

UC Davis

UC Davis Electronic Theses and Dissertations

Title

Removing Bottlenecks in Research Workflows: Improving SERS Sampling and Computational Zeolite Experiments Through the Application of Machine Learning and the Construction of Custom Data Pipelines

Permalink

<https://escholarship.org/uc/item/7531788r>

Author

Antonio, Dexter Dante

Publication Date

2021

Peer reviewed|Thesis/dissertation

Removing Bottlenecks in Research Workflows:
Improving SERS Sampling and Computational Zeolite Experiments Through the Application of
Machine Learning and the Construction of Custom Data Pipelines

By

DEXTER ANTONIO
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Chemical Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Ambarish Kulkarni, Chair

Adam Moule

Roland Faller

Committee in Charge

2021

Table of Contents

Table of Contents	ii
List of Figures	v
List of Tables	x
Acknowledgements.....	xi
Abstract.....	xii
Chapter 1	1
Introduction.....	1
1.1 Overview.....	1
1.2 Data Cleaning.....	3
1.3 Data Engineering	5
Chapter 2.....	7
Machine Learning Assisted Sampling of SERS Substrates Improves Data Collection Efficiency	7
2.1 Acknowledgements.....	7
2.2 Abstract.....	8
2.3 Introduction.....	9
2.4 Methods.....	12
2.4.1 Sample Preparation	12

2.4.2 SERS Spectra Acquisition	13
2.4.3 Spectra Labeling	14
2.4.4 Performance Measures.....	14
2.5 Results.....	16
2.5.1 Sample Collection.....	16
2.5.2 Data Organization and Parsing for ML Input	19
2.5.3 Model Selection and Performance Analysis	21
2.5.4 Testing Model Performance.....	24
2.5.5 Variable importance.....	25
2.5.6 Out of Sample Testing	26
2.6 Discussion	29
2.7 Conclusion	32
2.8 Data Accessibility Statement	32
2.9 Funding	33
Chapter 3.....	34
The Multiscale Atomic Zeolite Simulation Environment (MAZE): A Python Package for Improved Zeolite Structural Manipulations.....	34
3.1 Acknowledgements.....	34
3.2 Abstract:.....	35
3.3 Introduction.....	35

3.4 Current Limitations with Tracking Atoms Within the ASE Interface	37
3.5 MAZE Design Philosophy	40
3.6 Assessing the Usability of the MAZE Package	41
3.6.1 Design principles	41
3.6.2 API Usability Assessment.....	42
3.6.3 MAZE Software Structure	43
3.6.4 The Index Mapper Class	45
3.7 Results and Discussion	47
3.7.1 Results and Discussion Introduction.....	47
3.7.2 Building a Zeolite from a CIF File	48
3.7.3 Structural Manipulations.....	50
3.7.4 Cluster Extraction, Atom Capping and Integration	53
3.7.5 Creation and Optimization of a Database of Zeolite Structures	55
3.8 Conclusions.....	57
Chapter 4.....	59
Conclusions.....	59
4.1 Project Summaries	59
4.2 Future work.....	59
5.1 Bibliography	62

List of Figures

Figure 1: Example of data generation and data cleaning in a computational experiment (A) and a laboratory experiment (B). Figure 1B, presents simulation input files (left), a section of the raw output files of a GCMC simulation (middle) and the cleaned data matrix (right). Figure 1B shows the raw spectra generated by the instrument (middle) and the cleaned baseline corrected and normalized spectra (right). 4

Figure 2: Comparison of a traditional experimental workflow and an automated experimental workflow. The top figure shows the involvement of a scientist in the data cleaning and data fitting portions, where in the second figure those sections are automated via an automatic data pipeline..... 6

Figure 3: Hierarchical data structure for label-free SERS experiments. A single experiment consists of different samples, each of which is analyzed with a separate substrate. For each sample, different x, y positions (spots) are analyzed by collecting a series of SERS spectra one-after-another. The number of spectra collected per spot and spot collected per substrate can vary depending on signal quality. 12

Figure 4: Image of Python-based Labeler app for manually assigning labels to spectra. The interface displays a spectrum to SERS expert, who can assign a label to the spectra with the “good”, “bad” or “maybe” buttons. The spot number, spectra number, sample name, and current label is displayed in the bottom of the window. The save labels buttons saves the labels to a NetCDF file for later use in model training and testing..... 14

Figure 5: Experimental workflow of the ML-SERS platform. a) Rhodamine 6G was used as a SERS reported molecule on plasmonic Moxtek (1) or Plasmore (2) substrates. Varying solutions

of R6G were pipetted (~ 20 μ L total) onto the surface and 20×20 pixels surface areas were scanned. b) SEM micrographs illustrate the structure of the Moxtek and Plasmore substrates. c) Representative SERS spectrum of R6G; the highlighted peaks at 620, 780, 1198, 1367 and 1513 cm^{-1} are characteristic spectral features of R6G. 17

Figure 6: PCA analysis to guide SERS spectra selection. a) PC score plot demonstrating each measured sample group (400 spectra per group, total 2000 spectra in the data set), each marker represents a single SERS spectrum. b) PC1 loading spectrum displaying the positive features at 620, 780, 1198, 1367, and 1513 cm^{-1} that can be associated to the SERS reporter molecule R6G used in the study. c) Examples of “good”, “maybe” and “bad” spectra pinpointed with corresponding star markers (red, yellow, blue, respectively) at the PC score plot. 19

Figure 7: Open-source data pipeline developed for this study. The first stage of the pipeline converts and processes raw spectra files into a binary NetCDF file format. The second data labeling stage employs a custom Python “Labeler” app, allowing an expert Raman user to quickly assign labels (e.g., “good”, “bad”, or “maybe”) to the spectra serialized in the netCDF files. After labels have been assigned, the last stage of the pipeline is model training, where the binary files are loaded into NumPy arrays to train and test various ML models. 20

Figure 8: a) Calibration plots for four tested machine learning algorithms LR-SGD, SVG-SGD, DT, LDA, RF, and XGB on validation set. The x axis represents the mean predicted probability of the events occurring in the bin. The y axis represents the fraction of positive (i.e., “good”) spectra in the in the bin. b) The total number of probabilities in each decile, for each classifier. 23

Figure 9: a) ROC curves for six tested models, logistic regression (LR-SGD), support vector machines (SVG-SGD), decision trees (DT), linear discriminate analysis (LDA), random forest (RF) and XGBoost (XGB). Random guessing is represented by the $y=x$ line whereas higher

performing models lie closer to the left corners of the plot. Inset: Confusion matrix for XGBoost algorithm trained on train and validation set and tested on the test set. The trace of the matrix indicates correct predictions, while the offset values indicate incorrect predictions. b)

Comparison of hyperparameter-tuned model performance on validation data set. The highest value for each category is bolded and the lowest value for each category is italicized..... 24

Figure 10: Representative spectra classifications for XGBoost on in-sample test dataset. Text overlaid on each spectra indicate the expert user label trained model's performance in accurately predicting the label..... 25

Figure 11: Relative importance of different wavenumber regions for the XGBoost model trained on the in-sample train and validation data. Each bin contains the sum of the importance scores of the wavenumbers in the range [start wavenumber, end wavenumber). An R6G spectrum is overlaid for reference, where it is apparent that relative importance correlate with spectral features of R6G. 26

Figure 12: a) Visualization of the sampled region from the five substrates used in experiment along with the corresponding labels. Red squares represent a negative spectrum, green squares represent a positive spectrum, and purple squares represent spectra with ambiguous labels, which were excluded from fitting and prediction. b) Comparison of actual and predicted labels for five test samples. The comparison is represented by the color. False positive (red), false negative (pink), true positive (light blue), true negative (dark blue), and ambiguous, maybe labeled, spectra (purple) pixels are shown. 28

Figure 13: Relationship between indices in *Atoms* objects derived from glyoxal. The indices of the elements carbon, oxygen, hydrogen, and sulfur are colored black, red, purple and yellow respectively. The columns relate the indices of different *Atoms* objects to each other. The indices

shifting issue becomes more pronounced when multiple structural operations are performed in series. For example, if the O and H atoms are added back to the #3 to recover #1, their indices (4 and 5) would differ from the initial structure..... 39

Figure 14: Simplified unified modeling language (UML) class diagram for the *Zeolite* object.

Inheritance relationships are denoted by an open arrow. The *Zeolite* class inherits from the *PerfectZeolite* class, which inherits from the *Atoms* class 44

Figure 15: Dictionary representation of the main index mapper for a collection of related *Zeolites*. The keys of the outer dictionary represent the unique IDs, where the inner dictionaries map the relationship between indices for the same shared atom across different *Atoms*-like objects. 45

Figure 16: Comparison between MAZE and ASE code for generating a BEA zeolite structure with the Silicon T2 sites replaced by aluminum atoms. The MAZE code uses the built-in make function to read the unmodified CIF file and store the mapping in the `site_to_atom_indices` dictionary. The longer ASE code requires a modified CIF file as input, and the element mapping to be manually defined. Both codes generate and visualize the same BEA T2 Si→Al structure. 49

Figure 17: Workflow for cluster integration. A blue arrow shows the workflow direction, starting with a BEA zeolite constructed using the make method. The functions or operations required to transfer from one structure to another are shown between the structures on the blue line. 54

Figure 18: Code required to generate structure F from structure A..... 54

Figure 19: Data pipeline for the construction of optimized zeolite derivatives. The MAZE package allows for key optimization steps to be automated through a single python script. 56

Figure 20: Workflow diagram for the optimization of a collection of trajectory files. The workflow is divided into optimization (blue), save (gray) and analyze (green) stages. 57

Figure 21: A general diagram for an autonomous experiment. The sample criteria are selected by an agent and are collected by the experimental apparatus and controller. An automatic data pipeline processes the raw data output by the experiment and feeds the data back to the agent.. 60

List of Tables

Table 1: Metrics useful in assessing the performance of a binary classifier	15
Table 2: Model performance parameters for the XGBoost algorithm trained on the training set and validation set and tested on the test set.	25
Table 3: Out-of-sample performance for XGBoost model on five substrates	27
Table 4: Usability criteria for API design.....	42
Table 5: Criteria and sub-criteria used to assess the MAZE API's usability.....	43
Table 6: Representation of the main_index of an <i>IndexMapper</i> object. The column headers denote either the unique_id or the name of the zeolite. The parent column corresponds to the indices of the <i>PerfectZeolite</i> from which the other zeolite structures were derived. Each row of the table shows the relationship between the different indices across different zeolites.....	46
Table 7: Assessment of usability for both the ASE and MAZE package for loading a labeled CIF file	49
Table 8: ASE <i>Atoms</i> object and MAZE <i>Zeolite</i> object's structure manipulation methods	51
Table 9: Usability of structural manipulation methods for the <i>Atoms</i> objects and <i>Zeolite</i> Objects	52

Acknowledgements

I would like to acknowledge everyone who helped me complete my Master's thesis and supported me throughout graduate school. I would like to directly thank Prof. Ambarish Kulkarni whose guidance, support and flexibility allowed me to direct my research in a way that aligned with my career goals. I also want to acknowledge Prof. Randy Carney and Dr. Tatu Rojalin for working with Prof. Kulkarni and me on the SERS project, leading to the creation of a truly novel SERS sampling solution.

I would also like to acknowledge all members of the Kulkarni group for their feedback during group meeting and taking the time to try out the MAZE package.

I would also like to thank the University of California, Davis for providing tuition abatement, a small salary and healthcare in return for teaching assistance. Without this financial support, completing this thesis would not have been possible.

Finally, I would like to thank my family and Maggie Riley, who provided support and guidance in navigating graduate school.

Abstract

Advances in machine learning and growing computational power are enabling large scale data analysis experiments. To facilitate these experiments, data must be cleaned from its raw form into one suitable for analysis. Automatic data pipelines are able to facilitate the creation of large-scale experiments by automating the transformation and cleaning of data into a form amenable to analysis. In this thesis an automatic data pipeline is developed for two separate projects:

“Machine Learning Assisted Sampling of SERS Substrates Improves Data Collection Efficiency” and “The Multiscale Atomic Zeolite Simulation Environment (MAZE): A Python Package for Improved Zeolite Structural Manipulations”. These two projects are related in that both automate key sections of the experimental data analysis, building the groundwork for future autonomous experiments.

Machine Learning Assisted Sampling of SERS Substrates Improves Data Collection

Efficiency: Surface-enhanced Raman scattering (SERS) is a powerful technique for sensitive label-free analysis of chemical and biological samples. While much recent work has established sophisticated automation routines via machine learning (ML) and related artificial intelligence (AI) methods, these efforts have largely focused on downstream processing (e.g., classification tasks) of previously collected data. While fully automated analysis pipelines are desirable, current progress is limited by cumbersome and manually-intensive sample preparation and data collection steps. Specifically, a typical lab-scale SERS experiment requires the user to evaluate the quality and reliability of the measurement (i.e., the spectra) as the data is being collected. This need for expert user-intuition is a major bottleneck that limits applicability of SERS-based

diagnostics for point-of-care clinical applications, where trained spectroscopists are likely unavailable. While application-agnostic numerical approaches (e.g., signal-to-noise thresholding) are useful, there is an urgent need to develop algorithms that leverage expert user intuition and domain knowledge to simplify and accelerate data collection steps. To address this challenge, in this work, we introduce an ML-assisted method at the acquisition stage. We tested six common algorithms to measure best performance in the context of spectral quality judgement. For adoption into future automation platforms, we developed an open-source python package tailored for rapid expert user annotation to train ML algorithms. We expect that this new approach to use ML to assist in data acquisition can serve as a useful building block for point-of-care SERS diagnostic platforms.

The Multiscale Atomic Zeolite Simulation Environment (MAZE): A Python Package for

Improved Zeolite Structural Manipulations: Zeolites are nanoporous materials with widespread industrial applications as catalysts and gas separators. Due to the enormity of the zeolite chemical space and structural complexity, computational experiments are needed to identify high-performing zeolites and interpret zeolite characterization data. These computational experiments are enabled by software packages, most notably the Atomic Simulation Environment (ASE) which provides an easy-to-use Python interface to drive low level simulation code. ASE has some limitations that make certain zeolite simulation workflows challenging and labor intensive. These limitations motivated the creation of the Multiscale Atomistic Zeolite Simulation Environment (MAZE) package which builds on-top of ASE to facilitate common zeolite structural manipulations that are challenging with the base ASE package. The improved interface of MAZE, compared to ASE, is demonstrated by applying application programming interface (API) design heuristics and showcasing the ability of MAZE to facilitate common

zeolite workflows. It is demonstrated that the MAZE package has an improved API for zeolite workflows and that this can facilitate the creation of large databases of zeolite structural derivatives.

Chapter 1

Introduction

1.1 Overview

The two separate projects described in this thesis are the development of a machine learning assisted sampling technique for surface enhanced Raman spectroscopy (SERS) substrates and the creation of the Multiscale Atomic Zeolite Simulation Environment (MAZE) Python package. These seemingly unrelated projects share the common thread of removing bottlenecks in a scientific workflow, through the creation of an automated data pipeline. A data pipeline, in a research context, is a process that transforms the raw data generated from an experiment into a form amenable to analysis. In the SERS project the preprocessing of spectra and categorization of spectra as positive or negative is automated. The MAZE package simplifies common structural manipulations in simulated zeolite structures removing the key roadblock in the generation of zeolite structural derivative databases. By removing repetitive steps in scientific workflows, both of these projects speed up data generation allowing for more large-scale experiments to be conducted.

This thesis is divided into four chapters. The first chapter, the introduction, provides an overview of the structure of the thesis and the unifying themes of the two separate projects are discussed.

The remaining introduction section provides a broad overview of machine learning and data engineering and how it relates to the two discussed projects. Specifically, it focuses on the importance of cleaning data through the use of data pipelines.

The second chapter of this thesis covers the project “machine learning assisted sampling of SERS”. This chapter focuses on the use of a supervised machine learning algorithm to automatically remove bad spectra from a dataset, which sets the groundwork for future autonomous experiments. These sections and the abstract of this thesis are based off of a manuscript submitted to Applied Spectroscopy in which I am co-first author. Permission was granted from the other primary author, Dr. Tatu Rojalin, to include the contents of this manuscript in my thesis. This inclusion was also approved by my senior academic adviser (SAA), Wallace Woods and the current Chemical Engineering department chair, Prof. Tonya Kuhl. This manuscript was written with the substantial help of the other two secondary authors Dr. Ambarish Kulkarni and Dr. Randy Carney. The work was supported by the UC Davis Center for Data Science and Artificial Intelligence Research (CeDAR) Innovative Data Science Seed Funding Program.

The next chapter (Chapter 3) of this thesis focuses on the development of the MAZE Python software package. This project extends the Atomic Simulation Environment (ASE)[1] to more naturally represent zeolites and facilitate the calculations required to determine their properties. The most notable aspect of this project is that it allows for the creation of large-scale databases of zeolite structures. This package was based off of the extensive scripts in the Kulkarni group and since its release has been downloaded over 100 times through PyIP [2], and cloned an unknown number of times directly from the GitHub repo.

The brief final chapter (Chapter 4) concludes the thesis by reiterating the main points and briefly discussing how data pipelines make autonomous experiments possible.

1.2 Data Cleaning

The availability of data, exponentially increasing computing power, and developments in machine learning are revolutionizing materials science and making the generation, interpretation and analysis of experiments easier than ever before [3][4]. These advances are actively being combined by training machine learning algorithms on the data generated from data-heavy simulation and laboratory experiments, producing systems which can outperform traditional, first-principal, parameterized models [3], [4]. In order to develop high performing models, the data the model is trained on must be large enough, clean and representative of the sample space [5]. Surveys of practicing data scientists concluded that the largest barrier to successful data science projects is low quality or low quantity of training data [5]. In a research setting barriers to producing an abundance of high-quality data, must be overcome to utilize the advances in machine learning.

Data quality can be improved by performing higher quality experiments and by refining the pre-processing that the data undergoes prior to the analysis [6]. In the field of data science this pre-processing is referred to as “data cleaning”, and it is often stated that around 80% of a data science practitioner’s time is devoted to this task [7]. In general, data cleaning takes an experimental data set and turns it into a form that is amenable to analysis either by a machine learning algorithm or a human expert [7][8]. Although its name conjures images of an unpleasant chore, proper data cleaning is an integral component of producing high performing

models and the process incorporates the extensive domain knowledge of the researcher into the resulting dataset [8]. Through data cleaning the important components from an experimental result are selected, extracted and reformed producing a clean dataset.

Two concrete examples of data cleaning in the scientific domain are the extraction of carbon dioxide adsorption data from GCMC computational experiment output files and the baseline correction and smoothing of surface enhanced Raman spectroscopy (SERS) spectra (Figure 1).

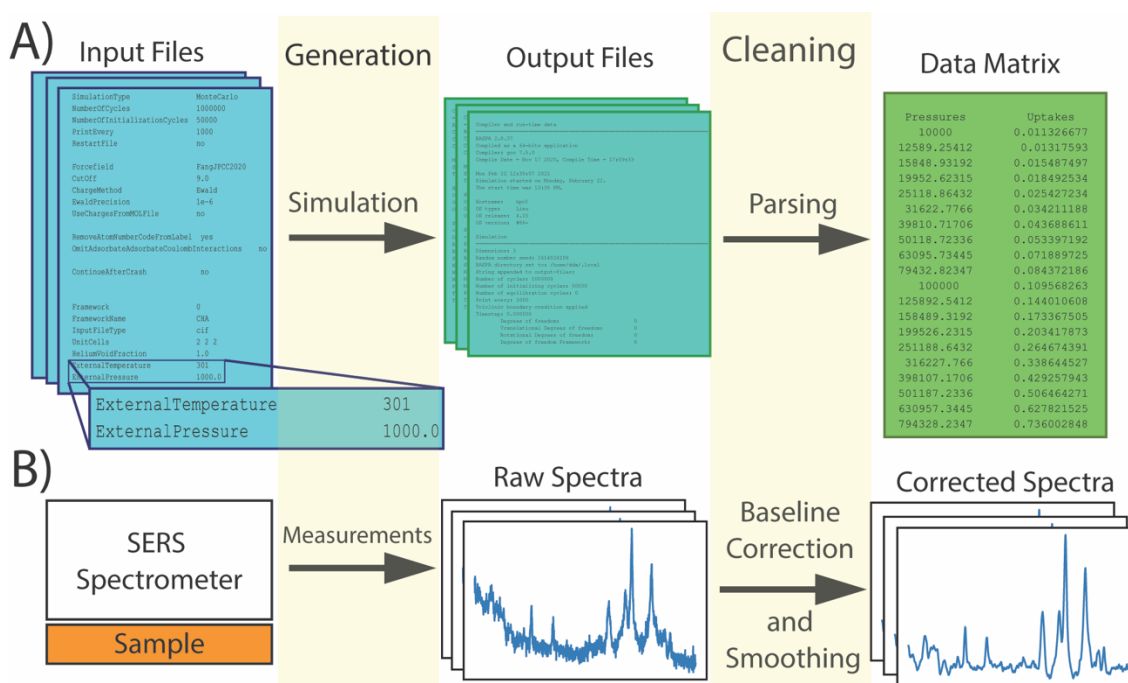


Figure 1: Example of data generation and data cleaning in a computational experiment (A) and a laboratory experiment (B). Figure 1A, presents simulation input files (left), a section of the raw output files of a GCMC simulation (middle) and the cleaned data matrix (right). Figure 1B shows the raw spectra generated by the instrument (middle) and the cleaned baseline corrected and normalized spectra (right).

In these two examples, the raw data is not amenable to analysis either because it contains noise (case B) or superfluous information in an unusable data structure (Case A). The noise in data can be removed through expert judgement or through the application of an algorithm. The structure of the data also has to be amenable to analysis, requiring data to be de-nested into a data matrix. In a research setting, these transformations are often performed in a sub-optimal, semi-manual

way, such as extracting the data manually from an output files or manually deleting negative spectra from a data file. When performed in such a way, the drudgery invoked by the cleaning term is accurate and large-scale experiments are hindered by the labor-intensive post-collection steps.

1.3 Data Engineering

Data engineering is the field focused on developing data pipelines which automatically clean raw data into a form suitable for downstream analysis [9]. This emerging field is typically focused on large scale systems, yet the principles and ideas of the field are widely applicable to a research setting. The key component of data engineering is the data pipelines [9]. These pipelines *automatically* clean and transform the data, meaning that aside from supplying a raw file to the pipeline, no user intervention is required. In a research setting, the data generated is often under the control of a researcher, thus the data engineering also encompasses the additional aspect of selective data generation. The development of automated data engineering pipelines allows for the rapid processing of raw, experimental (laboratory or simulation) data into a useful form. A diagram for a general scientific experiment is shown in Figure 2, with a comparison between the manual steps and the steps of a data pipeline.

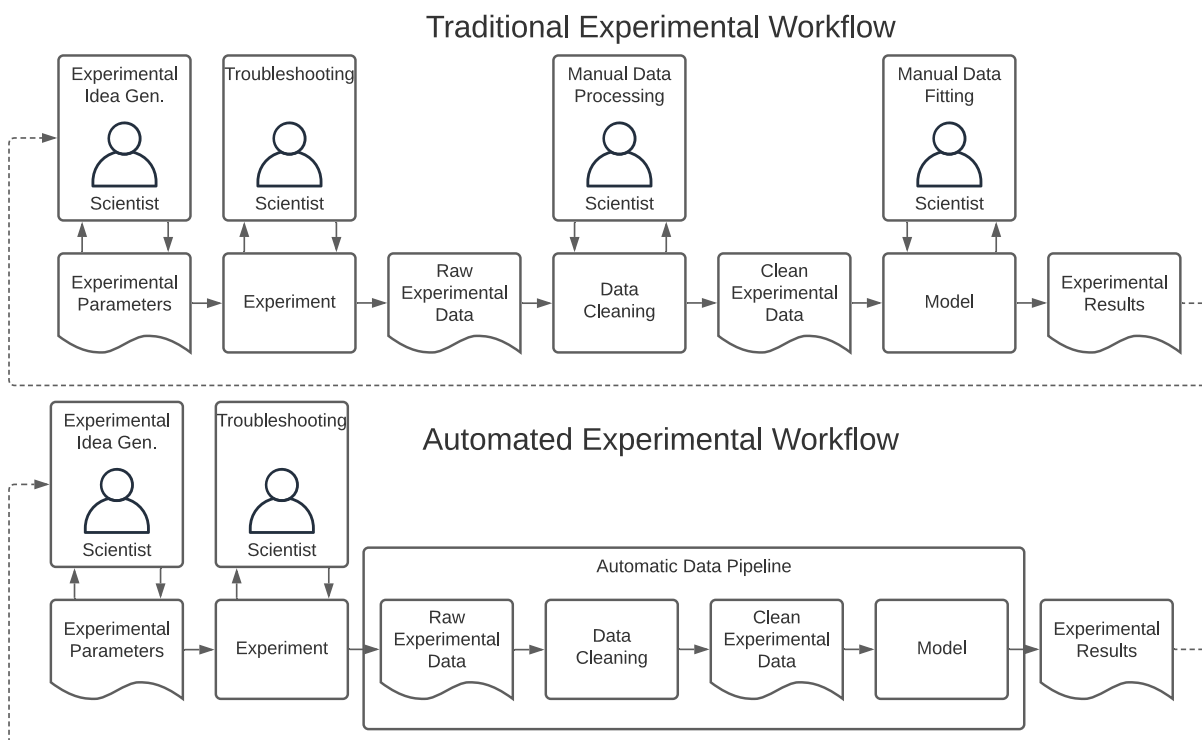


Figure 2: Comparison of a traditional experimental workflow and an automated experimental workflow. The top figure shows the involvement of a scientist in the data cleaning and data fitting portions, where in the second figure those sections are automated via an automatic data pipeline

In Figure 1, the data pipeline replaces the need to manually clean the data after the experiment.

Another component automated by the data pipeline is the fitting of a model to the data. In

traditional experimental workflows, notably spectroscopy, this process is done manually, yet can

be automated, drastically reducing the time required. The model selection and hyperparameter

tuning also take place prior to the pipeline being deployed, and is not included in the flow chart.

Chapter 2

Machine Learning Assisted Sampling of SERS Substrates Improves Data Collection Efficiency

2.1 Acknowledgements

These sections and the abstract of this thesis are based off of a manuscript submitted to Applied Spectroscopy in which I am co-first author. Permission was granted from the other primary author, Dr. Tatu Rojalin, to include the contents of this manuscript in my thesis. This inclusion was also approved by my senior academic adviser (SAA), Wallace Woods and the current Chemical Engineering department chair, Prof. Tonya Kuhl. This manuscript was written with the substantial help of the other two secondary authors Dr. Ambarish Kulkarni and Dr. Randy Carney. The work was supported by the UC Davis Center for Data Science and Artificial Intelligence Research (CeDAR) Innovative Data Science Seed Funding Program.

2.2 Abstract

Surface-enhanced Raman scattering (SERS) is a powerful technique for sensitive label-free analysis of chemical and biological samples. While much recent work has established sophisticated automation routines via machine learning (ML) and related artificial intelligence (AI) methods, these efforts have largely focused on downstream processing (e.g., classification tasks) of previously collected data. While fully automated analysis pipelines are desirable, current progress is limited by cumbersome and manually-intensive sample preparation and data collection steps. Specifically, a typical lab-scale SERS experiment requires the user to evaluate the quality and reliability of the measurement (i.e., the spectra) as the data is being collected. This need for expert user-intuition is a major bottleneck that limits applicability of SERS-based diagnostics for point-of-care clinical applications, where trained spectroscopists are likely unavailable. While application-agnostic numerical approaches (e.g., signal-to-noise thresholding) are useful, there is an urgent need to develop algorithms that leverage expert user intuition and domain knowledge to simplify and accelerate data collection steps. To address this challenge, in this work, we introduce an ML-assisted method at the acquisition stage. We tested six common algorithms to measure best performance in the context of spectral quality judgement. For adoption into future automation platforms, we developed an open-source Python package tailored for rapid expert user annotation to train ML algorithms. We expect that this new approach to use ML to assist in data acquisition can serve as a useful building block for point-of-care SERS diagnostic platforms.

2.3 Introduction

Surface-enhanced Raman scattering (SERS) is a powerful label-free detection and analysis technique that exploits the near field enhancement of inelastically scattered Raman signal via nanostructured plasmonic surfaces[10]. SERS is highly sensitive, capable of even single molecule detection, with broad applicability in detection and monitoring of disease, particularly for cancer. While many proof-of-concept SERS studies emerge annually, and technologies to enable point-of-use and even wearable devices are now a reality, widespread adoption of the technique to replace or supplement existing sensing platforms has not come to fruition. A major bottleneck of this goal is that application of SERS currently requires expert users to collect and interpret data.

In a typical SERS data acquisition process – whether it is a clinical diagnostic platform or a characterization of an unknown chemical entity – hundreds to thousands of spectra are typically collected, preprocessed, and subjected to downstream analyses, e.g., principal component analysis (PCA), hierarchical clustering, or other types of classification routines. Much literature has been devoted to the preprocessing considerations[11]–[13], including de-noising, smoothing, baseline correction algorithms, background subtraction methods, and cosmic ray removal.

Machine learning (ML) and artificial intelligence (AI) methods (e.g., convolutional neural networks, or CNNs, deep neural networks, random forest classifiers, etc.) have been widely applied to various classification tasks following preprocessing. For instance, such methods have enabled classification of small molecules[14] and their mixtures[15], various minerals[16], bacteria[17], [18] and viruses[19]. Discrimination of esophageal cancer[20], non-small-cell lung

cancer [21], and nasopharyngeal and liver cancer[22], have also been demonstrated. CNNs have been applied to Raman/SERS spectra of circulating biomarkers as well, such as extracellular vesicles (EVs) in prostate[23], lung[24], and pancreatic cancer[25], as well as general cancer biomarker identification[26]. Diabetes mellitus detection[27], applications in cytopathology[16], AI-based discrimination of tumor suppressor genes[28], nitroxoline quantification[29], and caffeine and associated metabolites detection[30] have also been proposed. Overall, many ML algorithms have emerged to complement or replace traditional methods (e.g., multivariate classification) for data analysis in vibrational spectroscopy.

While it is apparent that ML greatly improves prediction accuracy, and automated spectral processing improves the efficiency of SERS platforms in general, we posit that progress in developing SERS-based diagnostics is not limited by the lack of state-of-the-art ML algorithms, but instead by the absence of automated data collection and sampling protocols. For example, following spectral preprocessing steps, the user has to decide which spectra are adequate for further downstream analyses (e.g., biological sample classification for diagnostic purposes). A question remains at this stage, whether the analyte of interest and the SERS substrate have been sampled exhaustively enough to produce meaningful and statistically representative data. This step arguably creates the largest barriers to automation of SERS platforms as (1) it requires significant user expertise and domain knowledge, (2) assumes minimal user bias, and (3) relies on several related, but not identical measurements. Recognizing the ability of ML algorithms to translate user intuition to diverse classification problems[31], [32], it is clear that ML methods will provide high value to aid in such expert-driven sampling decisions, even during measurement. To the best of our knowledge, such approaches for SERS data collections have not yet been reported in the field.

SERS data are highly dynamic in nature[10], [33]–[35], manifesting in the heterogeneous fluctuation of spectra – even for a single analyte measured on a high-quality, geometrically ordered substrate [22], [36] For typical measurements, multiple spots need to be sampled many times to account for heterogeneity, arising from pre-measurement parameters (sample exposure time, data collection frequency, laser power, etc.), spatial differences in analyte concentration and orientation, ionic composition of the solution, osmotic and elastic potentials and material-related heterogeneities of the SERS substrate itself, impurities present on the surface[37]–[39] etc. These issues, unfortunately, have led to doubt in the ability to perform truly quantitative SERS [10], [36], [40]–[44].

In light of the above discussion, the main objective of this work is to develop a robust and automated ML-SERS approach to “sufficiently” sample the substrate, i.e., to automatically collect statistically representative quantity of high-quality spectra for a given substrate and the analyte(s). Such an approach offers minimal operator intervention for SERS spectra acquisition, increasing the efficiency of measurement and reproducibility of the downstream analyses.

The hierarchical data sampling scheme currently used in SERS experiments (Figure 3) is designed to collect representative spectra with high signal-to-noise ratios. For a given sample, spectra are collected at separate spots (e.g., x, y coordinates) to capture the spectral diversity of the sample. To increase the signal to noise ratio and reduce variance, multiple spectra at each spot are typically collected and averaged. By excluding negative spectra from the averaging, the signal-to-noise ratio can be increased. Manual exclusion of negative spectra can be accomplished by an expert, but is cumbersome due to the thousands of spectra generated in a typical SERS experiment. Automatic “bad” (i.e. negative) spectra identification is thus a significant objective

to improving SERS experimental data. As a valuable step towards this goal, in this study, we develop a suite of ML algorithms to classify spectra as either “good” or “bad” (i.e., negative) and critically assess their performance. The highest performing XGBoost model was identified and utilized to characterize both in-sample and out-of-sample datasets. This model was then utilized to characterize an out-of-sample dataset and offer the potential for automated data collection, removing the need to monitor the collection procedure completely.

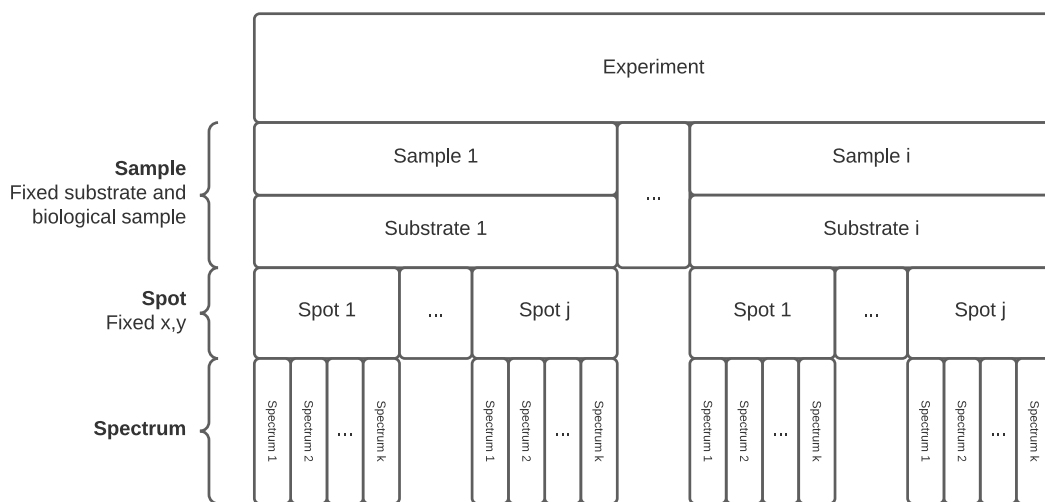


Figure 3: Hierarchical data structure for label-free SERS experiments. A single experiment consists of different samples, each of which is analyzed with a separate substrate. For each sample, different x, y positions (spots) are analyzed by collecting a series of SERS spectra one-after-another. The number of spectra collected per spot and spot collected per substrate can vary depending on signal quality.

2.4 Methods

2.4.1 Sample Preparation

Two commercial plasmonic substrates were chosen for this study, from Moxtek (Moxtek Inc., UT, USA) and Plasmore (Plasmore S.R.L, Pavia, Italy). A well-characterized SERS standard reporter, Rhodamine 6G (R6G), was selected as a model compound for surface scanning. Two

different concentrations were prepared in ultrapure water to demonstrate a high (3 mM) and low (10 nM) R6G concentration. The plasmonic substrates were characterized by scanning electron microscope (SEM), using a ThermoFisher Quattro S (ThermoFisher Scientific, Waltham, MA, USA). For SEM measurement, substrates were mounted on metal studs using two-sided black carbon tape, and the following imaging parameters were applied: working distance 11.4–12.0 mm, spot size 2.5, accelerating voltage 10.0 kV, and chamber pressure 100 Pa.

2.4.2 SERS Spectra Acquisition

SERS spectra were acquired using a custom-built inverted Raman scanning confocal microscope with an excitation wavelength of 785 nm and a 60 \times , 1.2 NA water immersion objective on an inverted IX73 Olympus microscope. Raman spectra were captured via an Andor Kymera3281-C spectrophotometer and Newton DU920P-BR-DD CCD camera. Initial in situ data processing and cosmic ray removal was carried out using Solis v4.31.30005.0 software. All SERS measurements were acquired using exposure time 1 s per scan with a laser power of \sim 10-20 mW. Moxtek or Plasmore substrates were scanned on a 20 \times 20 pixels area thus yielding total 400 spectra per one scanned area. The step size was adjusted to 400 nm, resulting in the total scanned area of 8 μ m \times 8 μ m. To simulate a real scanning procedure performed by a non-trained operator, the scanned areas were selected randomly without any pre-search for “good” signals. Unless elsewhere otherwise described SERS spectra preprocessing was performed using custom scripts written in MATLAB v2020a (MathWorks, MA, USA). Spectral preprocessing included penalized least-squares (PLS) background correction, smoothing, and normalization. Where stated throughout the study, these preprocessed spectral sets were further subjected to principal component analysis (PCA) based on the corresponding MATLAB built-in functions.

2.4.3 Spectra Labeling

To train a supervised machine learning algorithm to identify “good” and “bad” spectra and assess the trained model performance a dataset of labeled spectra must be created. To create this dataset a SERS spectra expert must assign labels to all spectra. To facilitate this task, a custom Python based GUI was created, which allows for the rapid labeling of a collection of spectra. A screenshot of the labeling program is shown in Figure 4. The program allows for keyboard labeling and control facilitating rapid labeling of spectra.

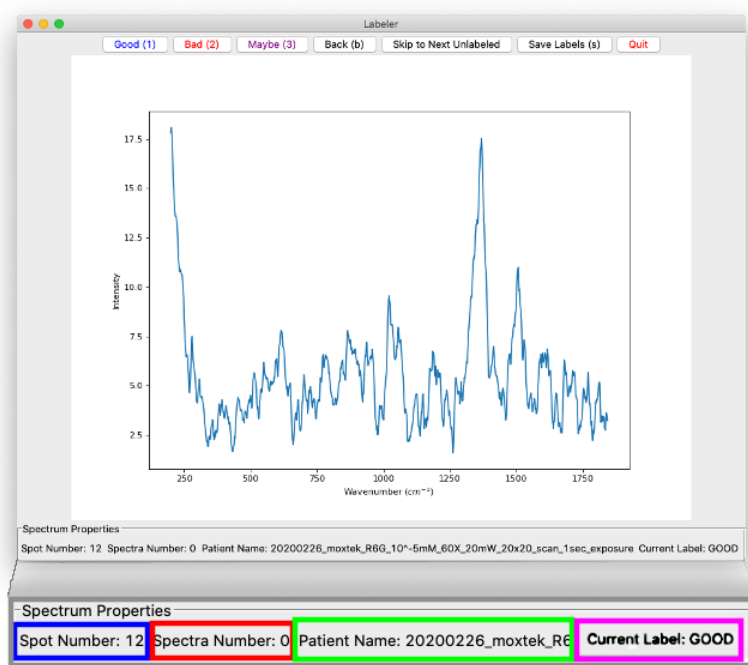


Figure 4: Image of Python-based Labeler app for manually assigning labels to spectra. The interface displays a spectrum to SERS expert, who can assign a label to the spectra with the “good”, “bad” or “maybe” buttons. The spot number, spectra number, sample name, and current label is displayed in the bottom of the window. The save labels buttons saves the labels to a NetCDF file for later use in model training and testing.

2.4.4 Performance Measures

There are a variety of metrics that can be used to assess the performance of a binary classifier. Most of them center around four different possible outcomes of a classification on an unknown sample: a true positive (TP), false positive (FP), false negative (FN) and true negative (TN).

From these four metrics the precision, recall (true positive rate), false positive rate, and accuracy can be calculated (Table 1). The binary prediction of a classifier is typically based on an underlying score of the classifier, and can be adjusted depending on the classifier use case. If the cutoff is increased, then the precision will increase while the recall will decrease. This related increase and decrease is known as the precision-recall tradeoff. To visualize the tradeoff, it is common to make a receiver operator characteristic (ROC) plot, which plots the true positive rate versus false positive rate at various cutoffs. In an ROC plot random guessing is represented by a $y=x$ line and higher performing models are closer to the plot edges. The ROC plot can be collapsed into a single metric by calculating the area under the ROC curve to give the AUC value. The AUC value varies between 0 and 1, with 0.5 being the score of random guessing. The AUC metric is useful because it is independent of the classification cutoff, and thus can give a better overall picture of a model's performance.

Table 1: Metrics useful in assessing the performance of a binary classifier

Name	Formula	Description
Precision	$\frac{TP}{TP + FP}$	The number of true positives to the number of positive predictions
Recall/True Positive Rate	$\frac{TP}{TP + FN}$	The number of true positives to total number of positives
False Positive Rate	$\frac{FP}{TN + FP}$	The number of false positives to number of total negatives
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Total Number of correct predictions over total number of predictions
F1	$\frac{2}{\frac{1}{recall} + \frac{1}{precision}}$	Harmonic mean of precision and recall scores
ROC Curve	plot(x = Recall, y = FP Rate)	Recall operator characteristics curve, which shows the relationship between the true and false positive rate

Another important way for assessing model performance is through calibration plots. The calibration plots assess how well a binary classifier's probability predictions match the frequency of class occurrences. Constructing a calibration plot involves predicting the probability that each element of the validation/test set belongs to a positive class, ordering the elements by the predicted probability, assigning each element to a bin, calculating the fraction of positive classes in each bin and comparing that fraction to the mean predicted probability in that bin. If the model is well calibrated then the fraction of positives classes should match the mean predicted probability of the classes in the bin. Scikit-learn's "calibration_curve" function automatically performs this process and generates the calibration plots.

2.5 Results

2.5.1 Sample Collection

High and low concentrations of a common SERS-active reporter molecule, R6G, were dried out on to two high-quality, lithographically-formed commercial SERS substrates (Moxtek and Plasmore), as schematized in Figure 5. SEM micrographs displaying the plasmonic nanostructures on either surface are shown in Figure 5b. In total, five samples were prepared. Two different concentrations of high and low 10nM R6G concentrations were prepared on both Moxtek or Plasmore substrates. Substrates were either scanned using 10 mW laser power for high concentration or 20 mW laser power for low concentration. A fifth sample was created to investigate the effect of laser power on the recorded SERS signals, therefore a "low"

concentration Plasmore was also scanned using lower 10mW power. Prepared substrates were subjected to SERS measurements using a custom confocal scanning Raman microscope to yield several random 20×20 -pixel areas (total scanned area of $8 \mu\text{m} \times 8 \mu\text{m}$). Figure 5c shows a representative spectra average and standard for high concentration of R6G deposited on a Moxtek substrate.

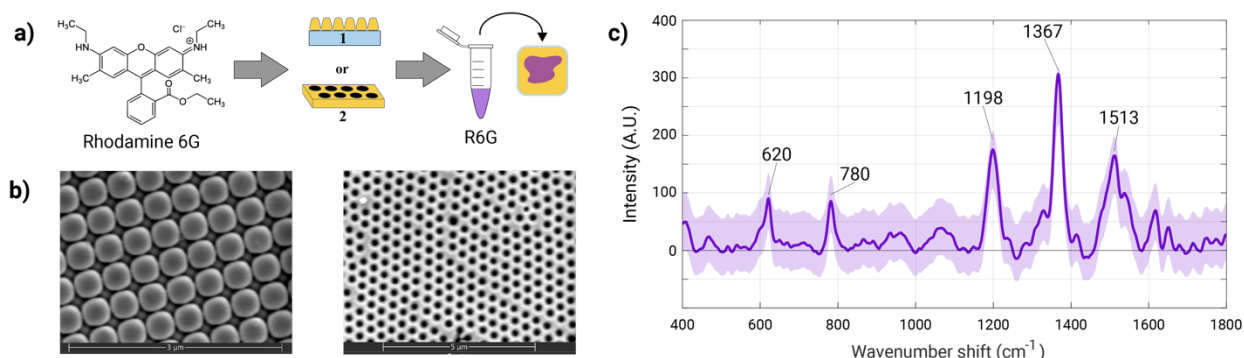


Figure 5: Experimental workflow of the ML-SERS platform. a) Rhodamine 6G was used as a SERS reported molecule on plasmonic Moxtek (1) or Plasmore (2) substrates. Varying solutions of R6G were pipetted ($\sim 20 \mu\text{L}$ total) onto the surface and 20×20 pixels surface areas were scanned. b) SEM micrographs illustrate the structure of the Moxtek and Plasmore substrates. c) Representative SERS spectrum of R6G; the highlighted peaks at 620, 780, 1198, 1367 and 1513 cm^{-1} are characteristic spectral features of R6G.

A conventional approach to classify spectral data is to carry out PCA following manual selection of quality spectra (and/or via iterative use of PCA to screen out low quality or outlier data). An example of this process is illustrated in Supplemental Figure 6. Use of thresholds or intuitive interpretation using PC score plot and principal component loading spectra are relatively systematic methods to guide the spectra selection procedure. However, for more complex datasets featuring mixtures of chemicals, where the PCs do not cleanly correspond to single entities, it is tedious and time-consuming to apply such manual selection routines for hundreds or thousands of spectra, and further adds a notable source of inter-operator bias. Therefore, we endeavored to explore application of ML algorithms to recognize quality spectra following expert user training.

An example PCA-based conventional spectra selection procedure is illustrated in Figure 6 showing the PCA score plot (Figure 6a) for the complete dataset (2000 spectra). Figure 6b displays the corresponding PC1 loading spectrum, containing 33.5% of the total variation in the dataset. Each marker in the score plot represents a single spectrum, and each colored group demonstrates a 20×20 raster scan acquired from one substrate and one condition as described. The score plot and loading spectrum provide intuitive information on the directions where the “good” or “bad” spectra can be presumably found. In this example PC1 loading spectrum contains positive peaks and bands that can be associated with R6G (numbered peaks in Figure 6b). Correspondingly, the red shaded area in the PC score plot (Fig. S2a) shows the positive score values on different sample groups. Thus, the spectra residing in this area contain more R6G features than the ones located on the negative side of the PC score plot. Fig. S2c shows representative examples of “good”, “maybe” and “bad” spectra selected from the extreme ends of respective sample groups; red star denoting a “good”, yellow “maybe”, and blue “bad” spectrum. This is a relatively systematic method to find various representative spectra types within the measured data set, and can be used to guide the spectra selection procedure. However, typically it is notably tedious to select hundreds or thousands of spectra manually, especially if the data set comprises a complex mixture of chemicals and not a single entity as tested here. It is also worthwhile stressing that this phase comprises a notable source for user-driven bias.

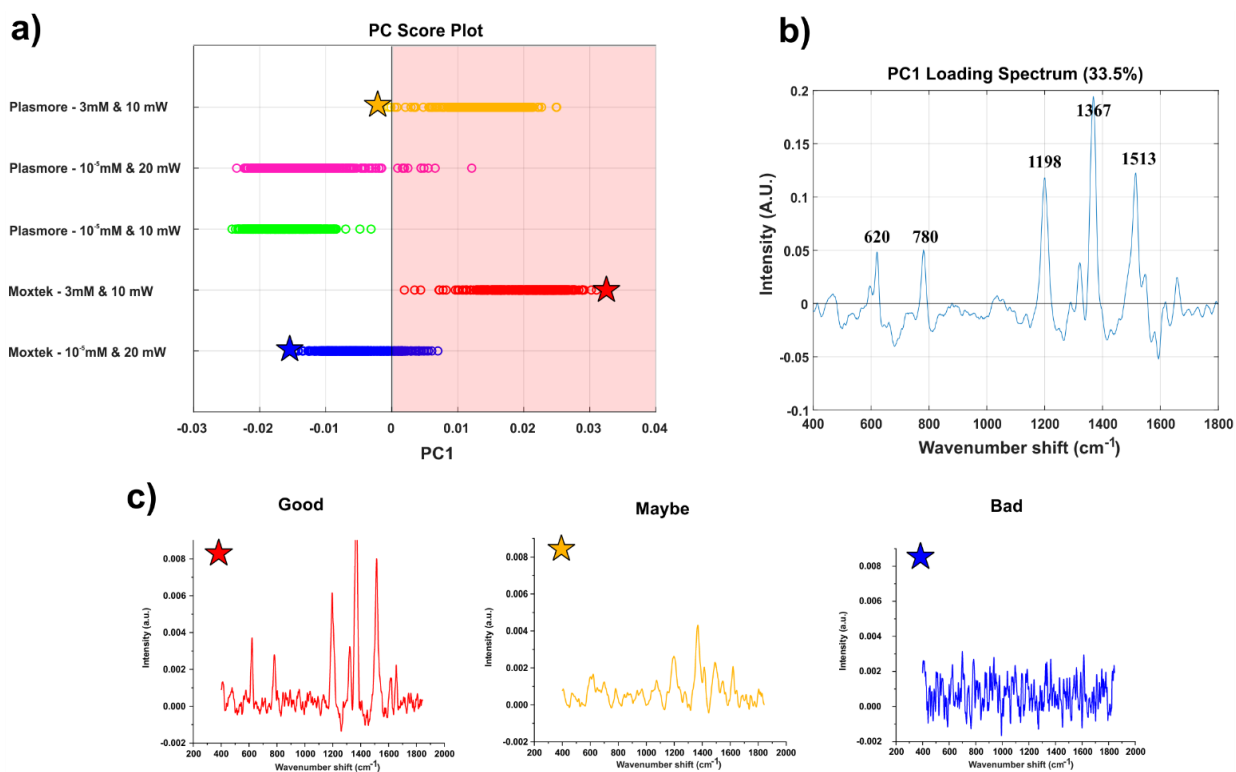


Figure 6: PCA analysis to guide SERS spectra selection. a) PC score plot demonstrating each measured sample group (400 spectra per group, total 2000 spectra in the data set), each marker represents a single SERS spectrum. b) PC1 loading spectrum displaying the positive features at 620, 780, 1198, 1367, and 1513 cm^{-1} that can be associated to the SERS reporter molecule R6G used in the study. c) Examples of “good”, “maybe” and “bad” spectra pinpointed with corresponding star markers (red, yellow, blue, respectively) at the PC score plot.

2.5.2 Data Organization and Parsing for ML Input

To utilize the spectra data for ML, we established a python-based data pipeline that converts plaintext Raman spectrum files as input, preprocesses the spectra, and converts them into a binary format that can be utilized for visualization, data labeling, and model training. The three stages of the data pipeline are shown in Figure 7. All code utilized for this data pipeline is available under the open-source MIT license on GitHub (See Data Accessibility Statement)

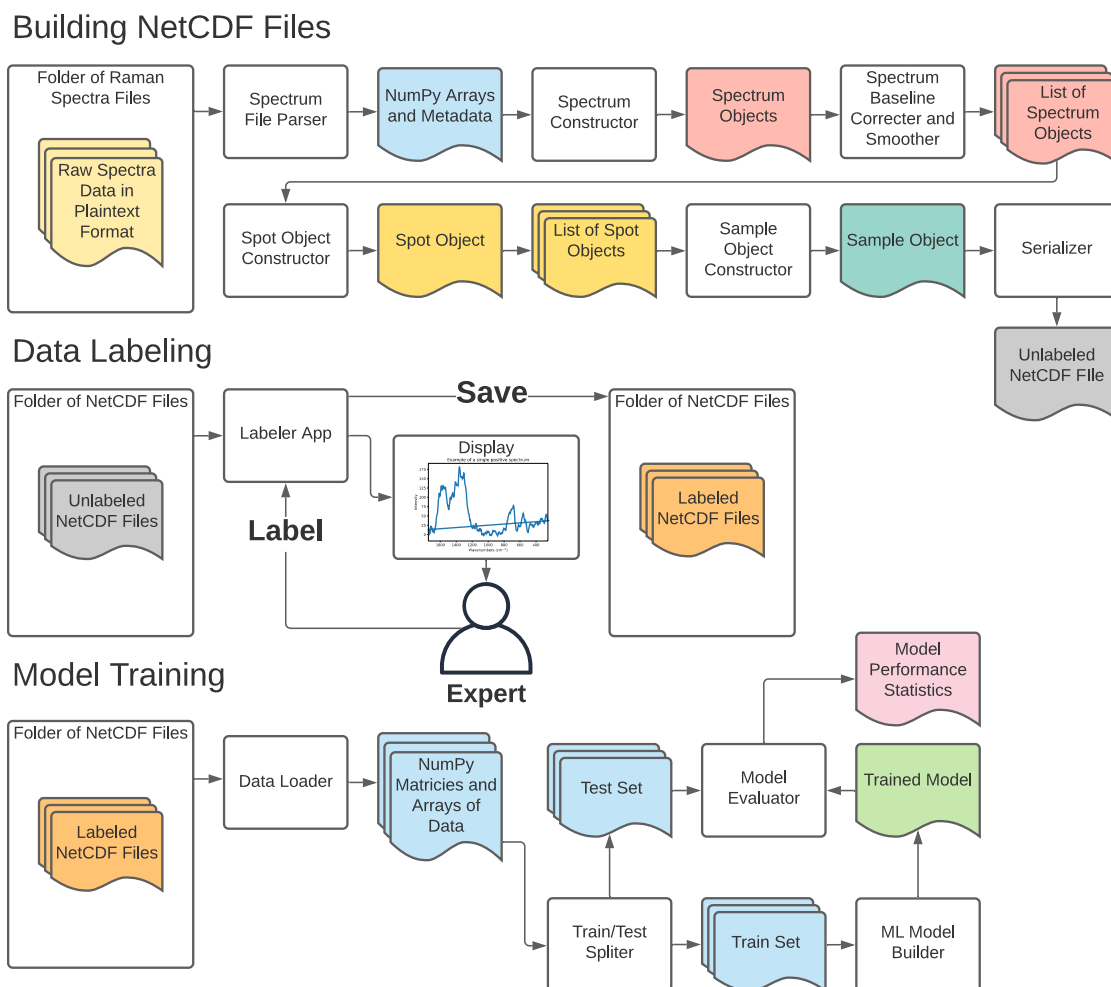


Figure 7: Open-source data pipeline developed for this study. The first stage of the pipeline converts and processes raw spectra files into a binary NetCDF file format. The second data labeling stage employs a custom Python “Labeler” app, allowing an expert Raman user to quickly assign labels (e.g., “good”, “bad”, or “maybe”) to the spectra serialized in the netCDF files. After labels have been assigned, the last stage of the pipeline is model training, where the binary files are loaded into NumPy arrays to train and test various ML models.

The first stage converts plaintext Raman spectrum files into a binary NetCDF file. The NetCDF format (short for network common data form) is a machine independent data storage scheme designed for efficiently saving multi-dimensional scientific data, and well suited for storing spectral datasets[45]. An essential part of this stage is the baseline correction and smoothing, which was performed with the airPLS baseline correction algorithm and Whittaker smoothing function, respectively, via code ported to Python 3[46], [47]. This modified code is available on GitHub in compliance with the LGPL license.

The second stage of the pipeline utilizes expert data labeling to train the ML models in the third stage. To implement a supervised learning algorithm for “good” and “bad” spectra classification, labels need to be associated with each spectrum. Given the need to quickly and easily label thousands of spectra for training (2000 different individual spectra needed to be labeled for this study) a Python-based labeling program was created. This program takes a series of netCDF files as input, displays each spectrum to the user and allows for rapid labeling, and then saves the dataset with the applied labels. A screenshot of the Labeler program interface is shown in Figure 4. The premise is to establish three different bins for the classification purposes: a) “good” and chemically representative R6G spectrum (also termed as “positive” in this context), b) “maybe” adequate R6G spectrum where a clear decision could not be made by an expert user, and c) “bad” (also termed as “negative” in this context), unrepresentative R6G spectrum (e.g., very low SNR). Labeling was based on expert user intuition and experience, focusing on feature rich spectra with clear sharp peaks and minimal noise. For this study, the Labeler program was used to tag a total of 1995 spectra (940 good, 936 bad, rest 119).

2.5.3 Model Selection and Performance Analysis

Following the labeling task, the acquired spectra were evaluated using an assortment of popular ML-assisted classification routines. The labeled datasets were shuffled and split into train, validation, and test sets (72.25%, 12.75%, and 15% respectively). The percent positive and negative in each subset was calculated and found to be within +/- 5% of 50% for both classes. Building on the hypothesis that existing ML classifiers are well-suited to distinguish between good and bad spectra, we evaluated six distinct methods: logistic regression stochastic gradient descent (LR-SGD), support vector machines stochastic gradient descent (SVM-SGD), decision

trees (DT), linear discriminate analysis (LDA), random forest (RF) and XGBoost (XGB). The first five models are implemented within the Scikit-learn package[48], whereas a custom package for XGBoost is used[49]. For each model, performance is assessed by calculating resulting receiver operator characteristics (ROC) curve and associated area of the curve (AUC). The ROC curve quantifies the diagnostic ability of a classifier for different discrimination thresholds, while the AUC is independent of the classification cutoff, and thus can give a better overall picture of a model's performance [50].

After hyperparameter tuning using the AUC score (walkthrough can be found in our Jupyter notebooks on GitHub) all six models were assessed by training on the training dataset and their performance validated using the validation set. The ROC plots and associated classification metrics for these six models are shown in Figure 9. A full description of the calculation for these classification metrics can be found in Table 1 Associated calibration plots are shown in Supplemental Figure 8.

The LDA model was the worst performing in all categories, likely due to the large number of features and lack of regularization. The next worst performing model (by AUC) was the DT model, which consisted of a series of Boolean decisions arranged into a tree structure. This was followed by a LR-SGD and SVM-SGD. Unlike the default logistic regression and SVM solvers, the SGD classifier was not affected by the high correlation between adjacent features (inherent to spectral data, adjacent wavenumber shifts are correlated) and was able to provide stable solutions. Despite their simplicity, the SGD based models performed well, with the SVM-SGD providing the highest precision of all tested models. The final two tree-based models tested were RF and XGB. The RF model outperformed XGB by 0.0006 AUC units, yet the XGB model was

better correlated and scored higher in the accuracy, recall and F1 categories (Supplemental Figure 8). The confusion matrix for the XGB model is shown as an inset in Figure 9a. These results are consistent with other studies[51] which identify XGBoost as a top performing algorithm for binary classification tasks. Therefore, we focus on the XGBoost algorithm in the remainder of this work.

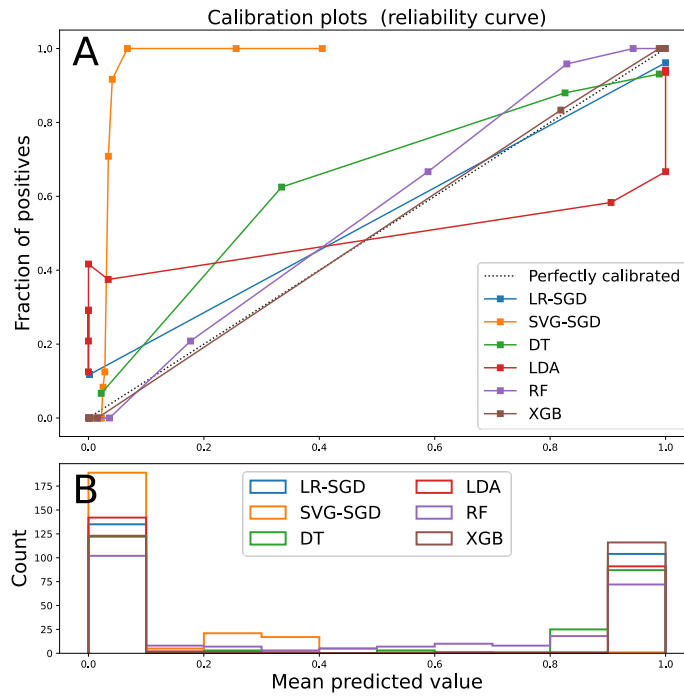


Figure 8: a) Calibration plots for four tested machine learning algorithms LR-SGD, SVG-SGD, DT, LDA, RF, and XGB on validation set. The x axis represents the mean predicted probability of the events occurring in the bin. The y

axis represents the fraction of positive (i.e., “good”) spectra in the in the bin. b) The total number of probabilities in each decile, for each classifier.

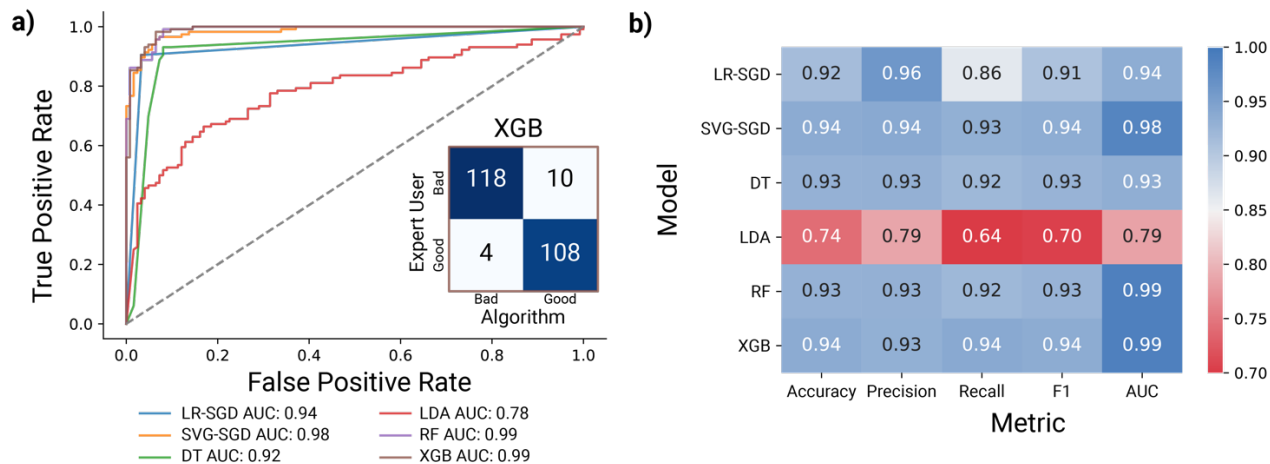


Figure 9: a) ROC curves for six tested models, logistic regression (LR-SGD), support vector machines (SVG-SGD), decision trees (DT), linear discriminate analysis (LDA), random forest (RF) and XGBoost (XGB). Random guessing is represented by the $y=x$ line whereas higher performing models lie closer to the left corners of the plot. Inset: Confusion matrix for XGBoost algorithm trained on train and validation set and tested on the test set. The trace of the matrix indicates correct predictions, while the offset values indicate incorrect predictions. b) Comparison of hyperparameter-tuned model performance on validation data set. The highest value for each category is bolded and the lowest value for each category is italicized.

2.5.4 Testing Model Performance

Recognizing the favorable performance of XGB on the validation set, we proceed to investigating the efficacy for in-sample and out-of-sample tests sets. The in-sample test sets, which involve intermixing the spectra from all samples into the test, train and validation sets, give a better overall picture of the model performance by capturing the spot-to-spot heterogeneity of the spectra. The true in-sample performance of XGB is estimated by training it on the combined train and validation set, and assessing its ability to predict the labels of the 282 spectra in the test set. The metrics derived for the performance of the XGB model are shown in

Table 2. Representative classifications for the XGB model on specific spectra are shown Figure 10. Overall, we observe good performance of the model with 95% accuracy.

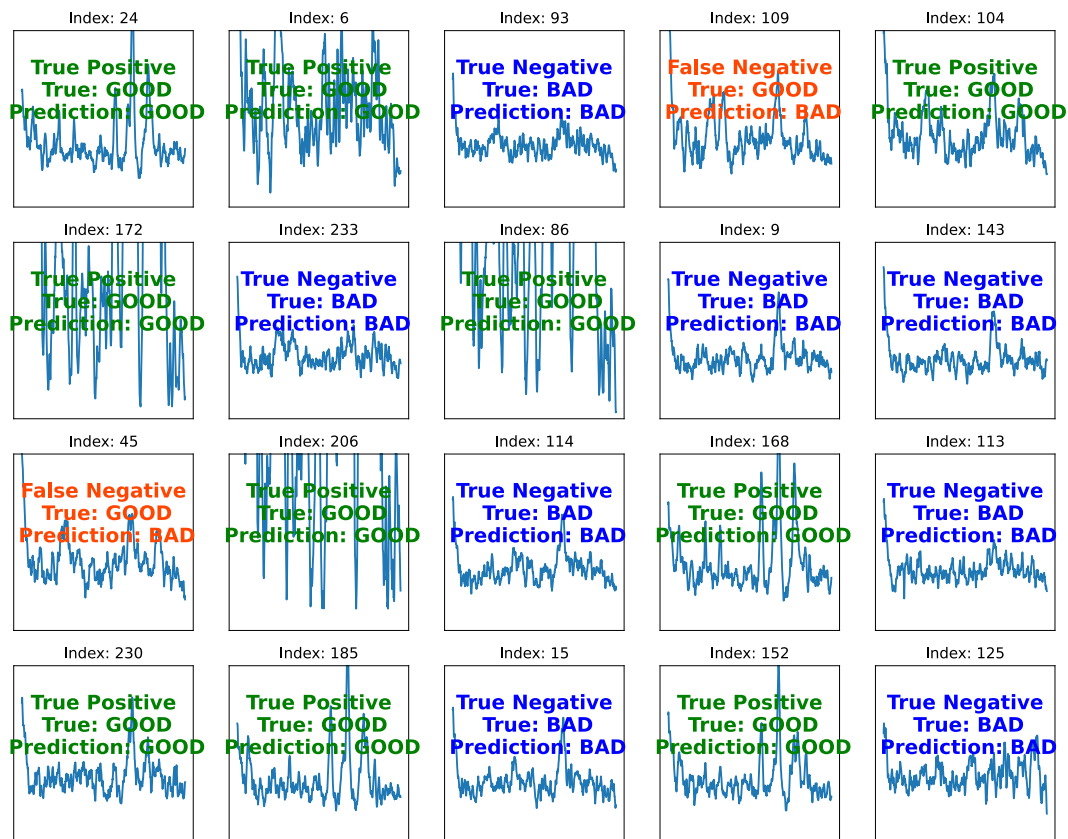


Figure 10: Representative spectra classifications for XGBoost on in-sample test dataset. Text overlaid on each spectra indicate the expert user label trained model’s performance in accurately predicting the label.

Table 2: Model performance parameters for the XGBoost algorithm trained on the training set and validation set and tested on the test set.

Accuracy	Precision	Recall	F1	AUC
0.95	0.93	0.97	0.95	0.99

2.5.5 Variable importance

In this experiment no feature engineering or variable selection was attempted, such that the entire 1024 features of the spectra (i.e., 1024 data points, arising from the CCD dimensions collecting the photons following dispersion) were used to train the XGB algorithm. We note that this feature makes the ML approaches for analytical spectroscopy methods like SERS notably

powerful as there is no need to perform a priori dimensionality reduction (e.g. PCA) but rather the full feature space can be used in the training phase. To determine what features were most important in predicting the label of the spectra the importance score for different regions were aggregated together and plotted together. These bar charts give the overall importance of a certain region to the prediction (Figure 11). Unsurprisingly, the region 1510-1350 cm^{-1} has the highest importance, as the peaks present in positive spectra tend to cluster around that region. Interestingly the region 200-400 cm^{-1} also has some high importance. It is likely that the intensities of these variables give an indication of the overall noise of the spectra, and act as a proxy for estimating the signal to noise ratio.

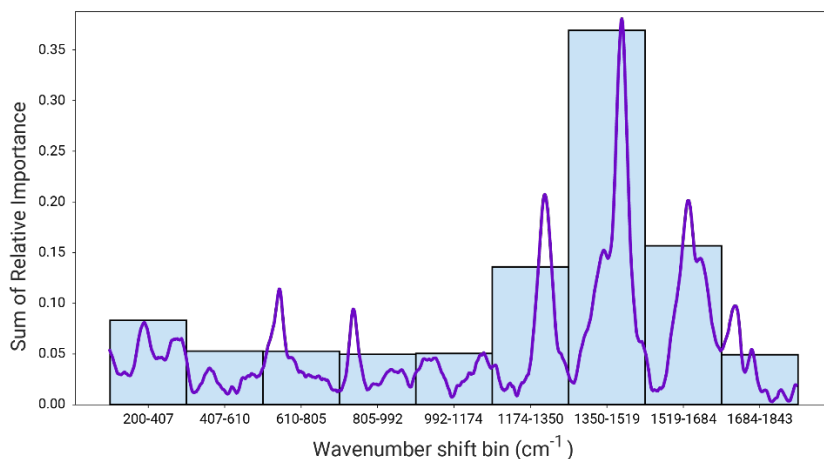


Figure 11: Relative importance of different wavenumber regions for the XGBoost model trained on the in-sample train and validation data. Each bin contains the sum of the importance scores of the wavenumbers in the range [start wavenumber, end wavenumber). An R6G spectrum is overlaid for reference, where it is apparent that relative importance correlate with spectral features of R6G.

2.5.6 Out of Sample Testing

To test the out-of-sample performance of XGB, the model was trained on four out of the five substrates and then used to predict the labels of the left-out test substrate (Table 3). The hyperparameters were previously tuned using data from the test substrates, but the data was

otherwise unseen by the model. This process was repeated for all five samples. The predictions were assessed with standard metrics (Table 3).

Table 3: Out-of-sample performance for XGBoost model on five substrates

Test Sample	Substrate	Dye	Concentration (M)	Power (mW)	Accuracy	Precision	Recall	F1	AUC	Fraction Positive
1 ¹	Moxtek	R6G	1.00E-08	20	0.90	NA	NA	NA	0.96	0.10
2	Moxtek	R6G	3.00 E-03	10	0.77	0.78	0.98	0.87	0.83	0.75
3	Plasmore	R6G	1.00E-08	20	0.95	0.91	0.64	0.75	0.99	0.12
4	Plasmore	R6G	3.00 E-03	10	0.72	0.78	0.73	0.75	0.80	0.60
5	Plasmore	R6G	1.00E-08	10	0.98	0.98	1.00	0.99	0.69	0.98

¹For sample 1, no false positives or true negatives were generated making the precision, recall and F1 formula inapplicable.

To further evaluate out-of-sample performance, we overlay the labeled data onto their spatial coordinates in Figure 12a, given that these datasets were collected by scanning over a 20 by 20-pixel grid ($8 \mu\text{m} \times 8 \mu\text{m}$) in even increments (400 nm pixel width, approximately diffraction limited). Green represents a “good” spectrum label, while red represents a “bad” spectrum label and purple represents an ambiguous “maybe” spectrum where a label could not be accurately assigned by the expert user. For each of these labeled samples, the model was used to predict the corresponding labels, with colors again plotted in Figure 12b. Ambiguous spectra were excluded from training the model and were not predicted by the model, but are still shown in purple. Incorrect predictions of false negatives and false positives are coded by warm colors, light red and pink, respectively. Similarly, the true negative was represented by a cool color, i.e., light blue and the true positive by a dark blue. More generally, a perfect prediction would correspond to all true positives and/or true negatives, which corresponds to blue shades.

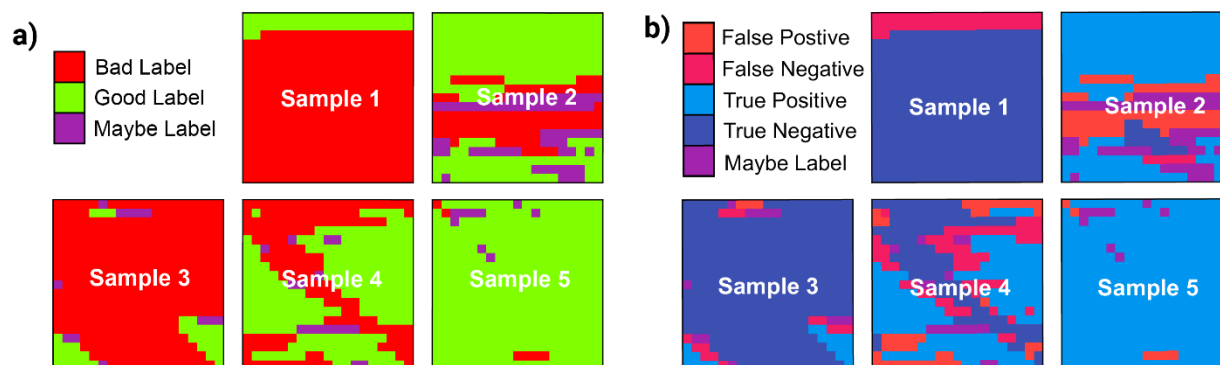


Figure 12: a) Visualization of the sampled region from the five substrates used in experiment along with the corresponding labels. Red squares represent a negative spectrum, green squares represent a positive spectrum, and purple squares represent spectra with ambiguous labels, which were excluded from fitting and prediction. b) Comparison of actual and predicted labels for five test samples. The comparison is represented by the color. False positive (red), false negative (pink), true positive (light blue), true negative (dark blue), and ambiguous, maybe labeled, spectra (purple) pixels are shown.

The performance of the XGB model on the in-sample and out-of-sample test set demonstrate that the categorization of spectra can be automated. With the in-sample test set, the XGBoost model achieves an AUC score of 0.99 similar to its performance on the in-sample validation set, indicating minimal overfitting. The 95% accuracy and similarly high precision and recall score prove that the majority of the predictions are correct. The out-of-sample performance is lower and more variable, yet still high. The average out-of-sample AUC score is 0.85 ± 0.12 and an average accuracy is 0.87 ± 0.12 demonstrating that the model is still highly successful at classifying spectra in unseen substrates in different conditions. After learning the labeled spectra from four R6G samples collected at various concentrations, substrates and power, the model is able to correctly predict the label for spectra 87% of the time. This feat unambiguously shows the potential of ML-assisted sampling to automate spectra categorization.

2.6 Discussion

Here we introduced ML-assisted SERS spectra classification methodology to streamline acquisition and efficiently classify recorded spectra. The exclusion of negative signals from the SERS analysis already takes place during normal experimentation. When scanning a substrate, the majority of signals are negative (e.g., noisy, not representative of the typical sampled areas, out of focus, capture cosmic rays). Typically, a trained experimentalist makes the determination of when a “good” signal is collected. Although subjective, this strategy utilizes our impressive pattern matching ability, which is challenging to replicate with structured algorithms. This technique excludes the majority of negative spectra by avoiding their initial collection, but it is not perfect and negative spectra occasionally creep into the recorded dataset. Although experimentalists can easily distinguish between good and bad spectra, the large datasets collected by typical SERS experiments make manual excluding the negative spectra post-collection onerous. In addition to presenting a barrier to large data set collection, these expert user-driven decisions also limit the application of SERS in a clinical setting. For SERS technology to transfer from the research laboratory to the clinic these subjective, labor intensive steps must be eliminated.

The scholarly literature encompassing automatization endeavors of Raman and SERS measurements predominantly demonstrates approaches to automate either (i) the collection or (ii) the data preprocessing phase, e.g., baseline correction, cosmic ray-induced spike removal, noise reduction, scaling and normalization, background subtraction, including various thresholding techniques to harness signal-to-noise ratio for spectra selection[11]–[13]. The main limitations of

the current preprocessing techniques are that they either rely on tuning the processing parameters (e.g., fitting parameters) or require calculating and thresholding the signal to noise (SNR) ratio, which is not possible if the underlying analyte signal is not known or highly fluctuating. Therefore, there exists a clear niche to design robust workflows to select spectra for the downstream analyses. In essence, our strategy is independent of the spectral preprocessing approaches and SNR thresholding, rendering it a promising means to be applied in a wide variety of different platforms.

ML algorithms are able to codify human intuition by learning from labeled training data and are well-suited to identify noisy, feature-poor spectra. The use of a ML algorithm has several advantages over a traditional structured algorithm. A trained ML model requires no parameter tuning once trained and can learn from the extensive experience of trained experimentalists. With the availability of open-source ML packages[48] training and integration are straightforward.

A plethora of different classes of ML algorithms exist and new ones are frequently being invented. Of the existing classes, they can roughly be divided into two domains, classical ML algorithms and deep learning algorithms. Classical ML algorithms include tree-based algorithms such as random forest and XGBoost, as well as more established classifiers like support vector machines. Deep learning algorithms encompass the tremendous diversity of multilayered neural network models, such as CNNs. Both classical and deep learning models can achieve similarly high performance, but classic ML algorithms can perform well on smaller datasets whereas deep learning architectures typically require tens of thousands of data points to converge. In this current work the complete dataset consisted of only 2000 different spectra, thus the tested

models were confined to classical models, yet the methods presented here are easily extendable to deep learning models when working with larger datasets.

Amongst all the models tested here, the XGBoost model performed best across both the in-sample and out-of-sample datasets. Its performance in this dataset matches our expectation that it is performing akin to a user expert making an intuitive decision. To detail this, consider the is major inter-sample variation in the fraction of expert assigned positive labels, likely due to the inhomogeneous covering of dried R6G on the SERS substrates. In sample one, 90% of the spectra are negatively labeled and the classifier predicts a negative label for all of them. In sample five the reverse situation occurs; 98% of the spectra are positively labeled and the XGB model assigns a positive label to all the spectra. In these extreme cases, XGB is essentially learning from the out-of-sample labels and not taking into consideration the unique characteristics of the substrate. In this case, an expert experimentalist would adjust their own threshold of classification based on the observed signal to noise in a specified sample. For example, if many weak signals were observed, the threshold for collecting a spectrum would be lower than in the case where the majority of spectra had an apparent high signal to noise ratio. In the intermediate case of sample 4 with a 60% positive rate, the algorithm performs well (AUC = 0.80), although lower than in the test case (AUC = 0.99). Nevertheless, the algorithm is still successful in categorizing samples with a range of positivity rates.

Future work using this approach will involve automatically tuning the classification threshold based on the number of positively classified spectra in a sample. We also will explore the feedback of this trained algorithm to control stage movement and automate measurement of full datasets.

2.7 Conclusion

This work describes application of an ML algorithm to address a central challenge for adapting SERS to automated platforms: the current dependency of expert user-driven endpoints for sampling. The elimination of bad spectra from a collected dataset can increase the signal to noise ratio by reducing the variance. Especially in SERS applications it is desired to collect and analyze as homogeneous sets of spectra as possible, which is accomplished by the ML assisted spectra selection. By applying this algorithm to the acquisition stage, the labor required to collect many spectra can be reduced making collecting larger and more comprehensive datasets feasible. Furthermore, by automating the acquisition stage of the SERS experiment another barrier to the clinical application of this technology can be broken down. Given the exponential growth of acquired data (e.g., spectra, images, or videos), there is an immense demand for integrating reliable, automated and fast analysis methods to the experimental procedures for SERS instrumentation. We envision that the workflow described here will allow for more robust automated SERS analyses. We foresee that the introduced platform can be further expanded to quantitative analyses of chemicals as well as complex biological and clinical samples such as patient-derived EVs or crude serum for modern diagnostic purposes.

2.8 Data Accessibility Statement

All data collected for this study, including SERS datasets and the Labeler program files, can be downloaded from the following open repository: <https://github.com/kul-group/ramanbox/>. All open-source Python code will be maintained at: <https://www.github.com/kul-group/ramanbox>

2.9 Funding

This work was supported by the UC Davis Center for Data Science and Artificial Intelligence Research (CeDAR) Innovative Data Science Seed Funding Program; the Ovarian Cancer Education and Research Network, Inc. (OCERN); the NIH [grant number 1R01CA241666].

Chapter 3

The Multiscale Atomic Zeolite Simulation Environment (MAZE): A Python Package for Improved Zeolite Structural Manipulations

3.1 Acknowledgements

The MAZE package involved combining the extensive scripting library of other students, most notably those of the PhD students Sam Holton and Jiawei Gao. This work was also made possible by the extraordinary support of Prof. Ambarish Kulkarni who advocated for the project and provided guidance on the API design and provided his extensive library of zeolite scripts.

3.2 Abstract:

Zeolites are nanoporous materials with widespread industrial applications as catalysts and gas separators. Due to the enormity of the zeolite chemical space and structural complexity, computational experiments are needed to identify high-performing zeolites and interpret zeolite characterization data. These computational experiments are enabled by software packages, most notably the Atomic Simulation Environment (ASE) which provides an easy-to-use Python interface to drive low level simulation code. ASE has some limitations that make certain zeolite simulation workflows challenging and labor intensive. These limitations motivated the creation of the Multiscale Atomistic Zeolite Simulation Environment (MAZE) package which builds on-top of ASE to facilitate common zeolite structural manipulations that are challenging with the base ASE package. The improved interface of MAZE, compared to ASE, is demonstrated by applying application programming interface (API) design heuristics and showcasing the ability of MAZE to facilitate common zeolite workflows. It is demonstrated that the MAZE package has an improved API for zeolite workflows and that this can facilitate the creation of large databases of zeolite structural derivatives

3.3 Introduction

Zeolites, a broad class of silica-based nanoporous materials, are widely used for various industrial applications including gas separation and catalysis [52]–[54]. The properties of these materials depend on various factors including the framework topology, Si/Al ratio and presence of extra-framework species. Specifically, transition metal (TM) exchanged zeolites, which

combine the desirable characteristics of heterogenous catalysts (high thermal stability and simpler separations) with those of enzymes (high selectivity and reactivity under mild conditions) [54] have received considerable attention as catalysts for NO_x abatement and methane valorization. In these applications, the zeolite framework provides a thermally-stable scaffold for the formation of catalytically active species [55], [56]. As the precise identification of the active site is complicated by the presence of multiple TM species, computational modeling is often used to provide valuable insights (e.g., thermodynamic stabilities, reaction barriers etc.) into the reaction mechanisms [57][56]. Given the various length- and time-scale associated with the relevant molecular processes (e.g., adsorption, diffusion, reaction), multiscale approaches that combine wave function theory, periodic density functional theory and classical force fields are often necessary. While a number of broadly-applicable software packages are available for performing these calculations, there is a need to develop a software toolkit for zeolite-specific tasks. In this work, we describe the philosophy behind and the capabilities of a new python-based open-source software package –Multiscale Atomistic Zeolite Simulation Environment (MAZE).

The increasing availability of open-source software packages [58] that run, analyze and offer an user-friendly interfaces has greatly simplified the process of performing computational chemistry calculations. A classic example is the Atomic Simulation Environment (ASE), provides interfaces to various computational chemistry codes (e.g., VASP[59], LAMMPS[60], GPAW[61] etc). [1] ASE provides python-based wrappers to the underlying quantum chemical simulation code and offers an intuitive application programming interface (API) for setting up, starting, and analyzing calculations [1], [62]. By automating the cumbersome computational setups and subsequent data analysis, ASE simplifies the process of performing and analyzing

complex calculations [1], [63]. Furthermore, by allowing manipulation through Python scripts, rather than a GUI, these calculations become self-documenting, reproducible and easy to streamline into complex workflows [1], [58].

3.4 Current Limitations with Tracking Atoms

Within the ASE Interface

Despite the continued developments with the ASE codebase and the active user community, we note that a few specific structure manipulation tasks currently challenging to implement within ASE. Often a variety of structure manipulations (e.g., extracting and reinserting clusters, adding terminal H atoms etc.) are necessary to address a zeolite-specific scientific question – the current ASE interface is not-suited for “tracking” the changes in the atom indices. This is illustrated using a simple example below.

In ASE groups of atoms are represented in memory by *Atoms* Python objects. In an *Atoms* object, the properties of all of the atoms are stored in *NumPy* arrays. When a specific atom in an *Atoms* object is accessed using the `get_item` method (e.g., `my_atoms[index]`) an *Atom* object is created, which has (among others) the attributes ‘tag’, ‘position’, ‘symbol’ and ‘index’. The properties of an individual atom in an *Atoms* object can be altered by directly changing an atoms property with the `set_item` method. For example, to set the tag of the first atom in an *Atoms* object to 3, the command `my_atoms[0].tag = 3` can be used. A subset of atoms can also be accessed by passing in an iterable collection of indices (e.g., `my_atoms[indices_list]`), which returns a new *Atoms* object containing the selected subset of atoms. The underlying data structure for storing the

atoms properties is highly efficient, since it doesn't require storing an individual Python object for each atom represented by an *Atoms* object. Unfortunately, it also means that the indices of each atom can change with each addition or deletion. Figure 1 shows the structures and atom indices for various glyoxal derivatives, demonstrating how structural manipulations can alter the indices of atoms.

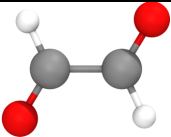
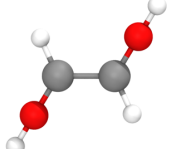
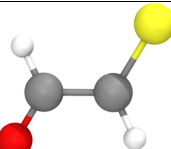
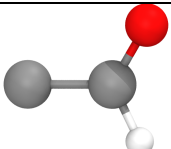
Structure	Indices of the NumPy array							
	0	1	2	3	4	5	None	None
Add two H atoms to #1								
	0	1	2	3	4	5	6	7
Replace the O atom by the S								
	0	1	2	3	4	5	None	None
Delete the O and H on C1								
	0	1	None	None	2	3	None	None

Figure 13: Relationship between indices in *Atoms* objects derived from glyoxal. The indices of the elements carbon, oxygen, hydrogen, and sulfur are colored black, red, purple and yellow respectively. The columns relate the indices of different *Atoms* objects to each other. The indices shifting issue becomes more pronounced when multiple structural operations are performed in series. For example, if the O and H atoms are added back to the #3 to recover #1, their indices (4 and 5) would differ from the initial structure.

Figure 1 shows how the indices of the individual atoms can change when additions and deletions are performed. The most pronounced difference is in the fourth structure, where the deletion of two atoms causes the decrement of the index 4 and 5 to 2 and 3 respectively. The addition of atoms simply extends the arrays and are thus added to the end. There is no inherit ordering in the indices thus the order in which atoms are added to a structure effects the resulting order. If a substitution is made, by changing the identity of a given atom, then the indices remain unchanged.

The changing indices with structural manipulations can make certain zeolite workflows difficult since it is not possible to automatically match the indices between related structures. Although it

is possible to manually tag individual atoms, this process is cumbersome and the integer tags cannot be guaranteed to be globally unique. Furthermore, not all output formats preserve these tags, meaning that the tagging information can be lost throughout a workflow. The mutability of *Atoms* objects, and subsequent index shifting, introduces significant complexity in tracking the relationship and identity of atoms.

3.5 MAZE Design Philosophy

Recognizing the challenges outlined above, the MAZE package puts the atom relationship tracking at the center of its design, while maintaining compatibility with all of ASE features. As demonstrated in the following sections, it greatly simplifies zeolite workflows, and allows materials informatics experiments to be easily performed. The key innovation of the MAZE python package is the development of an *IndexMapper* class to track the relationship between the indices of different *Atoms*-like objects. The *IndexMapper* class records the mapping between all of the indices of all of the *Zeolite* objects derived from a parent structure and the changes in indices associated with each structural manipulation. Accompanying this tracking is an improved API for the structural manipulation of zeolite structures.

To gauge this improvement, it is necessary to develop criteria to assess the API usability. Human computer interaction researchers have studied how programmers interact with APIs, noting that “programmers are users too”[64]. In this field many different metrics have been developed to assess usability [62], [65], [66]. The most comprehensive review of this heterogenous research field is provided by Mosqueira-Rey et al [62]. In their paper, they distill the variety of metrics used to quantify usability into six categories: knowability, operability, efficiency, robustness,

safety, and subjective satisfaction. Descriptions of these categories, and subcategories are provided in the method section.

Here, we examine how well MAZE meets API usability criteria by completing several complex computational zeolite workflows. The workflows range from simple tasks like identifying the unique T sites to optimizing a collection of zeolite structural derivatives with VASP all while preserving the relationship between atoms. For the final optimization step, the workflow management tool Fireworks [67] was utilized, which is specifically designed for high throughput simulations. Along with introducing the MAZE package and its capabilities, this paper aims to demonstrate how custom software can be built on top of ASE to improve specialized chemical simulation workflows.

3.6 Assessing the Usability of the MAZE Package

3.6.1 Design principles

The overarching goals of the MAZE project are to ease the structural manipulations commonly encountered in zeolite workflows, while maintaining the functionality of ASE. This goal was achieved by creating design goals that guided the development of the MAZE software package.

The following design goals were developed:

1. Allow the identity and relationships between atoms to easily be determined
2. Make the end-to-end zeolite workflows straightforward

- Maintain complete compatibility with ASE's extensive functionality

To achieve these design goals, the MAZE package built upon ASE's features surrounding generating structures and reading and writing structures.

3.6.2 API Usability Assessment

The API usability assessment criteria assembled by Mosqueira-Rey et al. [62] was utilized for a comparison between the MAZE API and the ASE API for zeolite structural manipulations. The six distinct usability criteria presented in their article are shown in Table 4 along with a corresponding description.

Table 4: Usability criteria for API design

Criteria Name	Criteria Description
Knowability	How well the user can learn, understand and know how to use the API
Operability	How well the user can use the API to perform different operations
Efficiency	The computational time and number of steps required to perform an operation
Robustness	The ability of the system to withstand misuse and incorrect usage
Safety	The capacity to avoid damaging the system or external systems
Subjective Satisfaction	The joy which users derive from using the system

For simplicity only the knowability, operability and efficiency criteria were used to assess the MAZE API. Computational efficiency was not a critical factor in the MAZE project, and the steps required to perform a certain operation overlap with the operability and knowability criteria. Safety was also not directly relevant to the API design since the MAZE package is intended to be used by a single user on their own personal machine, with a memory-safe

programming language. We believe that the subjective satisfaction of using MAZE is high, yet being a subjective criterion that required human subjects to assess, it was excluded from further discussion.

The relevant sub-criteria of each major criteria were chosen to further refine the evaluation (Table 5). Irrelevant sub-criteria (e.g., third party abuse), were not included in the table.

Table 5: Criteria and sub-criteria used to assess the MAZE API's usability

Criteria	Sub-Criteria	Sub-Criteria Description
Knowability	Clarity	Understandability of existing code
Knowability	Consistency	Consistent function signatures and usage across package
Knowability	Memorability	Easy to remember functionality due to familiar API design
Knowability	Helpfulness	Complete documentation, useful docstrings and informative error messages
Operability	Completeness	Can perform all required operations.
Operability	Precision	Can perform specific operations without unintended side-effects
Robustness	Internal Error	Few bugs in the code, and bugs and errors are handled properly
Robustness	Improper use	Able to withstand improper use through proper error handling

3.6.3 MAZE Software Structure

A complete, detailed description of the code functionality and design decisions is presented in the online documentation and supplementary material. An overview of the key design decisions and code structure are presented here.

The MAZE project aims to include all of the functionality of the base ASE package while including additional functionality related to the tracking of atoms. This was incorporated by using inheritance. A zeolite is a group of atoms, so it is appropriate to create a *Zeolite* class that

inherits from ASE's *Atoms* class. The *Zeolite* class represents a zeolite and includes additional methods and properties for identifying the unique crystallographic sites. Polymorphism ensures that the *Zeolite* class has all of the attributes and methods of the parent *Atoms* class and thus all of ASE's methods and classes also work well with it.

The additional functionality of the *Zeolite* class is divided between two classes, the parent *PerfectZeolite* class and its subclass *Zeolite*. The *PerfectZeolite* class includes the functionality for building from a labeled CIF file and preserving the site labels. It also includes additional attributes for storing a name, GUID, index mapper, additions and parent zeotype (the zeolite from which the current structure is derived). The methods included in the *PerfectZeolite* are all of those related to site identification, and serialization. In a group of zeolites there can be only one perfect zeolite, from which all the derivatives are made. A simplified unified modeling language (UML) class diagram for the *Zeolite* object is presented in figure 14.

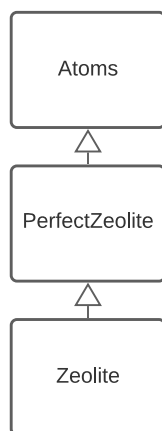


Figure 14: Simplified unified modeling language (UML) class diagram for the *Zeolite* object. Inheritance relationships are denoted by an open arrow. The *Zeolite* class inherits from the *PerfectZeolite* class, which inherits from the *Atoms* class

Users of the MAZE package will interact primarily with *Zeolite* objects. The main additional features of the *Zeolite* class versus the *PerfectZeolite* class are related to atom manipulation, such as adding atoms, deleting atoms, extracting clusters and capping clusters. By dividing the

functionality between two classes, the attributes that make a *Zeolite* and those involved in structural manipulation can be separated, greatly simplifying the underlying code. The underpinning of the *Zeolite* functionality is an internal *IndexMapper* object, which tracks the relationship between the indices of the atoms in the zeolites derived from the same parent structure.

3.6.4 The Index Mapper Class

The instances of the *IndexMapper* class are responsible for tracking the relationship between atom indices. A reference to an *IndexMapper* object is an attribute of each *Zeolite* class and related *Zeolites* share the same *IndexMapper*. The *IndexMapper* does not directly encounter *Atoms* objects, only working with their indices. The core data structure of the *IndexMapper* is the `main_index`, which consists of a collection of nested dictionaries. The key of the outer dictionary is the unique id of each row of atoms in the object (Figure 15). The inner dictionary consists of each zeolites name attribute followed by the index of an *Atom* or a *None* object.

```
{0: {"parent": 0, "Zeolite_1": 0, "Cluster_3": None, "Open Defect_5": 0},
 1: {"parent": 1, "Zeolite_1": 1, "Cluster_3": None, "Open Defect_5": 1},
 2: {"parent": 2, "Zeolite_1": 2, "Cluster_3": 0, "Open Defect_5": None},
 3: {"parent": 3, "Zeolite_1": 3, "Cluster_3": None, "Open Defect_5": 0},
 4: {"parent": 4, "Zeolite_1": 4, "Cluster_3": None, "Open Defect_5": 0},
 # ...
188: {"parent": 188, "Zeolite_1": 188, "Cluster_3": None, "Open Defect_5": 167},
189: {"parent": 189, "Zeolite_1": 189, "Cluster_3": None, "Open Defect_5": 168},
190: {"parent": 190, "Zeolite_1": 190, "Cluster_3": None, "Open Defect_5": 169},
191: {"parent": 191, "Zeolite_1": 191, "Cluster_3": None, "Open Defect_5": 170}}
```

Figure 15: Dictionary representation of the main index mapper for a collection of related *Zeolites*. The keys of the outer dictionary represent the unique IDs, where the inner dictionaries map the relationship between indices for the same shared atom across different *Atoms*-like objects.

The index mapper can also be represented in table form, to more naturally represent the *Atoms* relationships to each other (Table 6).

Table 6: Representation of the main_index of an *IndexMapper* object. The column headers denote either the unique_id or the name of the zeolite. The parent column corresponds to the indices of the *PerfectZeolite* from which the other zeolite structures were derived. Each row of the table shows the relationship between the different indices across different zeolites.

unique_id	parent	Zeolite_1	Cluster_3	Open Defect_5
0	0	0	NaN	0
1	1	1	NaN	1
2	2	2	0	NaN
3	3	3	NaN	2
4	4	4	NaN	3
...
188	188	188	NaN	167
189	189	189	NaN	168
190	190	190	NaN	169
191	191	191	NaN	170

The main_index in the *IndexMapper* object records the relationship indices of different *Atoms* objects. The unique_id assigns a unique identifier (ID) to each atom. This ID does not depend on the atom species and if an atoms type is changed from silicon to tin, for example, the ID remains unchanged. The row shows the relationship between the indices across different *Atoms* objects. For example, in row four, the ID equals 3 and the indices of the parent, *Zeolite_1* and *Open Defect_5* are all equal to 3. The equivalent atom index in *Cluster_3*, which consists of an extracted cluster from *Zeolite_1* is 0. *Cluster_3* consists of 21 atoms, and the *Open Defect_5* object consists of all of the atoms in *Zeolite_1* with the exception of the atoms in *Cluster_3*. Thus, *Open Defect_5* final indices are offset by 21 as can be seen in the final rows of the table.

The `main_index` is automatically updated when each atom manipulation operation is performed and does not require additional intervention from the user. The index mapper class can be used to directly map between two related zeolites with the `get_index` function, yet its core benefit comes about by enabling the structural manipulation functions such as `cap_atoms` and `integrate`.

Complimenting this index mapper are the `add_atoms` and `delete_atoms` methods in the *Zeolite* class, which return a copy of the original *Zeolite* object with the applied modifications and append a new column to the *IndexMapper*'s main index. This new column contains the indices of the newly created *Zeolite* object and each row indicates the relationship between the atoms in other zeolites. If a *Zeolite* object is deleted, then the deconstructor will remove its corresponding entry from the *IndexMapper*, preventing the main index from being cluttered with deleted *Zeolite* object indices.

3.7 Results and Discussion

3.7.1 Results and Discussion Introduction

To demonstrate the capability of the MAZE code and assess its API four distinct tasks were performed. These tasks include building a *Zeolite* object from a labeled CIF file, adding and removing atoms, and a complete workflow involving removing a cluster, changing some of its atoms and reinserting it back into the original zeolite. Finally, a complete optimization workflow involving Fireworks and the VASP calculator was demonstrated, which shows MAZE's capacity to facilitate high-throughput computational workflows.

3.7.2 Building a Zeolite from a CIF File

T-sites are Silicon, Aluminum or metal atoms, which have a tetrahedral shape. *Zeolites* often contain multiple distinct T-sites, each of which has unique chemistries. To screen *Zeolites* for certain properties, the chemistries at each T site need to be assessed.

Experiments of this type start with downloading a CIF file from the IZA database, placing the file in the project folder and reading the CIF file into an *Atoms* object with the `ase.io.read` function. One challenge with this approach is that CIF files downloaded from the IZA database contain extra information about the identity of unique atoms, which is not preserved when the CIF file is loaded with ASE's read function. Thus, various "hacks" are needed to align the *Atoms* object built from the CIF file with their labels. One hack is changing a unique site from a silicon to an unused atom such as xenon, and when the *Atoms* object is loaded reverting it back to a silicon, but noting the indices. This manual tagging mechanism is slow, opaque because the code is no longer self-documenting, and error prone, since it involves manually editing a critical data file.

The MAZE package significantly improves this process by introducing the `make` method. The `make` method takes a zeolite IZA code as input, looks for the corresponding CIF file, and if it is not found attempts to download the zeolite CIF file from the IZA database. After locating or downloading the correct CIF file, the `make` function then builds a *Zeolite* from a IZA CIF file, and stores the mapping between the indices and their identities in two of the *Zeolite* objects internal dictionaries. The identities of the sites can then be determined by using the `get_site_type` method or by accessing the dictionaries directly (Figure 16).

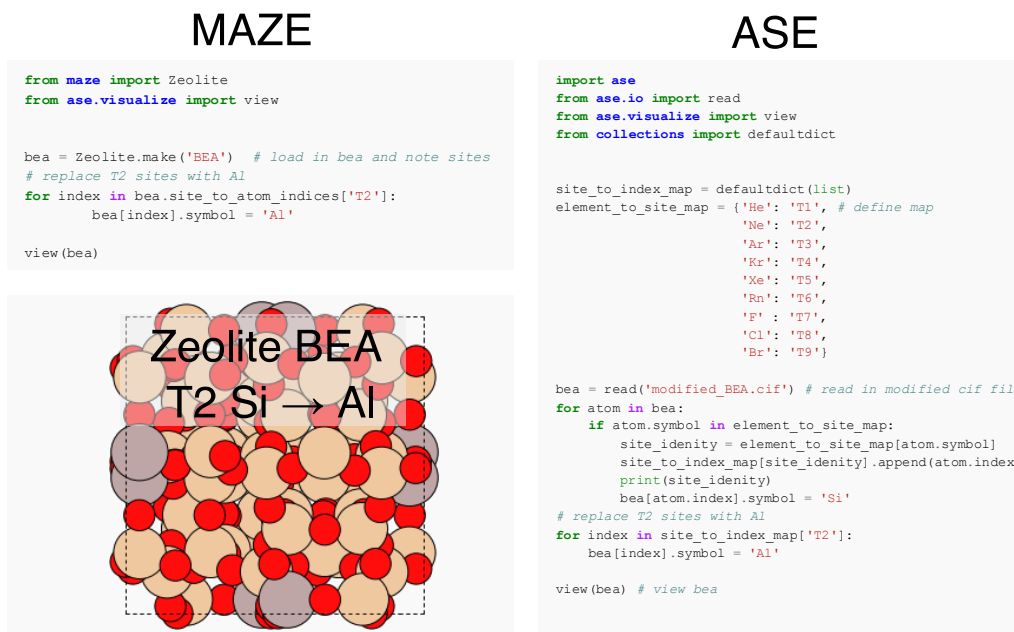


Figure 16: Comparison between MAZE and ASE code for generating a BEA zeolite structure with the Silicon T2 sites replaced by aluminum atoms. The MAZE code uses the built-in make function to read the unmodified CIF file and store the mapping in the site_to_atom_indices dictionary. The longer ASE code requires a modified CIF file as input, and the element mapping to be manually defined. Both codes generate and visualize the same BEA T2 Si→Al structure.

Figure 16 shows that the MAZE package is simpler to use than the base ASE package for this specified workflow. To objectively assess the MAZE package usability for constructing *Atoms*-like objects from labeled CIF files the usability criteria described in the methods section were applied to this specific use case. The results of these applied criteria are shown in table 7.

Table 7: Assessment of usability for both the ASE and MAZE package for loading a labeled CIF file

Usability Criteria	ASE Description	MAZE Description
Knowability	Requires manually implementing a complex, time consuming workflow	Only requires calling a single function
Operability	For each labeled site the CIF file must be modified	Labeling performed automatically when reading a standard CIF file
Robustness	Prone to silent errors due to improper CIF file modification	Labeling automatic, preventing human error

In the knowability category, the MAZE package outperforms the ASE read function, since only a single, easy to remember function is needed to construct a labeled *Zeolite* from a CIF file. When using the ASE package an entire, convoluted workflow must be performed. This workflow is prone to human error, since it involves manually downloading and modifying the CIF file, thus the robustness is also worse. The operability of the ASE package is also limited since tagging all the sites requires more effort than tagging a single site. Overall, the MAZE package offers a significant improvement over the ASE package and meets the key categories of API usability.

3.7.3 Structural Manipulations

The *Atoms* class' structural manipulation features allow atoms to be added and removed from the collection and the properties of individual atoms to be altered. The API by which these manipulations are performed is inspired by Python's list manipulation methods. Although familiar to Python users, these manipulations are not self-consistent as some have side effects (e.g. the pop method) while others are side effect free such as the `__add__` method. In zeolite workflows it is common for many derivatives of a single parent zeolite to be generated, and this is complicated by methods with side-effects, due to the need for explicit copying prior to each modification.

In alignment with the goal of the MAZE project, new methods for atomic manipulation were designed, which do not mutate the underlying object, and instead return a copy with the applied modifications. These methods (`add_atoms` and `delete_atoms`) simplify the computational workflows and also allow for method chaining improving code readability. A list of the available methods for the ASE *Atoms* object and the MAZE *Zeolite* object are shown in Table 8 below.

Table 8: ASE *Atoms* object and MAZE *Zeolite* object's structure manipulation methods

Method Name	Classes with Method	Modifies Object
<code>__add__</code> (overloads +)	<i>Atoms, PerfectZeolite, Zeolite</i>	No
<code>__iadd__</code> (overloads +=)	<i>Atoms, PerfectZeolite, Zeolite</i>	Yes
<code>extend</code>	<i>Atoms, PerfectZeolite, Zeolite</i>	Yes
<code>append</code>	<i>Atoms, PerfectZeolite, Zeolite</i>	Yes
<code>len</code>	<i>Atoms, PerfectZeolite, Zeolite</i>	No
<code>__delete__</code> overloads del	<i>Atoms, PerfectZeolite, Zeolite</i>	Yes
<code>pop</code>	<i>Atoms, PerfectZeolite, Zeolite</i>	Yes
<code>add_atoms</code>	<i>Zeolite</i>	No
<code>delete_atoms</code>	<i>Zeolite</i>	No
<code>cap_atoms</code>	<i>Zeolite</i>	No
<code>Integrate_zeotype</code>	<i>Zeolite</i>	No
<code>get_cluster</code>	<i>Zeolite</i>	No
<code>apply</code>	<i>Zeolite</i>	No

The structural manipulation features of the *Zeolite* class include those of the *Atoms* class and additional methods for structural manipulation built around the `add_atoms` and `delete_atoms` methods. For compatibility with the plethora of functions in ASE all of the *Atom*'s classes methods must be preserved. Although their use is not recommended, the list manipulation functions in the ASE atoms class are still compatible with MAZE. In comparing the APIs, the new methods in *Zeolite* are compared to the existing methods in *Atoms*. The usability of the structural manipulation features can be assessed based on the criteria described in the methods

section. A table summarizing the comparison is shown in Table 9, with a “winner” given for each criterion along with a justification for the assignment.

Table 9: Usability of structural manipulation methods for the *Atoms* objects and *Zeolite* Objects

Criteria	Sub-Criteria	Winner (MAZE, ASE, Tied)	Justification
Knowability	Clarity	Tied	Both ASE and MAZE methods clearly described the modification
Knowability	Consistency	MAZE	All MAZE structural manipulation methods return new Zeolites
Knowability	Memorability	ASE	ASEs functions are consistent with Python’s list manipulation methods and thus are more memorable
Knowability	Helpfulness	Tied	Both ASE and MAZE have complete documentation
Operability	Completeness	MAZE	MAZE can perform more advanced structural manipulations such as capping atoms and integrating clusters
Operability	Precision	MAZE	Precise structural manipulations can be performed such as removing specific T sites
Robustness	Internal Error	MAZE	MAZE manipulation methods’ side-effects are confined to the <i>IndexMapper</i> object reducing errors
Robustness	Improper use	MAZE	The side effects of the ASE methods can produce hidden errors that are challenging to detect

The usability assessment shows that the MAZE *Zeolite* methods outperforms the ASE in all categories except for the memorability category. The *Atoms* manipulation features have high memorability since they are the same as the list manipulation methods found in Python’s base package, yet the inconsistency in return type among the operations makes performing certain operations challenging. The MAZE structural manipulation features are consistent, increasing the knowability, allow for chained, structural manipulations, increasing the operability and have increased efficiency since the manipulations can be confined to a single line and are more robust due to the consistent interface and side-effect free methods.

3.7.4 Cluster Extraction, Atom Capping and Integration

The power of these additional structural manipulation features can be demonstrated by performing a complex workflow. A typical zeolite unit cell contains over one-hundred atoms, but the areas of chemical interest are frequently confined to the atoms adjacent to a single T-site. To reduce the computational expense of quantum chemical calculations, the calculations are typically performed on a smaller subset of atoms adjacent to the active sites of interest. This subset of atoms is referred to as a cluster [68]. To ensure convergence, capping atoms are added to the terminal cluster atoms, guaranteeing that they have an octet. The optimal position for the capping atoms is based on the parent zeolite's structure. After the capped cluster's structure has been optimized, the cluster can be integrated back into the initial zeolite with the integration method.

This workflow is extraordinarily difficult to perform with the ASE base package due to the challenge associated with tracking the relationship between atom indices. The *Zeolite* class's built-in index mapper ensures that the relationship between atoms can easily be determined and forms the basis for the simple functions that perform this workflow. In Figure 17 a pictorial representation of stages in the workflow is shown along with the methods needed to perform the transformation from one stage to the next.

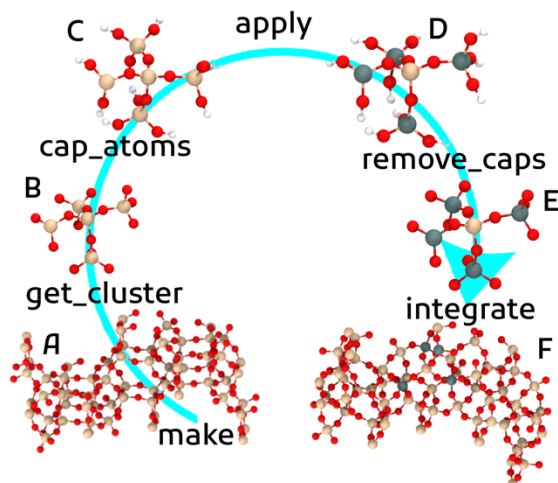


Figure 17: Workflow for cluster integration. A blue arrow shows the workflow direction, starting with a BEA zeolite constructed using the make method. The functions or operations required to transfer from one structure to another are shown between the structures on the blue line.

The overall workflow has six distinct structures bridged by functions which take the previous structure as an input and output the new structure. The cluster structures (B, C, D, E) have different indices than the BEA frameworks (A, F), yet the indices can easily be mapped to each other using the built-in *IndexMapper*'s `get_index` method. Since the functions do not alter the *Zeolite* to which they are applied, and instead return a new zeolite object, they can be chained together. The chained methods required to transform structure A into structure F is shown in Figure 1.

```

from maze import Zeolite
def change_atoms(atoms):
    for atom in atoms:
        if atom.symbol == 'Si' and atom.tag != 154:
            atoms[atom.index].symbol = 'Sn'

bea = Zeolite.make('BEA')
cluster = bea.get_cluster(154)[0].cap_atoms().apply(change_atoms).remove_caps()
bea_sn = bea.integrate(cluster)

```

Figure 18: Code required to generate structure F from structure A

The code presented in figure 1 demonstrates how a complex workflow can be achieved with the chaining of several functions together. This simplicity allows for knowability of the operations,

precise and complete operability, and robustness due to high readability. The scrambling of the indices with the cluster extraction does not allow for consistent code using the base ASE package. Instead, the indices of each atom must be matched manually at each stage of the process. Thus, the MAZE package interface has increased the knowability, operability and robustness compared to the cumbersome manual workflow required when using the base ASE package

3.7.5 Creation and Optimization of a Database of Zeolite Structures

MAZE's structural manipulation enhancements can be utilized to construct thousands of *Zeolite* derivatives in a systematic way. These structures can then be optimized with the aid of the workflow management tool Fireworks [67] and a custom-build python package VASP-FW, which combines ASE's VASP interface with Fireworks'. To demonstrate this capacity, a database of open framework copper containing zeolites was created with the MAZE package. The Python class utilized for this process was written by the current PhD student Jiawei Guo, whose research focuses on copper containing zeolite catalysts. As of the writing of this thesis, optimization has not been complete, yet the pipeline and workflow has been constructed and the individual components tested. A diagram of this automatic data pipeline for structural optimization is shown below (Figure 19).

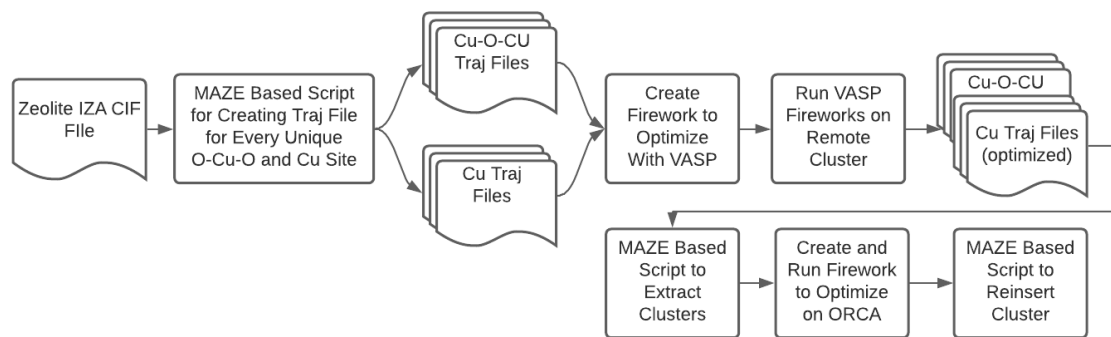


Figure 19: Data pipeline for the construction of optimized zeolite derivatives. The MAZE package allows for key optimization steps to be automated through a single python script.

The MAZE package facilitates these structural manipulations by removing the need to manually match indices between derived structures. This allows for the automatic creation of large-scale databases of zeolite derivatives, producing large datasets amenable for complex analysis.

In the previous examples all structural manipulations took place on *Zeolite* objects loaded into memory. Quantum chemical optimizations typically take place on remote servers separate from the structural manipulation code. The computations are also expensive, thus preserving the resulting optimized structure on non-volatile memory is essential. The MAZE package incorporates read-write capabilities that allow groups of related zeolites to be saved in a single file while preserving the mapping between the zeolites. The read-write capabilities also allow for a properly tagged structure to be reincorporated back into a group of zeolites by registering it with an existing index mapper. This read-write capacity allows for the development of complex, distributed workflows.

The Vienna Ab initio Simulation Package (VASP) was utilized with ASE and MAZE for structural optimization [59]. The open-source workflow management software Fireworks [67] was utilized to manage jobs, while a remote AWS PostgreSQL database was used to store the

Atom object data. To ensure code modularity, this VASP fireworks workflow was built into as separate python package (vasp-fw), which facilitated the workflow needed for optimizing trajectory files. An overview of the data system for the optimization and an example workflow is shown in Figure 20 below.

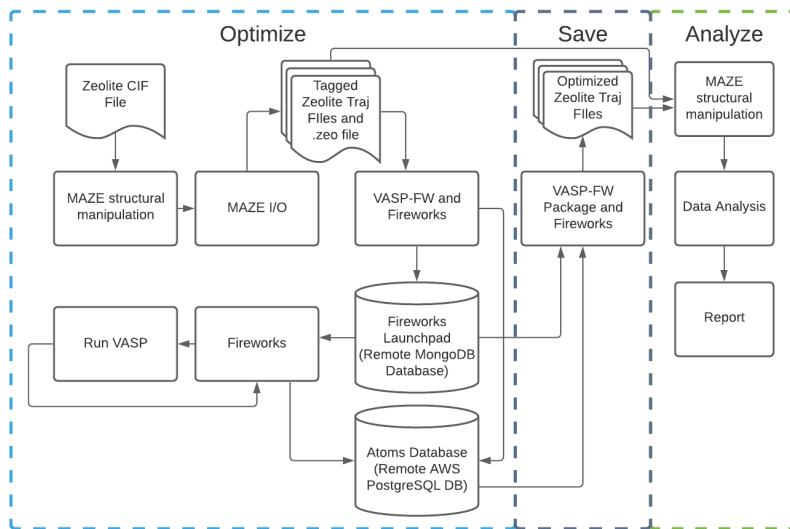


Figure 20: Workflow diagram for the optimization of a collection of trajectory files. The workflow is divided into optimization (blue), save (gray) and analyze (green) stages.

The diagram (Figure 20) shows the steps for optimizing trajectory with Firework and ASE’s VASP calculator. This process will be repeated for each trajectory file in the zeolite database shown in Figure 19, allowing for a comprehensive study of open framework, copper containing zeolites.

3.8 Conclusions

The improved API of the MAZE package was presented here by demonstrating how to perform representative tasks. Several other features of the MAZE package include database integration

and adsorbate additions. A complete description can be found in the documentation, which is referenced in the supplementary material.

MAZE's improved API builds on-top of the Atomic Simulation Environment. This new interface facilitates computational zeolite calculations by greatly simplifying the steps needed to perform common zeolite workflows. The API usability was assessed with the criteria knowability, operability and robustness and by demonstrating example workflows. Computational experiments are less labor intensive than wet lab experiments, but lack of optimal APIs for scientific software and complex workflows can incur a significant time commitment from researchers to setup and run. By creating custom software tailored to the specific task, research can be simplified and larger scale experiments can be conducted.

Chapter 4

Conclusions

4.1 Project Summaries

In this work, two separate projects were undertaken. The first project focused on constructing a data pipeline to pre-process SERS spectra. A novel component of this pipeline was the utilization of a supervised machine learning algorithm to identify negative spectra. The second project, the development of the MAZE Python package, allowed for common structural manipulations of simulated zeolite structures to be simplified. As demonstrated, this allowed for the construction of libraries of zeolite derivatives and their optimization through integration with VASP and Fireworks. Both of these projects share the common characteristic of automating a key component of an experimental workflow, reducing the labor required to conduct large scale experiments.

4.2 Future work

In addition to simplifying existing workflows, this work sets the groundwork for future autonomous experiments. Autonomous experiments involve using an agent, rather than a

researcher, to choose which sample to collect and a controller to automatically collect the chosen sample [69] (Figure 21).

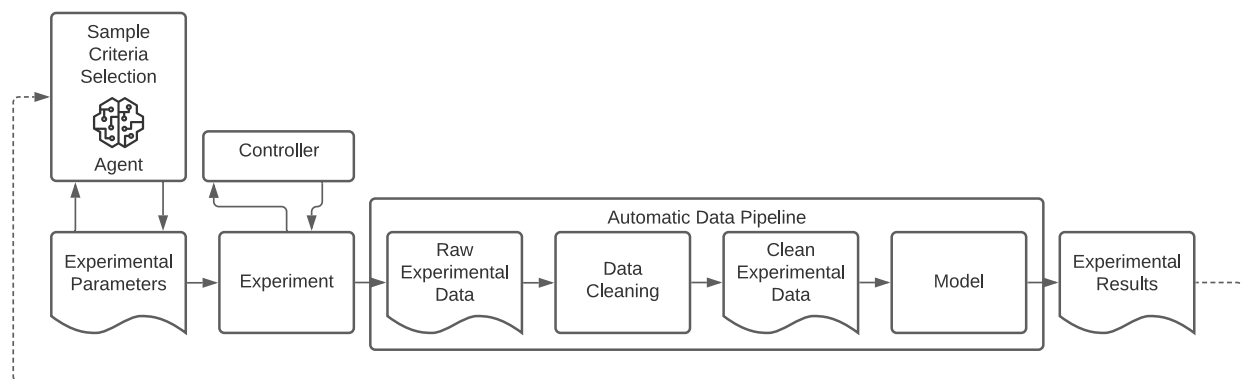


Figure 21: A general diagram for an autonomous experiment. The sample criteria are selected by an agent and are collected by the experimental apparatus and controller. An automatic data pipeline processes the raw data output by the experiment and feeds the data back to the agent.

Autonomous experiments could extend the work described in this thesis. In the case of zeolite studies, the automated workflow could be combined with an autonomous agent that explores the vast zeolite space in search of structures with high selectivity for CO₂ over water. In the SERS situation an autonomous experiment would involve automatically sampling spots on the sample that produced positive spectra at higher rate than those that produced negative signals, freeing the researcher from overseeing the collection process. The automation of the SERS sampling is close to fruition as it is straightforward to integrate Python based ML algorithm with the SERS spectrometer instrument using LabVIEW.

The agent utilized for experiment selection be simple, such as a grid search algorithm [69] or employ more sophisticated reinforced learning algorithms [69], [70]. Reinforce learning algorithms are responsible for the superhuman feats achieved by AI in games most notably in Go[71] and StarCraft II[70] and have been applied to optimize chemical reactions[72], and beamline data collection[69]. To harness these advances an automatic data pipeline must be

created. Thus, the deployment of data pipelines can significantly reduce the labor required for large scale experiments.

5.1 Bibliography

- [1] A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. Bjerre Jensen, J. Kermodé, J. R. Kitchin, E. Leonhard Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. Bergmann Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, “The atomic simulation environment—a Python library for working with atoms,” *J. Phys.: Condens. Matter*, vol. 29, no. 27, p. 273002, Jul. 2017, doi: 10.1088/1361-648X/aa680e.
- [2] “PyPI Download Stats for MAZE-sim.” <https://pypistats.org/packages/maze-sim> (accessed May 05, 2021).
- [3] O. T. Unke, S. Chmiela, H. E. Sauceda, M. Gastegger, I. Poltavsky, K. T. Schütt, A. Tkatchenko, and K.-R. Müller, “Machine Learning Force Fields,” *Chem. Rev.*, p. acs.chemrev.0c01111, Mar. 2021, doi: 10.1021/acs.chemrev.0c01111.
- [4] C. Chen, Y. Zuo, W. Ye, X. Li, Z. Deng, and S. P. Ong, “A Critical Review of Machine Learning of Energy Materials,” *Adv. Energy Mater.*, vol. 10, no. 8, p. 1903242, Feb. 2020, doi: 10.1002/aenm.201903242.
- [5] H. Atwal, *Practical DataOps: Delivering Agile Data Science at Scale*. Berkeley, CA: Apress L.P., 2020. Accessed: May 03, 2021. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=2327464>
- [6] M. Boumans and S. Leonelli, “From Dirty Data to Tidy Facts: Clustering Practices in Plant Phenomics and Business Cycle Analysis,” in *Data Journeys in the Sciences*, S. Leonelli and N. Tempini, Eds. Cham: Springer International Publishing, 2020, pp. 79–101. doi: 10.1007/978-3-030-37177-7_5.
- [7] H. Wickham, “Tidy Data,” *Journal of Statistical Software, Articles*, vol. 59, no. 10, pp. 1–23, 2014, doi: 10.18637/jss.v059.i10.
- [8] “‘PART III’ [Chapter 23] in ‘Debates in the Digital Humanities 2019’ on Manifold,” *Debates in the Digital Humanities*. <https://dhdebates.gc.cuny.edu/read/untitled-f2acf72c-a469-49d8-be35-67f9ac1e3a60/section/07154de9-4903-428e-9c61-7a92a6f22e51> (accessed Apr. 30, 2021).
- [9] J. Furbush, “Data engineering: A quick and simple definition,” *O’Reilly Media*, Jul. 16, 2018. <https://www.oreilly.com/content/data-engineering-a-quick-and-simple-definition/> (accessed Apr. 30, 2021).
- [10] J. Langer, D. Jimenez de Aberasturi, J. Aizpurua, R. A. Alvarez-Puebla, B. Auguié, J. J. Baumberg, G. C. Bazan, S. E. J. Bell, A. Boisen, A. G. Brolo, J. Choo, D. Cialla-May, V. Deckert, L. Fabris, K. Faulds, F. J. García de Abajo, R. Goodacre, D. Graham, A. J. Haes, C. L. Haynes, C. Huck, T. Itoh, M. Käll, J. Kneipp, N. A. Kotov, H. Kuang, E. C. Le Ru, H. K. Lee, J.-F. Li, X. Y. Ling, S. A. Maier, T. Mayerhöfer, M. Moskovits, K. Murakoshi, J.-

- M. Nam, S. Nie, Y. Ozaki, I. Pastoriza-Santos, J. Perez-Juste, J. Popp, A. Pucci, S. Reich, B. Ren, G. C. Schatz, T. Shegai, S. Schlücker, L.-L. Tay, K. G. Thomas, Z.-Q. Tian, R. P. Van Duyne, T. Vo-Dinh, Y. Wang, K. A. Willets, C. Xu, H. Xu, Y. Xu, Y. S. Yamamoto, B. Zhao, and L. M. Liz-Marzán, “Present and Future of Surface-Enhanced Raman Scattering,” *ACS Nano*, vol. 14, no. 1, pp. 28–117, Jan. 2020, doi: 10.1021/acsnano.9b04224.
- [11] F. W. L. Esmonde-White, M. V. Schulmerich, K. A. Esmonde-White, and M. D. Morris, “Automated Raman spectral preprocessing of bone and other musculoskeletal tissues,” San Jose, CA, Feb. 2009, p. 716605. doi: 10.1117/12.809436.
- [12] G. Lopez-Reyes and F. Rull Pérez, “A method for the automated Raman spectra acquisition: Automated Raman spectra acquisition,” *J. Raman Spectrosc.*, vol. 48, no. 11, pp. 1654–1664, Nov. 2017, doi: 10.1002/jrs.5185.
- [13] H. G. Schulze, S. Rangan, J. M. Piret, M. W. Blades, and R. F. B. Turner, “Developing Fully Automated Quality Control Methods for Preprocessing Raman Spectra of Biomedical and Biological Samples,” *Appl Spectrosc.*, vol. 72, no. 9, pp. 1322–1340, Sep. 2018, doi: 10.1177/0003702818778031.
- [14] W. Hu, S. Ye, Y. Zhang, T. Li, G. Zhang, Y. Luo, S. Mukamel, and J. Jiang, “Machine Learning Protocol for Surface-Enhanced Raman Spectroscopy,” *J. Phys. Chem. Lett.*, vol. 10, no. 20, pp. 6026–6031, Oct. 2019, doi: 10.1021/acs.jpcllett.9b02517.
- [15] X. Fan, W. Ming, H. Zeng, Z. Zhang, and H. Lu, “Deep learning-based component identification for the Raman spectra of mixtures,” *Analyst*, vol. 144, no. 5, pp. 1789–1798, 2019, doi: 10.1039/C8AN02212G.
- [16] S. D. Krauß, R. Roy, H. K. Yosef, T. Lehtonen, S. F. El-Mashtoly, K. Gerwert, and A. Mosig, “Hierarchical deep convolutional neural networks combine spectral and spatial information for highly accurate Raman-microscopy-based cytopathology,” *J. Biophotonics*, vol. 11, no. 10, p. e201800022, Oct. 2018, doi: 10.1002/jbio.201800022.
- [17] A. A. Moawad, A. Silge, T. Bocklitz, K. Fischer, P. Rösch, U. Roesler, M. C. Elschner, J. Popp, and H. Neubauer, “A Machine Learning-Based Raman Spectroscopic Assay for the Identification of *Burkholderia mallei* and Related Species,” *Molecules*, vol. 24, no. 24, p. 4516, Dec. 2019, doi: 10.3390/molecules24244516.
- [18] R. M. Jarvis and R. Goodacre, “Discrimination of Bacteria Using Surface-Enhanced Raman Spectroscopy,” *Anal. Chem.*, vol. 76, no. 1, pp. 40–47, Jan. 2004, doi: 10.1021/ac034689c.
- [19] B. Deng, X. Luo, M. Zhang, L. Ye, and Y. Chen, “Quantitative detection of acyclovir by surface enhanced Raman spectroscopy using a portable Raman spectrometer coupled with multivariate data analysis,” *Colloids and Surfaces B: Biointerfaces*, vol. 173, pp. 286–294, Jan. 2019, doi: 10.1016/j.colsurfb.2018.09.058.
- [20] S.-X. Li, Q.-Y. Zeng, L.-F. Li, Y.-J. Zhang, M.-M. Wan, Z.-M. Liu, H.-L. Xiong, Z.-Y. Guo, and S.-H. Liu, “Study of support vector machine and serum surface-enhanced Raman spectroscopy for noninvasive esophageal cancer detection,” *J. Biomed. Opt.*, vol. 18, no. 2, p. 027008, Feb. 2013, doi: 10.1117/1.JBO.18.2.027008.
- [21] Y. Zhang, X. Ye, G. Xu, X. Jin, M. Luan, J. Lou, L. Wang, C. Huang, and J. Ye, “Identification and distinction of non-small-cell lung cancer cells by intracellular SERS nanoprobe,” *RSC Adv.*, vol. 6, no. 7, pp. 5401–5407, 2016, doi: 10.1039/C5RA21758J.
- [22] X. Xu, K. Kim, H. Li, and D. L. Fan, “Ordered Arrays of Raman Nanosensors for Ultrasensitive and Location Predictable Biochemical Detection,” *Adv. Mater.*, vol. 24, no. 40, pp. 5457–5463, Oct. 2012, doi: 10.1002/adma.201201820.

- [23] W. Lee, A. T. M. Lenferink, C. Otto, and H. L. Offerhaus, “Classifying Raman spectra of extracellular vesicles based on convolutional neural networks for prostate cancer detection,” *J Raman Spectrosc*, vol. 51, no. 2, pp. 293–300, Feb. 2020, doi: 10.1002/jrs.5770.
- [24] J. Park, M. Hwang, B. Choi, H. Jeong, J. Jung, H. K. Kim, S. Hong, J. Park, and Y. Choi, “Exosome Classification by Pattern Analysis of Surface-Enhanced Raman Spectroscopy Data for Lung Cancer Diagnosis,” *Anal. Chem.*, vol. 89, no. 12, pp. 6695–6701, Jun. 2017, doi: 10.1021/acs.analchem.7b00911.
- [25] J. Carmicheal, C. Hayashi, X. Huang, L. Liu, Y. Lu, A. Krasnoslobodtsev, A. Lushnikov, P. G. Kshirsagar, A. Patel, M. Jain, Y. L. Lyubchenko, Y. Lu, S. K. Batra, and S. Kaur, “Label-free characterization of exosome via surface enhanced Raman spectroscopy for the early detection of pancreatic cancer,” *Nanomedicine: Nanotechnology, Biology and Medicine*, vol. 16, pp. 88–96, Feb. 2019, doi: 10.1016/j.nano.2018.11.008.
- [26] N. Banaei, J. Moshfegh, A. Mohseni-Kabir, J. M. Houghton, Y. Sun, and B. Kim, “Machine learning algorithms enhance the specificity of cancer biomarker detection using SERS-based immunoassays in microfluidic chips,” *RSC Adv.*, vol. 9, no. 4, pp. 1859–1868, 2019, doi: 10.1039/C8RA08930B.
- [27] E. Guevara, J. C. Torres-Galván, M. G. Ramírez-Eliás, C. Luevano-Contreras, and F. J. González, “Use of Raman spectroscopy to screen diabetes mellitus with machine learning tools,” *Biomed. Opt. Express*, vol. 9, no. 10, p. 4998, Oct. 2018, doi: 10.1364/BOE.9.004998.
- [28] H. Shi, H. Wang, X. Meng, R. Chen, Y. Zhang, Y. Su, and Y. He, “Setting Up a Surface-Enhanced Raman Scattering Database for Artificial-Intelligence-Based Label-Free Discrimination of Tumor Suppressor Genes,” *Anal. Chem.*, vol. 90, no. 24, pp. 14216–14221, Dec. 2018, doi: 10.1021/acs.analchem.8b03080.
- [29] I. J. Hidi, M. Jahn, K. Weber, T. Bocklitz, M. W. Pletz, D. Cialla-May, and J. Popp, “Lab-on-a-Chip-Surface Enhanced Raman Scattering Combined with the Standard Addition Method: Toward the Quantification of Nitroxoline in Spiked Human Urine Samples,” *Anal. Chem.*, vol. 88, no. 18, pp. 9173–9180, Sep. 2016, doi: 10.1021/acs.analchem.6b02316.
- [30] O. Alharbi, Y. Xu, and R. Goodacre, “Simultaneous multiplexed quantification of caffeine and its major metabolites theobromine and paraxanthine using surface-enhanced Raman scattering,” *Anal Bioanal Chem*, vol. 407, no. 27, pp. 8253–8261, Nov. 2015, doi: 10.1007/s00216-015-9004-8.
- [31] S. M. Moosavi, A. Chidambaram, L. Talirz, M. Haranczyk, K. C. Stylianou, and B. Smit, “Capturing chemical intuition in synthesis of metal-organic frameworks,” *Nat Commun*, vol. 10, no. 1, p. 539, Dec. 2019, doi: 10.1038/s41467-019-08483-9.
- [32] D. Shen, G. Wu, and H.-I. Suk, “Deep Learning in Medical Image Analysis,” *Annu. Rev. Biomed. Eng.*, vol. 19, no. 1, pp. 221–248, Jun. 2017, doi: 10.1146/annurev-bioeng-071516-044442.
- [33] D. L. Jeanmaire and R. P. V. Duyne, “SURFACE RAMAN SPECTROELECTROCHEMISTRY,” p. 20.
- [34] D. L. Jeanmaire and R. P. Van Duyne, “Surface raman spectroelectrochemistry: Part I. Heterocyclic, aromatic, and aliphatic amines adsorbed on the anodized silver electrode,” *Journal of Electroanalytical Chemistry and Interfacial Electrochemistry*, vol. 84, no. 1, pp. 1–20, Nov. 1977, doi: 10.1016/S0022-0728(77)80224-6.

- [35] M. G. Albrecht and J. A. Creighton, “Anomalously intense Raman spectra of pyridine at a silver electrode,” *J. Am. Chem. Soc.*, vol. 99, no. 15, pp. 5215–5217, Jun. 1977, doi: 10.1021/ja00457a071.
- [36] A. M. Michaels, Jiang, and L. Brus, “Ag Nanocrystal Junctions as the Site for Surface-Enhanced Raman Scattering of Single Rhodamine 6G Molecules,” *J. Phys. Chem. B*, vol. 104, no. 50, pp. 11965–11971, Dec. 2000, doi: 10.1021/jp0025476.
- [37] B. Vincent, J. Edwards, S. Emmett, and A. Jones, “Depletion flocculation in dispersions of sterically-stabilised particles (‘soft spheres’),” *Colloids and Surfaces*, vol. 18, no. 2–4, pp. 261–281, Jun. 1986, doi: 10.1016/0166-6622(86)80317-1.
- [38] L. A. Wijenayaka, M. R. Ivanov, C. M. Cheatum, and A. J. Haes, “Improved Parametrization for Extended Derjaguin, Landau, Verwey, and Overbeek Predictions of Functionalized Gold Nanosphere Stability,” *J. Phys. Chem. C*, vol. 119, no. 18, pp. 10064–10075, May 2015, doi: 10.1021/acs.jpcc.5b00483.
- [39] S. R. Saunders, M. R. Eden, and C. B. Roberts, “Modeling the Precipitation of Polydisperse Nanoparticles Using a Total Interaction Energy Model,” *J. Phys. Chem. C*, vol. 115, no. 11, pp. 4603–4610, Mar. 2011, doi: 10.1021/jp200116a.
- [40] L. M. Almehmadi, S. M. Curley, N. A. Tokranova, S. A. Tenenbaum, and I. K. Lednev, “Surface Enhanced Raman Spectroscopy for Single Molecule Protein Detection,” *Sci Rep*, vol. 9, no. 1, p. 12356, Dec. 2019, doi: 10.1038/s41598-019-48650-y.
- [41] K. A. Bosnick, Jiang, and L. E. Brus, “Fluctuations and Local Symmetry in Single-Molecule Rhodamine 6G Raman Scattering on Silver Nanocrystal Aggregates[†],” *J. Phys. Chem. B*, vol. 106, no. 33, pp. 8096–8099, Aug. 2002, doi: 10.1021/jp0256241.
- [42] A. B. Zrimsek, N. Chiang, M. Mattei, S. Zaleski, M. O. McAnally, C. T. Chapman, A.-I. Henry, G. C. Schatz, and R. P. Van Duyne, “Single-Molecule Chemistry with Surface- and Tip-Enhanced Raman Spectroscopy,” *Chem. Rev.*, vol. 117, no. 11, pp. 7583–7613, Jun. 2017, doi: 10.1021/acs.chemrev.6b00552.
- [43] A. Szeghalmi, S. Kaminskyj, P. Rösch, J. Popp, and K. M. Gough, “Time Fluctuations and Imaging in the SERS Spectra of Fungal Hypha Grown on Nanostructured Substrates,” *J. Phys. Chem. B*, vol. 111, no. 44, pp. 12916–12924, Nov. 2007, doi: 10.1021/jp075422a.
- [44] J. Taylor, A. Huefner, L. Li, J. Wingfield, and S. Mahajan, “Nanoparticles and intracellular applications of surface-enhanced Raman spectroscopy,” *Analyst*, vol. 141, no. 17, pp. 5037–5055, 2016, doi: 10.1039/C6AN01003B.
- [45] Unidata, *Unidata, netCDF4 version 1.5.6*. Boulder, CO: UCAR/Unidata, 2021.
- [46] Z.-M. Zhang, S. Chen, and Y.-Z. Liang, “Baseline correction using adaptive iteratively reweighted penalized least squares,” *Analyst*, vol. 135, no. 5, p. 1138, 2010, doi: 10.1039/b922045c.
- [47] P. H. C. Eilers, “A Perfect Smoother,” *Anal. Chem.*, vol. 75, no. 14, pp. 3631–3636, Jul. 2003, doi: 10.1021/ac034173t.
- [48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, and D. Cournapeau, “Scikit-learn: Machine Learning in Python,” *MACHINE LEARNING IN PYTHON*, p. 6.
- [49] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, Aug. 2016, doi: 10.1145/2939672.2939785.

- [50] T. A. Lasko, J. G. Bhagwat, K. H. Zou, and L. Ohno-Machado, “The use of receiver operating characteristic curves in biomedical informatics,” *Journal of Biomedical Informatics*, vol. 38, no. 5, pp. 404–415, Oct. 2005, doi: 10.1016/j.jbi.2005.02.008.
- [51] S. M. Borstelmann, “Machine Learning Principles for Radiology Investigators,” *Academic Radiology*, vol. 27, no. 1, pp. 13–25, Jan. 2020, doi: 10.1016/j.acra.2019.07.030.
- [52] B. Smit and T. L. M. Maesen, “Towards a molecular understanding of shape selectivity,” *Nature*, vol. 451, no. 7179, pp. 671–678, Feb. 2008, doi: 10.1038/nature06552.
- [53] P. Xie, T. Pu, G. Aranovich, J. Guo, M. Donohue, A. Kulkarni, and C. Wang, “Bridging adsorption analytics and catalytic kinetics for metal-exchanged zeolites,” *Nat Catal*, vol. 4, no. 2, pp. 144–156, Feb. 2021, doi: 10.1038/s41929-020-00555-0.
- [54] N. Kosinov, C. Liu, E. J. M. Hensen, and E. A. Pidko, “Engineering of Transition Metal Catalysts Confined in Zeolites,” *Chem. Mater.*, vol. 30, no. 10, pp. 3177–3198, May 2018, doi: 10.1021/acs.chemmater.8b01311.
- [55] R. Qin, P. Liu, G. Fu, and N. Zheng, “Strategies for Stabilizing Atomically Dispersed Metal Catalysts,” *Small Methods*, vol. 2, no. 1, p. 1700286, Jan. 2018, doi: 10.1002/smt.201700286.
- [56] B. M. Weckhuysen and J. Yu, “Recent advances in zeolite chemistry and catalysis,” *Chem. Soc. Rev.*, vol. 44, no. 20, pp. 7022–7024, 2015, doi: 10.1039/C5CS90100F.
- [57] C. R. A. Catlow, V. Van Speybroeck, and R. van Santen, *Modelling and Simulation in the Science of Micro- and Meso-Porous Materials*. Saint Louis, UNITED STATES: Elsevier, 2017. [Online]. Available: <http://ebookcentral.proquest.com/lib/ucdavis/detail.action?docID=5051428>
- [58] S. Pirhadi, J. Sunseri, and D. R. Koes, “Open source molecular modeling,” *Journal of Molecular Graphics and Modelling*, vol. 69, pp. 127–143, Sep. 2016, doi: 10.1016/j.jmgm.2016.07.008.
- [59] G. Kresse and D. Joubert, “From ultrasoft pseudopotentials to the projector augmented-wave method,” *Phys. Rev. B*, vol. 59, no. 3, pp. 1758–1775, Jan. 1999, doi: 10.1103/PhysRevB.59.1758.
- [60] S. Plimpton, “Fast Parallel Algorithms for Short-Range Molecular Dynamics,” p. 42.
- [61] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, “Real-space grid implementation of the projector augmented wave method,” *Phys. Rev. B*, vol. 71, no. 3, p. 035109, Jan. 2005, doi: 10.1103/PhysRevB.71.035109.
- [62] E. Mosqueira-Rey, D. Alonso-Ríos, V. Moret-Bonillo, I. Fernández-Varela, and D. Álvarez-Estévez, “A systematic approach to API usability: Taxonomy-derived criteria and a case study,” *Information and Software Technology*, vol. 97, pp. 46–63, May 2018, doi: 10.1016/j.infsof.2017.12.010.
- [63] S. R. Bahn and K. W. Jacobsen, “An object-oriented scripting interface to a legacy electronic structure code,” *Comput. Sci. Eng.*, vol. 4, no. 3, pp. 56–66, Jun. 2002, doi: 10.1109/5992.998641.
- [64] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, “Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools,” *Computer*, vol. 49, no. 7, pp. 44–52, Jul. 2016, doi: 10.1109/MC.2016.200.
- [65] L. Murphy, M. B. Kery, O. Alliyu, A. Macvean, and B. A. Myers, “API Designers in the Field: Design Practices and Challenges for Creating Usable APIs,” in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Lisbon, Oct. 2018, pp. 249–258. doi: 10.1109/VLHCC.2018.8506523.

- [66] B. A. Myers and J. Stylos, “Improving API usability,” *Commun. ACM*, vol. 59, no. 6, pp. 62–69, May 2016, doi: 10.1145/2896587.
- [67] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson, “FireWorks: a dynamic workflow system designed for high-throughput applications: FireWorks: A Dynamic Workflow System Designed for High-Throughput Applications,” *Concurrency Computat.: Pract. Exper.*, vol. 27, no. 17, pp. 5037–5059, Dec. 2015, doi: 10.1002/cpe.3505.
- [68] C. Schroeder, C. Mück-Lichtenfeld, L. Xu, N. A. Grosso-Giordano, A. Okrut, C. Chen, S. I. Zones, A. Katz, M. R. Hansen, and H. Koller, “A Stable Silanol Triad in the Zeolite Catalyst SSZ-70,” *Angew. Chem. Int. Ed.*, vol. 59, no. 27, pp. 10939–10943, Jun. 2020, doi: 10.1002/anie.202001364.
- [69] P. M. Maffettone, J. K. Lynch, T. A. Caswell, C. E. Cook, S. I. Campbell, and D. Olds, “Gaming the beamlines—employing reinforcement learning to maximize scientific outcomes at large-scale user facilities,” *Mach. Learn.: Sci. Technol.*, vol. 2, no. 2, p. 025025, Jun. 2021, doi: 10.1088/2632-2153/abc9fc.
- [70] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, doi: 10.1038/s41586-019-1724-z.
- [71] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, doi: 10.1126/science.aar6404.
- [72] Z. Zhou, X. Li, and R. N. Zare, “Optimizing Chemical Reactions with Deep Reinforcement Learning,” *ACS Central Science*, p. 8, 2017.