Journal of Applied Physics TUTORIAL pubs.aip.org/aip/jap

Decoding diffraction and spectroscopy data with machine learning: A tutorial

Cite as: J. Appl. Phys. **137**, 131101 (2025); doi: 10.1063/5.0255593
Submitted: 30 December 2024 · Accepted: 5 March 2025 ·
Published Online: 3 April 2025

View Online Export Citation CrossMark

D. Vizoso and R. Dingreville[a)]

AFFILIATIONS

Center for Integrated Nanotechnologies, Sandia National Laboratories, Albuquerque, New Mexico 87185, USA

[a)]**Author to whom correspondence should be addressed:** rdingre@sandia.gov

ABSTRACT

This Tutorial provides a step-by-step guide on how to apply supervised machine-learning techniques to analyze diffraction and spectroscopy data. This Tutorial details four models—a reconstruction-focused model, a regression-focused model, a hybrid reconstruction/regression model, and a multimodal model—that use x-ray diffraction profiles and vibrational density of states spectra to predict various microstructural descriptors. In this Tutorial, we cover data pre-processing steps, constructions of the models via dimensionality reduction and regression, training, and analysis of these models. Comparisons of the model's performance are provided, highlighting the strength and weakness of the various approaches utilized.

I. INTRODUCTION

Spectroscopy and diffraction techniques have a long history of use for materials characterization in a wide range of scientific fields, including their use for the identification of the composition[1,2] or phase[3–5] of materials or the measurement of the stress state of a material,[6–9] among many other applications. Advances in experimental tools and automation have made the collection of large amounts of diffraction[10–13] or spectroscopy[14–16] data simpler than ever before, enabling high-throughput materials screening and characterization.

However, traditional analysis methods for these types of data, notably peak analysis, can become quite complicated when dealing with noise due to instrumental factors or material- and sample-related overlapping features. These complications can lead to biased data interpretation[17,18] or limit the analysis to only simple problems where consistent outcomes are possible.[19,20] When these methods need human intervention to correct for any of these biases and simplifications or to fine-tune the analysis, they become too slow to use for in-line analysis (which may be desired for accelerating time-sensitive beam-line experiments[21]) or for handling large volumes of data from high-throughput experimental techniques. Machine-learning methods provide a systematic approach for analyzing and interpreting spectroscopy and diffraction data that avoids the limitations of traditional methods driven by human-identifiable features. These techniques offer significant advantages in automation and efficiency and are capable of handling complex, noisy, or overlapping data. Numerous studies have demonstrated that machine-learning models can not only match the accuracy of traditional analysis methods,[21–28] but also reveal additional, hidden materials information that is not easily identifiable by conventional human analysis.[21,29,30]

In this Tutorial, we provide a step-by-step guide on how to apply supervised machine-learning techniques to analyze diffraction and/or spectroscopy data for the purpose of extracting microstructural state descriptors from an observed x-ray diffraction (XRD) profile. In Sec. II, we introduce the dataset used for this Tutorial. This publicly available dataset consists of simulated XRD profiles and vibrational density of states (VDoS) spectra. These profiles and spectra were generated from atomistic structures that had been subjected to different amounts of disorder insertion and mechanical loading.[31,32] For this Tutorial, we chose simulation data because they provide a controlled environment where the underlying patterns are well-understood, allowing us to focus on demonstrating the core concepts and methodologies of the machine-learning methods illustrated hereafter. In Sec. II B, we discuss how to perform data preparation and pre-processing for spectroscopy and diffraction data to improve model performance while preserving the integrity of the data. Section III provides a summary of the essential machine-learning building blocks to be

used for diffraction and spectroscopy data analysis. In Sec. IV, we discuss how to design and select a model architecture by defining and comparing four distinct machine-learning models. The first three models operate exclusively on a single input modality (XRD profiles). Each of these models emphasizes different aspects of the machine-learning model's building blocks, namely, dimensionality reduction, regression, or a hybrid approach that balances both tasks. The fourth model consists of a multimodal approach, which incorporates both XRD profiles and VDoS spectra as inputs. This last model is meant to demonstrate a straightforward method for integrating multiple data types to improve the performance of the analysis. These four models are presented to give examples of prioritizing specific aspects of the analysis based on data characteristics and the objective of the analysis. Finally, in Sec. V, we compare the performance of these different models and provide a discussion around model selection. Commented code snippets are included to serve as a bridge between theoretical concepts and practical implementation, enabling the reader to directly apply the discussed techniques to their own research. Overall, this Tutorial aims to equip researchers and practitioners with the knowledge and tools necessary to leverage supervised machine learning for advanced materials characterization, enabling more efficient and insightful analysis of complex microstructural data.

II. DATASET DESCRIPTION

This Tutorial utilizes a subset of a publicly available dataset, which includes simulated spectroscopy and diffraction profiles derived from molecular-dynamics simulations of mechanically deformed and disordered atomic structures.[31,32] In this Tutorial, we primarily utilize XRD profiles and the associated microstructural descriptors of bulk single crystal gold (Au). One machine-learning model in this Tutorial also incorporates VDoS spectra available in the dataset.

A. Atomic structures and associated diffraction and spectroscopy data

We performed molecular-dynamics simulations of Au structures using the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS).[33] We used an embedded atom method (EAM) interatomic potential[34] to describe interactions between Au atoms and created various atomic structures with different strain (either uniaxial or hydrostatic compression or tension) and defect states. Details on how these structures were created are provided elsewhere.[31] From these structures, we extracted seven microstructural descriptors, including stress tensor components (σ_{XX}, σ_{YY}, σ_{ZZ}), face-centered cubic (fcc), hexagonal close-packed (hcp), and disordered phase fractions, and the total dislocation density. These microstructural descriptors serve as regression targets for the machine-learning models designed to predict microstructural information from observed XRD profiles.

XRD spectra were simulated using the LAMMPS diffraction package on all Au atomic structures, with specific parameters for wavelength (1.54 Å), 2θ range (30°– 90°), and reciprocal lattice spacing (0.005 $Å^{-1}$). VDoS spectra were obtained by measuring the velocity autocorrelation function (VACF) in the molecular-dynamics simulations and then computing their Fourier transforms. The resulting VDoS spectra span a frequency range from 0 to 1667.7 cm^{-1}. This dataset exemplifies how machine-learning can link microstructural states to spectroscopy or diffraction data. The methods in this Tutorial are broadly applicable to other materials from the complete dataset[31,32] or various systems with diverse characterization techniques, microstructural properties, and modifications. Importantly, the chosen property and microstructural descriptors must be encoded in the selected characterization technique.

B. Data pre-processing

Data pre-processing is often necessary to enable or improve model training. Key considerations include data consistency, the range and distribution of the data values, normalization of the data, and noise filtering.

Regarding data consistency, input data size generally must be uniform across all samples (some machine-learning architectures can handle disparate input data sizes, but most cannot). In the case of the XRD 1D line profiles, each profile must be adjusted such that it spans the same range of 2θ values. Downsampling is a convenient method for achieving consistency and matching the range and point density of the least dense sample(s) in the dataset. This can be done by using a subset of the points in the high-density sample(s) (i.e., drop every other point or use every fourth point, etc.) or by averaging sets of points in bins. Both of these approaches can maintain the original integrity of the data, although high frequency features that exist below the downsampled resolution will be lost. Using spline or interpolation techniques to increase the density of sparse spectra can seem like an attractive method. However, this approach can also introduce artificial distortions to the data in regions with sharp features, as seen in Fig. 1(a), where interpolating between the sparsely distributed points creates a spectrum that deviates from the original near sharper features and results in shifts in peak positions and changes in relative feature intensities. For the purposes of this Tutorial, the raw XRD profiles collected from the original dataset were downsampled from an initial vector size of 6000 points down to 150 points by summing 40 points per bin across the 2θ range. VDoS spectra were instead truncated to contain a total of 5997 points covering a frequency range from 0 to approximately 500 cm^{-1}.

Normalization is another crucial step in preparing data as it can significantly enhance training speed and increase model sensitivity to relevant data ranges. However, it must be done carefully in order to prevent information loss or spurious information addition. When dealing with arbitrary or uncontrolled magnitudes—such as those found in experimental XRD profiles, which can vary with measurement time and can be difficult to control—one can normalize each profile by dividing the XRD profile by its maximum value. This approach can prevent the model from learning spurious correlations arising from inconsistent magnitudes across the dataset. Conversely, if the magnitudes are controlled, normalization can be performed identically across all samples (i.e., division by the same value for all samples). This ensures that the relative differences in magnitude between samples are preserved. Figure 1(b) illustrates both normalization approaches: the two spectra in the top panel are normalized by the maximum value of the XRD 1

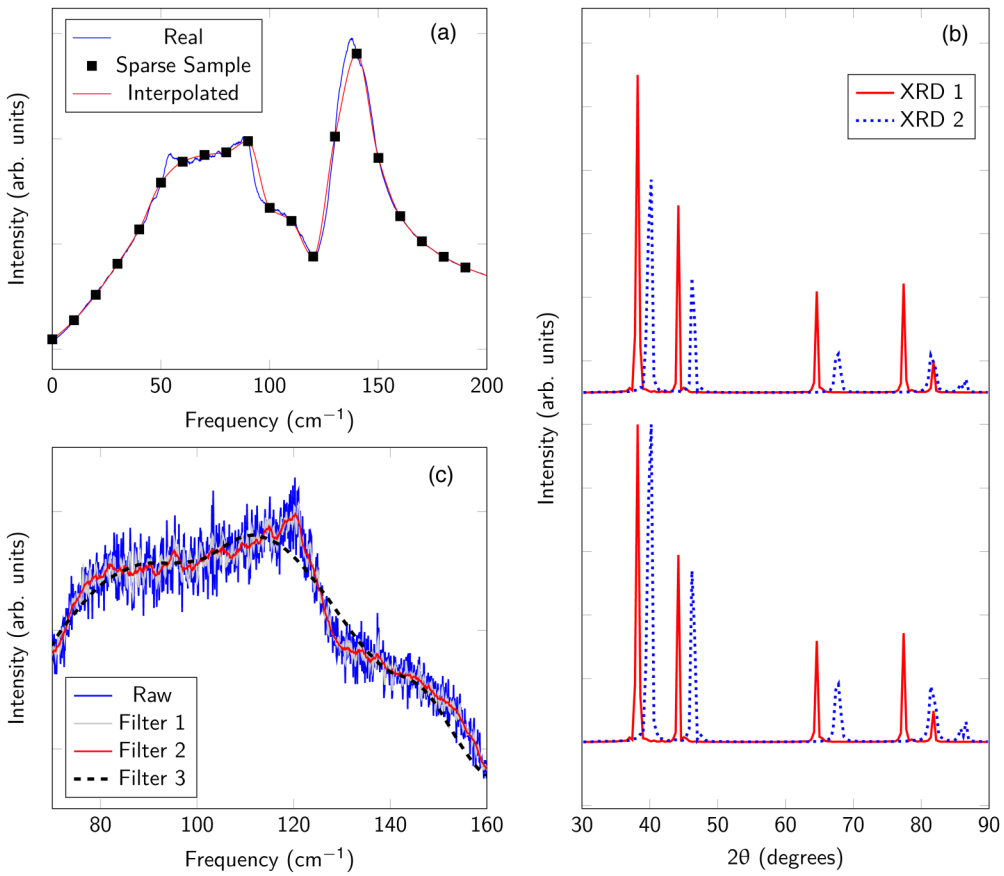

FIG. 1. (a) Comparison between a smoothed VDoS spectrum, a sparse sampling of this spectrum, and an interpolation of the sparse sampling. (b) Comparison between two XRD profiles (solid and dotted lines) that are normalized based on the maximum value of XRD1 only (top panel) or based on the maximum values of each profile (bottom panel). (c) Comparison between a noisy VDoS spectrum and several spectra that have been filtered using different filtering parameters.

spectrum, while the two spectra in the bottom panel are normalized individually by their respective maxima. For this Tutorial, both the VDoS spectra and the XRD profiles were normalized by dividing each spectrum or profile by the maximum intensity observed across all of the spectra or profiles in the dataset. Since these spectra and profiles are the result of well-controlled simulations, normalization by the maximum intensity preserves differences in relative intensities that may contain information relevant to the regression task.

Noise filtering may also be necessary depending on the quality of the input data and the chosen machine-learning model. For the purposes of discussion, we consider two types of noise: zero-mean and non-zero mean noises. Zero-mean noise refers to noises like Gaussian or Poisson noise, where each point in the measurement is displaced by a positive or negative value off of its true point by a random amount that can be represented by a known distribution with a mean value of zero. Non-zero mean noise has a consistent positive or negative bias and is commonly found in experimental data.

When addressing zero-mean noise, if the amplitude of the noise is consistent across the dataset and significantly lower than the feature magnitudes, it can typically be ignored without negatively affecting model performance. However, if the amplitude of the noise varies significantly between samples or if it is comparable in magnitude to the features present in the data, removing it may be beneficial or even essential for achieving good model performance. When applying filtering or smoothing algorithms for zero-mean noise removal, it is crucial to ensure that these processes do not distort or eliminate important features in the data. For example, Fig. 1(c) demonstrates how increasingly strong filtering of a noisy VDoS spectrum can initially remove local noise but eventually distorts the spectrum near inflection points. In this Tutorial, no noise removal or filtering was applied to the VDoS or XRD data. The decision to use noise filtering should be based on a careful evaluation of the data characteristics and the specific requirements of the machine-learning task.

Experimental measurements often exhibit non-zero mean noise due to background signals inherent to sensors and

TABLE I. Range of microstructural descriptor values from dataset.

Property	Min and Max value	Unit
fcc phase fraction	0.354 to 1.0	Unitless
hcp phase fraction	0.0 to 0.353	Unitless
Disordered phase fraction	0.0 to 0.302	Unitless
σ_{XX}	−204.22 to 14.48	GPa
σ_{YY}	−205.43 to 14.48	GPa
σ_{ZZ}	−207.32 to 14.48	GPa
Dislocation density	0.0 to 4.14×10^{-3}	$Å^{-2}$

instrumentation, which contribute additional noise to the collected data. If the magnitude of non-zero mean noise remains consistent across all samples relative to the features of interest, its removal may be unnecessary, as machine-learning models can learn to disregard such background signals. However, when the noise magnitude varies between samples, removal becomes critical—especially if the noise intensity is comparable to key features or if it correlates with the target descriptors (e.g., a higher noise floor in samples with specific features). Non-zero mean noise removal typically involves subtracting or adjusting the background signal, which may vary across the data range (e.g., higher background at lower 2θ values in XRD measurements) and between samples, necessitating careful manual or automated preprocessing.

Finally, we note that, for regression tasks as the one illustrated in this Tutorial, particularly in multi-output scenarios, normalizing the target outputs (here the microstructural descriptors) is often advantageous. This is especially crucial when the different outputs have significantly different value ranges. In our dataset, which includes seven microstructural descriptors as target outputs, each descriptor has its own distinct range of values and associated order of magnitude, as listed in Table I. To address this disparity in range, we normalized each descriptor range by mapping its minimum value to zero and scaling all values to fall between 0 and 1. This normalization was part of the data pre-processing conducted before model training. By standardizing the ranges of microstructural descriptors, we ensure that the machine-learning model treats each output variable with equal importance during the training process, regardless of their original scales.

C. Considerations for experimental data

The dataset described in Sec. II A is derived from computational simulations, ensuring that the data are free from spurious artifacts and contains noise that is well-characterized and accurately modeled. Consequently, minimal pre-processing was required to prepare the data for training the machine-learning models. In contrast, experimental data often involve complex noise and artifacts that are harder to characterize, necessitating additional care during preparation for model training or inference.

Let us consider two situations where experimental data are being used with a machine-learning model. In the first case, when training a machine-learning model directly with experimental data for inference, as in Desai *et al.* [35] for instance with experimental XRD measurements, the consistency of noise and artifacts across training and inference datasets is important. Methods for noise removal, normalization, and standardization, as detailed in Sec. II B, are generally applicable and appropriate, with careful attention paid to potential variations in noise floors and signal amplitudes. In the second case, training a machine-learning model with simulated data (or a combination of simulated and experimental data) for inference on experimental data introduces challenges due to potential inconsistencies in noise and artifacts between the two data types. A model trained solely on simulated data is unlikely to perform accurately on experimental data without addressing these differences. For instance, the XRD profiles in the dataset described in Sec. II A are generated through simulations and lack experimentally observed noise sources, such as instrument broadening or sensor background noise. Using such a model for inference on experimental data would likely result in inaccurate predictions.

To address discrepancies between simulated training data and experimental inference data, two approaches can be employed. The first involves pre-processing experimental data to remove noise and artifacts absent in simulated data. This method is feasible when the experimental data volume is low or pre-processing is straightforward but caution must be exercised to avoid altering relevant features. The second approach entails modifying simulated data to emulate experimental noise and artifacts, requiring a thorough understanding of experimental noise characteristics. For instance, Natinsky *et al.*[36] added artificial noise and corruptions to experimental atomic force microscopy (AFM) images for data augmentation. However, inadequate characterization of experimental noise and artifacts may lead to poor model performance on real-world datasets with unrepresented characteristics in the training set.

III. REGRESSION WITH DIFFRACTION OR SPECTROSCOPY DATA

There are several strategies to use supervised machine-learning algorithms to extract microstructure information from XRD profiles. One straightforward approach involves training a single model to directly map the XRD data ($\boldsymbol{I}$) to the desired, target microstructural descriptors ($\boldsymbol{s}$). This approach can be mathematically represented as

$$\boldsymbol{s} = F(\boldsymbol{I}), \tag{1}$$

where F is a machine-learning model that learns the mapping between XRD profiles $\boldsymbol{I}$ and microstructural descriptors $\boldsymbol{s}$. This

method has been successfully employed in recent studies, demonstrating its efficacy in predicting various material properties from diffraction data.[21,25–27,37,38] However, an alternative and often more robust approach, which is detailed in this Tutorial, involves a two-step process.[29,30] The first step consists of using dimensionality reduction techniques to learn a latent representation of the input XRD data,

$$\boldsymbol{Z} = F_1(\boldsymbol{I}), \tag{2}$$

where F_1 is a machine-learning model that compresses the input XRD profiles, $\boldsymbol{I}$, into a latent representation $\boldsymbol{Z}$. This step effectively compresses the high-dimensional diffraction profile into a more manageable, smaller form while retaining essential information. The second step is a regression from this reduced representation to the microstructural descriptors of interest, which can be mathematically expressed as

$$\boldsymbol{s} = F_2(\boldsymbol{Z}), \tag{3}$$

where F_2 is a machine-learning model that performs the mapping from the latent representation, $\boldsymbol{Z}$, to the microstructural descriptors, $\boldsymbol{s}$. This two-step approach often provides enhanced interpretability and flexibility, allowing for more nuanced analysis of the relationship between XRD profiles and material microstructure. Sections III A–III C provide introductions into the dimensionality reduction and regression methods used in this Tutorial.

A. Dimensionality reduction

Dimensionality reduction transforms high-dimensional XRD (or VDoS) data into a more manageable, lower-dimensional representation. This process, mathematically expressed in Eq. (2), converts an XRD profile $\boldsymbol{I}$, consisting of M data points across a specific 2θ range, into a vector $\boldsymbol{Z}$ with significantly fewer dimensions N, where N is much smaller than M. Traditionally, a subject matter expert performing a diffraction analysis intuitively performs a form of dimensionality reduction by focusing on specific human-identified features like peak positions, widths, and intensities. In the context of machine learning, these features are now captured via the vector $\boldsymbol{Z}$ which contains the full information of the original XRD profile and therefore effectively represent the microstructural states encoded in the diffraction data.

In this Tutorial, we employed convolutional autoencoders for dimensionality reduction of diffraction data. An autoencoder consists of two primary components: an encoder and a decoder. The encoder compresses the input data, reducing its dimension from M to a smaller latent representation of dimension N. The decoder reconstructs the original input from this latent encoding. A generalized diagram showing the structure of the autoencoder is provided in Fig. 2. The autoencoder is trained by minimizing a loss function

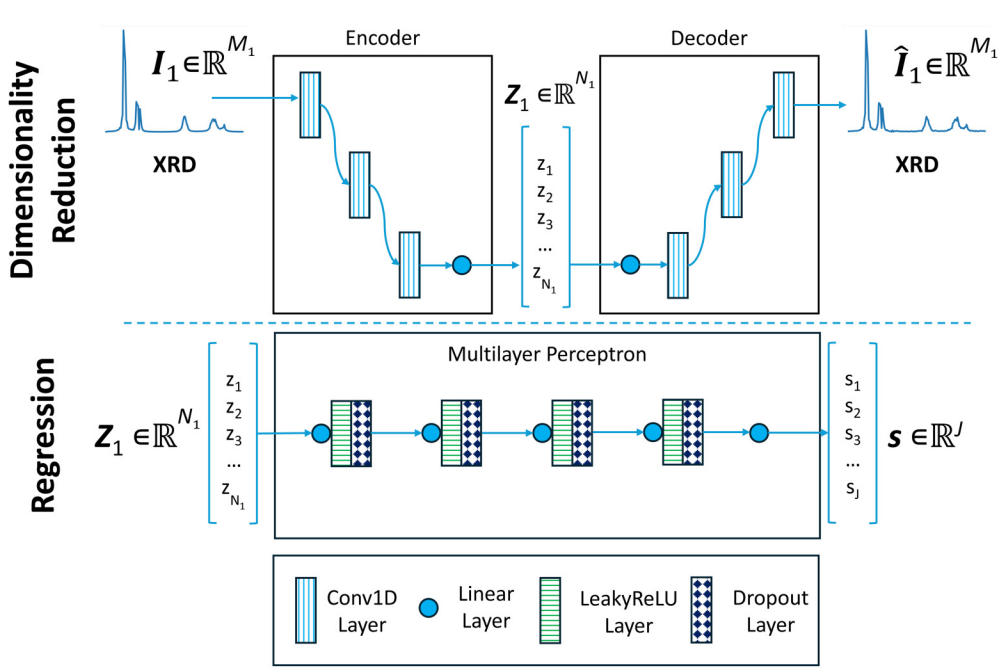

FIG. 2. Top row: dimensionality reduction via a convolutional autoencoder. The autoencoder is composed of an encoder that compresses an input spectrum $\boldsymbol{I}_1$ of dimension M_1 into a latent space $\boldsymbol{Z}_1$ of dimension N_1, and a decoder that reconstructs the input $\hat{\boldsymbol{I}}_1$ from its encoding. Middle row: regression using a multi-layer perceptron. The MLP is composed of several linear layers, which can be followed by various activation functions or other layer types such as Dropout or normalization layers. Bottom row: legend for the symbols used in neural networks' schematics.

that compares the original input to its reconstruction. While traditionally optimized for accurate reconstruction, autoencoders can be modified to encode specific information into the latent representation through additional loss terms.[39] Both the encoder and decoder are independent neural networks, which may have symmetric or asymmetric structures and can incorporate various neural network components. In the convolutional autoencoder defined in Fig. 2, convolutional blocks raster a kernel across the input data (XRD profile) capturing stationary features (peak shapes and characteristics) in the input data. These convolutional blocks are followed by a linear layer that is capable of learning the spatial relationship of the stationary features within the input. This approach allows for efficient dimensionality reduction while preserving the essential features of the diffraction data, making it particularly useful for analyzing complex materials systems and their microstructural states.

When performing dimensionality reduction, selecting the appropriate size of the latent space N is crucial. Some dimensionality reduction methods, such as Principal Component Analysis (PCA),[40–43] provide some guidance by quantifying the variance captured by each latent variable. Autoencoders, on the other hand, provide reconstruction error metrics, allowing users to adjust N until satisfactory reconstruction accuracy is achieved. It is generally advisable to conduct a sensitivity analysis to evaluate how varying N affects both the reconstruction error and performance in subsequent tasks utilizing the latent encoding. This approach ensures an optimal balance between data compression and information retention.

B. Regression

The regression model aims to map the latent vector $\boldsymbol{Z}$ to a set of material state descriptors $\boldsymbol{s}$ of dimension J, extracted from the XRD profiles. In the same manner that there are many different dimensionality reduction techniques, there exists a broad choice of regression techniques. For a comprehensive survey of regression methods that could be selected, the reader is encouraged to seek out books or review articles that provide thorough descriptions of the various classes and specific examples of regression methods that could be utilized for a wide variety of tasks, some of which are cited here.[44–48] The choice of method depends on the specific requirements of the materials characterization task at hand.

In this Tutorial, we employ a multi-layer perceptron (MLP) for all regression tasks. MLPs are versatile feed-forward neural networks that map input data to output values, making them well-suited for extracting microstructural descriptors from the latent representations of XRD profiles. A schematic of the general structure of an MLP is provided in Fig. 2. The structure can be represented mathematically as

$$\mathbf{s} = \varphi\left(\sum_{i=1} w_i z_i + b\right), \tag{4}$$

where φ is a non-linear activation function, w is a vector of weights, and b is a vector of biases. These network parameters, w and b, are learnable parameters that are optimized during training to minimize the prediction error. The architecture of an MLP is very flexible, with the specific activation function as well as the number and size of the hidden layers being adjustable to better address specific situations or datasets.

C. Model architecture selection

The selection of a specific model architecture involves many decisions, including general model architecture, specific model hyperparameters defining network depth and layer composition, as well as training parameters. These choices significantly impact model performance, affecting prediction accuracy, training difficulty, and data requirements. Frequently, these choices come with potential trade-offs, such as deeper networks potentially performing better at the desired task while also potentially being more prone to overfitting and requiring more data to train the network. Often, the rationale behind these impactful decisions remains unclear. When selecting model architectures for a specific dataset and machine-learning task, it is best practice to try many different combinations of model architectures and hyperparameters.

When optimizing model architectures and hyperparameters, it is crucial to assess how much optimization is necessary for the task at hand. While sequential fine tuning and optimization can yield incremental performance gains, they come at the expense of significant time, computational resources, and potentially reduced model generalizability. Establishing a reasonable accuracy threshold for the task at hand and stopping optimization once reasonable accuracy is satisfied is a practical approach to limit the otherwise boundless scope of model selection and tuning.

IV. MODELS

This Tutorial explores four machine-learning models for predicting material properties from XRD data, and in one case, using a combination of XRD and VDoS data. The first three models differ in how the process of dimensionality reduction is handled and how the dimensionality reduction and regression portions of the model are trained. These models are (i) a reconstruction-focused model using a traditional autoencoder/MLP model; (ii) a regression-focused model using an encoder/MLP model; and (iii) a hybrid reconstruction/regression model based on concurrent autoencoder-MLP training. The reconstruction-focused model uses an autoencoder for dimensionality reduction, optimized for high-quality reconstruction of the XRD profile. The resulting latent representation is then used to train an MLP to regress the microstructural descriptors. In the regression-focused model, the decoder is removed, allowing simultaneous training of the encoder and the MLP. This produces a latent representation optimized specifically for the regression task. The third machine-learning model, the hybrid reconstruction/regression model, trains the autoencoder and MLP concurrently using a combined loss function, creating a latent representation that balances reconstruction and regression tasks. Separately, the last and fourth model combines information from two latent spaces, derived from XRD profiles and VDoS spectra respectively, demonstrating a simple method for fusing multiple data modalities. Code examples are provided as part of the Tutorial, which demonstrate the construction and use of these models in Python using the PyTorch library.[49] The included code

```python
# Define XRD Autoencoder architecture
class XRD_Autoencoder(nn.Module):
    def __init__(self,encoded_space_dim):
        super().__init__()
        # Encoder Layers
        # Each convolutional layer expands the number of channels using
        #   a kernel size of 5
        self.conv1 = nn.Conv1d(1,4,5)
        self.conv2 = nn.Conv1d(4,8,5)
        self.conv3 = nn.Conv1d(8,16,5)
        # This linear layer takes the dimension of the output from the
        #   convolutional layers (2208) and reduces it down to the
        #   desired dimension size (encoded_space_dim)
        self.lin1 = nn.Linear(2208,encoded_space_dim)
        # The flatten operation is used to prepare the output of the
        #   convolutional layers for the linear operation
        self.flatten = nn.Flatten(start_dim=1)
        # Decoder Layers
        # The transpose convolutional layers perform inverse operations
        #   compared to the encoding convolutional layers
        self.convt1 = nn.ConvTranspose1d(4,1,5)
        self.convt2 = nn.ConvTranspose1d(8,4,5)
        self.convt3 = nn.ConvTranspose1d(16,8,5)
        # This linear layer  expands the encoded representation of the data
        #   up to the size needed by the transpose convolutional layers
        self.revlin1 = nn.Linear(encoded_space_dim,2208)
        # Unflatten Layer
        # The unflattened size is set based on the output of the above linear
        #   layer (2208 divided by 16 gives 138)
        self.unflatten = nn.Unflatten(dim=1,
            unflattened_size=(16,138))

    # The forward operation passes the input data through the encoder
    #   and the decoder
    # The encoded representation and the reconstruction of the input
    #   are returned as outputs
    def forward(self,x):
        # Encode the XRD profiles
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.lin1(x)
        # Save the encoded values
        encoded_space = x
        x = self.revlin1(x)
        x = self.unflatten(x)
        x = self.convt3(x)
        x = self.convt2(x)
        x = self.convt1(x)
        return x, encoded_space
```

CODE LISTING 1: Autoencoder Example.

snippets and model training results shown in Sec. V were created using PyTorch v. 2.0.1.

A. Reconstruction-focused model

This model implements a sequential approach to analyze XRD data. Initially, we train a convolutional autoencoder to perform dimensionality reduction [see Eq. (2)], targeting a latent dimension of $N_1 = 30$. This specific dimension was chosen based on PCA results, which indicated that 30 components capture over 98% of the total variance in the XRD dataset. This approach ensures that we retain the most significant information while substantially reducing data complexity. The architecture of the convolutional autoencoder used for this dimensionality reduction task is detailed in Code Listing 1. The code shown in Code Listing 1 assumes that the XRD profiles that are input in to the autoencoder have a length of $M_1 = 150$. This input dimension results in a 2208-point tensor after three convolutional layers, as shown in lines 10 and 17 of Code Listing 1. This value is derived from the input dimension M_1, channel expansion (16×138), and kernel size. If the autoencoder is to be used with XRD profiles of a different length or if changes are made to the hyperparameters of the convolutional layers, these lines will need to be modified with the correct dimension. Additionally, the value 138 on line 19 requires updating as it depends on the convolutional layers' output tensor size.

This autoencoder mimics the structure shown in Fig. 2. The encoding portion of the model is composed of three 1D convolutional layers each with a kernel size of 5 and the number of channels increasing through each consecutive layer. Then, these layers are flattened and a single dense linear layer is used to reduce the output of the convolutional layers down to the desired number of latent variables. Schematically, the encoder architecture can be described as follows: $\mathrm{Conv}^5_{4\times146} \times \mathrm{Conv}^5_{8\times142} \times \mathrm{Conv}^5_{16\times138} \times \mathrm{Flat}_{1\times2208} \times \mathrm{Linear}_{1\times N_1=30}$, where the nomenclature "Conv" describes a convolutional layer, "Linear" describes a linear layer, and "Flat" describes a flattening operation. The superscript of the "Conv" operations indicates the convolution kernel size, while the subscripts of all operations represent the dimensionality of the data after passing through each operation, with the first number indicating the number of channels and the second number indicating the size of each channel. The decoding portion of the model mirrors the encoding portion, using the same number of layers, the same channel counts, and the same kernel sizes, just in the reverse order. When the forward operation of the model is called, it will return both the latent projection of the data that has been passed to the model as well as the reconstruction of the input data. For this architecture, the number of convolutional layers, the kernel size in each of these convolutions, the number of channels, the number and size of linear layers after the convolutions, and the symmetry or asymmetry of the encoder and decoder are all adjustable parameters that can be modified to tune the models performance. Again, the code shown in Code Listing 1 assumes that the XRD profiles that are input in to the autoencoder have a length of $M_1 = 150$; modifications to the dimension of the linear layers and the unflatten operation will be necessary if the input changes in dimension.

For the purposes of this Tutorial, the data that are used for training and testing the models are stored in a PyTorch Dataset object, defined in Code Listing 2. In this object, the XRD profiles and the regression targets (i.e., the vector of the microstructural descriptor, $\boldsymbol{s}$) are stored together. This is done to make it easier to create shuffled testing and training datasets that are shared between the autoencoder and the MLP. The code shown in Code Listing 2 creates a 70%–30% training and test split and stores these datasets in PyTorch DataLoader objects which handle shuffling and batching the data. Different test–train splits can be created by modifying the random seed number in the definition of the gen1 random number generator object. Using the code snippet will require storing your own dataset in the PyTorch tensors xrd_tensor and target_tensor. This and subsequent code snippets do not store or print the training and test losses during the training process; modifications will be required if that is desired.

The process of training the autoencoder involves several key steps provided in Code Listing 3. First, the desired loss function, which in this case is the Mean Squared Error (MSE) function, is initialized. Next, the autoencoder is called and its learnable parameters are passed to the optimizer, specifically AdamW[50] for the purposes of this Tutorial. AdamW is a modified version of the classical Adam optimizer[51] that incorporates a built-in method for handling weight decay during training. The effectiveness and speed of the model's training are significantly influenced by the choice of optimizer and its hyperparameters. As indicated in Code Listing 3, the training of the autoencoder is performed with a learning rate (lr) of 0.001 and a weight decay of 0.001 for a total of 500 training epochs. These hyperparameter values were selected after performing a manual hyperparameter search with the objective of minimizing the autoencoder reconstruction error.

The training process consists of nested loops: an outer loop for epochs and inner loops for batches. In the training inner loop, each batch of XRD profiles $\boldsymbol{I}_1$ is fed into the autoencoder, producing reconstructions $\widehat{\boldsymbol{I}}_1$ and encoded versions $\boldsymbol{Z}_1$ of the XRD profiles. The loss is calculated by comparing the original and reconstructed profiles, which is then used to update the model parameters. The loss function takes the form

$$\mathrm{MSELoss}(\boldsymbol{I}, \widehat{\boldsymbol{I}}) = \frac{1}{L}\sum_{i=1}^{L} (\boldsymbol{I}_i - \widehat{\boldsymbol{I}}_i)^2, \tag{5}$$

where L is the number of XRD profiles in a batch, and $\boldsymbol{I}_i$ and $\widehat{\boldsymbol{I}}_i$ are the input XRD profiles and the reconstructed XRD profiles, respectively. This loss value is computed once per batch, and the model parameters are updated to minimize this loss value. A separate testing loop evaluates the test dataset and computes a test loss without updating the model parameters. Comparing training and test losses helps monitor training progress and detect overfitting, which is indicated by an increase in the test loss while the training loss continues to decrease. Overfitting can be combated by making the model smaller (i.e., reducing the number of learnable parameters) or by introducing dropout layers to the model's architecture.

Once the autoencoder has been trained, the task of translating from the learned encoded space to the regression targets can be addressed by training a regression model. For this Tutorial, regression from the latent representation $\boldsymbol{Z}_1$ created by the autoencoder is performed using an MLP. Code Listing 4 shows the architecture

```python
class Model_Dataset(Dataset):
    # Dataset class that stores XRD data and regression target vectors
    def __init__(self,xrd_tensor,target_tensor):
        self.xrd = xrd_tensor
        self.targets = target_tensor

    def __len__(self):
        return len(self.targets)

    def __getitem__(self,idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        single_xrd = self.xrd[idx]
        single_targets = self.targets[idx]
        sample = {"XRD":single_xrd,"Regression Targets":single_targets}
        return sample

# Store the xrd and regression target data in the Model_Dataset class
model_dataset = Model_Dataset(xrd_tensor,target_tensor)
# Set the batch size to 32
batch_num = 32

# Set the random number seed to 12345
gen1 = torch.Generator().manual_seed(12345)

# Get the size of the dataset (used for creating the test-train split)
m = len(xrd_tensor)
# Create training and test sets from the full dataset
train_set,test_set = random_split(model_dataset,[int(m-m*0.3),int(m*0.3+1),
    generator=gen1)

# Put the appropriate datasets into dataloader objects
train_loader = DataLoader(train_set,batch_size=batch_num,shuffle=True)
test_loader = DataLoader(test_set,batch_size=batch_num,shuffle=True)
```

CODE LISTING 2: Dataset Definition.

that was used for this MLP, which is composed of five hidden linear layers, which are individually followed by LeakyReLU activation functions and dropout layers that randomly zero 20% of the tensors that pass through them. The addition of dropout layers was found to help mitigate overfitting to the training data as well as to reduce the optimal loss value that could be achieved during training, at the cost of increasing the number of training epochs required. Schematically, this architecture can be represented as follows:

$$\begin{aligned}&\text{Linear}^{\text{LReLU}}_{1\times 64} \times \text{Drop}_{p=0.2} \times \text{Linear}^{\text{LReLU}}_{1\times 64} \times \text{Drop}_{p=0.2} \times \text{Linear}^{\text{LReLU}}_{1\times 64}\\ &\quad\times \text{Drop}_{p=0.2} \times \text{Linear}^{\text{LReLU}}_{1\times 64} \times \text{Drop}_{p=0.2} \times \text{Linear}_{1\times J=7},\end{aligned}$$

where "LReLU" refers to the LeakyReLU activation function and "Drop" refers to Dropout operations, where the subscript refers to the fraction of elements within the input tensor that are set to zero.

The MLP training code follows a similar structure to the autoencoder training. It initializes the MLP and optimizer, then uses a two-stage loop where XRD profiles are processed through the autoencoder before being passed to the MLP. The MSE loss function is used to evaluate the models performance, with the loss function now being computed by comparing the known regression targets to the values predicted by the MLP such that

$$\text{MSELoss}(\boldsymbol{s},\widehat{\boldsymbol{s}}) = \frac{1}{L}\sum_{i=1}^{L}(\boldsymbol{s}_i - \widehat{\boldsymbol{s}}_i)^2, \tag{6}$$

```python
# Define the loss function that will be used for evaluating model
#   performance
loss_fn = torch.nn.MSELoss()
# Set up XRD autoencoder object, set the encoded dimension size to 30
xrd_autoencoder = XRD_Autoencoder(encoded_space_dim=30)
# Get the trainable parameters from the autoencoder object
params_to_optimize = [{"params": xrd_autoencoder.parameters()}]
# Define the optimizer and set the optimization hyperparameters
#   learning rate set to 1e-3
#   weight decay set to 1e-3
xrd_optim = torch.optim.AdamW(params_to_optimize,lr=1e-3,weight_decay=1e-3)

# Set the number of training epochs for training the autoencoder
num_epochs = 500
# Initialize the outer loop
for epoch in range(num_epochs):
    # Prepare autoencoder for  training
    xrd_autoencoder.train()
    # Initialize the inner training loop
    for i_batch, sample_batch in enumerate(train_loader):
        # Isolate the XRD profiles from the Dataset object
        xrd_batch = sample_batch["XRD"]
        # Pass XRD profiles through the autoencoder
        recon, encoded = xrd_autoencoder(xrd_batch)
        # Calculate the reconstruction loss
        train_loss = loss_fn(recon,xrd_batch)
        # Update model weights
        xrd_optim.zero_grad()
        train_loss.backward()
        xrd_optim.step()
    # Set model to eval mode for testing
    xrd_autoencoder.eval()
    # Initialize the inner testing loop
    with torch.no_grad():
        for i_batch, sample_batch in enumerate(test_loader):
            xrd_batch = sample_batch["XRD"]
            recon, encoded = autoencoder(xrd_batch)
            test_loss = loss_fn(recon,xrd_batch)
```

CODE LISTING 3: Autoencoder Training Example.

where $\boldsymbol{s}$ and $\hat{\boldsymbol{s}}$ are the known and predicted regression targets, respectively. The same training and test datasets are used to train both the autoencoder and the MLP. This is done to prevent errors that may exist in encodings of the test dataset from influencing the training of the MLP. As indicated in Code Listing 5, training of the MLP was performed with a learning rate of 0.0001 and a weigh decay of 0.0001 for a total of 10 000 training epochs.

Sequential training of the autoencoder and MLP optimizes the latent representation for input reconstruction. This approach can enhance performance and generalizability of the learned representation under specific conditions, such as when multiple models utilize the same latent space (for example, training a regression model and a classification model with the same latent representation). It is particularly advantageous when the intended machine-learning task is challenging or only possible with a low-accuracy, as concurrent training would be dominated by the more difficult task's loss.

As discussed in Sec. III, the performance of the autoencoder and MLP described in this section is closely tied with the selection of the model architectures as well as the training hyperparameters. The specific architectures and hyperparameters described above

```python
# Define MLP Architecture
class Regressor(nn.Module):
    def __init__(self,encoded_space_dim,reg_space_dim):
        super().__init__()
        # The first linear layer expands the input from the encoded
        #   dimension size to a size of 64
        self.lin1 = nn.Linear(encoded_space_dim,64)
        # Subsequent linear layers maintain the size of the data
        self.lin2 = nn.Linear(64,64)
        self.lin3 = nn.Linear(64,64)
        self.lin4 = nn.Linear(64,64)
        # The final linear layer compresses the data from a size of 64
        #   down to the size of reg_space_dim
        self.lin5 = nn.Linear(64,reg_space_dim)

        # Each dropout layer has a 20% probability of zeroing values
        self.drop1 = nn.Dropout(p=0.2)
        self.drop2 = nn.Dropout(p=0.2)
        self.drop3 = nn.Dropout(p=0.2)
        self.drop4 = nn.Dropout(p=0.2)

        self.lrelu = nn.LeakyReLU()

    # The forward operation passes data sequentially through blocks of
    #   linear layers followed by LeakyReLU activation functions and
    #   dropout layers, returning a vector of size reg_space_dim
    def forward(self,x):
        x = self.lin1(x)
        x = self.lrelu(x)
        x = self.drop1(x)
        x = self.lin2(x)
        x = self.lrelu(x)
        x = self.drop2(x)
        x = self.lin3(x)
        x = self.lrelu(x)
        x = self.drop3(x)
        x = self.lin4(x)
        x = self.lrelu(x)
        x = self.drop4(x)
        x = self.lin5(x)

        return x
```

CODE LISTING 4: MLP Example.

were selected after performing manual architecture and hyperparameter optimization tests for the specific dataset described in Sec. II A. It is true that different model architectures and different training hyperparameter sets may perform better with this dataset, and it may also be true that different architectures and hyperparameters may be required to observe good performance with different datasets. When developing machine-learning workflows, it is always recommended that architecture and hyperparameter optimization is performed, either with manual tuning or using optimization tools such as Optuna,[52] Neptune,[53] and Weights and Biases.[54]

```python
# Set up MLP object and set MLP hyperparameters
#   input dimension of encoded space is 30
#   size of the MLP output is 7, same as the number of regression targets
regressor = Regressor(encoded_space_dim=30,reg_space_dim=7)
# Get the trainable parameters from the MLP
params_to_optimize = [{"params": regressor.parameters()}]
# Define the optimizer and set optimization hyperparameters
#   learning rate of 1e-4
#   weight decay of 1e-4
optim = torch.optim.AdamW(params_to_optimize,lr=1e-4,weight_decay=1e-4)

# Define the number of training epochs
num_epochs = 10000
# Initialize the outer loop
for epoch in range(num_epochs):
    # Prepare MLP for training, autoencoder for inference
    regressor.train()
    xrd_autoencoder.eval()
    # Initialize the inner training loop
    for i_batch, sample_batch in enumerate(train_loader):
        # Isolate the XRD profiles from the Dataset object
        xrd_batch = sample_batch["XRD"]
        # Isolate the regression targets
        target_data = sample_batch["Regression Targets"]
        # Pass XRD profiles through the autoencoder without updating
        #   autoencoder parameters
        with torch.no_grad():
            recon, encoded = xrd_autoencoder(xrd_batch)
        # Pass the encoded XRD profiles into the MLP
        predictions = regressor(encoded)
        # Calculate the regression loss
        train_loss = loss_fn(predictions,target_data)
        # Update model weights
        xrd_optim.zero_grad()
        train_loss.backward()
        xrd_optim.step()
    # Set regression model to eval mode for testing
    regressor.eval()
    # Initialize the inner testing loop
    with torch.no_grad():
        for i_batch, sample_batch in enumerate(test_loader):
            xrd_batch = sample_batch["XRD"]
            target_data = sample_batch["Regression Targets"]
            recon, encoded = autoencoder(xrd_batch)
            predictions = regressor(encoded)
            test_loss = loss_fn(predictions,target_data)
```

CODE LISTING 5: MLP Training Example.

B. Regression-focused model

The previous architecture focused on creating a latent representation that best reconstructed the original data. While this approach should theoretically provide a good basis for property regression, it is not guaranteed. The autoencoder can prioritize features that minimize reconstruction loss while overlooking subtle, regression-relevant information. In this section, we introduce a machine-learning model specifically designed for the regression task by removing the decoder and training the encoder and MLP concurrently, without considering reconstruction of the XRD profile.

```python
# Define architecture for an encoder that passes directly into an MLP
class Encoder_MLP(nn.Module):
    def __init__(self,encoded_space_dim,reg_space_dim):
        super().__init__()
        # Encoder Portion
        # Each convolutional layer expands the number of channels using
        #   a kernel size of 5
        self.conv1 = nn.Conv1d(1,4,5)
        self.conv2 = nn.Conv1d(4,8,5)
        self.conv3 = nn.Conv1d(8,16,5)
        # This linear layer takes the dimension of the data from the
        #   convolutional layers (2208) and reduces it down to the
        #   desired dimension size (encoded_space_dim)
        self.enc_lin1 = nn.Linear(2208,encoded_space_dim)
        # The flatten operation is used to prepare the output of the
        #   convolutional layers for the linear operation
        self.flatten = nn.Flatten(start_dim=1)

        # MLP Portion
        # The first linear layer expands the encoded space from
        #   a size of encoded_space_dim up to a size of 64
        self.lin1 = nn.Linear(encoded_space_dim,64)
        # Subsequent linear layers maintain the data size
        self.lin2 = nn.Linear(64,64)
        self.lin3 = nn.Linear(64,64)
        self.lin4 = nn.Linear(64,64)
        # The final linear layer compresses the data from a size
        #   of 64 down to a size of reg_space_dim
        self.lin5 = nn.Linear(64,reg_space_dim)

        # Each dropout operation has a 20% probability of zeroing values
        self.drop1 = nn.Dropout(p=0.2)
        self.drop2 = nn.Dropout(p=0.2)
        self.drop3 = nn.Dropout(p=0.2)
        self.drop4 = nn.Dropout(p=0.2)

        self.lrelu = nn.LeakyReLU()

    # The forward operation passes data through the encoder and directly into
    #   the MLP, returning a vector of size reg_space_dim
    def forward(self,x):
        # Encoder
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.enc_lin1(x)

        # MLP
        x = self.lin1(x)
        x = self.lrelu(x)
        x = self.drop1(x)
        x = self.lin2(x)
        x = self.lrelu(x)
        x = self.drop2(x)
        x = self.lin3(x)
        x = self.lrelu(x)
        x = self.drop3(x)
        x = self.lin4(x)
        x = self.lrelu(x)
        x = self.drop4(x)
        x = self.lin5(x)

        return x
```

CODE LISTING 6: Integration of Encoder and MLP Example.

Code Listing 6 shows the architecture of this regression-focused model. This code uses the encoder from Code Listing 1 and combines it directly with the MLP from Code Listing 4. Schematically, this architecture can be described as follows:

$$\begin{aligned}&\mathrm{Conv}^{5}_{4\times146} \times \mathrm{Conv}^{5}_{8\times142} \times \mathrm{Conv}^{5}_{16\times138} \times \mathrm{Flat}_{1\times2208}\\ &\quad\times \mathrm{Linear}_{1\times N_1=30} \times \mathrm{Linear}^{\mathrm{LReLU}}_{1\times64} \times \mathrm{Drop}_{p=0.2} \times \mathrm{Linear}^{\mathrm{LReLU}}_{1\times64}\\ &\quad\times \mathrm{Drop}_{p=0.2} \times \mathrm{Linear}^{\mathrm{LReLU}}_{1\times64} \times \mathrm{Drop}_{p=0.2} \times \mathrm{Linear}^{\mathrm{LReLU}}_{1\times64}\\ &\quad\times \mathrm{Drop}_{p=0.2} \times \mathrm{Linear}_{1\times J=7}.\end{aligned}$$

By excluding the decoder from the machine-learned model, the learned latent representation is solely optimized to minimize the regression loss. The same dataset structure defined in Code Listing 2 can be used for training this model, as it already handles shuffling and batching of both the XRD profiles and the regression targets simultaneously. As for Code Listing 1, this model architecture is defined for an input length of $M_1 = 150$. If a different input length is used or if changes are made to the hyperparameters of the convolutional layers, the number 2208 on line 9 of Code Listing 6 will need to be modified to the correct value for the desired input length passed through the convolutional layers.

The process of training the regression-focused model is similar to the process of training the reconstruction-focused model, following the same stages of initializing the loss function, the model, and the optimizer, then entering a two-stage loop that iterates through training and testing epochs each broken into sets of batches. Where the procedure defined in Code Listing 7 differs from the previous approaches defined in Code Listings 3 and 5 is in how the model is handled and how the loss is computed. Since the present model does not have a decoder, no reconstructions of the input XRD profiles are produced, and no optimization occurs based on the reconstruction error. Instead, the model directly predicts microstructural descriptors $\boldsymbol{s}$ from an input XRD profile $\boldsymbol{I}$, which are compared against the known microstructural descriptors using Eq. (6) to compute the loss for updating the model parameters. As indicated in Code Listing 7, training of the encoder and MLP were performed with a learning rate of 0.0001 and a weight decay of 0.0001 for a total of 10 000 epochs.

Concurrent training of dimensionality reduction and regression models can offer advantages over sequential training in certain scenarios. For instance, autoencoders often require high-dimensional latent representations for accurate reconstruction. By removing the decoder and training the encoder and regression models simultaneously, lower-dimensional latent representations can be used, as input reconstruction is no longer a factor. This reduction can eliminate extraneous information, potentially improving regression accuracy. Additionally, it is advantageous when using an MLP for regression under memory constraints, as the MLP's parameter count and memory requirements scale with the input size.

C. Hybrid reconstruction/regression model

Sections IV A and IV B described models with different priorities: one focused on reconstructing the input XRD data, while the other emphasized creating a latent representation optimized for the desired regression task. In this section, we introduce a hybrid approach that considers both objectives. This hybrid model consists of an autoencoder and an MLP that are trained concurrently, ensuring that the latent representation of the input XRD data is optimized for both the task of reconstructing the input data as well as providing useful features for the regression task.

Code Listing 8 shows the model architecture of the hybrid model. This model combines the autoencoder defined in Code Listing 1 with the MLP defined in Code Listing 4 in a single PyTorch model class. When this model is called, it outputs predictions of the regression targets provided by the MLP as well as the learned latent representation and the reconstruction of the input data that is created by the autoencoder. The architecture defined in Code Listing 8 expects the input XRD profiles to have a length of $M_1 = 150$. If a different input length is utilized or if changes are made to the convolutional layers in the autoencoder, the numbers 2208 on lines 9 and 16 as well as the number 138 on line 18 will need to be modified.

Code Listing 9 provides the code for training the hybrid model. This model follows a similar structure to previous training methods, but it differs in how the loss is computed. The total loss combines two components: the reconstruction loss from the autoencoder and the regression loss from the MLP. A new hyperparameter, loss_scaler, is introduced allowing the user to adjust the relative importance of these two losses. By modifying this parameter, a user can prioritize either the reconstruction or regression task or balance both tasks equally. To determine an appropriate value for loss_scaler that balances the two tasks, the user can run data through the untrained model and compare the initial reconstruction and regression losses. In equation form, the loss function can be written as

$$\begin{aligned}\mathrm{HybridLoss}(\boldsymbol{I}, \widehat{\boldsymbol{I}}, \boldsymbol{s}, \widehat{\boldsymbol{s}}) = {}& \mathrm{loss\_scaler} \times \mathrm{MSELoss}(\boldsymbol{I}, \widehat{\boldsymbol{I}})\\ &+ \mathrm{MSELoss}(\boldsymbol{s}, \widehat{\boldsymbol{s}}),\end{aligned} \tag{7}$$

where loss_scaler adjusts the bias between the autoencoder and MLP losses, which are themselves computed using Eqs. (5) and (6), respectively.

This hybrid model is intended to combine the benefits of the two previous models in the creation of the latent representation. Optimization of the reconstruction loss drives the learned latent representation to encode as much of the input data as possible, which should in theory maximize the generalizability of the latent representation for different machine-learning tasks. Optimization of the regression loss drives the learned latent representation to specifically consider the information necessary to maximize the regression accuracy. The aim of these two driving forces is to create a latent representation that is tuned specifically for the regression task while still being generalizable for other tasks.

D. Multimodal model

The three previous models used XRD profiles as input for model training. However, our dataset also includes VDoS spectra for each microstructure, allowing us to leverage multimodal machine learning for our regression task. For the purposes of this Tutorial, we introduce in this section a basic multimodal approach based on the hybrid model previously defined for a single modality. This

```python
# Define loss function that will be used for evaluating model performance
loss_fn = torch.nn.MSELoss()
# Set up the model object and set model hyperparameters
#   encoded space size of 30
#   size of the MLP output is 7
encoder_mlp = Encoder_MLP(encoded_space_dim=30,reg_space_dim=7)
# Get the trainable parameters from the model
params_to_optimize = [{"params": encoder_mlp.parameters()}]
# Define the optimizer and set optimization hyperparameters
#   learning rate of 1e-4
#   weight decay of 1e-4
optim = torch.optim.AdamW(params_to_optimize,lr=1e-4,weight_decay=1e-4)

# Set the number of training epochs
num_epochs = 10000
# Initialize the outer loop
for epoch in range(num_epochs):
    # Prepare model for training
    encoder_mlp.train()
    # Initialize the inner training loop
    for i_batch, sample_batch in enumerate(train_loader):
        # Isolate the XRD profiles from the Dataset object
        xrd_batch = sample_batch["XRD"]
        # Isolate the regression targets
        target_data = sample_batch["Regression Targets"]
        # Pass XRD profiles through the model
        predictions = encoder_mlp(xrd_batch)
        # Calculate the regression loss
        train_loss = loss_fn(predictions,target_data)
        # Update model weights
        optim.zero_grad()
        train_loss.backward()
        optim.step()
    # Set model to eval mode for testing
    encoder_mlp.eval()
    # Initialize the inner testing loop
    with torch.no_grad():
        for i_batch, sample_batch in enumerate(test_loader):
            xrd_batch = sample_batch["XRD"]
            target_data = sample_batch["Regression Targets"]
            prediction = encoder_mlp(xrd_batch)
            test_loss = loss_fn(prediction,target_data)
```

CODE LISTING 7: Regression-Focused Model Training.

multimodal model employs two separate autoencoders: one for XRD profiles and another for VDoS spectra. The learned latent representations from both modalities are concatenated and fed into a modified MLP, which uses this combined information for predictions. The loss function that is optimized in training this model is defined as

$$\begin{aligned}\text{MultimodalLoss}(\boldsymbol{I}_1, \widehat{\boldsymbol{I}}_1, \boldsymbol{I}_2, \widehat{\boldsymbol{I}}_2, \boldsymbol{s}, \widehat{\boldsymbol{s}}) = {} & w_1 \times \text{MSELoss}(\boldsymbol{I}_1, \widehat{\boldsymbol{I}}_1) \\ & + w_2 \times \text{MSELoss}(\boldsymbol{I}_2, \widehat{\boldsymbol{I}}_2) \\ & + \text{MSELoss}(\boldsymbol{s}, \widehat{\boldsymbol{s}}),\end{aligned} \tag{8}$$

where w_1 and w_2 are weighting hyperparameters that adjust the magnitude of the losses computed individually for the two

```python
# Define the Hybrid Reconstruction/Regression Model Architecture
class XRD_Autoencoder_MLP(nn.Module):
    def __init__(self,encoded_space_dim,reg_space_dim):
        super().__init__()
        # Encoder Layers
        # Each convolutional layer expands the number of channels using
        #   a kernel size of 5
        self.conv1 = nn.Conv1d(1,4,5)
        self.conv2 = nn.Conv1d(4,8,5)
        self.conv3 = nn.Conv1d(8,16)
        # This linear layer takes the dimension of the data from the
        #   convolutional layers (2208) and reduces it down to the
        #   desired size (encoded_space_dim)
        self.ae_lin1 = nn.Linear(2208,encoded_space_dim)
        # The flatten operation is used to prepare the output of the
        #   convolutional layers for the linear operation
        self.flatten = nn.Flatten(start_dim=1)
        # Decoder Layers
        # The transpose convolutional layers perform the inverse operation
        #   of the encoding convolutional layers
        self.convt1 = nn.ConvTranspose1d(4,1,5)
        self.convt2 = nn.ConvTranspose1d(8,4,5)
        self.convt3 = nn.ConvTranspose1d(16,8,5)
        # This linear layer expands the encoded representation of the data
        #   up to the size needed by the transpose convolutional layers
        self.ae_revlin1 = nn.Linear(encoded_space_dim,2208)
        # The unflattened size is set based on the output of the above linear
        #   layer (2208 divided by 16 gives 138)
        self.unflatten = nn.Unflatten(dim=1,unflattened_size=(16,138))

        # MLP Layers
        # The first linear layer expands the encoded dimension from a size
        #   of encoded_space_dim up to a size of 64
        self.lin1 = nn.Linear(encoded_space_dim,64)
        # Subsequent linear layers maintain the size of the data
        self.lin2 = nn.Linear(64,64)
        self.lin3 = nn.Linear(64,64)
        self.lin4 = nn.Linear(64,64)
        # The final linear layer compresses the data from a size of 64
        #   down to a size of reg_space_dim
        self.lin5 = nn.Linear(64,reg_space_dim)

        # Each dropout layer has a 20% probability of zeroing values
        self.drop1 = nn.Dropout(p=0.2)
        self.drop2 = nn.Dropout(p=0.2)
        self.drop3 = nn.Dropout(p=0.2)
        self.drop4 = nn.Dropout(p=0.2)

        self.lrelu = nn.LeakyReLU()
```

CODE LISTING 8: Hybrid Reconstruction/Regression Model Architecture.

```
    # The forward operation passes data sequentially through the encoder,
    #   decoder, and MLP.
    # The output of the MLP, the encoded representation learned by the
    #   encoder, and the reconstruction of the input produced by the
    #   decoder are returned as output.
    def forward(self,x):
        # Encode the XRD profiles
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.ae_lin1(x)
        # Save the encoded values
        encoded_space = x
        x = self.ae_revlin1(x)
        x = self.unflatten(x)
        x = self.convt3(x)
        x = self.convt2(x)
        x = self.convt1(x)
        # Save the decoded XRD profiles
        reconstruction = x
        # Use an MLP to regress the target values from the encoded values
        x = self.lin1(encoded_space)
        x = self.lrelu(x)
        x = self.drop1(x)
        x = self.lin2(x)
        x = self.lrelu(x)
        x = self.drop2(x)
        x = self.lin3(x)
        x = self.lrelu(x)
        x = self.drop3(x)
        x = self.lin4(x)
        x = self.lrelu(x)
        x = self.drop4(x)
        x = self.lin5(x)
        return x, encoded_space, reconstruction
```

CODE LISTING 8: (*Continued.*)

autoencoders using Eq. (5) compared to the loss computed for the MLP using Eq. (6). Supplementary material I provides the code for the models, training workflow, and guidance on preparing VDoS profiles for model training. Figure 3 depicts a schematic of the architecture of the multimodal model.

This multimodal workflow takes the general architecture defined in Sec. IV C and modifies it to consider two input modalities, those being XRD profiles and VDoS spectra. Both of these modalities share the same general form of 1D line profiles, but they have distinct characteristics both in terms of the number of features that exist within the 1D line profiles as well as the form of those features. Due to these differences, different autoencoder architectures and latent dimension sizes are needed to produce good reconstructions of the two modalities. For the purposes of this Tutorial, manual architecture and hyperparameter optimization was performed to determine how the two autoencoders should be defined. The specific differences in the architectures of the two autoencoders used to encode these two modalities are defined in supplementary material I.

```python
# Define loss functions for the autoencoder and the MLP
loss_fn_1 = torch.nn.MSELoss()
loss_fn_2 = torch.nn.MSELoss()
# Set up the model object
xrd_ae_mlp = XRD_Autoencoder_MLP(encoded_space_dim=30,reg_space_dim=7)
# Get the trainable parameters from the model
params_to_optimize = [{"params": xrd_ae_mlp.parameters()}]
# Define the optimizer and set optimization hyperparameters
#   learning rate of 1e-4
#   weight decay of 1e-4
optim = torch.optim.AdamW(params_to_optimize,lr=1e-4,weight_decay=1e-4)
# Define weighting factor for use when combining the autoencoder and
#   MLP losses
loss_scaler = 3

# Set the number of training epochs
num_epochs = 10000
# Initialize the outer loop
for epoch in range(num_epochs):
    # Prepare model for training
    xrd_ae_mlp.train()
    # Initialize the inner training loop
    for i_batch, sample_batch in enumerate(train_loader):
        # Isolate the XRD profiles from the Dataset object
        xrd_batch = sample_batch["XRD"]
        # Isolate the regression targets
        target_data = sample_batch["Regression Targets"]
        # Pass XRD profiles through the model
        predictions, encoding, reconstruction = xrd_ae_mlp(xrd_batch)
        # Calculate the autoencoder loss
        train_ae_loss = loss_fn_1(reconstruction,xrd_batch)
        # Calculate the regression loss
        train_mlp_loss = loss_fn_2(predictions,target_data)
        # Combine the losses with the scaling factor applied
        train_loss = train_ae_loss*loss_scaler+train_mlp_loss
        # Update model weights
        optim.zero_grad()
        train_loss.backward()
        optim.step()
    # Set model to eval mode for testing
    xrd_ae_mlp.eval()
    # Initialize the inner testing loop
    with torch.no_grad():
        for i_batch, sample_batch in enumerate(test_loader):
            xrd_batch = sample_batch["XRD"]
            target_data = sample_batch["Regression Targets"]
            prediction,encoding,reconstruction = xrd_ae_mlp(xrd_batch)
            test_ae_loss = loss_fn_1(reconstruction,xrd_batch)
            test_mlp_loss = loss_fn_2(prediction,target_data)
            test_loss = test_ae_loss*loss_scaler+test_mlp_loss
```

CODE LISTING 9: Hybrid Reconstruction/Regression Model Training.

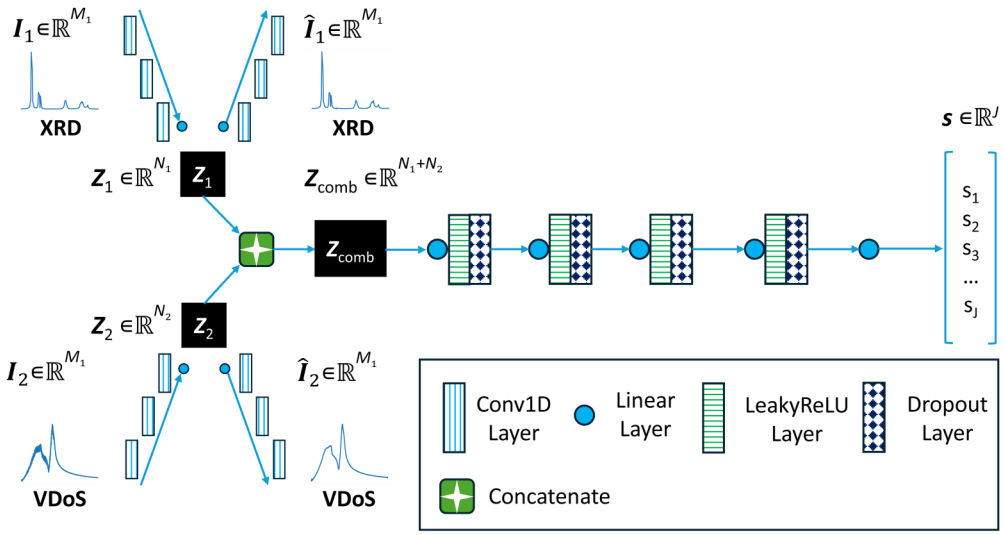

FIG. 3. Schematic of the structure for the multimodal model. Two autoencoders create separate latent representations (XRD: $\mathbf{Z}_1$, VDoS: $\mathbf{Z}_2$) of the two input modalities ($\boldsymbol{I}_1$, $\boldsymbol{I}_2$) and an MLP operates on the concatenated vector of these latent representations ($\mathbf{Z}_{comb}$) to create predictions ($\mathbf{s}$).

V. DISCUSSION

As shown in Table I, each microstructural descriptor that is included as a regression target varies across ranges of varying magnitudes, and, therefore, the range for each descriptor was normalized from 0 to 1 for model training. In Secs. V A–V C, when comparing the performance of the previously defined machine-learning models, model errors will be presented using un-normalized values. To examine the impact of the test–train split on model performance, each of the four workflows presented in this Tutorial were trained with ten different test–train splits. Model accuracies reported in Secs. V A–V C will be the averages and optimal performances observed across these 10 different trained models. Model architectures and all hyperparameters besides the seed used to create the test–train split were kept constant during these runs.

A. Comparison of single modality workflow performance

Tables II, III, and IV present performance metrics for the reconstruction-focused, regression-focused, and hybrid single-modality models in predicting microstructural descriptors from XRD profiles. For each descriptor, these tables provide three key values: the average and standard deviation of the root mean squared error (RMSE) calculated from 10 models trained on different test–train splits, the RMSE for each descriptor from the individual model that had the lowest average test loss (MSE) during training, and the lowest test error achieved for that specific descriptor across all 10 of the trained models. All values are reported in the units of their respective descriptors, as specified in Table I. Supplementary material II includes additional tables with the MSE for normalized regression targets and R^2 scores.

A comparison of the performance metrics in Tables II, III, and IV reveals that no single model consistently outperforms the others in predicting all microstructural descriptors. Each model shows strengths in different areas. The reconstruction-focused model performs particularly well in predicting the three stress components. It achieves the lowest average error across 10 training iterations and produces the most accurate results for stress predictions. The regression-focused model improves predictions for the fcc and hcp phase fractions compared to the reconstruction-focused model

TABLE II. Reconstruction-focused workflow RMSE.

Target descriptor	Average RMSE	Best model	Best RMSE
fcc phase fraction	$0.014\,90 \pm 0.005\,74$	0.009 50	0.009 43
hcp phase fraction	$0.007\,14 \pm 0.000\,527$	0.006 26	0.006 26
Disordered phase fraction	$0.006\,39 \pm 0.000\,994$	0.005 86	0.005 24
σ_{XX}	2.272 ± 0.315	1.645	1.645
σ_{YY}	2.326 ± 0.293	1.673	1.673
σ_{ZZ}	2.376 ± 0.334	1.733	1.733
Total dislocation density	$1.557 \times 10^{-4} \pm 1.917 \times 10^{-5}$	1.382×10^{-4}	1.365×10^{-4}

TABLE III. Regression-focused workflow RMSE.

Target descriptor	Average RMSE	Best model	Best RMSE
fcc phase fraction	0.013 39 ± 0.003 24	0.008 80	0.008 80
hcp phase fraction	0.006 69 ± 0.001 27	0.005 90	0.005 74
Disordered phase fraction	0.007 58 ± 0.001 01	0.005 96	0.005 96
σ_{XX}	2.798 ± 0.735	2.583	1.805
σ_{YY}	2.750 ± 0.727	2.470	1.846
σ_{ZZ}	2.792 ± 0.708	2.658	1.902
Total dislocation density	$1.605 \times 10^{-4} \pm 2.154 \times 10^{-5}$	1.427×10^{-4}	1.415×10^{-4}

but performs similarly or slightly worse for the other descriptors. The hybrid model performs best in predicting various phase fractions and total dislocation density but shows lower accuracy than the reconstruction-focused model for stress values. Ultimately, the selection of the most suitable model should be based on the particular needs of the application and the relative importance of accurately predicting each microstructural descriptor.

Figure 4 shows parity plots to compare the regression accuracy of the best-performing models from each single-modality model (reconstruction-focused, regression-focused, and hybrid). These parity plots reveal the similarity across the three models, not only in overall model accuracy, but also in specific areas where model predictions suffer. For example, the disordered phase fraction plots for all models show a consistent pattern: the data appears clustered into three distinct groups. The majority of prediction errors stems from misclassifying values into the wrong clusters. Also, while the cluster with the highest value is correctly centered, the cluster's shape indicates that the models struggle to accurately distribute values within this parameter space. Similar patterns of errors are evident in other descriptors. The total dislocation density plots show errors concentrated around the zero-value point, while the fcc phase fraction plots display inaccuracies in the region close to a value of one. These consistent error patterns across different models suggest that the challenges in accurate prediction may be inherent to the data structure or the specific relationships between XRD profiles and certain microstructural descriptors, rather than limitations of any particular modeling approach.

One important thing to note is that models presented here are not fully optimized. The model architectures and training methods used in this Tutorial were standardized for educational purposes, but they have potential for improvement. Modifications of the models' architecture or to the way in which they are trained could significantly improve their performance and potentially alter which model yields the highest accuracy for various regression targets. For example, the regression-focused model exhibited some training instability, occasionally experiencing sudden increases in test loss. These fluctuations negatively impacted the model's final accuracy, and could have been mitigated by stopping the training process when the test loss was lowest. This highlights the importance of fine-tuning in real-world applications. In practical scenarios, it would be essential to customize the model architecture and training procedure to suit the specific regression task at hand.

B. Comparison between single mode and multimodal models

Table V presents performance metrics for the multimodal model, which uses both XRD and VDoS profiles as inputs. A comparison of these metrics with those shown in Tables II, III, and IV reveals that the multimodal model offers some improvements in regression accuracy over the single-modality methods. However, these improvements are generally relatively marginal compared to those of single-modality approaches, with a small reduction in RMSEs for most descriptors. This trend suggests that the multimodal workflow exhibits greater stability across different training runs, demonstrating reduced variance in model performance. This increased consistency indicates that the multimodal approach may be more robust and reliable, even if the absolute improvements in accuracy are not dramatic.

C. Selecting an appropriate model

The analysis of the proposed models reveals that they achieve comparable levels of accuracy overall but with distinct strengths and weaknesses. Some models perform well at predicting certain

TABLE IV. Hybrid workflow RMSE.

Target descriptor	Average RMSE	Best model	Best RMSE
fcc phase fraction	0.014 04 ± 0.005 56	0.008 45	0.007 79
hcp phase fraction	0.005 89 ± 0.000 30	0.006 19	0.005 43
Disordered phase fraction	0.006 59 ± 0.001 02	0.005 78	0.005 32
σ_{XX}	2.393 ± 0.471	2.036	1.839
σ_{YY}	2.344 ± 0.471	1.950	1.810
σ_{ZZ}	2.401 ± 0.443	1.968	1.874
Total dislocation density	$1.476 \times 10^{-4} \pm 1.593 \times 10^{-5}$	1.394×10^{-4}	1.314×10^{-4}

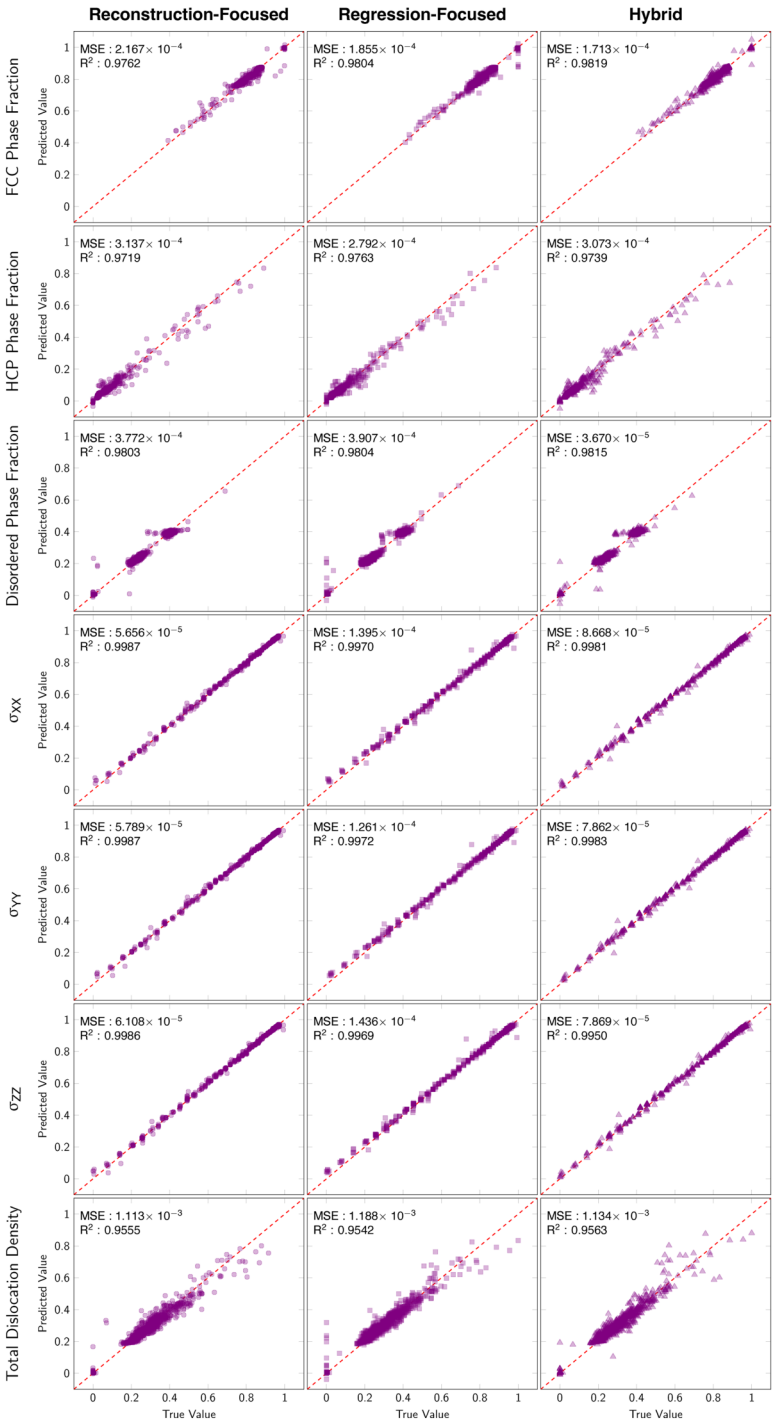

FIG. 4. Parity plots from the test portion of the dataset for the seven microstructural descriptors predicted by the three single-modality models. Values shown have been normalized to be between 0 and 1 based on the total range of descriptor values from the dataset. R^2 and MSE for each parity plot are provided.

descriptors while underperforming on others. Other models demonstrate more consistent accuracy across all descriptors. It is important to note that these models have potential for further optimization through adjustments to model architectures and training hyperparameters. The choice of model for a specific application should be guided by the particular requirements of that application. For instance, if sample generation is needed, variational autoencoders could replace the deterministic convolutional

TABLE V. Multimodal hybrid workflow RMSE.

Target descriptor	Average RMSE	Best model	Best RMSE
fcc phase fraction	0.012 30 ± 0.005 93	0.006 97	0.006 97
hcp phase fraction	0.006 27 ± 0.000 29	0.006 05	0.005 91
Disordered phase fraction	0.005 65 ± 0.001 09	0.004 57	0.004 48
σ_{XX}	1.982 ± 0.436	1.906	1.454
σ_{YY}	1.956 ± 0.441	1.882	1.406
σ_{ZZ}	2.064 ± 0.454	2.017	1.420
Total dislocation density	$1.344 \times 10^{-4} \pm 2.198 \times 10^{-5}$	1.092×10^{-4}	1.092×10^{-4}

autoencoders used in this Tutorial. For applications requiring multiple output types (e.g., both regression and classification), these models can be adapted to allow multiple models to operate on the learned latent representation of the input data. Ultimately, the selection of the most appropriate model depends on a careful consideration of the specific needs and constraints of the task at hand.

VI. CONCLUSIONS

This Tutorial is primarily intended for researchers and practitioners interested in leveraging machine-learning techniques for advanced materials characterization. This Tutorial provides a comprehensive and practical guide for applying supervised machine-learning techniques to analyze diffraction and spectroscopy data for extracting microstructure information. The key elements for such approaches include data preparation composed of consistent data sampling, careful normalization technique, and noise filtering; and a machine-learning model composed of a dimensionality reduction elements to capture the salient features encoded in the XRD data and a regression model to efficiently map this low-dimensional representation of the diffraction data to an extensive set of microstructural descriptors. In this Tutorial, we covered four distinct machine-learning workflows:

1. A model focusing on reconstruction, which relies on sequentially training a dimensionality reduction model of the XRD data, followed by training of the regression model. The learned reduced representation is optimized for the reconstruction of the input XRD.
2. A model focusing on regression for which the decoder is removed, creating a latent representation optimized specifically for regression of the microstructural descriptors.
3. A hybrid model that focused on joint optimization of both the reconstruction and regression, allowing for tunable training toward either task.
4. A multimodal approach includes multiple input data modalities, enabling the regression model to use latent representations from various sources.

All four workflows successfully extract microstructural state descriptors from the given data. Comparisons show comparable regression accuracy across workflows, with some performing better for specific descriptors or showing more consistency across different test–train splits. The choice of workflow depends on the user's specific needs and the nature of the data. Testing multiple workflows for a given task is recommended to identify the best-performing approach. These workflows are flexible and can be modified with different machine-learning architectures for dimensionality reduction or regression tasks.

While the machine-learning models illustrated in this Tutorial demonstrated their ability to reveal hidden materials information not easily identifiable by conventional human analysis, several avenues for future improvements remain. For instance, the development of more interpretable models[55–57] would help practitioners and researchers with the ability to better understand the relationship between materials characterization data (whether it is diffraction, spectroscopy, or other) and the underlying microstructure and associated physical and chemical processes at play. Another area for improvement is related to transfer learning[58,59] and how models trained on simulated data can be adapted to actual experimental data, potentially reducing the need for large experimental datasets.[60,61] Finally as a last example, the methods illustrated in this Tutorial could be improved to better estimate the uncertainty associated with the model predictions for more reliable predictions and materials characterization.[62,63]

Moving forward, this Tutorial could serve as a starting point for automating and enhancing the analysis of complex materials data such as diffraction and spectroscopy across a broad ranging of disciplines including geoscience,[64–68] archeology,[69–71] astronomy,[72–74] or medical imaging.[75–77]

SUPPLEMENTARY MATERIAL

The supplementary material provides details on network architectures used in this Tutorial and convergence studies.

ACKNOWLEDGMENTS

The authors would like to thank Andreas Robertson and Lane Schultz from Sandia National Laboratories for insightful comments and suggestions during the preparation of this manuscript. The authors would also like to thank the anonymous reviewers for their constructive feedback resulting in the final version of this Tutorial. Machine-learning capabilities and computational resources used to create this Tutorial are supported by the Center for Integrated Nanotechnologies, an Office of Science user facility operated for the U.S. Department of Energy. This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title,

and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

D. Vizoso: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Software (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **R. Dingreville:** Conceptualization (equal); Funding acquisition (lead); Investigation (equal); Project administration (lead); Resources (lead); Supervision (lead); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are openly available in the Materials Data Facility website, Ref. 32.

REFERENCES

[1]L. D. Whittig, *X-ray Diffraction Techniques for Mineral Identification and Mineralogical Composition* (American Society of Agronomy, Inc., 1965), Chap. 49.

[2]W. Harris and G. N. White, *X-ray Diffraction Techniques for Soil Mineral Identification* (Soil Science Society of America, Inc., 2008), Chap. 4.

[3]J. D. Hanawalt, "Phase identification by x-ray powder diffraction evaluation of various techniques," in *Twenty-Fifth Annual Conference on Applications of X-ray Analysis* (ACS, 1976), Vol. 20, pp. 63–73.

[4]K.-L. Lin, "Phase identification using series of selected area diffraction patterns and energy dispersive spectrometry within TEM," Microscopy Res. **2**, 57–66 (2014).

[5]Y. Yun, W. Wan, F. Rabbani, J. Su, H. Xu, S. Hovmöller, M. Johnsson, and X. Zou, "Phase identification and structure determination from multiphase crystalline powder samples by rotation electron diffraction," J. Appl. Crystallogr. **47**, 2048–2054 (2014).

[6]I. De Wolf, H. E. Maes, and S. K. Jones, "Stress measurements in silicon devices through Raman spectroscopy: Bridging the gap between theory and experiment," J. Appl. Phys. **79**(9), 7148–7156 (1996).

[7]K. F. Dombrowski, I. De Wolf, and B. Dietrich, "Stress measurement using ultraviolet micro-Raman spectroscopy," Appl. Phys. Lett. **75**, 2450–2451 (1999).

[8]U. Welzel, J. Ligot, P. Lamparter, A. C. Vermeulen, and E. J. Mittemeijer, "Stress analysis of polycrystalline thin films and surface regions by x-ray diffraction," J. Appl. Crystallogr. **38**, 1–29 (2004).

[9]T. Beechem, S. Graham, S. P. Kearney, L. M. Phinney, and J. R. Serrano, "Simultaneous mapping of temperature and stress in microdevices using micro-Raman spectroscopy," Rev. Sci. Instrum. **78**, 061301 (2007).

[10]J. M. Gregoire, D. G. Van Campen, C. E. Miller, R. J. R. Jones, S. K. Suram, and A. Mehta, "High-throughput synchrotron x-ray diffraction for combinatorial phase mapping," J. Synchrotron Radiat. **21**, 1262–1268 (2014).

[11]M. O. Cichocka, J. Ångström, B. Wang, X. Zou, and S. Smeets, "High-throughput continuous rotation electron diffraction data acquisition via software automation," J. Appl. Crystallogr. **51**, 1652–1661 (2018).

[12]Y. Luo, B. Wang, S. Smeets, J. Sun, W. Yang, and X. Zou, "High-throughput phase elucidation of polycrystalline materials using serial rotation electron diffraction," Nat. Chem. **15**, 483–490 (2023).

[13]D. P. Adams, R. Kothari, S. Addamane, M. Jain, K. Dorman, S. Desai, C. Sobczak, M. Kalaswad, N. Bianco, F. W. DelRio, J. O. Custer, M. A. Rodriguez, J. Boro, R. Dingreville, and B. L. Boyce, "Guided combinatorial synthesis and automated characterization expedites the discovery of hard, electrically conductive Pt_xAu_{1-x} films," J. Vac. Sci. Technol. A **42**(5), 053411 (2024).

[14]K. Liu, X. Hong, Q. Zhou, C. Jin, J. Li, W. Zhou, J. Liu, E. Wang, A. Zettl, and F. Wang, "High-throughput optical imaging and spectroscopy of individual carbon nanotubes in devices," Nat. Nanotechnol. **8**, 917–922 (2013).

[15]D. S. Sebba, D. A. Watson, and J. P. Nolan, "High throughput single nanoparticle spectroscopy," ACS Nano **3**(6), 1477–1484 (2009).

[16]W. Malzer, D. Grötzsch, R. Gnewkow, C. Schlesiger, F. Kowalewski, B. Van Kuiken, S. DeBeer, and B. Kanngießer, "A laboratory spectrometer for high throughput x-ray emission spectroscopy in catalysis research," Rev. Sci. Instrum. **89**(11), 113111 (2018).

[17]M. Marjańska, D. K. Deelchand, and R. Kreis, and the 2016 ISMRM MRS Study Group Fitting Challenge Team, "Results and interpretation of a fitting challenge for MR spectroscopy set up by the MRS study group of ISMRM," Magn. Reson. Med. **87**, 11–32 (2021).

[18]C. Kunka, B. L. Boyce, S. M. Foiles, and R. Dingreville, "Revealing inconsistencies in x-ray width methods for nanomaterials," Nanoscale **46**, 22456–22466 (2019).

[19]N. I. Tjahyono, T. Groutso, D. S. Wong, P. Lavoie, and M. P. Taylor, *Improving XRD Analysis for Complex Bath Chemistries—Investigations and Challenges Faced* (Springer, 2014), pp. 573–578.

[20]M. F. Wahab, G. Hellinghausen, and D. W. Armstrong, "Progress in peak processing," LC-GC Europe **32**, 22–28 (2019).

[21]V.-A. Surdu and R. György, "X-ray diffraction data analysis by machine learning methods—A review," Appl. Sci. **13**(17), 9992 (2023).

[22]J. A. Aguiar, M. L. Gong, and T. Tasdizen, "Crystallographic prediction from diffraction and chemistry data for higher throughput classification using machine learning," Comput. Mater. Sci. **173**, 109409 (2020).

[23]J. J. Kim, F. T. Ling, D. A. Plattenberger, A. F. Clarens, A. Lanzirotti, M. Newville, and C. A. Peters, "SMART mineral mapping: Synchrotron-based machine learning approach for 2D characterization with coupled micro XRF-XRD," Comput. Geosci. **156**, 104898 (2021).

[24]B. Zhao, S. Wolter, and J. A. Greenberg, "Application of machine learning to x-ray diffraction-based classification," Proc. SPIE **10632**, 20–25 (2018).

[25]Y. Suzuki, "Automated data analysis for powder x-ray diffraction using machine learning," Synchrotr. Radiat. News **35**(4), 9–15 (2022).

[26]C. A. M. Ramirez, M. Greenop, L. Ashton, and I. Rehman, "Applications of machine learning in spectroscopy," Appl. Spectrosc. Rev. **56**, 733–763 (2020).

[27]W. Zhang, L. C. Kasun, Q. J. Wang, Y. Zheng, and Z. Lin, "A review of machine learning for near-infrared spectroscopy," Sensors **22**, 9764 (2022).

[28]W. Fu and W. S. Hopkins, "Applying machine learning to vibrational spectroscopy," J. Phys. Chem. A **122**(1), 167–171 (2017).

[29]C. Kunka, A. Shanker, E. Y. Chen, S. R. Kalidindi, and R. Dingreville, "Decoding defect statistics from diffractograms via machine learning," npj Comput. Mater. **7**, 67 (2021).

[30]D. Vizoso, G. Subhash, K. Rajan, and R. Dingreville, "Connecting vibrational spectroscopy to atomic structure via supervised manifold learning: Beyond peak analysis," Chem. Mater. **35**(3), 1186–1200 (2023).

[31]D. Vizoso and R. Dingreville, "Dataset of simulated vibrational density of states and x-ray diffraction profiles of mechanically deformed and disordered atomic structures in gold, iron, magnesium, and silicon," Data Brief **55**, 110689 (2024).

[32]D. Vizoso and R. Dingreville, "Simulated vibrational density of states and x-ray diffraction profiles of mechanically deformed and disordered atomic structures in gold, iron, magnesium, and silicon," 2023, see https://doi.org/10.18126/tacz-v14v.
[33]A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," Comput. Phys. Commun. **271**, 108171 (2022).
[34]H. W. Sheng, M. J. Kramer, A. Cadien, T. Fujita, and M. W. Chen, "Highly optimized embedded-atom-method potentials for fourteen fcc metals," Phys. Rev. B **83**, 134118 (2011).
[35]S. Desai, M. Jain, S. J. Addamane, D. P. Adams, R. Dingreville, F. W. DelRio, and B. L. Boyce, "Navigating high-dimensional process-structure-property relations in nanocrystalline Pt-Au alloys with machine learning," Mater. Des. **248**, 113494 (2024).
[36]E. Natinsky, R. M. Khan, M. Cullinan, and R. Dingreville, "Reconstruction of high-resolution atomic force microscopy measurements from fast-scan data using a Noise2Noise algorithm," Measurement **227**, 114623 (2024).
[37]X. Zhao, Y. Luo, J. Liu, W. Liu, K. M. Rosso, X. Guo, T. Geng, A. Li, and X. Zhang, "Machine learning automated analysis of enormous synchrotron x-ray diffraction datasets," J. Phys. Chem. C **127**(30), 14830–14838 (2023).
[38]N. Niitsu, M. Mitani, H. Ishii, N. Kobayashi, K. Hirose, S. Watanabe, T. Okamoto, and J. Takeya, "Powder x-ray diffraction analysis with machine learning for organic-semiconductor crystal-structure determination," Appl. Phys. Lett. **125**, 013301 (2024).
[39]A. Pati and A. Lerch, "Latent space regularization for explicit control of musical attributes," in *Proceedings of the 36th International Conference on Machine Learning* (ICML, 2019).
[40]C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques," arXiv:1403.2877 (2014).
[41]R. Bro and A. K. Smilde, "Principal component analysis," Anal. Methods **6**(9), 2812–2831 (2014).
[42]F. R. S. Karl Pearson, "LIII. On lines and planes of closest fit to systems of points in space," Philos. Mag. J. Sci. **2**, 559–572 (1901).
[43]M. I. Latypov, M. Kühbach, I. J. Beyerlein, J.-C. Stinville, L. S. Toth, T. M. Pollock, and S. R. Kalidindi, "Application of chord length distributions and principal component analysis for quantification and representation of diverse polycrystalline microstructures," Mater. Charact. **145**, 671–685 (2018).
[44]M. Fernández-Delgado, M. S. Sirsat, E. Cernadas, S. Alawadi, S. Barro, and M. Febrero-Bande, "An extensive experimental survey of regression methods," Neural Netw. **111**, 11–34 (2019).
[45]M. Thompson, "Regression methods in the comparison of accuracy," Analyst **107**, 1169–1180 (1982).
[46]T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Linear Methods for Regression (Springer, 2009), pp. 43–99.
[47]T. P. Ryan, *Modern Regression Methods* (Wiley, 2008).
[48]D. S. Young, *Handbook of Regression Methods* (Chapman and Hall/CRC, 2017).
[49]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: an imperative style, high-performance deep learning library," CoRR, abs/1912.01703 (2019).
[50]I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," arXiv:1711.05101 (2019).
[51]D. P. Kingma, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2017).
[52]T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2019).
[53]Neptune Team, neptune.ai, 2019.
[54]L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com.
[55]F. Oviedo, Z. Ren, S. Sun, C. Settens, Z. Liu, N. T. P. Hartono, S. Ramasamy, B. L. DeCost, S. I. P. Tian, G. Romano *et al.*, "Fast and interpretable classification of small x-ray diffraction datasets using data augmentation and deep neural networks," npj. Comput. Mater. **5**(1), 60 (2019).
[56]Y. Suzuki, H. Hino, T. Hawai, K. Saito, M. Kotsugi, and K. Ono, "Symmetry prediction and knowledge discovery from x-ray diffraction patterns using an interpretable machine learning approach," Sci. Rep. **10**(1), 21790 (2020).
[57]J. Venderley, K. Mallayya, M. Matty, M. Krogstad, J. Ruff, G. Pleiss, V. Kishore, D. Mandrus, D. Phelan, L. Poudel *et al.*, "Harnessing interpretable and unsupervised machine learning to address big data from modern x-ray diffraction," Proc. Natl. Acad. Sci. U.S.A. **119**(24), e2109665119 (2022).
[58]K. Kaufmann, H. Lane, X. Liu, and K. S. Vecchio, "Efficient *few-shot* machine learning for classification of EBSD patterns," Sci. Rep. **11**(1), 8172 (2021).
[59]L. Wu, S. Yoo, A. F. Suzana, T. A. Assefa, J. Diao, R. J. Harder, W. Cha, and I. K. Robinson, "Three-dimensional coherent x-ray diffraction imaging via deep convolutional neural networks," npj. Comput. Mater. **7**(1), 175 (2021).
[60]B. Boyce, R. Dingreville, S. Desai, E. Walker, T. Shilt, K. L. Bassett, R. R. Wixom, A. P. Stebner, R. Arroyave, J. Hattrick-Simpers, and J. A. Warren, "Machine learning for materials science: Barriers to broader adoption," Matter **6**(5), 1320–1323 (2023).
[61]J. E. Fowler, M. A. Kottwitz, N. Trask, and R. Dingreville, "Beyond combinatorial materials science: The 100 prisoners problem," Integrating Mater. Manuf. Innovation **13**(1), 83–91 (2024).
[62]J. R. Hattrick-Simpers, B. DeCost, A. G. Kusne, H. Joress, W. Wong-Ng, D. L. Kaiser, A. Zakutayev, C. Phillips, S. Sun, J. Thapa, H. Yu, I. Takeuchi, and T. Buonassisi, "An open combinatorial diffraction dataset including consensus human and machine learning labels with quantified uncertainty for training new machine learning models," Integrating Mater. Manuf. Innovation **10**(2), 311–318 (2021).
[63]H. Yang, Z. Wu, K. Zhang, D. Wang, and H. Yu, "Uncertainty quantification on small angle x-ray scattering measurement using Bayesian deep learning," J. Appl. Phys. **136**, 143101 (2024).
[64]S. S. Al-Jaroudi, A. Ul-Hamid, A.-R. I. Mohammed, and S. Saner, "Use of x-ray powder diffraction for quantitative analysis of carbonate rock reservoir samples," Powder Technol. **175**, 115–121 (2007).
[65]W.-T. Jiang, D. R. Peacor, P. Árkai, M. Tóth, and J. W. Kim, "TEM and XRD determination of crystallite size and lattice strain as a function of illite crystallinity in pelitic rocks," J. Metamorphic Geol. **15**, 267–281 (1997).
[66]B. N. Hupp and J. J. Donovan, "Quantitative mineralogy for facies definition in the Marcellus Shale (Appalachian Basin, USA) using XRD-XRF integration," Sediment. Geol. **371**, 16–31 (2018).
[67]C. Fabre, "Advances in laser-iinduced breakdown spectroscopy analysis for geology: A critical review," Spectrochim. Acta Part B: At. Spectrosc. **166**, 105799 (2020).
[68]D. G. Henry, I. Jarvis, G. Gillmore, and M. Stephenson, "Raman spectroscopy as a tool to determine the thermal maturity of organic matter: Application to sedimentary, metamorphic, and structural geology," Earth-Sci. Rev. **198**, 102936 (2019).
[69]G. Piga, A. Santos-Cubedo, S. M. Solá, A. Brunetti, A. Malgosa, and S. Enzo, "An x-ray diffraction (XRD) and x-ray fluorescence (XRD) investigation in human and animal fossil bones from Holocene to middle triassic," J. Archaeol. Sci. **36**, 1857–1868 (2009).
[70]D. Bersani, C. Conti, P. Matousek, F. Pozzi, and P. Vandenabeele, "Methodological evolutions of Raman spectroscopy in art and archaeology," Anal. Methods **8**, 8395–8409 (2016).
[71]J. C. Hiller, M. J. Collins, A. T. Chamberlain, and T. J. Wess, "Small-angle xray scattering: A high-throughput technique for investigating archaeological bone preservation," J. Archaeol. Sci. **31**, 1349–1359 (2004).
[72]A. Peacock and J. Ellwood, "The high throughput x-ray spectroscopy mission: XMM," Space Sci. Rev. **48**, 343–365 (1988).

[73]J. P. Maillard, L. Drissen, F. Grandmont, and S. Thibault, "Integral wide-field spectroscopy in astronomy: The imaging FTS solution," Exp. Astron. **35**, 527–559 (2013).
[74]A. Kremin, S. Bailey, J. Guy, T. Kisner, and K. Zhang, "Rapid processing of astronomical data for the dark energy spectroscopic instrument," in *2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)* (IEEE, 2020), pp. 1–9.
[75]S. Pahlow, K. Weber, J. Popp, B. R. Wood, K. Kochan, A. Rüther, D. Perez-Guaita, P. Heraud, N. Stone, A. Dudgeon, B. Gardner, R. Reddy, D. Mayerich, and R. Bhargava, "Application of vibrational spectroscopy and imaging to point-of-care medicine: A review," Appl. Spectrosc. **72**, 52–84 (2018).
[76]L. E. Jamieson and H. J. Byrne, "Vibrational spectroscopy as a tool for studying drug-cell interaction: Could high throughput vibrational spectroscopic screening improve drug development?" Vib. Spectrosc. **91**, 16–30 (2017).
[77]D. Cozzolino, "Infrared methods for high throughput screening of metabolites: Food and medical applications," Comb. Chem. High Throughput Screening **14**, 125–131 (2011).