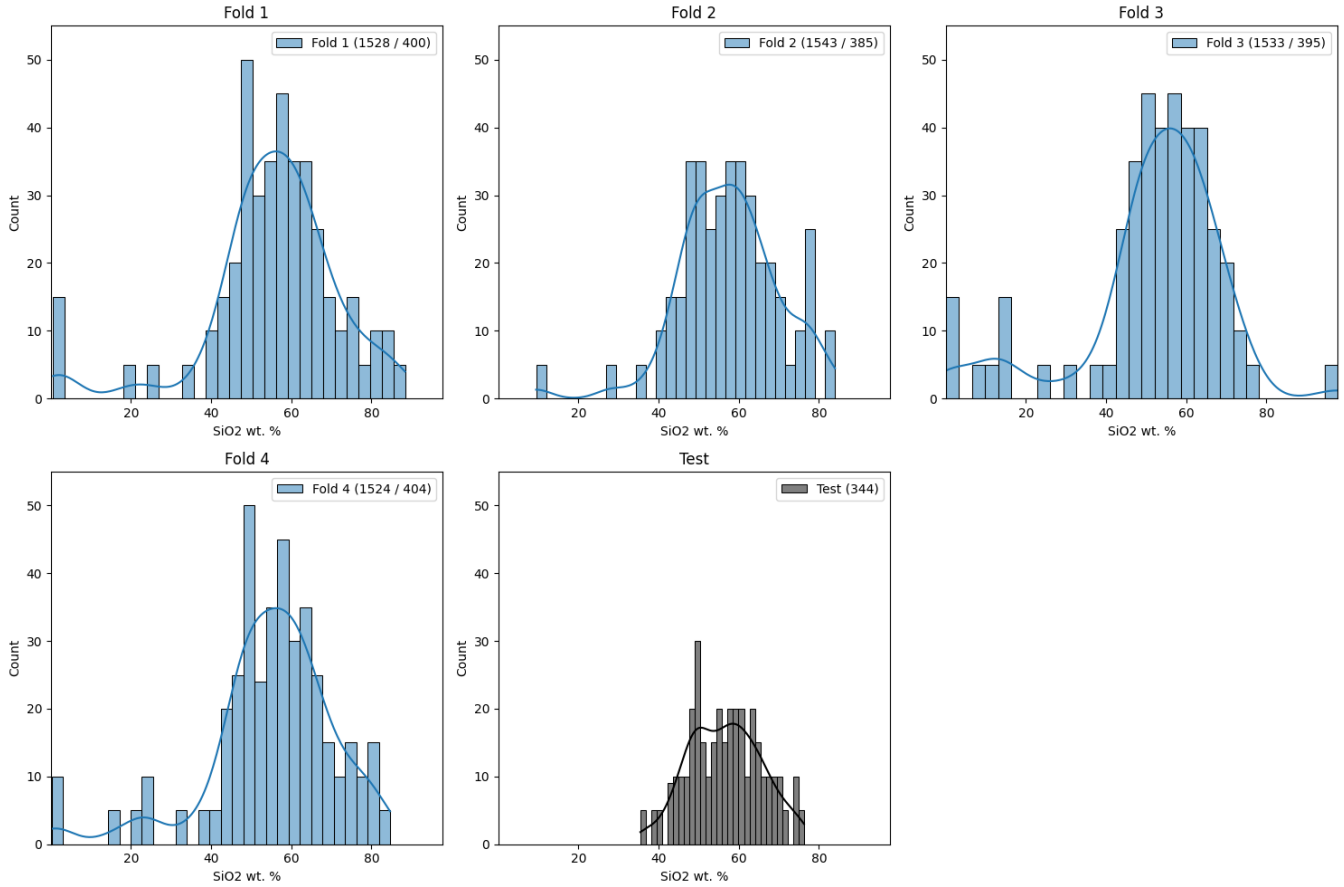

Fig. A.2. Histogram and KDE of SiO2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents SiO2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

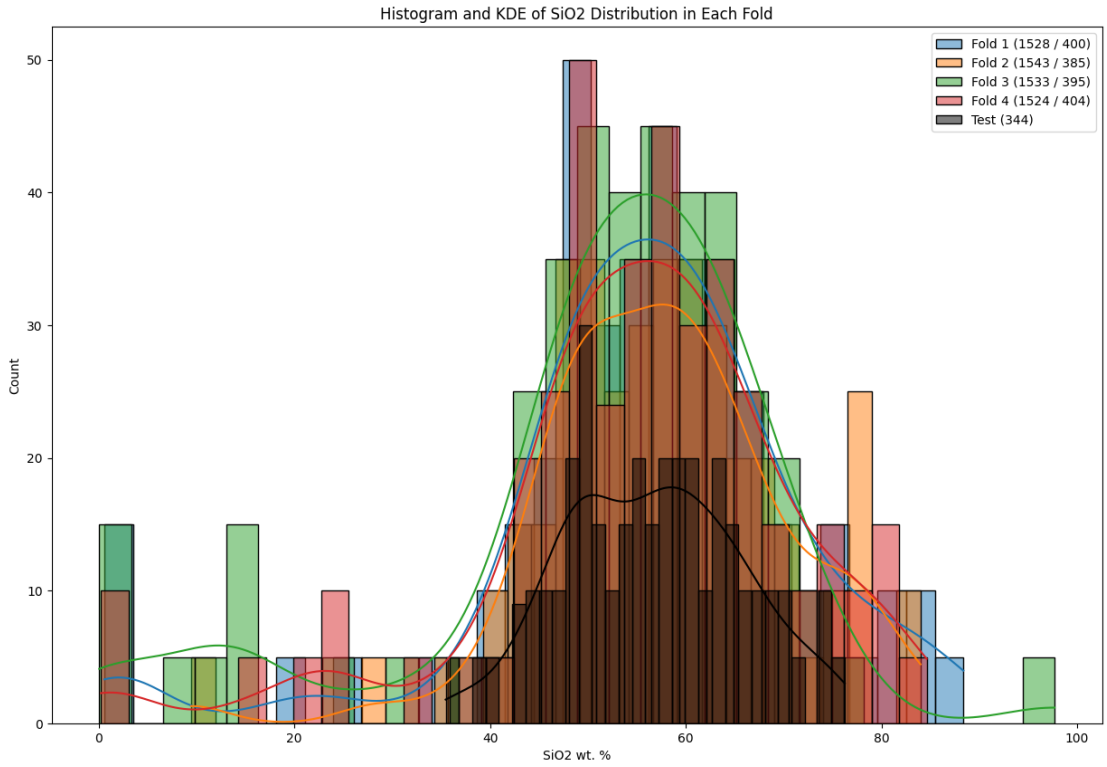

Fig. A.3. Combined Histogram and KDE of SiO2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents SiO2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

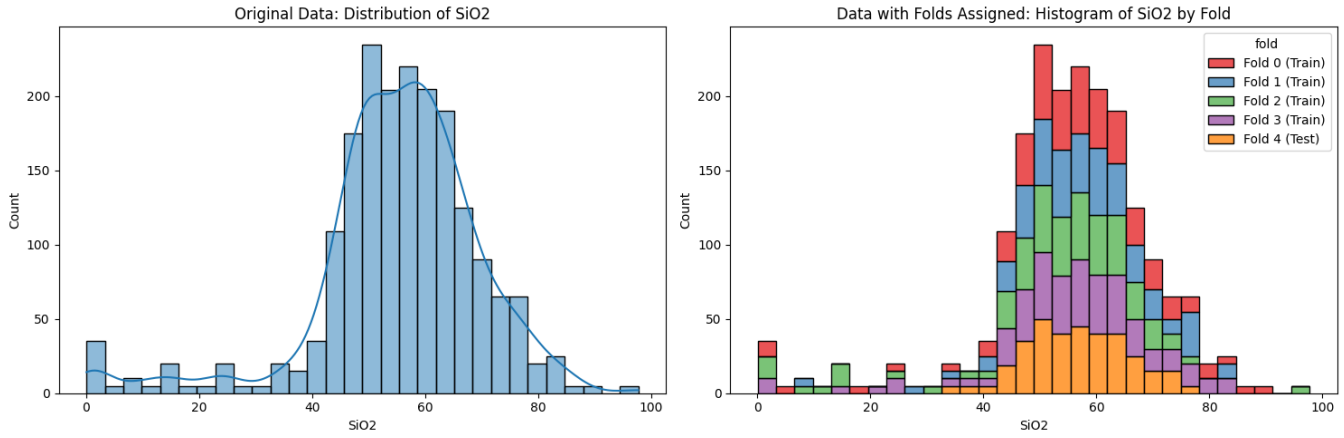

Fig. A.4. Distribution of SiO2 concentrations before and after fold assignment. The left plot shows the original distribution of SiO2, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

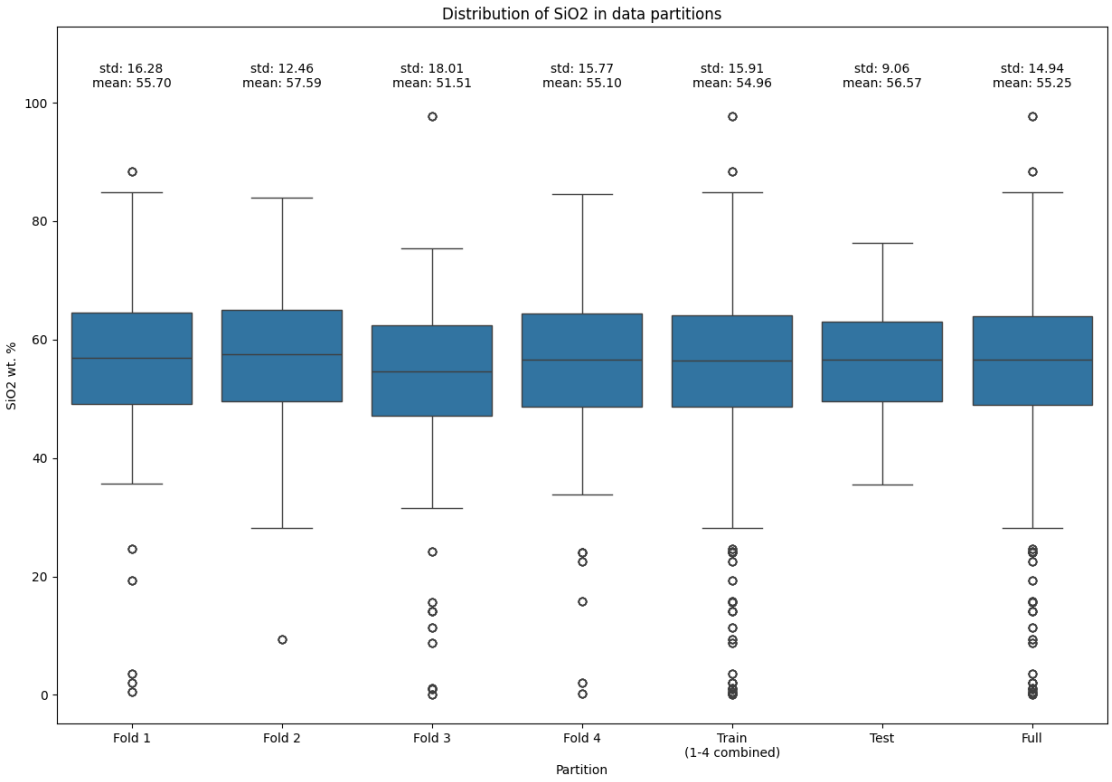

Fig. A.5. Distribution of SiO2 concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

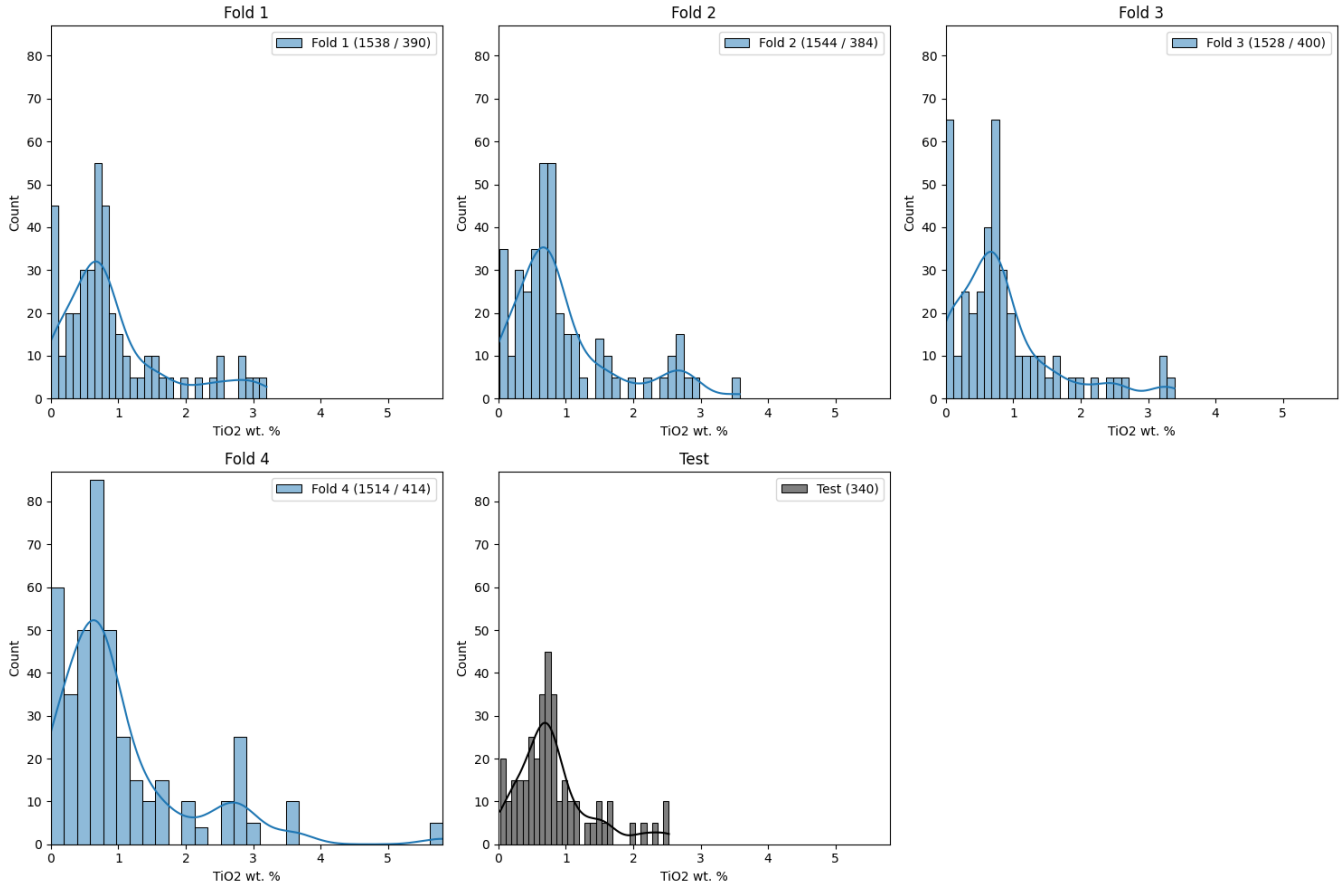

Fig. A.6. Histogram and KDE of TiO2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents TiO2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

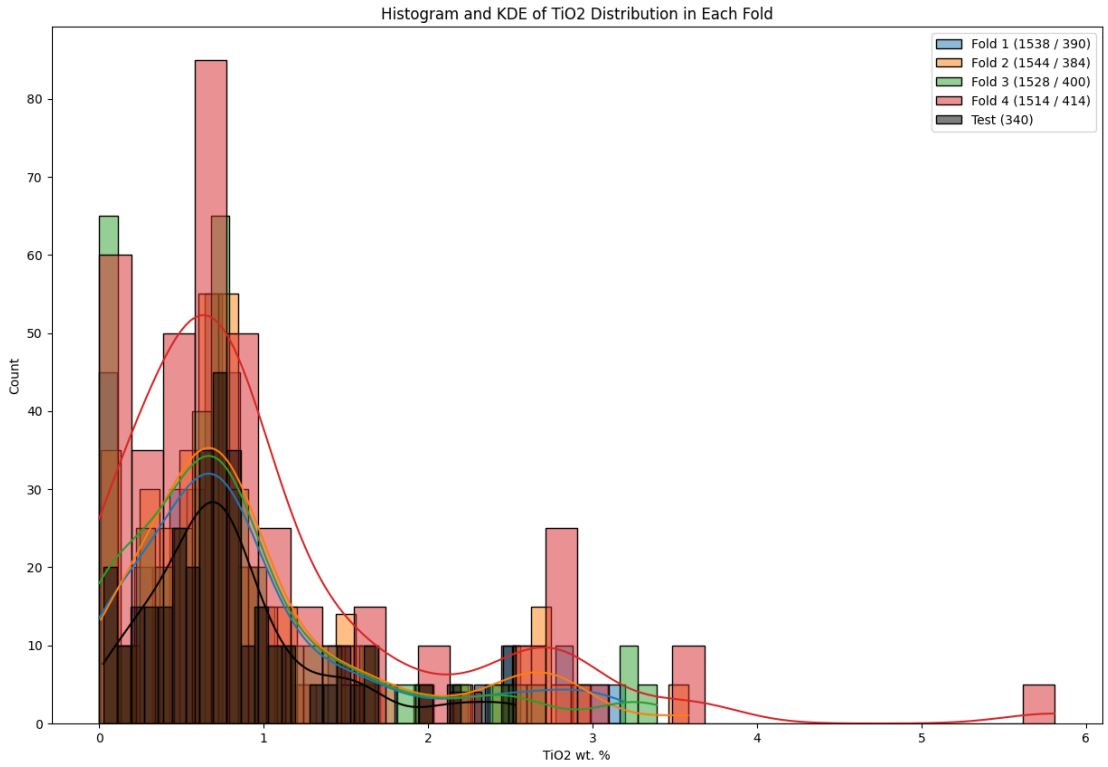

Fig. A.7. Combined Histogram and KDE of TiO2 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents TiO2 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

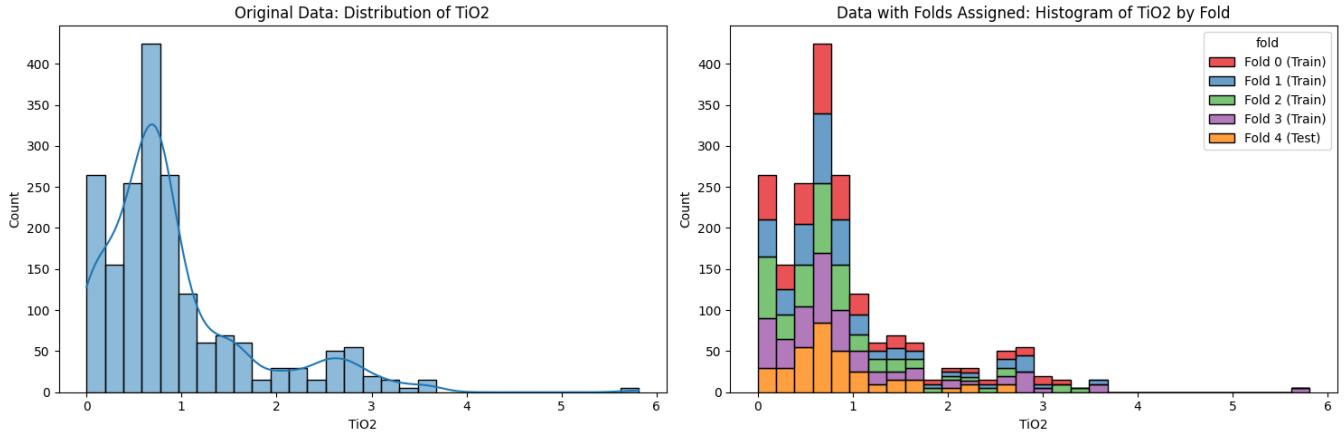

Fig. A.8. Distribution of TiO2 concentrations before and after fold assignment. The left plot shows the original distribution of TiO2, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

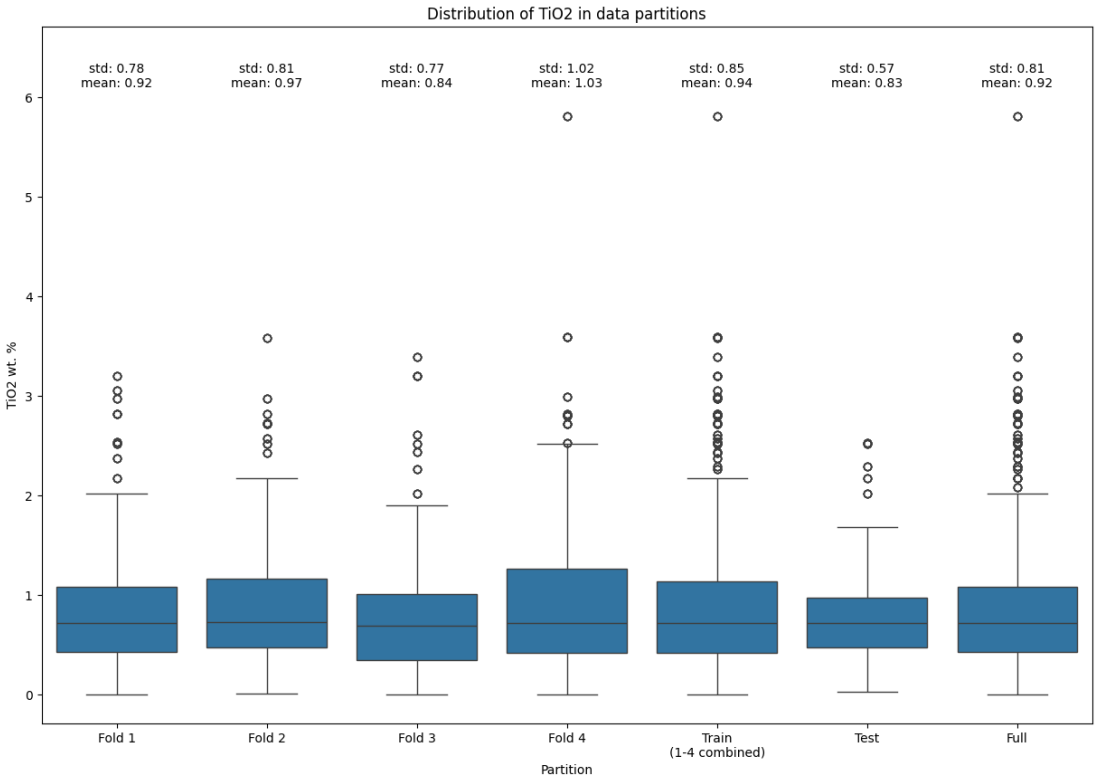

Fig. A.9. Distribution of TiO2 concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

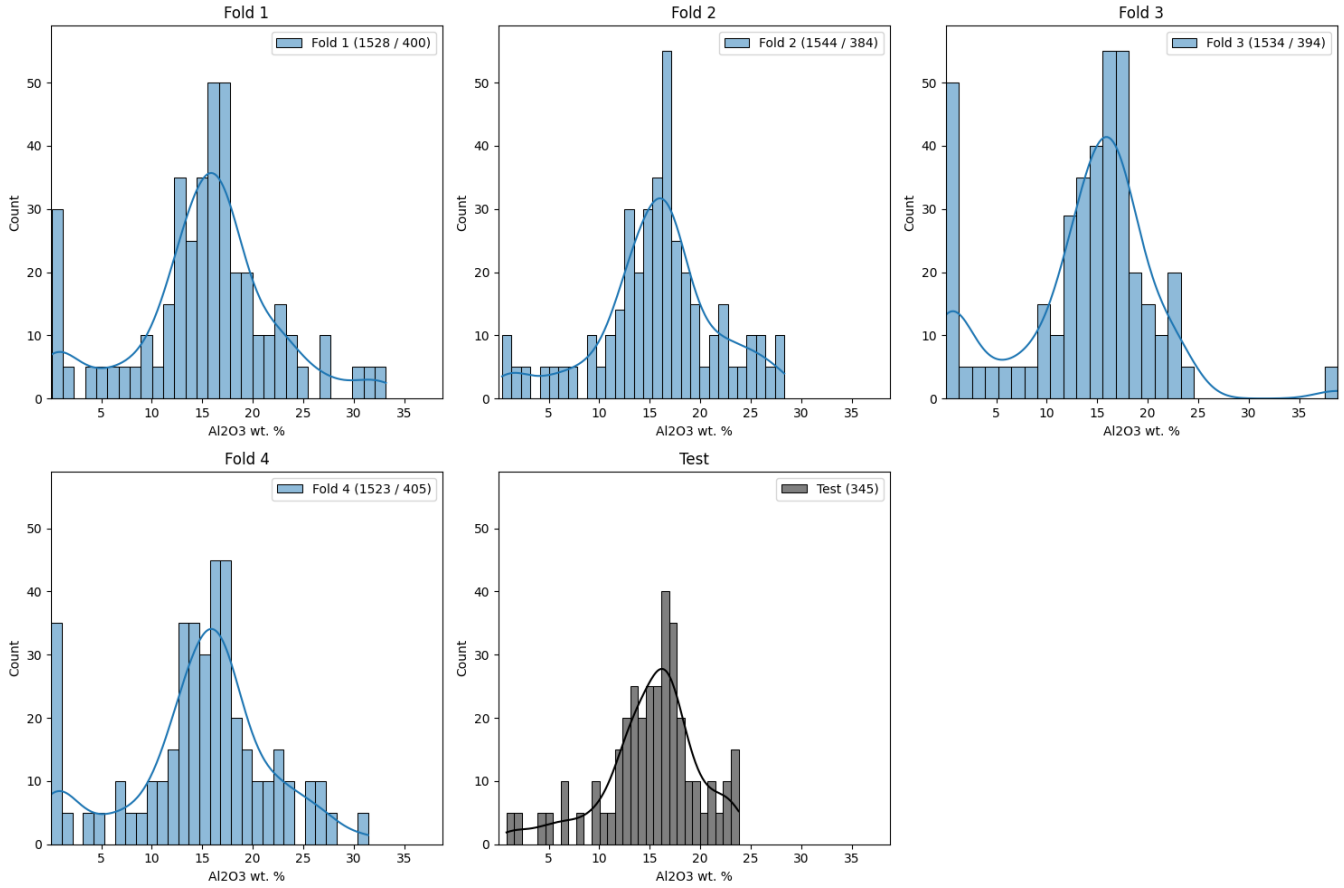

Fig. A.10. Histogram and KDE of Al2O3 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents Al2O3 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

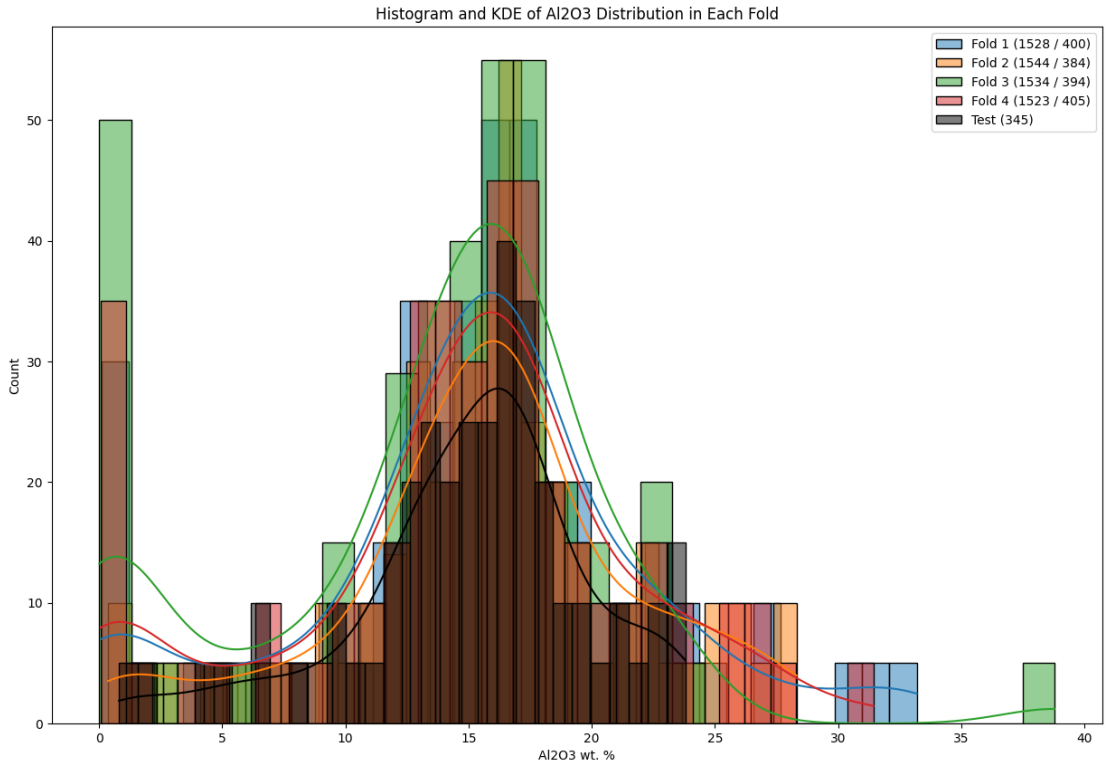

Fig. A.11. Combined Histogram and KDE of Al2O3 distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents Al2O3 concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

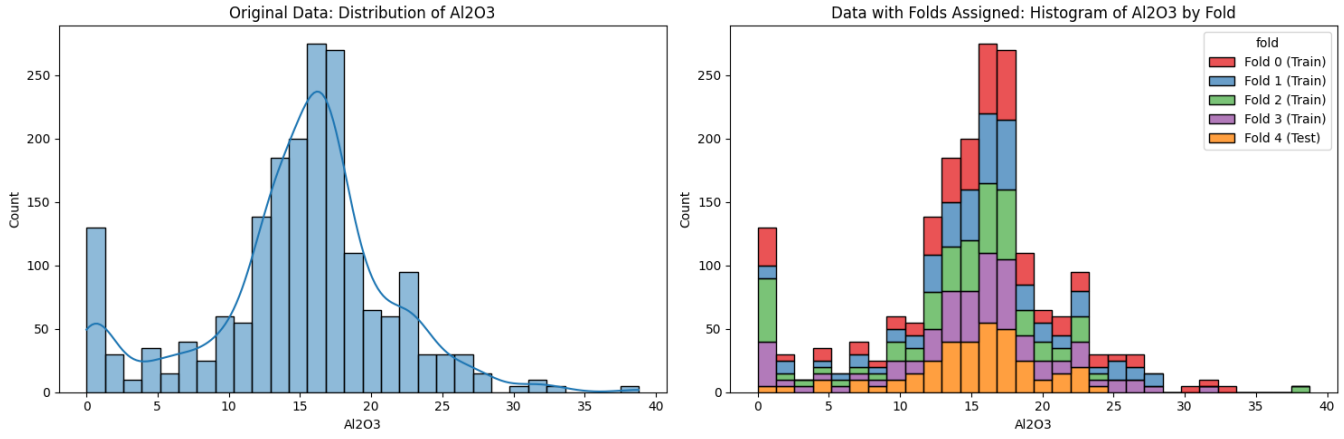

Fig. A.12. Distribution of Al2O3 concentrations before and after fold assignment. The left plot shows the original distribution of Al2O3, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

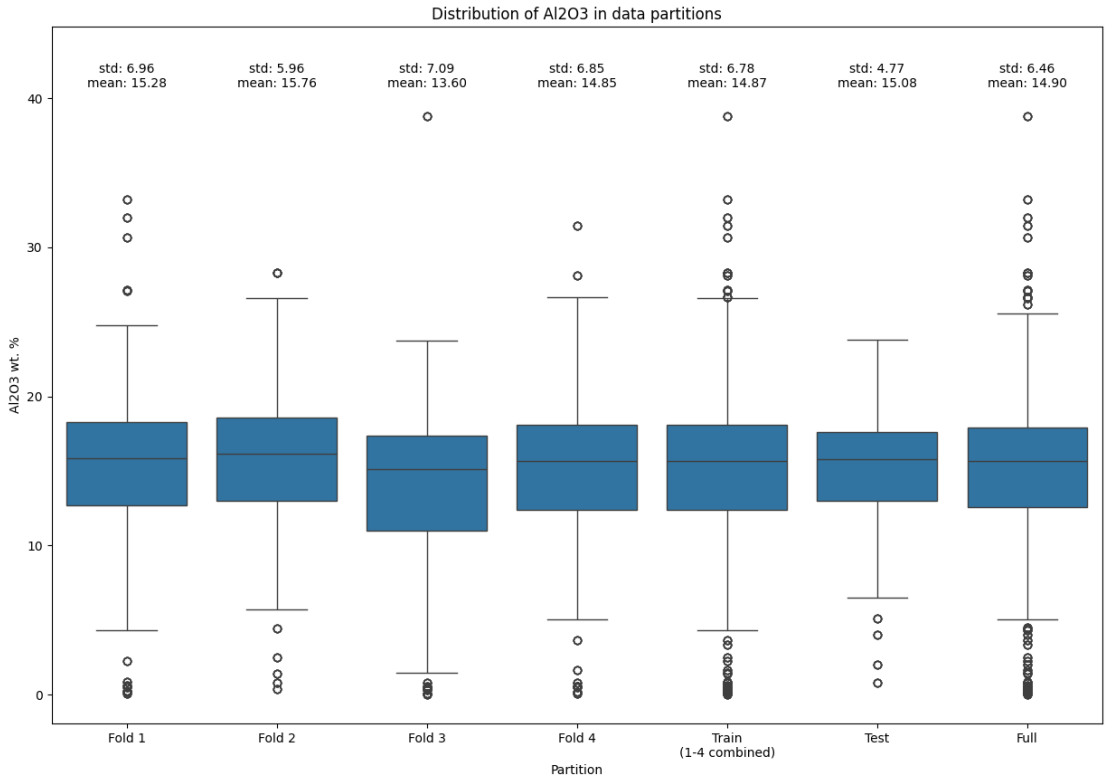

Fig. A.13. Distribution of Al2O3 concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

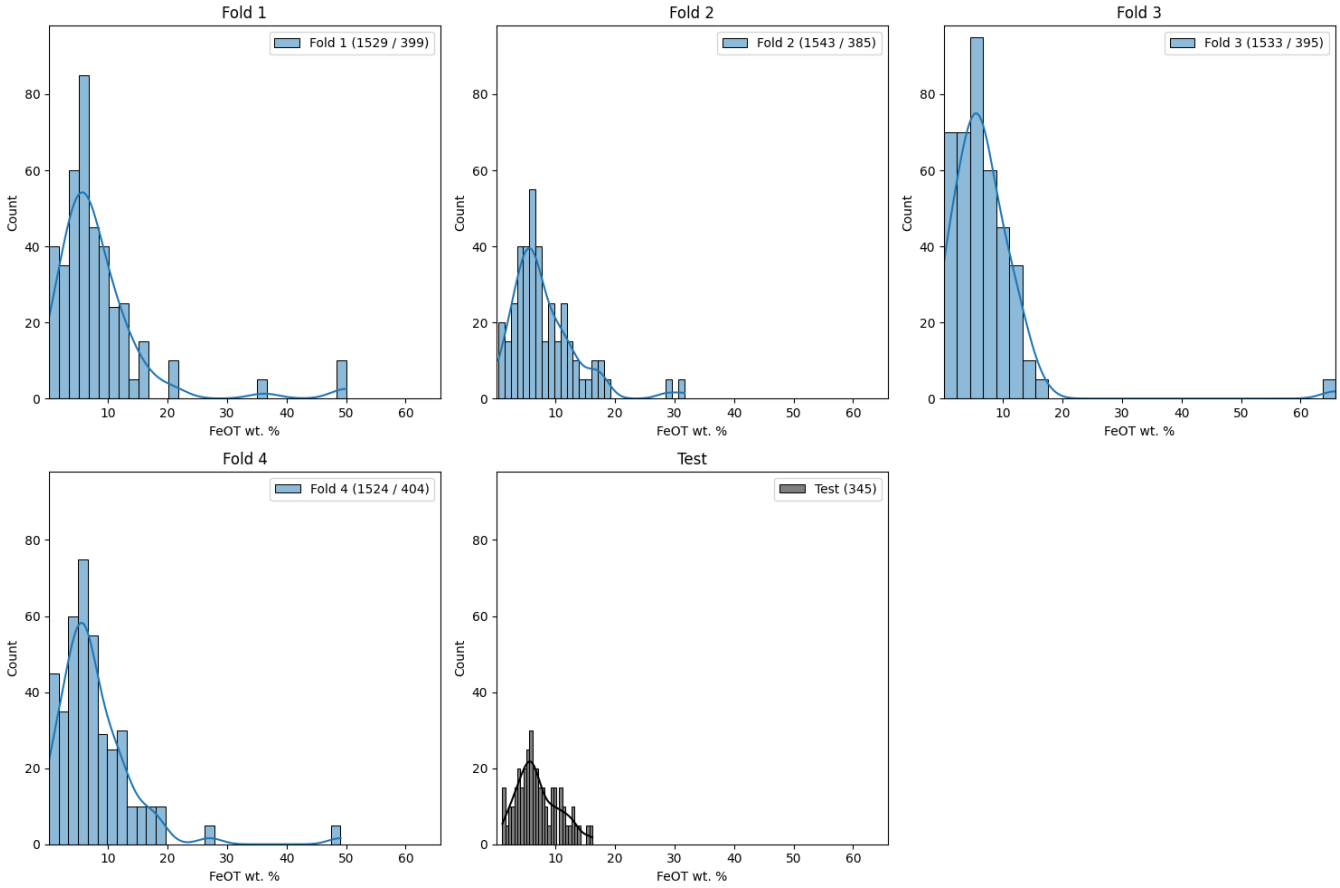

Fig. A.14. Histogram and KDE of FeOT distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents FeOT concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

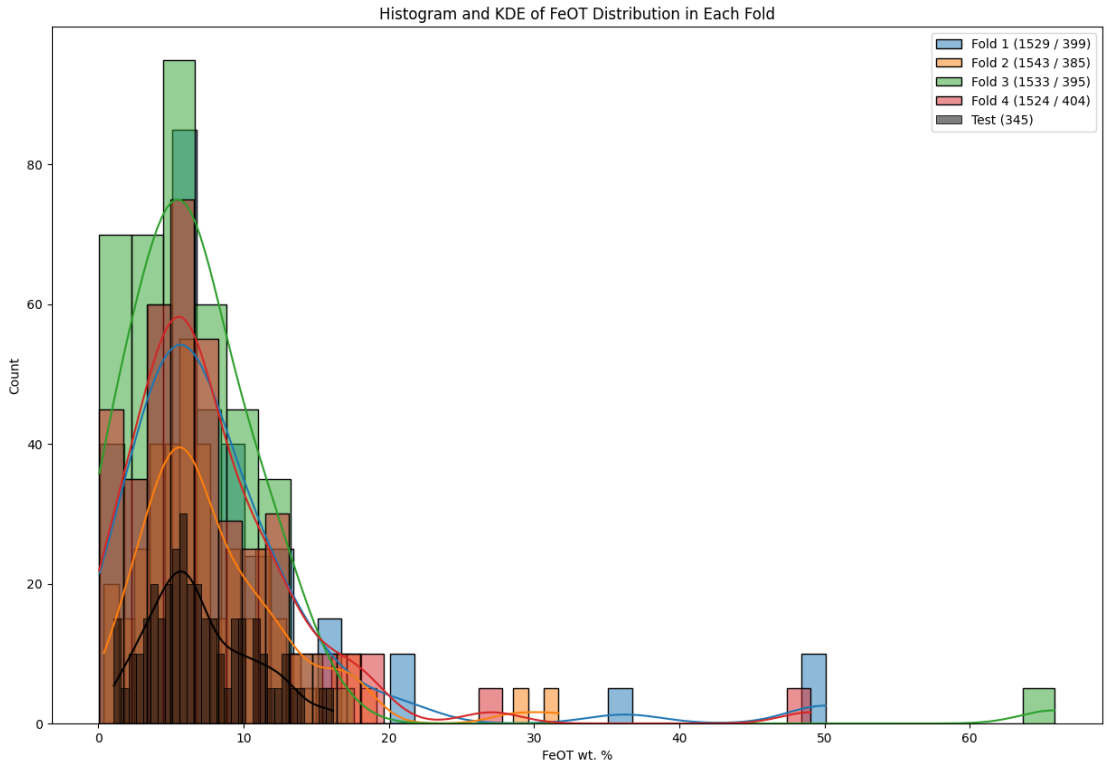

Fig. A.15. Combined Histogram and KDE of FeOT distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents FeOT concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

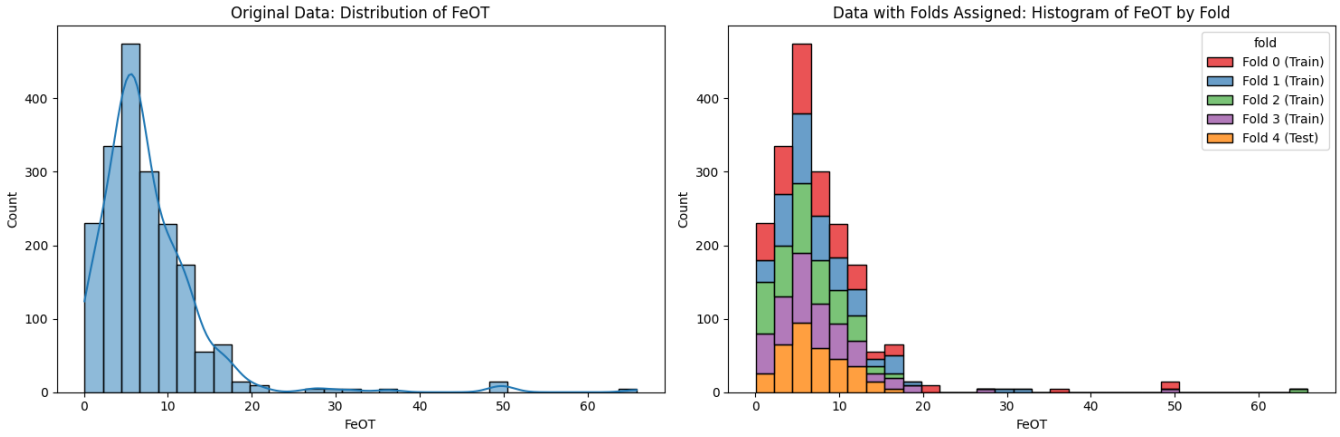

Fig. A.16. Distribution of FeOT concentrations before and after fold assignment. The left plot shows the original distribution of FeOT, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

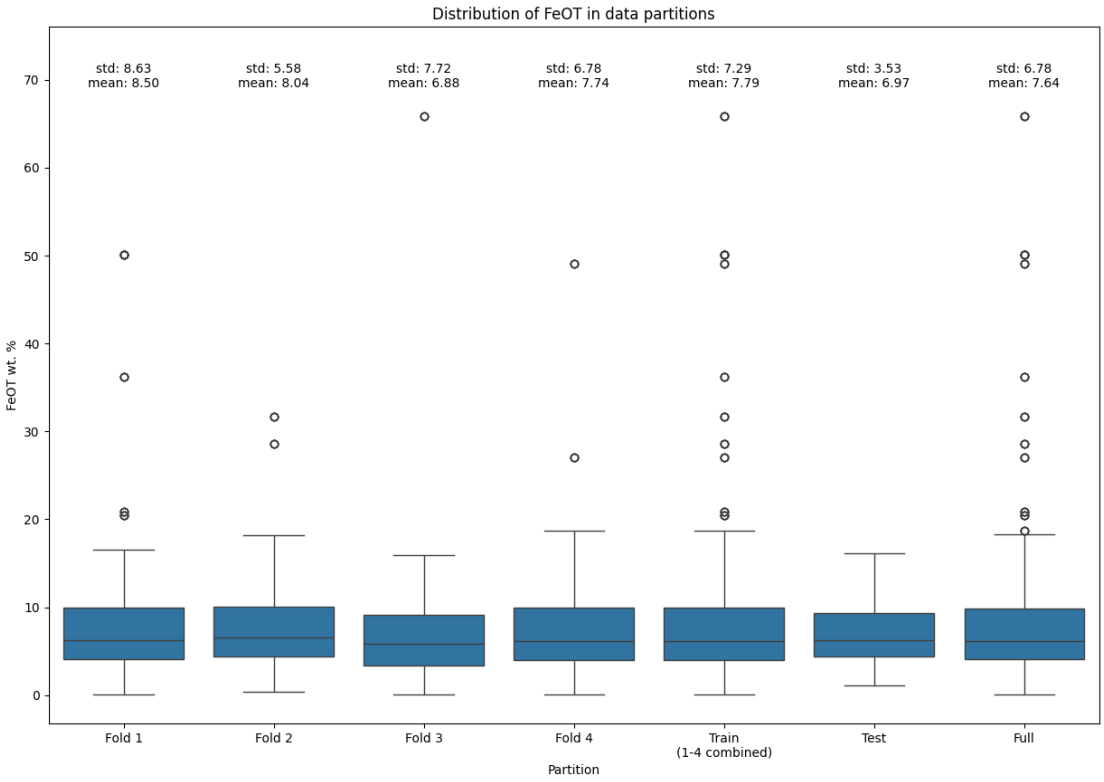

Fig. A.17. Distribution of FeOT concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

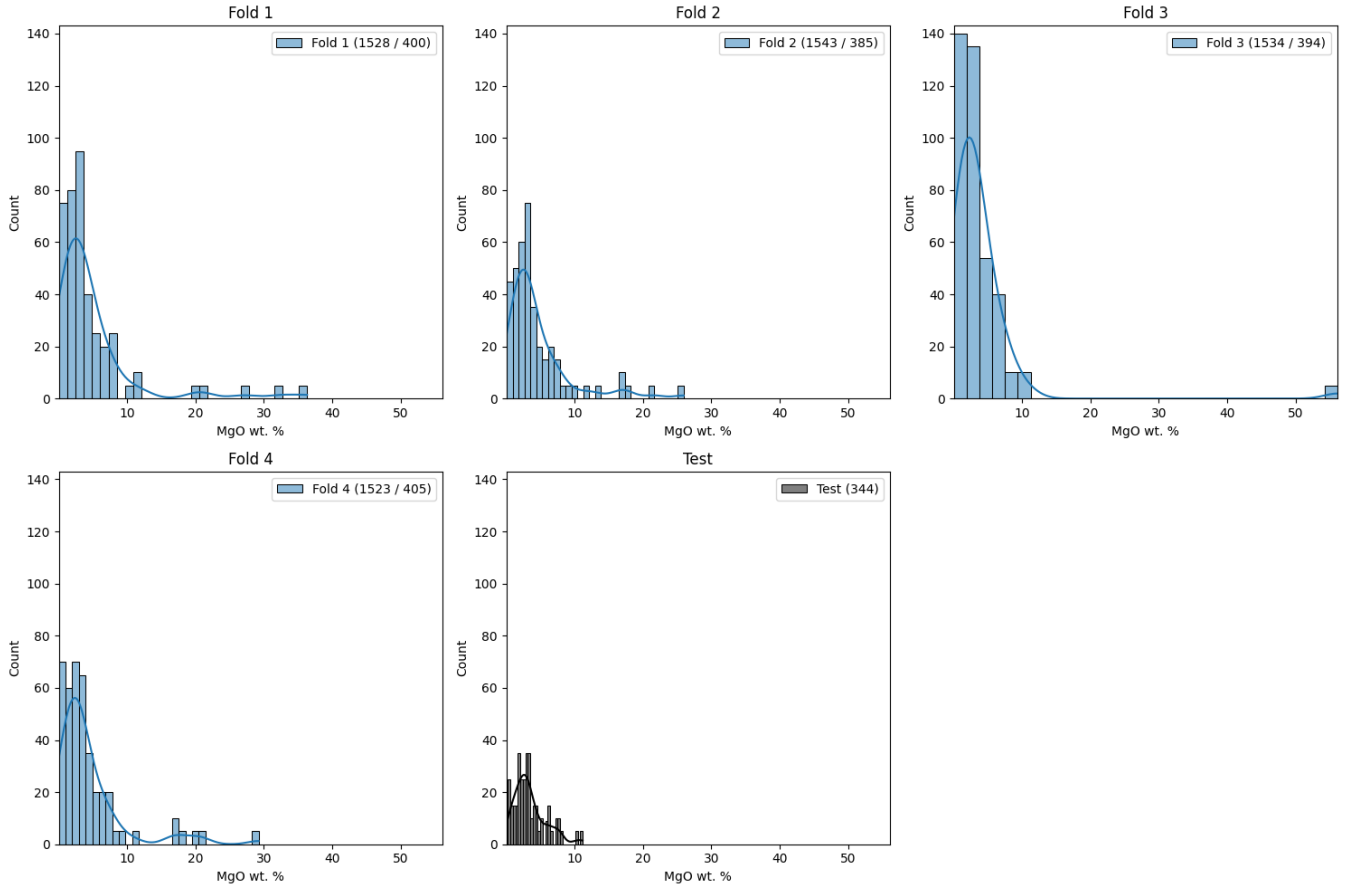

Fig. A.18. Histogram and KDE of MgO distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents MgO concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

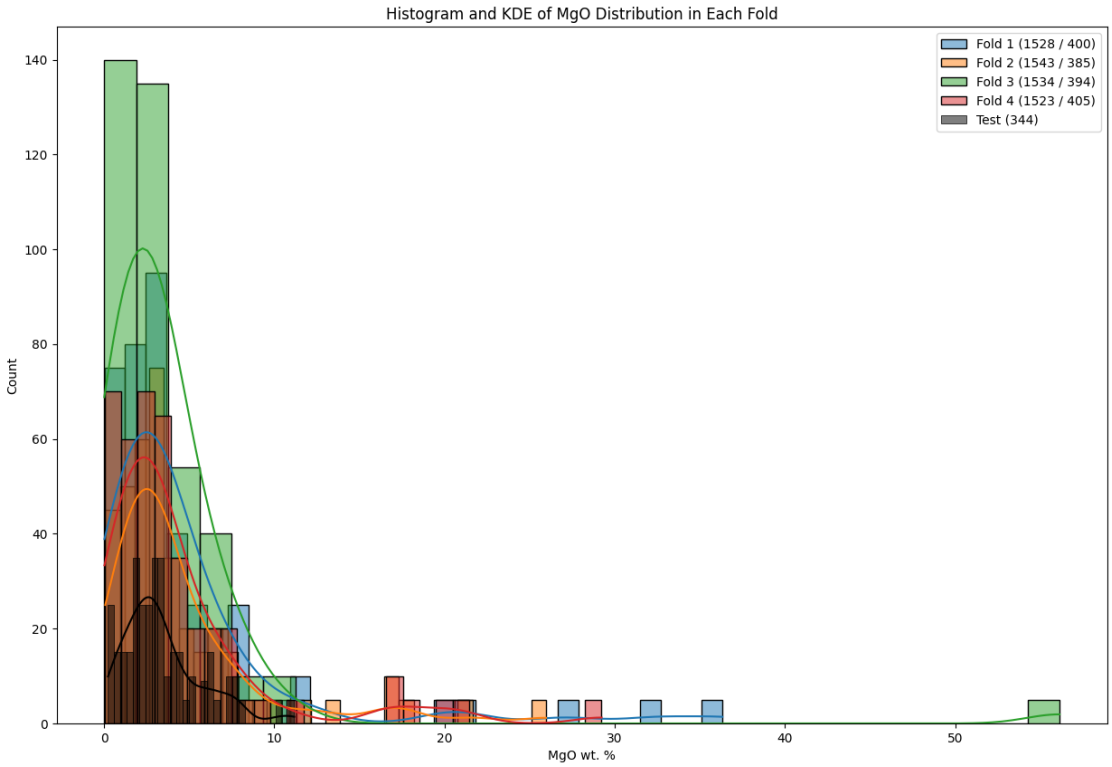

Fig. A.19. Combined Histogram and KDE of MgO distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents MgO concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

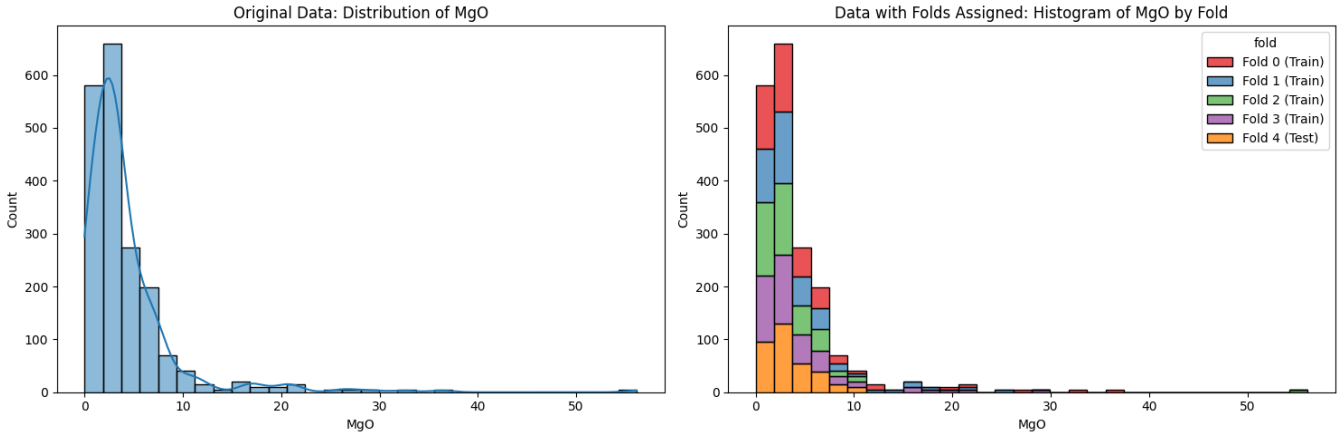

Fig. A.20. Distribution of MgO concentrations before and after fold assignment. The left plot shows the original distribution of MgO, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

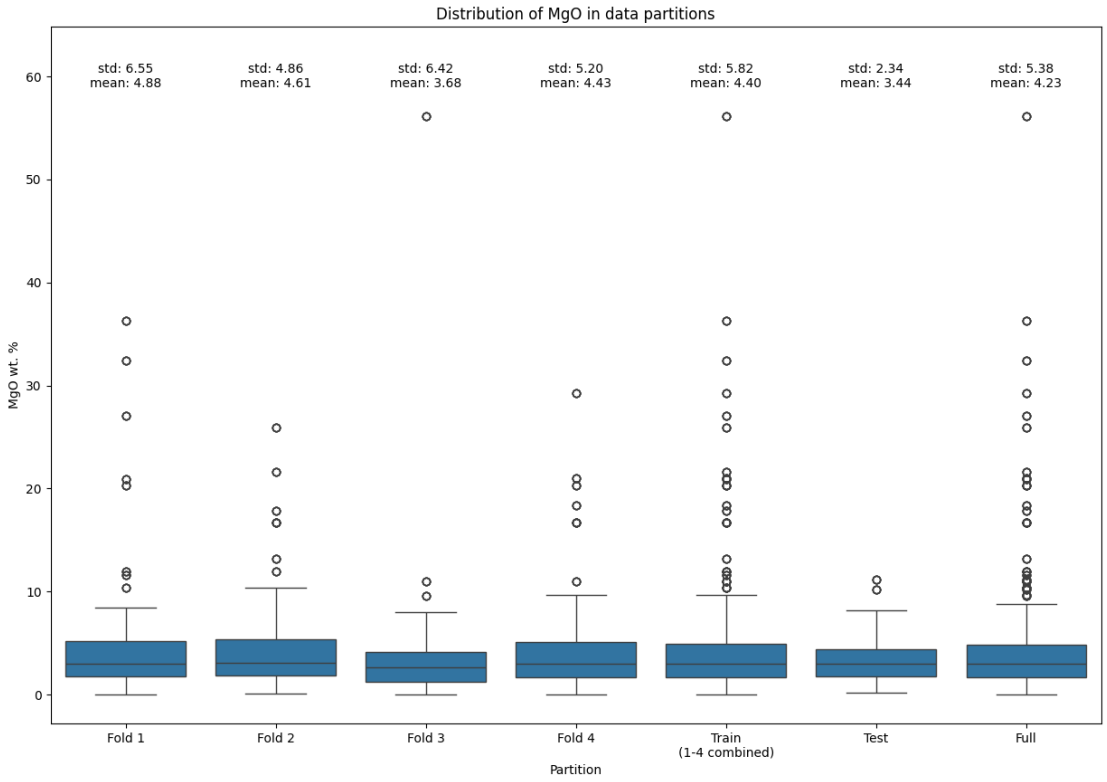

Fig. A.21. Distribution of MgO concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

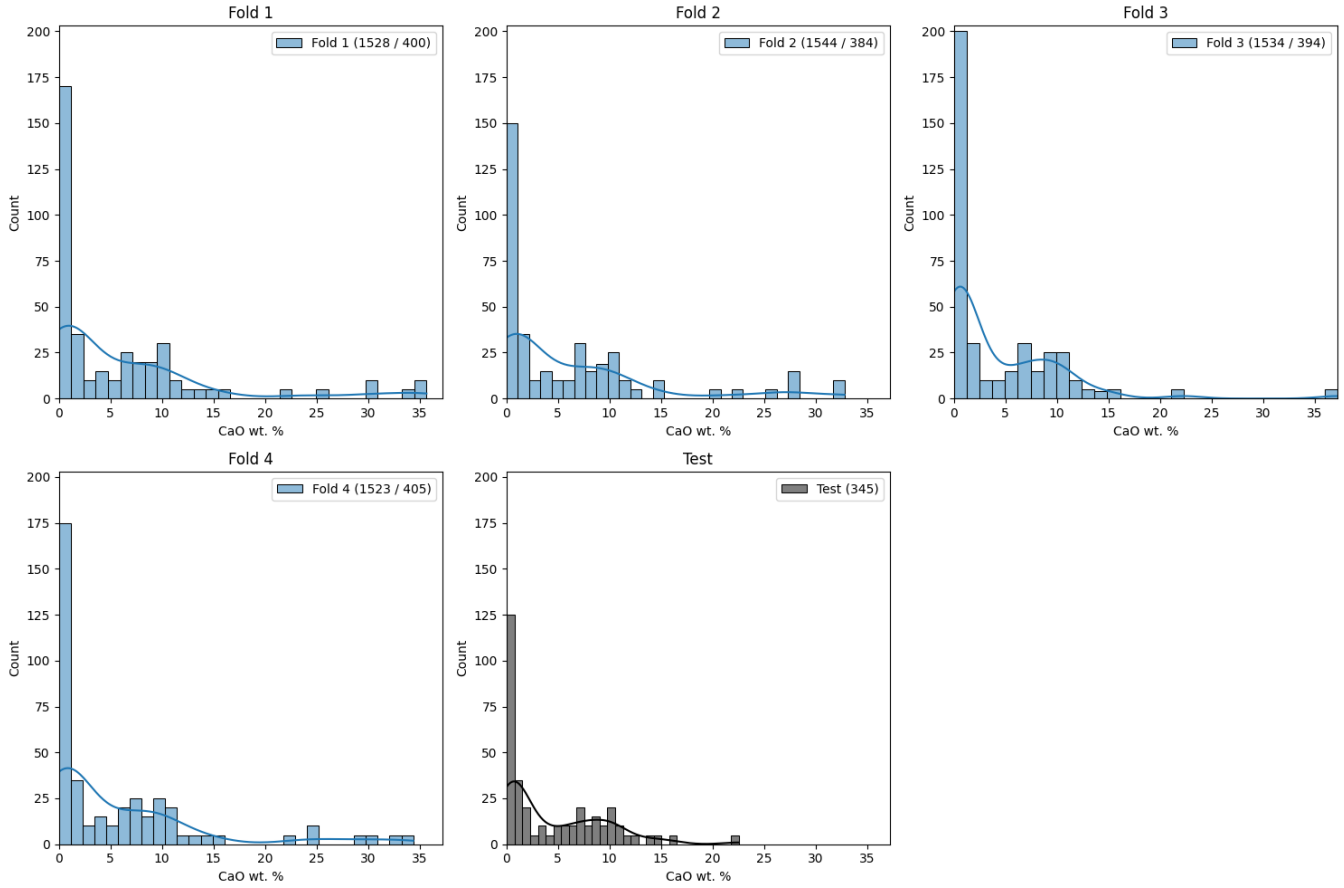

Fig. A.22. Histogram and KDE of CaO distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents CaO concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

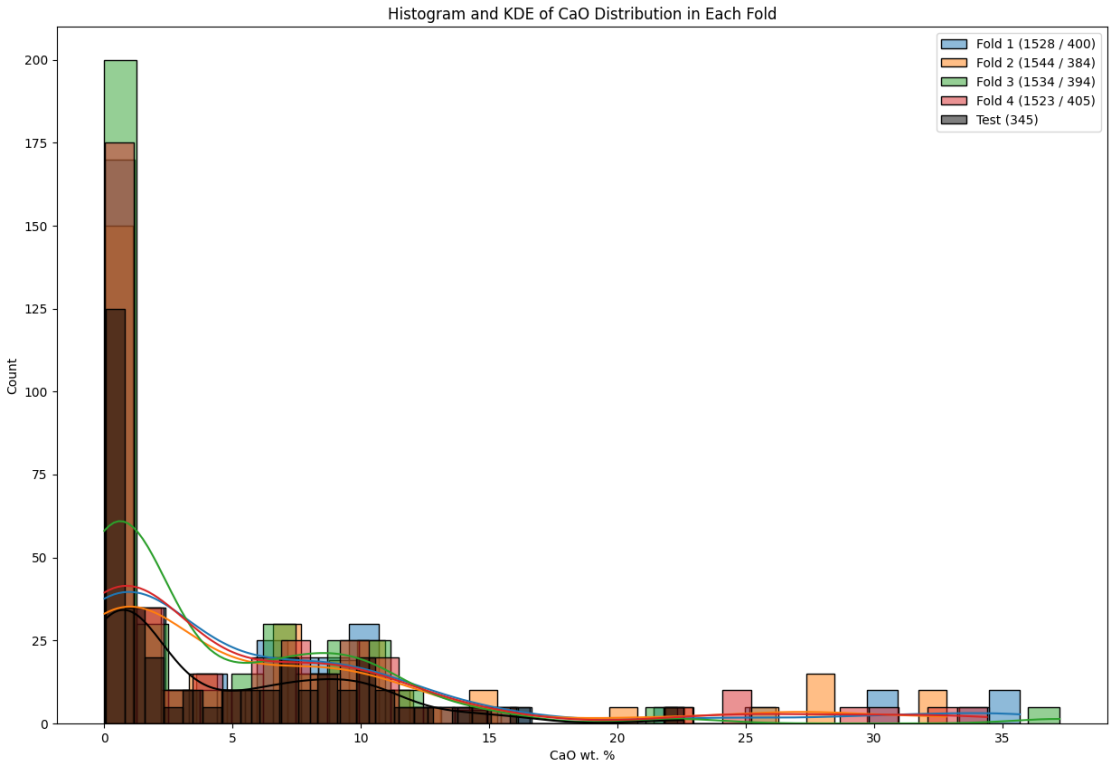

Fig. A.23. Combined Histogram and KDE of CaO distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents CaO concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

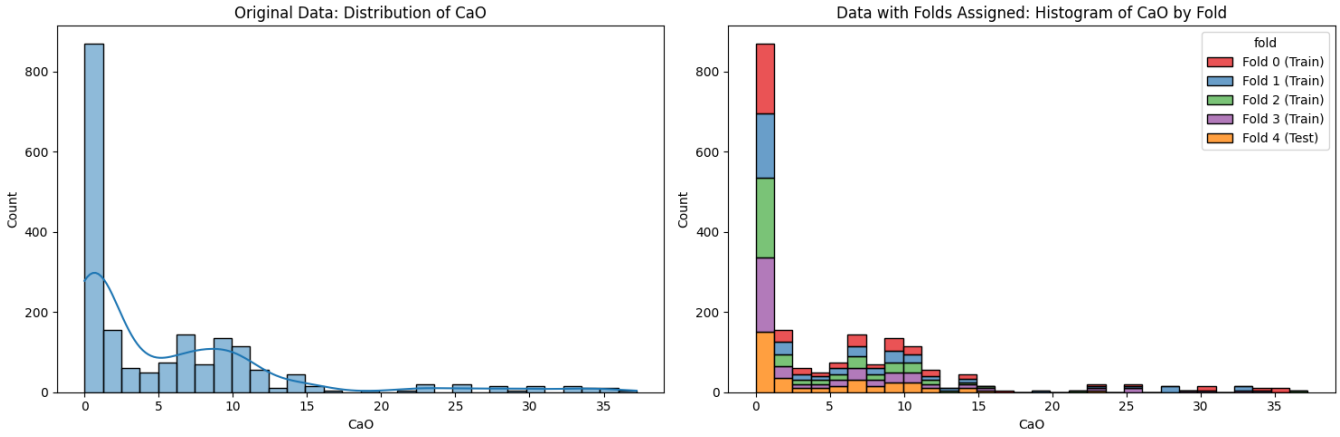

Fig. A.24. Distribution of CaO concentrations before and after fold assignment. The left plot shows the original distribution of CaO, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

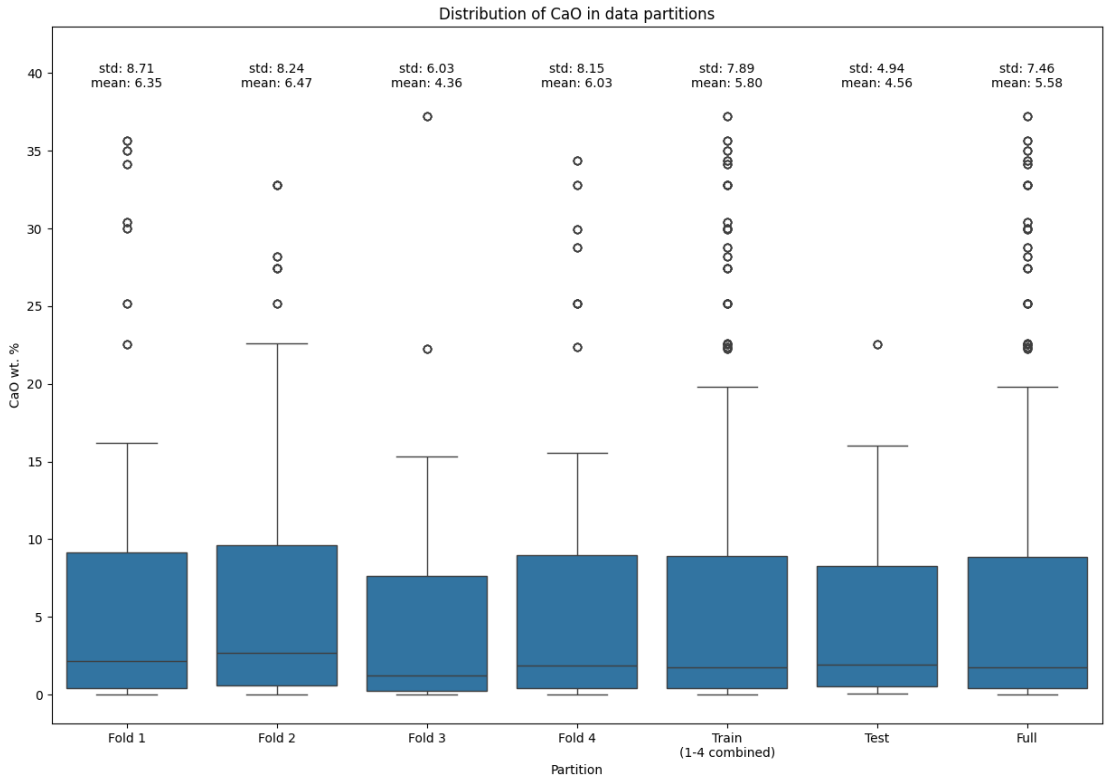

Fig. A.25. Distribution of CaO concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

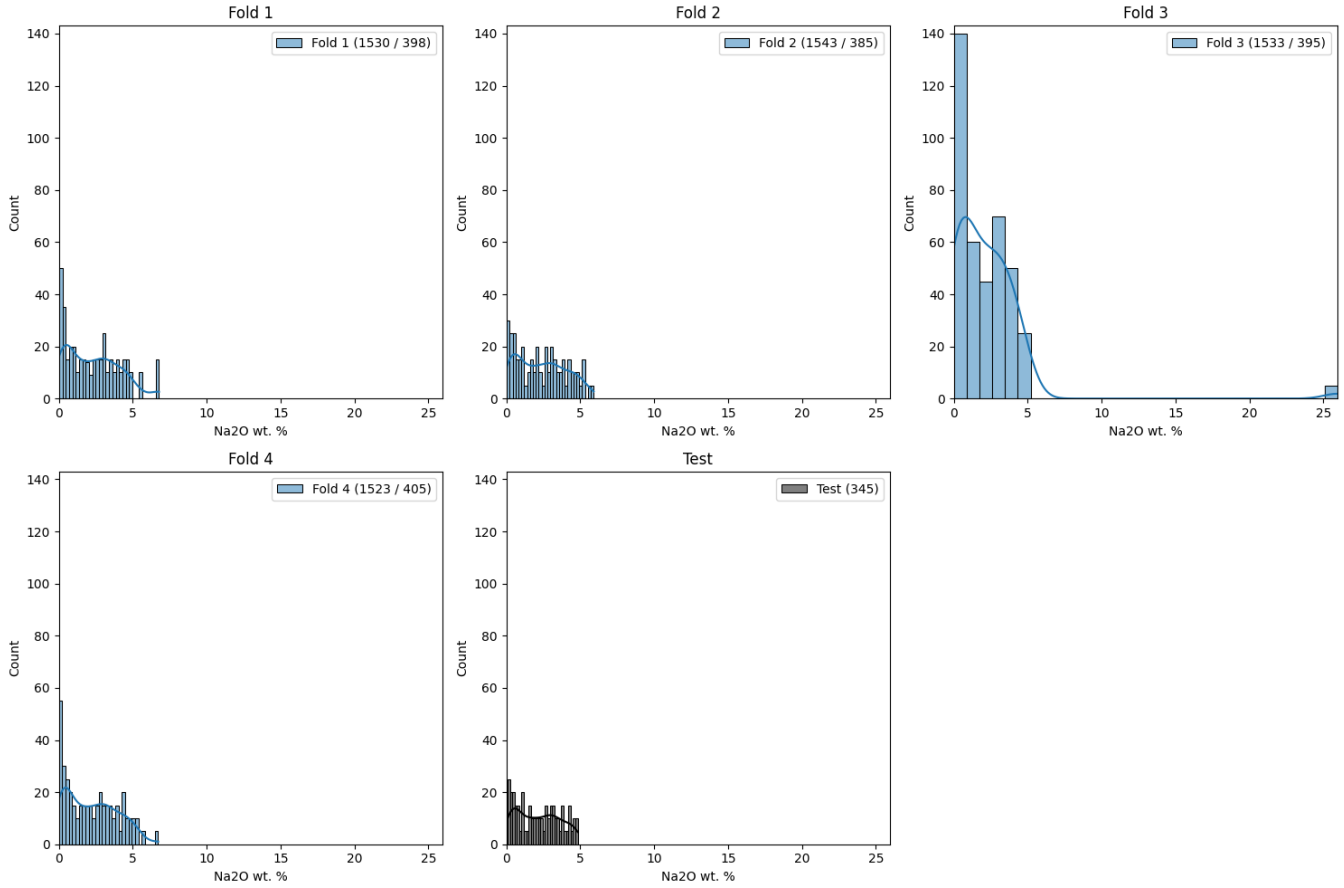

Fig. A.26. Histogram and KDE of Na2O distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents Na2O concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

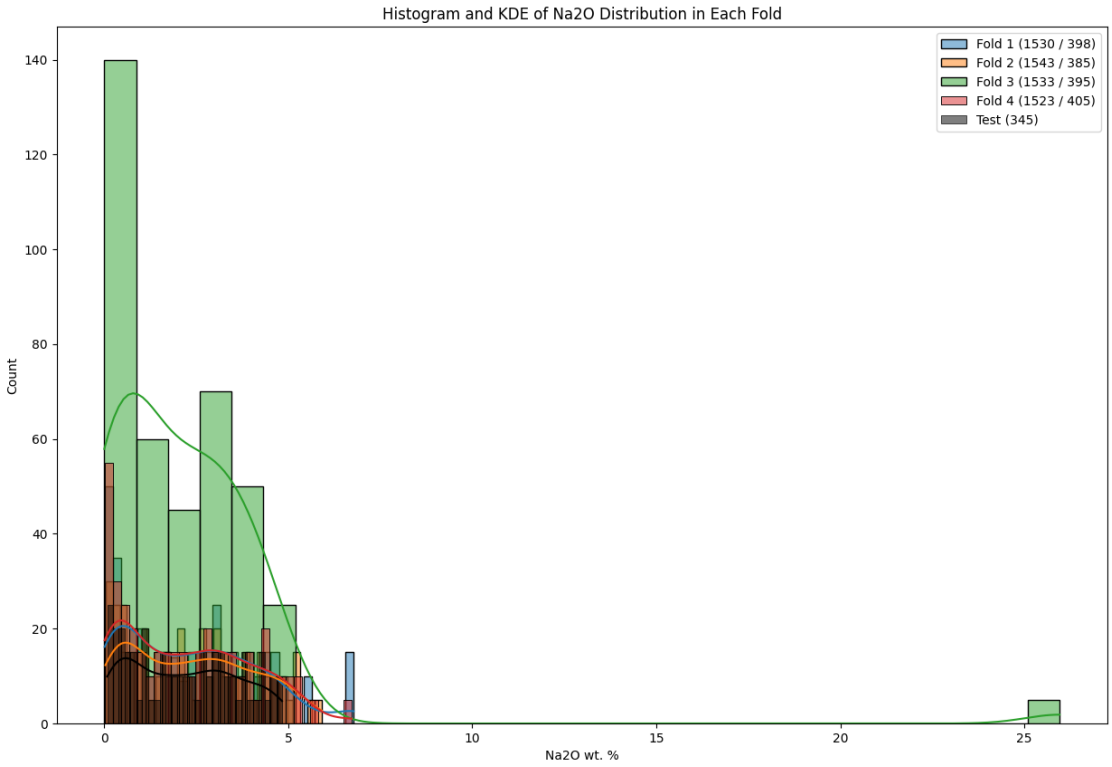

Fig. A.27. Combined Histogram and KDE of Na2O distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents Na2O concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

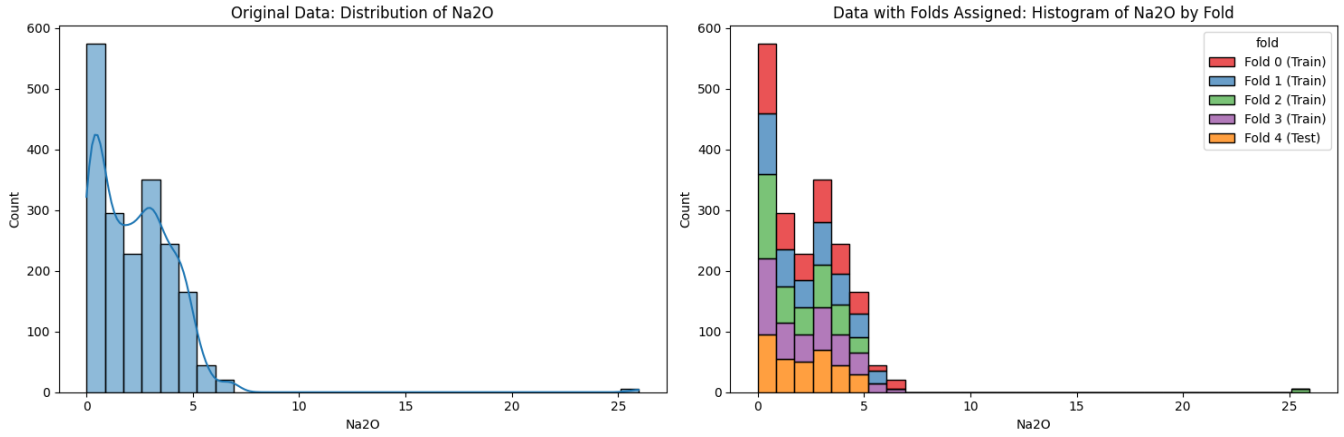

Fig. A.28. Distribution of Na2O concentrations before and after fold assignment. The left plot shows the original distribution of Na2O, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

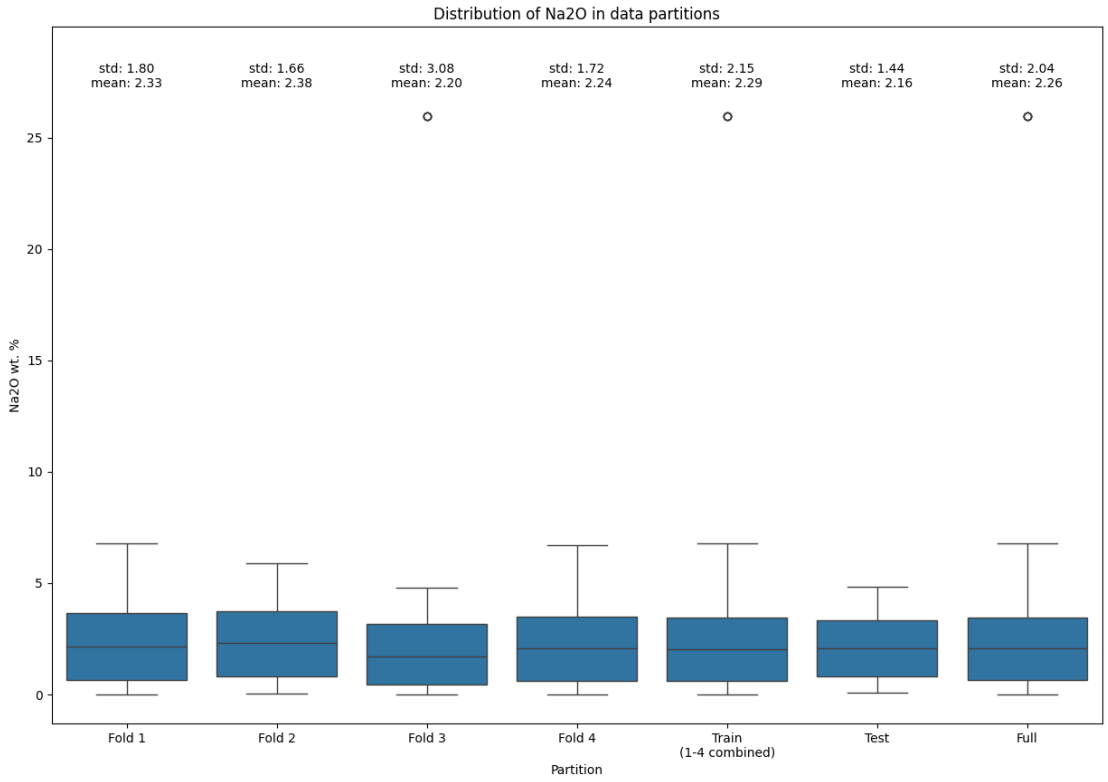

Fig. A.29. Distribution of Na2O concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

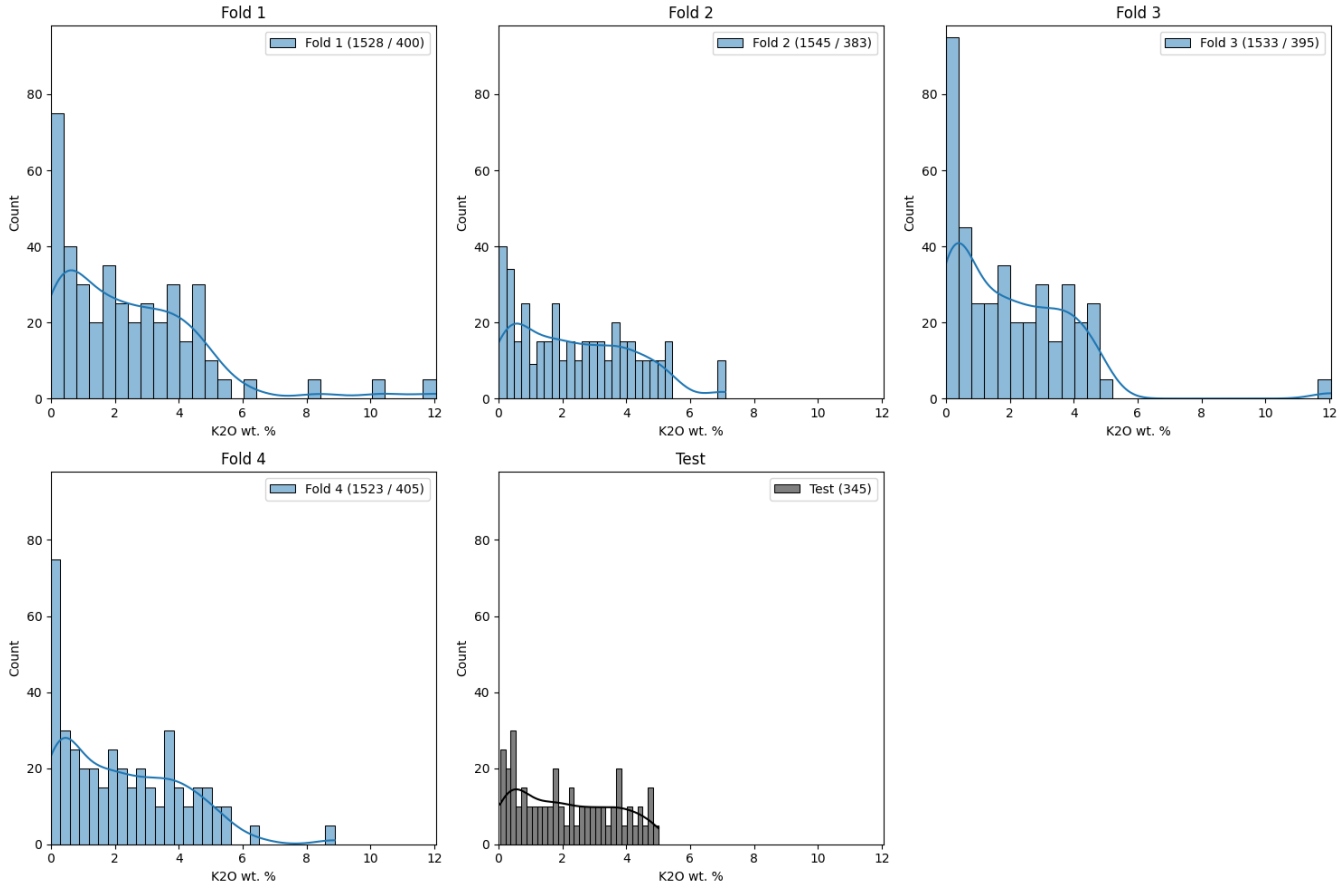

Fig. A.30. Histogram and KDE of K2O distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents K2O concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

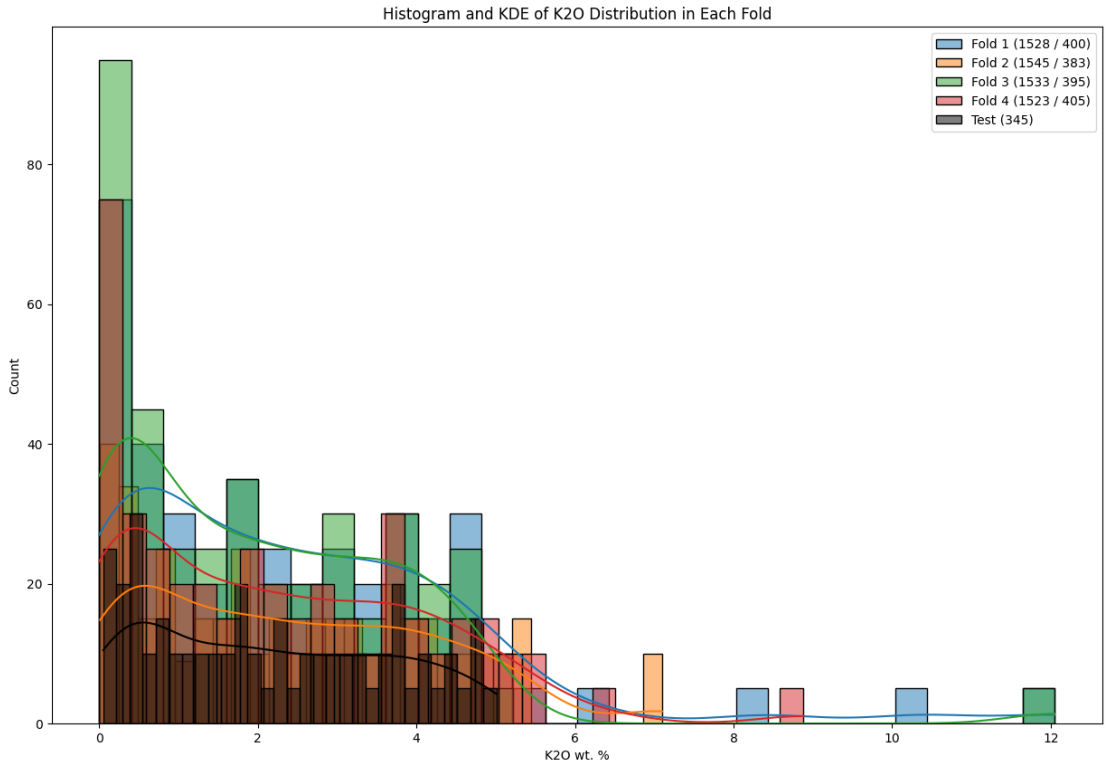

Fig. A.31. Combined Histogram and KDE of K2O distribution in each fold. The y-axis represents the count of samples per bin, and the x-axis represents K2O concentration. The notation in the legend indicates the amount of instances in the training/validation sets.

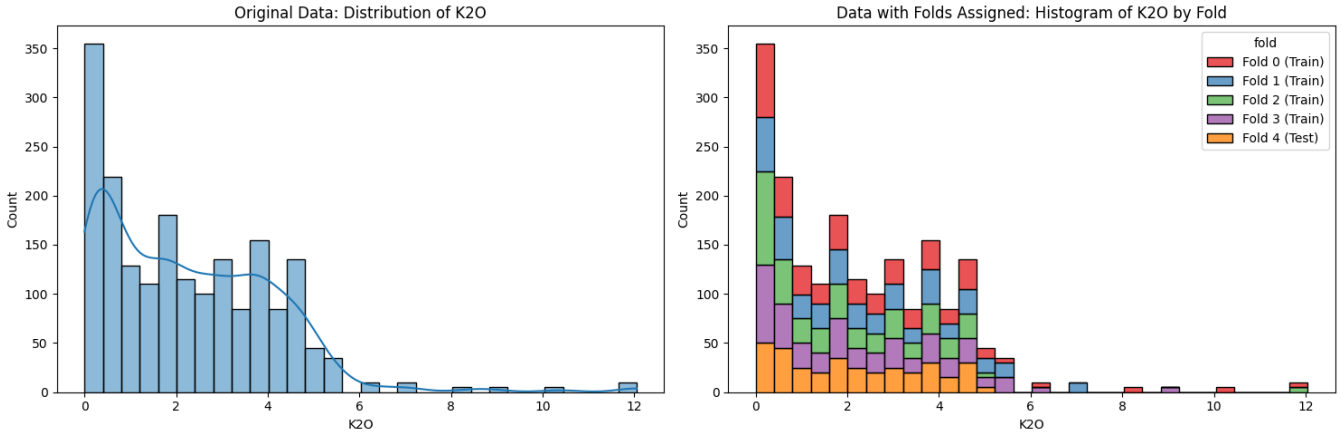

Fig. A.32. Distribution of K2O concentrations before and after fold assignment. The left plot shows the original distribution of K2O, while the right plot shows the distribution with folds assigned, color-coded to indicate the different folds.

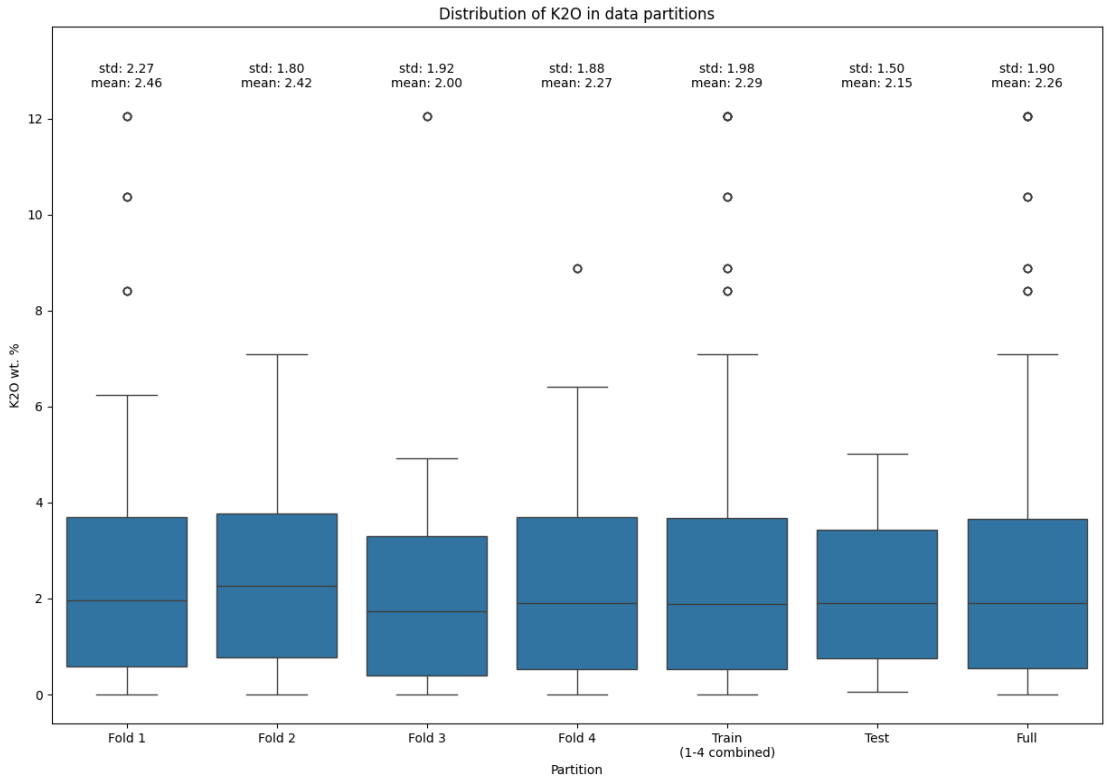

Fig. A.33. Distribution of K2O concentrations across cross-validation folds, training set, test set, and the entire dataset. The mean and standard deviation statistics for each partition are indicated figure.

A.3 Initial Experiment: Model Hyperparameters

Table A.1. Explicitly set hyperparameters for the PLS, SVR, ridge, LASSO, ENet, RF, and ETR models. When not explicitly set, the default hyperparameters provided by the libraries listed in Section 7.2 are used.

Model	Hyperparameter	Value
PLS	`n_components`	34
	`scale`	True
	`max_iter`	500
SVR	`kernel`	poly
	`C`	100
	`epsilon`	0.1
	`gamma`	scale
	`degree`	2
	`coef0`	1.0
Ridge Regression	`alphas`	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$
	`max_iter`	1000
	`tol`	10^{-4}
LASSO	`alphas`	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$
	`max_iter`	1000
	`tol`	10^{-4}
ENet	`alphas`	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$
	`l1_ratio`	$\{0.1, 0.5, 0.7, 0.9, 1.0\}$
	`max_iter`	1000
	`tol`	10^{-4}
RF	`n_estimators`	100
	`max_depth`	10
	`min_samples_split`	2
	`min_samples_leaf`	1
	`max_features`	sqrt
	`random_state`	42
ETR	`n_estimators`	100
	`max_depth`	10
	`min_samples_split`	2
	`min_samples_leaf`	1
	`random_state`	42

Table A.2. Explicitly set hyperparameters for the GBR and XGBoost models. When not explicitly set, the default hyperparameters provided by the libraries listed in Section 7.2 are used. The NGBoost model does not have any explicitly set hyperparameters.

Model	Hyperparameter	Value
GBR	n_estimators	100
	max_depth	3
	min_samples_split	2
	min_samples_leaf	1
	max_features	None
	loss	squared_error
	learning_rate	0.1
	subsample	1.0
	criterion	friedman_mse
	random_state	42
	verbose	0
	validation_fraction	0.1
	n_iter_no_change	None
	tol	10^{-4}
	ccp_alpha	0.0
NGBoost	-	-
XGBoost	max_depth	4
	min_child_weight	5
	gamma	0.1
	subsample	0.7
	colsample_bytree	0.5
	colsample_bylevel	0.5
	colsample_bynode	0.5
	lambda	1
	alpha	0.5
	learning_rate	0.05
	n_estimators	100
	objective	reg:squarederror
	eval_metric	rmse

Layer	Output Shape	Hyperparameter
Input	(*input_dim*,)	-
Dense	(1024,)	activation = ReLU
Dropout	(1024,)	rate = 0.3
Dense	(512,)	activation = ReLU
Dropout	(512,)	rate = 0.3
Dense	(256,)	activation = ReLU
Dense	(128,)	activation = ReLU
Output	(*output_dim*,)	-
Optimizer: Adam		
Learning Rate: 0.001		

Table A.3. Summary of the ANN architecture.

Table A.4. Summary of the CNN architecture.

Layer	**Output Shape**	**Hyperparameter**
Input	(*input_dim*,)	-
Reshape	(48, 128, 1)	-
Conv2D	(48, 128, 32)	filters = 32, kernel_size = (3, 3), activation = ReLU, padding = 'same'
BatchNormalization	(48, 128, 32)	-
MaxPooling2D	(24, 64, 32)	pool_size = (2, 2)
Conv2D	(24, 64, 32)	filters = 32, kernel_size = (3, 3), activation = ReLU, padding = 'same'
BatchNormalization	(24, 64, 32)	-
MaxPooling2D	(12, 32, 32)	pool_size = (2, 2)
Conv2D	(12, 32, 64)	filters = 64, kernel_size = (3, 3), activation = ReLU, padding = 'same'
BatchNormalization	(12, 32, 64)	-
MaxPooling2D	(6, 16, 64)	pool_size = (2, 2)
Conv2D	(6, 16, 128)	filters = 128, kernel_size = (3, 3), activation = ReLU, padding = 'same'
BatchNormalization	(6, 16, 128)	-
MaxPooling2D	(3, 8, 128)	pool_size = (2, 2)
Flatten	(3072,)	-
Dense	(256,)	activation = ReLU
Dropout	(256,)	rate = 0.5
Dense	(*output_dim*,)	-
Dense	(*output_dim*,)	kernel_regularizer = $L_2(0.01)$
Optimizer: Adam **Learning Rate:** 0.001		

A.4 Overview of Best Performing Model Configurations

Table A.5. Overview of model types for SiO_2 oxide.

SiO_2	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	pls	none	kernel_pca	min_max_scaler	4.552	4.551	4.084
	svr	none	none	min_max_scaler	4.592	4.588	3.533
	gbr	none	none	norm3_scaler	4.652	4.646	3.720
	lasso	power_transformer	pca	norm3_scaler	4.737	4.738	4.248
	xgboost	quantile_transformer	none	norm3_scaler	4.791	4.781	3.968
	elasticnet	quantile_transformer	none	norm3_scaler	4.841	4.844	3.947
	ngboost	power_transformer	none	norm3_scaler	4.860	4.851	4.148
	ridge	power_transformer	none	norm3_scaler	4.940	4.938	3.816
	extra_trees	power_transformer	none	norm3_scaler	5.141	5.118	3.821
	random_forest	none	none	norm3_scaler	5.204	5.192	3.788

Table A.6. Overview of model types for TiO_2 oxide.

TiO_2	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	svr	power_transformer	none	norm3_scaler	0.409	0.406	0.397
	gbr	power_transformer	none	norm3_scaler	0.410	0.409	0.332
	xgboost	none	none	robust_scaler	0.411	0.410	0.317
	random_forest	quantile_transformer	none	norm3_scaler	0.422	0.421	0.334
	elasticnet	none	none	robust_scaler	0.423	0.423	0.351
	extra_trees	power_transformer	none	standard_scaler	0.426	0.426	0.338
	ridge	none	none	min_max_scaler	0.428	0.427	0.359
	lasso	power_transformer	none	standard_scaler	0.431	0.430	0.372
	ngboost	none	none	robust_scaler	0.431	0.431	0.355
	pls	power_transformer	kernel_pca	robust_scaler	0.441	0.441	0.411

Table A.7. Overview of model types for Al_2O_3 oxide.

Al_2O_3	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	xgboost	power_transformer	none	norm3_scaler	2.075	2.067	1.740
	gbr	power_transformer	none	robust_scaler	2.092	2.089	1.987
	ngboost	power_transformer	none	robust_scaler	2.121	2.113	2.052
	svr	quantile_transformer	none	min_max_scaler	2.179	2.176	1.873
	ridge	quantile_transformer	none	norm3_scaler	2.218	2.211	1.843
	elasticnet	quantile_transformer	none	norm3_scaler	2.225	2.219	1.804
	pls	quantile_transformer	none	robust_scaler	2.247	2.244	2.111
	lasso	quantile_transformer	none	norm3_scaler	2.249	2.242	1.903
	extra_trees	power_transformer	none	min_max_scaler	2.288	2.261	2.092
	random_forest	power_transformer	none	max_abs_scaler	2.302	2.295	2.111

Table A.8. Overview of model types for FeO_T oxide.

FeO_T	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	svr	quantile_transformer	none	norm3_scaler	2.242	2.243	1.803
	pls	power_transformer	none	standard_scaler	2.701	2.669	2.063
	ridge	quantile_transformer	none	norm3_scaler	2.707	2.687	1.878
	gbr	power_transformer	none	max_abs_scaler	2.749	2.750	1.793
	xgboost	none	none	max_abs_scaler	2.749	2.743	1.622
	elasticnet	power_transformer	none	max_abs_scaler	2.862	2.831	1.773
	lasso	quantile_transformer	none	norm3_scaler	2.875	2.862	1.842
	extra_trees	none	none	max_abs_scaler	2.900	2.903	1.870
	ngboost	none	none	robust_scaler	2.980	2.953	1.773
	random_forest	quantile_transformer	none	norm3_scaler	3.079	3.044	2.018

Table A.9. Overview of model types for MgO oxide.

MgO	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	svr	power_transformer	none	robust_scaler	1.322	1.321	0.791
	pls	none	kernel_pca	norm3_scaler	1.327	1.321	0.993
	ridge	power_transformer	none	robust_scaler	1.448	1.443	1.321
	elasticnet	power_transformer	none	robust_scaler	1.466	1.462	1.630
	gbr	quantile_transformer	none	norm3_scaler	1.468	1.464	0.880
	extra_trees	power_transformer	none	norm3_scaler	1.533	1.522	0.765
	lasso	none	kernel_pca	min_max_scaler	1.604	1.596	1.092
	xgboost	none	none	norm3_scaler	1.618	1.610	1.129
	ngboost	quantile_transformer	none	norm3_scaler	1.624	1.603	0.980
	random_forest	quantile_transformer	none	norm3_scaler	1.640	1.630	0.973

Table A.10. Overview of model types for CaO oxide.

CaO	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	svr	quantile_transformer	none	min_max_scaler	1.193	1.192	1.600
	pls	quantile_transformer	none	max_abs_scaler	1.270	1.263	1.768
	gbr	quantile_transformer	none	norm3_scaler	1.281	1.280	1.793
	extra_trees	none	none	norm3_scaler	1.308	1.309	1.829
	xgboost	power_transformer	none	norm3_scaler	1.363	1.361	1.913
	elasticnet	quantile_transformer	none	norm3_scaler	1.384	1.377	1.634
	ridge	quantile_transformer	none	norm3_scaler	1.406	1.400	1.623
	random_forest	none	none	norm3_scaler	1.439	1.435	1.737
	ngboost	none	none	robust_scaler	1.488	1.481	1.920
	lasso	power_transformer	none	min_max_scaler	1.529	1.514	1.684

Table A.11. Overview of model types for Na_2O oxide.

Na_2O	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	svr	power_transformer	none	norm3_scaler	0.777	0.775	0.393
	pls	power_transformer	none	norm3_scaler	0.845	0.842	0.561
	gbr	quantile_transformer	none	norm3_scaler	0.904	0.895	0.374
	xgboost	quantile_transformer	none	max_abs_scaler	0.952	0.943	0.431
	extra_trees	quantile_transformer	none	norm3_scaler	0.965	0.953	0.479
	elasticnet	quantile_transformer	none	standard_scaler	0.994	0.990	0.504
	lasso	quantile_transformer	none	max_abs_scaler	0.995	0.991	0.507
	ngboost	quantile_transformer	none	norm3_scaler	1.000	0.993	0.443
	random_forest	quantile_transformer	none	norm3_scaler	1.002	0.995	0.470
	ridge	quantile_transformer	none	norm3_scaler	1.011	1.001	0.467

Table A.12. Overview of model types for K_2O oxide.

K_2O	Model Type	Transformer Type	PCA Type	Scaler Type	RMSECV	Std. dev. CV	RMSEP
	pls	none	none	norm3_scaler	0.587	0.586	0.724
	gbr	quantile_transformer	none	min_max_scaler	0.590	0.587	0.423
	svr	quantile_transformer	none	norm3_scaler	0.593	0.593	0.594
	xgboost	power_transformer	none	standard_scaler	0.600	0.599	0.455
	elasticnet	power_transformer	none	robust_scaler	0.602	0.602	0.650
	ngboost	quantile_transformer	none	max_abs_scaler	0.602	0.600	0.420
	lasso	power_transformer	none	norm3_scaler	0.607	0.606	0.624
	ridge	power_transformer	none	norm3_scaler	0.611	0.611	0.629
	random_forest	power_transformer	none	norm3_scaler	0.675	0.669	0.515
	extra_trees	power_transformer	none	robust_scaler	0.714	0.709	0.464

A.5 PyHAT Contribution Certificate

Received 13/06/2024

United States Department of the Interior

GEOLOGICAL SURVEY

U.S. GEOLOGICAL SURVEY
ASTROGEOLOGY SCIENCE CENTER
2255 N. GEMINI DR.
FLAGSTAFF, AZ 86001

To Whom it May Concern,

We are writing to certify that Christian Houmann, Ivik Hostrup, and Patrick Østergaard have made several meaningful contributions to the Python Hyperspectral Analysis Tool (PyHAT).

Their team met with us virtually, along with their advisor Dr. Jens Frydenvang, to discuss their results in duplicating state-of-the-art machine learning models that we have personally developed. Even without all of the workflows and scripts available to them, their team was able to achieve remarkable success. During this meeting, the students discussed many improvements on published methods that they personally developed or explored during their project. These contributions stand to make a sizeable impact on the field of chemometrics as applied to the ChemCam and SuperCam emission spectroscopy instruments on two active missions as part of NASA's Mars Exploration Program.

First, the team suggested the use of a Stacking Regressor to combine the results of several models into a final "meta model". This is an alternative to the relatively crude and labor-intensive "submodel blending" approach that was used for the current ChemCam calibration. The team also suggested the use of a power transform as a preprocessing step. Both suggestions have been implemented and are showing promising results.

They also directly contributed to the PyHAT repository in the form of commits and merge requests found at the link below. These important contributions include parallelizing and improving the ChemCam data reading functionality, developing an automated approach to a previously tedious manual outlier identification method, and adding a function that helps to interpret loadings from Independent Component Analysis.

See Merge Requests 8, 9, 10, and 11:
https://code.usgs.gov/astrogeology/pyhat/-/merge_requests

We thank Christian, Ivik, and Patrick for their valuable contributions. Please do not hesitate to contact us with any questions.

Sincerely,

Dr. Ryan B. Anderson
Research Physical Scientist
rbanderson@usgs.gov

Dr. Travis S.J. Gabriel
Research Physical Scientist
tgabriel@usgs.gov