ChatGPT

Models for NIR Spectral Prediction and Preprocessing Strategies

Traditional Machine Learning Models (Scikit-Learn & Ensembles)

- **Linear Regression (OLS) & Regularized Regression (Ridge, Lasso, ElasticNet)**: Ordinary least squares fails when NIR spectra have highly collinear features (common due to overlapping absorption bands) [1]. Ridge regression mitigates multicollinearity by shrinkage, and Lasso performs feature selection, but **all linear models require careful preprocessing**. Mean-centering each wavelength (to remove constant offsets) is essential, and scaling features to unit variance is recommended so that no single wavelength dominates due to larger scale [2]. Baseline offsets and scatter must be corrected (e.g. with Standard Normal Variate *SNV* or Multiplicative Scatter Correction *MSC*) before fitting, otherwise the model will devote parameters to those irrelevant variations. Aggressive spectral derivatives that amplify high-frequency noise can hurt linear models – they may overfit the noise due to their unbounded linear nature. Instead, mild smoothing (e.g. Savitzky–Golay filtering) and first-derivative (to remove baseline drift) often improve linear regression performance by focusing on broad absorbance features while reducing noise [3].

- **Principal Component Regression (PCR)**: PCR combines PCA for dimensionality reduction with a linear regression on the principal components. This addresses multicollinearity by retaining only the major variance directions. However, because PCA is unsupervised, the top components may not be the most predictive [4] – e.g. the first principal component might capture overall light scatter or water content variance that isn't relevant to the target. **Preprocessing is crucial:** one should mean-center (and often autoscale) the spectra before PCA so that large absorption ranges don't dominate the first components [5]. It's also wise to apply baseline corrections (like detrending or derivatives) prior to PCA, so that principal components represent chemical signal rather than just baseline shifts. PCR is less sensitive to noise than OLS (since noise usually ends up in lower-variance PCs that can be dropped), but heavy noise in high-variance regions can still corrupt the first PCs – thus, spectral smoothing (e.g. Savitzky–Golay) before PCA can yield more robust components. In summary, PCR works well if informative spectral variance is extracted, but inappropriate preprocessing (or none at all) can cause PCR to model wrong factors (e.g. uncorrected scatter or outlier variance).

- **Partial Least Squares (PLS) Regression & Variants (PLS-DA, OPLS, Local PLS, etc.)**: PLS is a **widely used method in chemometrics for NIR** because it tackles multicollinearity by extracting latent factors that maximize covariance with the target [1] [6]. It essentially finds spectral feature combinations most correlated with the property of interest, making it very data-efficient. **Proper preprocessing boosts PLS performance:** NIR spectra are usually mean-centered before PLS, and scatter/baseline effects are removed via transformations like MSC or SNV, since PLS can otherwise waste components modeling those variations. Moderate Savitzky–Golay smoothing and using the first derivative are common: for example, applying SG smoothing, SNV, and a first derivative prior to PLS yielded accurate soil organic matter prediction [7]. These steps emphasize sharp spectral features and correct baseline drift, which helps PLS focus on true chemical signals. However, overly aggressive preprocessing (e.g. a second derivative with an excessively short window that amplifies noise) can hurt PLS because the method, while filtering

out random noise to some extent, is still limited by the number of factors – too much noise makes it harder to extract meaningful latent factors. Variants like **PLS-DA** (PLS Discriminant Analysis) apply PLS for classification by modeling class labels; they benefit from the same preprocessing as PLS regression (scatter correction, centering, etc.) to ensure class differences aren't masked by extraneous variance. **Orthogonal PLS (OPLS)** explicitly separates variation correlated with the target from orthogonal (uncorrelated) variation, effectively filtering out structured noise like light scatter [8]. OPLS thus can reduce the need for extensive external preprocessing – but it still assumes you've done basics like centering and perhaps SNV to handle gross scaling differences. **Local PLS (L-PLS) and Locally Weighted PLS (LW-PLS)** address global non-linearities: they build PLS models on a neighborhood of spectra similar to the query sample, or weight samples by spectral similarity. For these local models, **feature scaling and normalization are critical** – distance metrics used to find neighbors work best if spectra are on a comparable scale and free of offsets. Thus, one would apply standard NIR pretreatments (SNV, etc.) so that *distance* reflects genuine shape similarity. Local PLS approaches can capture non-linear behavior, but excessive noise or irrelevant regions will impair distance calculations, so mild smoothing and possibly wavelength selection (to exclude uninformative regions) improve their reliability. There are even **non-linear PLS** techniques (e.g. Kernel PLS) that use kernel transformations to capture curved relationships. These essentially behave like a non-linear SVR under the hood, so they demand similar preprocessing: features should be scaled and centered (the RBF or polynomial kernels assume features of similar magnitude [2]), and one must avoid feeding in pure noise. In summary, PLS and its many variants are robust calibration workhorses for NIR as long as you remove scatter, baseline shifts, and unnecessary noise beforehand – these methods will then excel at extracting the spectral information relevant to your prediction task.

- **Linear Discriminant Analysis (LDA/QDA)**: LDA and QDA are classical classification techniques that assume Gaussian class distributions. They have been used in NIR (often in tandem with dimensionality reduction like PCA or PLS) for tasks like authenticity or variety classification. **Preprocessing for LDA/QDA** is geared toward making the data meet assumptions and avoid ill-conditioning. Typically, one must drastically reduce the feature count (since a full covariance matrix in thousands of dimensions is singular with typical NIR sample sizes). In practice, a common pipeline is PCA or PLS-DA to extract a handful of components, then LDA on those. Those components (or the selected original wavelengths) should be mean-centered and scaled before LDA [5] – this ensures that no single feature (or principal component) variance dominates the covariance estimate. Moreover, any systematic differences not related to class (e.g. overall intensity shifts between batches) should be removed beforehand (using methods like SNV or baseline subtraction); otherwise LDA might separate classes based on those shifts rather than the chemical differences of interest. LDA is fairly sensitive to outliers, so spectral outlier removal or robust smoothing can improve it. QDA (quadratic DA) allows each class its own covariance – requiring even more caution with high-dimensional data. Heavy noise in the spectra will violate the Gaussian assumption and inflate covariance estimates, so smoothing is beneficial. In summary, LDA/QDA can work for NIR classification if preceded by dimension reduction and preprocessing; without those, the high dimensionality and collinearity of raw spectra would make LDA break down.

- **k-Nearest Neighbors (k-NN)**: k-NN can be applied to spectra for both regression (e.g. predicting a concentration by averaging nearest neighbors' values) or classification (assigning the class most common among neighbors). It's simple but **very sensitive to the distance metric**, so preprocessing heavily influences its success. **Feature scaling is critical** – distances in the high-dimensional spectral space must not be skewed by different scales [5]. Since all wavelengths have comparable units (absorbance or reflectance), one might think scaling is unnecessary; however, some spectral regions naturally exhibit larger variance (e.g. water absorption bands)

while others are very stable. Without scaling, a region with large variance (not necessarily related to the target) can dominate the Euclidean distance. Thus, one typically standardizes each wavelength (zero mean, unit variance) across the dataset or applies SNV to each spectrum so that all spectral features contribute more evenly [9]. Additionally, **scatter correction and baseline removal** are important for k-NN: if two spectra differ only by a multiplicative factor or baseline offset, uncorrected they will appear distant even though their shapes are similar. Techniques like MSC or SNV will make spectra with the same shape (but different pathlength or scattering) actually close in distance. Derivative preprocessing can also help by emphasizing shape and reducing constant offsets—e.g. using a first derivative makes the distance metric focus on peak positions/slopes rather than absolute level. However, one should not use an overly noisy derivative: k-NN has no internal noise robustness, so every tiny spectral fluctuation contributes to distance. A smoothed first-derivative spectrum often works well: it captures the salient spectral features for comparison while softening noise. **Dimensionality reduction** is also often needed for k-NN due to the "curse of dimensionality." Thousands of highly correlated wavelengths make distance calculations less meaningful (neighbors tend to all be at similar distances in very high dimensions). Reducing dimension via PCA (unsupervised) or better yet via supervised methods (selecting informative wavelengths or using PLS to get a few latent variables) can dramatically improve k-NN performance [10]. For instance, selecting key wavelength bands known to relate to the target (or using an algorithm like genetic algorithms or tree-based importance to pick bands) yields a distance metric that is more relevant. In summary, k-NN works for spectral data only with thorough preprocessing: **normalize scales, remove irrelevant variation (scatter, baseline), smooth noise, and reduce dimensionality** so that "nearest neighbors" in the resulting space genuinely correspond to similar samples in terms of chemical or class characteristics.

- **Support Vector Machines (SVM/SVR)**: SVMs (for classification) and SVRs (for regression) are powerful non-linear models that have been applied to NIR spectra with success, provided the data are preprocessed appropriately. The **RBF kernel SVM/SVR**, in particular, can model complex spectral–property relationships. **Feature scaling is mandatory for SVMs** [9]. Since the RBF kernel and other kernels compute distances in feature space, large-valued features (e.g. an absorbance band with higher numeric range) will dominate the kernel similarity if not scaled. Indeed, not scaling features can completely change which points are considered similar by the SVM [9]. Thus, one should standardize all wavelengths to comparable ranges (e.g. unit variance) before SVM modeling. Additionally, centering (zero mean) each feature is beneficial because many kernels (and regularization schemes) implicitly assume data centered around 0 [2]. **Traditional NIR preprocessors** often boost SVM performance: for example, applying Savitzky–Golay smoothing and a first derivative to spectra, followed by an RBF SVR, is a common approach to remove baseline drifts and emphasize spectral peaks [11]. These preprocessings help because SVM will struggle if the target-relevant signal is buried under baseline shifts or scaling variations. By using a derivative or SNV, one ensures the SVM focuses on spectral shape differences that correlate with the target, rather than on absolute intensity differences that might just be due to scatter. One should be cautious with very high-order derivatives or aggressive noise-enhancing steps, as SVM (especially with a large kernel width) can fit to noise patterns; a balance (mild derivative + smoothing) works best. Because SVMs are memory-based (store support vectors) and don't inherently perform feature selection, **dimension reduction or feature selection is wise** for large spectral datasets. Combining SVM with something like recursive feature elimination or using PLS scores as inputs can yield better results [12]. For instance, a study might use a genetic algorithm to select optimal wavelengths, then train an SVM, which has shown improved accuracy over using the full spectrum [12]. In summary, SVM/SVR can achieve excellent accuracy on NIR data – often outperforming PLS – but only if the spectra are properly normalized, corrected for scatter/baseline, denoised, and stripped down to

the most informative features. Feeding raw, unscaled spectra into an SVM will usually prevent it from learning effectively.

- **Decision Trees** (CART): A single decision tree can be used for NIR regression or classification, though it's more common as a component of ensemble methods. Trees are **invariant to monotonic scaling** of features [13] – meaning you do *not* need to standardize or normalize wavelengths for a tree model to split on them. In fact, per-wavelength standardization is generally *not* done for tree-based models and can be counterproductive. For example, if a particular wavelength has almost no variance (flat across samples) except a tiny bit of noise, standardizing that feature will inflate the noise to a unit variance scale, making that noise appear important; a tree could then erroneously split on random noise fluctuations. Keeping spectra in their original scale preserves the true relative variances, allowing the tree's splitting criterion to naturally focus on the more informative wavelengths. That said, basic preprocessing can still help trees: **denoising and outlier removal** will make a tree's splits more stable. A decision tree greedily partitions based on single features; if high-frequency noise causes random large jumps in absorbance at some wavelength, the tree might fixate on those as split points. Smoothing the spectra (e.g. with a Savitzky–Golay filter) thus helps by removing spikes that could lead to overfitting splits. Likewise, removing or down-weighting spectral outliers (samples with anomalous spectra) can prevent a tree from dedicating branches to peculiar noise patterns. Another consideration is that an unpruned tree in a high-dimensional spectral space can overfit badly – it may keep branching on tiny variations in different wavelengths. Imposing some preemptive dimensionality reduction (like selecting a subset of wavelength bands or using an informed feature selection) can reduce the propensity to overfit. In short, a standalone tree doesn't require scale normalization (and one should avoid per-feature standardization [13]), but it does benefit from spectral smoothing and perhaps focusing the input features on known informative regions. Usually, however, single trees are not as accurate or stable on NIR data, which is why ensemble methods are preferred.

- **Random Forest (RF)**: Random forests have become a popular non-linear method for NIR spectra. An RF is an ensemble of decision trees and inherits the tree's advantages and limitations. **Preprocessing for RF** should leverage the model's insensitivities while mitigating its vulnerabilities. Like single trees, RFs do not require feature scaling – they handle arbitrary feature ranges internally by thresholding on values [13]. Indeed, it's often recommended *not* to standardize each wavelength for an RF; doing so could amplify irrelevant noise variance in minor spectral regions, as discussed above. RFs *can* handle high-dimensional input, but they **suffer if many features are pure noise or highly collinear**. NIR spectra have many correlated wavelengths conveying redundant information; an RF will try splits on many of them, essentially performing similar splits repeatedly, which can reduce overall accuracy and inflate the apparent importance of noise. Therefore, **spectral feature reduction** is helpful. A key point is that RF splitting is easier to interpret (and often more effective) on actual wavelength features rather than on abstract principal components. While one could feed PCA scores into an RF, it's often better to perform *feature selection or spectral binning* for dimensionality reduction. For example, one might select a subset of important wavelengths (using domain knowledge or even the RF's own feature importance ranking) or bin average the spectrum into broader bands. Such approaches retain physical interpretability (you know which spectral region is used) and can improve RF performance by removing redundant features [10]. In contrast, PCA transformations create new composite features, and a tree splitting on a principal component is less transparent and can be less effective if the relationship to the target isn't strictly along major variance directions. Empirically, RF often works well with *band selection* + *smoothing* rather than PCA compression – since the trees can capture non-linear interactions between original spectral regions that might get obscured in PCA space. **Noise and derivatives:** RF is relatively robust to

moderate noise due to averaging many trees, but it's not immune. Very noisy inputs will cause each tree to pick up spurious splits occasionally, adding variance to the ensemble. Smoothing the spectra (e.g. Savitzky–Golay or other denoising) can therefore improve RF predictions by eliminating high-frequency noise that would otherwise lead to unstable splits. Conversely, applying a harsh second derivative that amplifies noise is detrimental – it could turn a slight baseline variation into wildly fluctuating values that confuse the model. If derivative preprocessing is desired (to remove baseline or emphasize peaks), it should be done with a sufficient smoothing window to control noise amplification. Many practitioners prefer the first derivative for RF (with smoothing) and avoid the noisier second derivative unless the data is very clean. **Scatter correction:** Methods like MSC or SNV, which remove multiplicative and additive effects, can benefit RF models as well. Although trees can in theory learn to ignore a global offset (by finding optimal split points that are invariant to an added constant), in practice scatter effects can introduce lots of small intensity differences across the spectrum that don't correlate to the target. Removing those via MSC/SNV can simplify what the RF has to learn. For instance, applying MSC to NIR spectra of tablets can flatten baseline shifts; an RF can then focus on differences in specific absorption bands related to composition, rather than splitting on overall brightness differences. Overall, Random Forests are powerful for NIR (capturing some non-linearity and interaction effects), but **they perform best with gentle preprocessing**: avoid arbitrary scaling, do apply noise reduction and baseline/scatter corrections, and reduce feature redundancy by band selection or similar. This maximizes the signal-to-noise that each tree sees, while the ensemble structure handles the remaining variance to yield a robust model.

- **Boosted Tree Ensembles (Gradient Boosting, XGBoost, LightGBM, CatBoost)**: Gradient boosting machines build an ensemble of trees in a stage-wise fashion, often yielding higher accuracy than RF by reducing bias, at the cost of being somewhat more prone to overfitting noise. Popular implementations like **XGBoost**, **LightGBM**, and **CatBoost** have all been applied to NIR datasets. The **preprocessing needs for boosting** are similar to those for RF. Tree-based boosters are likewise invariant to feature scale, so **standardization of wavelengths is not required** (and again can be avoided to not magnify tiny variances) [5] . However, boosting models will iteratively fit residuals, and they can **chase noise** if the data is not cleaned – even more so than RF, since boosting continuously refines on errors. For that reason, careful noise reduction (smoothing) and outlier removal pay dividends for boosted trees. For example, if an outlier spectrum has an anomalous spike at some wavelength, an RF might isolate that with one tree out of many (damping its effect), but a boosting model might devote several trees to try and fix the residual on that outlier, leading to overfit behavior. Removing or down-weighting outliers and using smoothed spectra thus improves generalization for boosting. **Dimensionality reduction** is also important: Boosting algorithms like XGBoost/LightGBM have built-in techniques (e.g. column subsampling) to handle many features, but feeding thousands of highly correlated wavelengths still increases model complexity and training time. Focusing on informative spectral regions (through prior knowledge or using feature importance from a preliminary model) can simplify the task. Moreover, some boosted models (XGBoost, LightGBM) allow setting an `feature_fraction` or similar to randomly ignore some features in each iteration, which helps decorrelate features and can mitigate the worst effects of multicollinearity. Nonetheless, giving the model fewer, more meaningful features (like key band ratios or averages) often boosts performance and reduces overfitting. **Scatter correction and baseline** adjustments are useful here as well – a boosting model, like any model, can be misled by multiplicative/additive effects that aren't related to the target. If, say, one batch of samples is overall darker and the target property doesn't actually depend on that, the booster might create splits early on separating batches (thinking it useful), thereby using up model capacity on irrelevant differences. Techniques like SNV, MSC, or detrending remove those effects so the booster can focus on the spectral differences that matter. One difference from RF: boosted trees

tend to be shallower (often each tree is limited to a small depth). This means each individual tree might capture only a simple relation (like “if peak height at 1500 nm > X then …”). If the data still has an underlying baseline shift, a shallow tree might not effectively handle it (unlike a deep tree that could carve out a complex rule). Pre-correcting baselines ensures even shallow trees can split on meaningful variations. **CatBoost** has a speciality in handling categorical variables and reducing the need for manual one-hot encoding – not particularly needed for pure spectral data (which are continuous features), but it also has an ordered boosting scheme that can reduce overfit. In NIR contexts, CatBoost, XGBoost, and LightGBM behave similarly if given the same input; all three will benefit from the same preprocessing steps: *don’t scale features unnecessarily, do remove noise and redundant features*. Empirically, these boosting methods have achieved excellent results on certain NIR tasks (such as predicting soil properties or detecting product adulteration), often rivalling PLS in accuracy. They may require tuning of regularization hyperparameters when spectra are high-dimensional, and preprocessing that cuts down on irrelevant variation makes that tuning easier. In summary, boosted tree ensembles are high-performance models for NIR **if** the spectra are pretreated to be as informative and noise-free as possible – smoothing, scatter correction, and thoughtful feature reduction will prevent the booster from overfitting spurious signals.

- **TabPFN (Tabular Prior-Data Fitted Network)**: *TabPFN* is a recent “foundation” model for tabular data that can be applied to NIR predictions. It’s essentially a pre-trained transformer that performs probabilistic inference on small datasets without gradient-based training on the user’s data. One major advantage is that TabPFN comes pre-loaded with inductive biases from vast simulated data, so it often works out-of-the-box on new tabular prediction tasks. **Preprocessing for TabPFN** is minimal by design – the model internally performs z-score normalization on input features [14], so you do **not** need to manually standardize or normalize the spectral features (scaling the inputs yourself will not change the results). In fact, the authors note that TabPFN normalizes all inputs such that further scaling has no effect on predictions [14]. That said, “garbage in, garbage out” still applies. You should ensure the spectra are sensible and informative for the task. TabPFN can handle numerical features well, but if your NIR data has known irrelevant regions (e.g. regions of pure noise or instrument artifact), it’s wise to remove or mask them – the model will not magically know those features are meaningless unless it figures it out from the data. Using **domain knowledge to combine or filter features can improve performance** even for TabPFN [14]. For example, if only certain wavelength ranges contain signals of interest, restricting the input to those ranges can help the model focus, especially since TabPFN has a fixed capacity. The model is robust to moderate multicollinearity (its transformer layers can learn dependencies between features), but extremely correlated and high-dimensional inputs can slow down its inference. It may therefore be beneficial to perform some dimensionality reduction if the number of wavelengths is huge relative to sample count – e.g. binning spectral channels or using PCA to compress very highly correlated bands (bearing in mind that you’d then have to feed those PCs into TabPFN, which is fine as long as they are normalized). One must also be careful with outliers: TabPFN’s predictions are based on patterns it learned in pre-training; if your NIR data has out-of-distribution outliers (e.g. a very noisy or corrupted spectrum), it’s uncertain how the model will handle it. It could give an unreliable prediction for such points. Thus, basic outlier detection and smoothing can be useful before feeding data to TabPFN. **In summary**, TabPFN greatly simplifies modeling (no need for tuning or model selection), and it already takes care of scaling internally. Your main job in preprocessing is to ensure the spectral inputs are clean and relevant – e.g. apply standard chemometric corrections (baseline, scatter) if those distortions are present, remove obvious noise/artifact regions, and possibly reduce feature count if it’s unnecessarily large. With that, TabPFN can leverage its learned prior to yield accurate predictions on surprisingly small NIR datasets [15].

Neural Network Models (Deep Learning Approaches)

- **Multi-Layer Perceptron (Fully-Connected Neural Network)**: An MLP is a generic feed-forward neural network with one or more hidden layers of neurons. In principle, an MLP can learn any nonlinear mapping from spectra to target given enough neurons and data. In practice, **MLPs require careful preprocessing and often struggle with high-dimensional spectral data**. Because every input wavelength connects to every neuron, having thousands of highly correlated inputs (typical in NIR) makes the network's job difficult – the network might waste capacity learning the same pattern multiple times. **Feature normalization is a must** for MLPs. Typically one standardizes all inputs to zero-mean, unit-variance (or normalizes to [0,1]) so that the initial weights see a roughly consistent scale [16]. If one input wavelength has values an order of magnitude larger than others, it will dominate the gradient updates and lead to an imbalance in learning. Indeed, unscaled features can cause the MLP's optimization to stall or converge to a poor solution. Therefore, you should apply global scaling (e.g. z-score across the dataset) to the spectral inputs. Additionally, mean-centering each spectrum (removing its average absorbance) can help the network focus on the shape of the spectrum rather than its absolute level – similar to how SNV works in chemometrics. In fact, many chemometricians use SNV or detrending on spectra before feeding them to an ANN, to eliminate baseline offsets and multiplicative effects that the network would otherwise have to waste neurons on modeling. **Dimensionality reduction** or feature selection is strongly recommended for MLPs when data is limited. An MLP doesn't inherently know which wavelengths are important, and with many inputs it can easily overfit, especially since NIR datasets might only have a few hundred samples. Using techniques like PCA (on training data only) to reduce input dimensions, or selecting a subset of informative wavelengths (perhaps via PLS loading weights or variable importance from another model), can improve MLP performance. By reducing input noise and redundancy, you make the network's task easier. However, one must then ensure the transformed inputs are also normalized appropriately. **Noise handling:** Unlike tree-based models, a plain MLP has no special robustness to noise – if the spectral input is noisy, the MLP will try to fit it unless regularized. So, smoothing the spectra beforehand (or using noise-reduction autoencoders) can be beneficial. On the flip side, adding a bit of noise during training (noise augmentation) or using dropout is often used to regularize MLPs. But you want the input noise level to be reasonable; extreme high-frequency noise will just confuse the network. **Baseline and scatter effects:** If uncorrected, these can degrade an MLP's learning. For example, if some spectra are measured with more scatter (lower overall absorbance), a neural net without preprocessing might erroneously learn that overall lower absorbance corresponds to a certain output (because it sees that correlation in raw data), rather than focusing on spectral shape differences. Techniques like MSC could be applied prior to the MLP to remove that confounding effect. In summary, a fully-connected neural net can model NIR spectra given proper care: you should scale and center the inputs, remove or reduce uninformative wavelengths (to control dimensionality), and apply the usual NIR preprocessing (baseline correction, etc.) to expose the relevant information. Without these steps, an MLP would need a very large number of neurons and an enormous training set to "discover" the correct spectral features on its own, and it would be prone to overfitting noise or collinear features.

- **Convolutional Neural Networks (CNN)**: CNNs are well-suited to NIR spectra because they exploit the local structure of data. In a 1D CNN for spectral data, convolutional filters (kernels) slide over the wavelength axis to pick out local patterns (e.g. the shape of an absorption peak) [17]. This inductive bias makes CNNs powerful for spectroscopic analysis: they can automatically learn features like "a peak at 1210 nm" or "a rising slope around 1700–1750 nm" which might correspond to chemical bonds. **Preprocessing for CNNs** aims to present the network with spectra in a normalized, but not overly transformed, form. First, **feature scaling**: while a CNN's

convolution operation is also not scale-sensitive in the sense of unit changes (it will learn weights to match the scale of features), proper scaling still improves training convergence. Typically, one would normalize the entire spectrum to a certain range (for instance, many applications simply scale reflectance or absorbance values to 0–1 or standardize them to N(0,1) across the dataset) so that the initial network weights (often initialized small) can effectively combine features. If using a pre-trained CNN (say one adapted from image networks), matching the expected input scale (e.g. mean 0 and variance 1, or similar to ImageNet stats if 2D) is important. **Baseline correction and scatter:** A CNN can, in theory, learn to handle an additive baseline shift (the first convolutional layer could learn a constant filter to detect global offset), but in practice providing baseline-corrected spectra makes learning much easier. Empirically, researchers often still apply SNV or detrending prior to CNN input [18]. For example, one study designed "TeaNet" (a CNN for tea classification) and found that applying SNV to the NIR spectra before feeding them to the CNN yielded perfect classification accuracy [18]. The SNV removed variability between samples due to scatter, allowing the CNN to zero in on the spectral differences due to tea type. In general, **scatter correction + CNN is a good combination**: the correction handles global intensity differences, and the CNN then detects local spectral features without being confounded by overall shifts. **Smoothing/denoising**: CNNs have some ability to ignore noise (a convolutional filter that's larger than the noise scale can average it out), but if the noise is high-frequency and high-amplitude, it can still degrade performance. It's common to apply a mild smoothing filter (like Savitzky–Golay) to spectra before input to a CNN, to improve signal-to-noise. Too much smoothing, however, could eliminate sharp features that a CNN might otherwise catch; a balance is needed depending on SNR. **Derivative preprocessing**: If one uses a first or second derivative on spectra, CNNs can certainly work with that as input – in fact, a CNN could learn edge-detection filters that mimic a derivative. Some approaches feed both raw and derivative spectra as two-channel input images to a CNN, so the network gets both baseline-removed information (derivative) and absolute intensities. If only a derivative spectrum is fed, you must ensure the derivative is not so noisy as to confuse the CNN. A first derivative (with smoothing) is often fine and can help by flattening baselines, whereas second derivatives need careful smoothing. **Dimensionality**: CNNs can handle the full spectral resolution by using small kernels and pooling to progressively reduce feature length, so you don't necessarily need to down-sample or select wavelengths beforehand. However, if the spectral resolution is extremely high (say >2000 features) and your dataset is small, you might still consider downsampling or binning to reduce input length – this lowers model size and might slightly reduce overfitting risk. Notably, CNNs can take advantage of *data augmentation* in the spectral domain. For instance, one can augment training spectra by adding slight random noise, shifting spectra slightly (simulating small wavelength calibration offsets), or scaling intensity (simulating different pathlengths). These augmentations can improve generalization. A study using a VGG-style CNN on NIR spectra introduced an *extended multiplicative signal augmentation (EMSA)* – basically creating augmented spectra with varying scatter conditions – and found that it **accelerated convergence and improved results**, highlighting that the final model was strongly influenced by the preprocessing and augmentation used [3]. In summary, CNNs are very powerful for NIR: **provide them with scaled, baseline-corrected, and denoised spectra**, and they will often learn superior features. They are somewhat less reliant on manual feature selection, because the convolutional filters will automatically focus on the important regions (e.g. the network can learn to give near-zero weights to irrelevant wavelengths). Still, if you know certain spectral regions are useless (e.g. very noisy or outside the instrument's effective range), you should exclude or mask them to avoid wasting model capacity. Modern CNN architectures from computer vision can be adapted here – for example, 1D analogues of **ResNet or Xception** networks (which stack many convolutional layers with skip connections or depthwise separable convolutions [19]) could be used to capture very complex spectral patterns. These deep CNNs will similarly benefit from the same preprocessing: e.g. **Xception**, a high-performance image CNN, would typically expect

inputs scaled to a given range and perhaps normalized per channel; a 1D Xception applied to spectra should likewise get normalized spectral inputs and could be confused by raw, unstandardized values. In short, CNNs can reduce the need for manual feature engineering *if* the data is preprocessed to eliminate the easy nuisances (scaling, baseline, noise) – the CNN will handle the rest (local feature extraction and non-linear combination of those features) to model the target.

- **Recurrent Neural Networks (RNN) – LSTM/GRU**: RNNs treat the spectrum as a sequence of values, analogous to a time series. Models like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) can maintain a memory of prior wavelengths as they scan through the spectrum, potentially capturing long-range dependencies (e.g. relationships between an overtone band at lower wavelength and a combination band at higher wavelength). While RNNs are less commonly used than CNNs for pure spectral data, they have been explored, particularly if one imagines the spectrum as a 1D sequence to "read." **Preprocessing for RNNs** is again about making the sequence amenable to learning. **Scaling and normalization** are essential – an LSTM uses gates with sigmoid and tanh activations, which can saturate if inputs are large. Scaling each wavelength to roughly within [-1, 1] (through standardization or min-max normalization) will prevent the network from saturating and ensure the gradients flow nicely. Without scaling, an LSTM may, for example, see a raw absorbance of 2.0 (which might saturate a sigmoid gate towards 1) versus 0.2 at another wavelength; the difference in magnitude could lead the network to inadvertently give more "importance" to the larger value even if it's not actually more informative. **Mean centering** the spectra (or using SNV) can also be beneficial to remove global level differences; otherwise an RNN might simply learn that spectra with overall higher values have a certain output (if that correlation exists in raw data) – similar to the issues discussed for other models. By removing the mean or trend, the RNN can focus on the shape patterns as it steps through the sequence. **Handling baseline trends**: If a spectrum has a sloping baseline, an LSTM could in theory learn a slowly changing component to represent it, but that uses up part of its capacity. It's more efficient to remove such trends via preprocessing. For instance, if all spectra have a tilted baseline due to particle scattering, applying a first derivative or high-pass filter would eliminate that, so the LSTM doesn't have to model a big slowly-changing baseline in addition to rapid changes. **Noise and smoothing**: High-frequency noise is problematic for RNNs as well – it introduces rapid fluctuations in the sequence that an LSTM might try to memorize or will simply add to the state and make learning harder. Smoothing the spectral sequence (e.g. via a moving average or SG filter) can help the RNN focus on meaningful spectral variation. However, one must be cautious not to over-smooth and erase subtle features that an LSTM could catch. An advantage of RNNs is that they can integrate information over a longer range than a small CNN kernel, so an LSTM might detect a faint broad feature by accumulating evidence across wavelengths; if you smoothed too aggressively, that broad feature might vanish. Thus, moderate noise reduction is advised, potentially combined with data augmentation (like adding slight noise during training to improve robustness). **Sequence length reduction**: If the spectral resolution is very high (thousands of points), an RNN has to unroll through all those time steps. This can increase training time and also potentially cause vanishing gradients if the sequence is too long (despite LSTM's gating mechanisms). In such cases, one might downsample the spectra (for instance, take every 2nd or 4th point if fine detail is not crucial) or group wavelengths into bins (averaging a few neighboring wavelengths) to shorten the sequence length. This is akin to reducing dimensionality and can help the RNN concentrate on broader features. It's similar to how one might reduce the time-step resolution in a time series if fine granularity isn't adding information. Of course, any such reduction should preserve known important features (don't bin a narrow but important peak out of existence). **Combining with CNN**: In some approaches, CNNs and RNNs are combined – e.g. using a 1D CNN to extract local features, then an LSTM to capture global trends of those features across the spectrum [20]. In those cases, the CNN part

would require the same preprocessing as described for CNN (normalized, smoothed input), and the LSTM on top would then see a sequence of feature vectors. Such a hybrid was reported to improve prediction accuracy for soil properties [20], but it also highlighted that with fewer training samples the model's robustness dropped – implying that these deep models need a lot of data or strong regularization. Preprocessing like feature selection can help in low-data regimes by reducing what the RNN has to learn. For example, if only certain wavelength regions are useful, feeding an LSTM a sequence of just those regions (maybe concatenated or zero out the rest) effectively focuses the sequence model on relevant portions, improving learning with small N. Overall, for pure RNN models: **normalize the spectral sequence, remove global offsets/trends, filter out high-frequency noise**, and consider simplifying the sequence (either by known feature regions or slight downsampling) if the model complexity is too high for your data size. With these steps, an LSTM/GRU can capture spectral patterns and even interactions across distant wavelengths that simpler models might miss.

- **Transformer Models (Self-Attention Networks)**: Transformers, exemplified by the **Vision Transformer (ViT)** and various sequence models, use self-attention mechanisms to learn relationships in the data. In a vision context, ViT splits an image into patches and treats them as a sequence of tokens, then uses multi-head attention to model global relationships [21]. For NIR spectra, one can similarly treat each spectrum as a sequence of tokens (each token could be a single wavelength's value or a small group of wavelengths). A transformer can then learn attention weights that link, say, a combination band region with a fundamental band region if that relationship is predictive of the target. The main benefit is that **transformers can capture long-range dependencies** in one layer (whereas an RNN or CNN might need many layers or timesteps). However, transformers typically require large amounts of data to train from scratch, so they haven't been widely used on small NIR datasets *without* transfer learning. **Preprocessing for transformer models** is similar to that for other neural nets: inputs should be appropriately scaled and normalized. In fact, many transformer implementations (including ViT) incorporate normalization layers, but it's still wise to feed in data that is zero-centered and scaled to a sensible range to avoid numerical extremes. For spectral data, one might split the input into "patches" (segments of contiguous wavelengths) before feeding to a vision transformer – in that case, one could normalize each patch or the whole spectrum. If using each wavelength as a token in a sequence model (like a standard text-like transformer), definitely scale the features: a large raw absorbance could create large query/key dot products and potentially destabilize training unless scaled (transformers usually rely on scaling factors and layernorm to cope, but it's best to start with normalized inputs). **Positional encoding** is another aspect: transformers themselves are permutation-invariant and need position information injected. For spectra, position corresponds to wavelength order. One would typically add a positional encoding (e.g. a sinusoidal or learnable positional vector) to each wavelength token to inform the model about the spectral order. If there's any non-uniform spacing in wavelengths, that should be encoded as well (usually NIR data is uniformly spaced, so a standard positional encoding works). This isn't a preprocessing per se, but a part of model input preparation. Now, regarding **preprocessing effects:** If a spectrum has a large baseline offset or tilt, a transformer has the capacity to learn that pattern if it's common (attention can attend to an overall offset token, etc.), but it's not guaranteed to prioritize handling baseline shifts unless they interfere with predicting the target. Just as with other models, feeding baseline-corrected spectra can make the transformer's job easier by eliminating a nuisance parameter. Similarly, scatter effects that multiply the spectrum might confound a transformer – because that would scale multiple tokens in a correlated way, and the model would have to learn to discount that correlation. Methods like MSC would remove those multiplicative effects, yielding spectra where each token's value is more directly tied to chemical composition rather than sample presentation. Given enough data, a transformer could theoretically figure out an MSC-like transformation (it has the capacity to model such linear

relationships across tokens), but with limited data it likely won't. So it's wise to do those standard corrections beforehand. **Noise**: Transformers don't have an inherent smoothing ability (no convolutional filtering), so if the spectral data is very noisy, the self-attention might latch onto noise patterns (e.g. attending between two random spikes). Some transformers include embedding layers that might average or project multiple features, but if working token-by-token, noise is directly in the token values. Thus, denoising the spectra (via smoothing or even using autoencoder pre-training) can be important. On the flip side, transformers can be regularized by adding noise during training (like dropout in attention or token mixing), but again, you don't want persistent noise in inputs that isn't informative. **Dimensionality**: A challenge is that attention's complexity grows with sequence length. If you have 1000+ wavelength tokens, that's 1000×1000 attention matrix – manageable, but if you have very high resolution or extended ranges, it becomes heavier. It might make sense to reduce the input length (via binning) as a preprocessing step. This not only eases computation but can reduce overfitting by removing superfluous fine detail. For example, you could average every 4 adjacent wavelengths to get 1 "token" with a mean value (losing some resolution but smoothing noise and cutting sequence length by 4). As long as this doesn't erase critical features, the transformer may actually perform better focusing on broader tokens. **Foundation models and transfer learning**: One exciting possibility is using large pre-trained models. For instance, one could use a pre-trained ViT (trained on images) by representing a spectrum as a trivial 1×N "image" or as an image of its scatter plot (though that's not common). More directly, if a transformer model were pre-trained on a large repository of spectra (perhaps via self-supervised learning like masking some spectral regions and predicting them), it could serve as a foundation model for NIR. In absence of that, people have tried transfer learning from models like Xception or ResNet (pre-trained on ImageNet) by treating spectra as 1D or 2D images. If doing so, **adhering to the original model's expected preprocessing is crucial**. For example, Xception expects images scaled in a certain way (typically 0-1 then normalized per channel to a specific mean/std). If you feed a spectrum as a single-channel image to Xception, you should similarly scale that "image" channel to the range the model was trained on (and perhaps tile or interpolate the spectrum to a size the CNN expects). Some studies have taken NIR spectra, plotted them as a curve or as a small spectrogram-like image, and then used a deep CNN like ResNet via transfer learning. In those cases, they often convert the spectral data to a grayscale image and apply the same normalization as for natural images. Results can be surprisingly good, but it underscores how important matching the preprocessing is – a model trained on photography images won't handle raw spectral values without re-normalization. Lastly, specific large architectures like **Vision Transformer (ViT)** themselves have been successfully applied in spectroscopy research in a limited way. For instance, one could patch the spectrum (divide into segments) and use a ViT to classify a material; this was shown to work for complex spectral datasets by capturing global patterns via attention. These models achieved performance on par with CNNs, with the advantage of potentially better capturing inter-band relationships. But they required more data to train. In such works, standard spectral preprocessing (MSC, SNV, derivatives) was still used before feeding data to the transformer – confirming that domain-specific preprocessing remains valuable even when using cutting-edge model architectures [22]. In summary, transformers hold promise for NIR because they can flexibly model interactions between distant spectral regions. To use them effectively: **normalize and scale the spectral inputs**, apply **baseline and scatter corrections** so those global perturbations don't distract the attention mechanism, consider **denoising or tokenizing the spectrum into slightly larger patches** to reduce noise and sequence length, and include positional information about wavelength. With those steps, a transformer (even a pre-trained one like a ViT) can be fine-tuned on spectral data and potentially outperform more rigid models by leveraging its powerful pattern-matching across the entire spectrum.

- **Notable Deep Learning Architectures (ResNet, Xception, ViT, etc.)**: It's worth mentioning some *reference architectures* and how they relate to NIR data as special cases of the classes above. **ResNet** (a CNN with skip connections) and **Xception** [19] (an Inception-inspired CNN using depthwise separable convolutions) are state-of-the-art image classifiers. If we adapt them to 1D spectra (e.g. use 1D convolutions in ResNet), the same principles apply: these networks need inputs scaled appropriately and often benefit from the kind of preprocessing discussed. For instance, if one uses a 1D ResNet to predict protein content from spectra, one would feed in spectra normalized to, say, mean 0 and unit variance, possibly with added channels like a derivative spectrum, and ResNet will learn hierarchical features. The skip connections in ResNet help it learn even if some layers are unnecessary, which can be useful if certain preprocessing already simplified the problem (the network can essentially skip over layers if the data is easy to fit). In such a case, heavy preprocessing (like aggressive feature selection) might make some network layers redundant – but ResNet's design can accommodate that by not harming performance, whereas a plain CNN might overfit if it has too many filters for the simplified input. **Xception**, being a deeper model, likewise would expect well-normalized inputs and may need a lot of data unless one uses transfer learning. If someone tried using a pre-trained Xception on spectral data by encoding the spectrum as an "image" (perhaps as a thin image with one dimension = wavelength and the other dimension dummy), they should use the preprocessing that Xception was trained with (typically mean subtraction and scaling to a certain range). However, such approaches are unconventional; more commonly, one would train these networks from scratch on spectral data or use a smaller variant. **Vision Transformer (ViT)** we already covered under transformers – it's a flagship model using attention and would need spectral tokens normalized and possibly segmented. The general rule with these *foundation* models is: **respect their input normalization and structure**. If using them out-of-the-box (pre-trained), transform your spectral data to fit their expected input domain (scale and shape). If training from scratch, use them as a powerful starting architecture but still apply the domain-specific preprocessing known to aid learning (baseline correction, etc.). As evidence that deep models and preprocessing go hand-in-hand, a 2021 study on **SpectraVGG** (a VGG-type deep CNN for NIR) found that data augmentation and preprocessing heavily influenced the final model – they used an augmentation analogous to MSC and saw large improvements [3]. Another study combined a CNN with a decision tree on NIR data and noted that results were **strongly dependent on proper preprocessing and feature selection** for the CNN to succeed [22]. These examples reinforce that even with extremely capable architectures like Xception or ViT, you cannot ignore the traditional preprocessing – the network will ultimately learn from the data you give it, and if that data still contains unnecessary noise or variation, the complexity of the model alone won't save it. In conclusion, modern deep learning architectures offer tremendous modeling capacity for NIR spectra – enabling end-to-end learning of features. But to harness that capacity effectively, one must apply the time-honored preprocessing techniques of spectroscopy (noise reduction, scatter correction, scaling, etc.) in conjunction with these models. By doing so, you set the stage for architectures like CNNs, RNNs, and Transformers (be it a custom 1D ResNet or a ViT) to learn the mapping from spectra to target without being tripped up by irrelevant spectral variations. **Deep models still "garbage in, garbage out" – quality in, breakthrough performance out.**

---

[1] [3] [6] [7] [8] [11] [12] [17] [18] [20] [22] A Review of Machine Learning for Near-Infrared Spectroscopy - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC9784128/

[2] [9] machine learning - When using SVMs, why do I need to scale the features? - Cross Validated
https://stats.stackexchange.com/questions/154224/when-using-svms-why-do-i-need-to-scale-the-features

4 10 machine learning - PCA versus other ways of feature selections? - Cross Validated
https://stats.stackexchange.com/questions/514225/pca-versus-other-ways-of-feature-selections

5 Importance of Feature Scaling — scikit-learn 1.7.2 documentation
https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

13 Feature Transformation & Binning — The Machine Learning Simplified book
https://code.themlsbook.com/chapter7/feature_transformation.html

14 Accurate predictions on small data with a tabular foundation model | Nature
https://www.nature.com/articles/s41586-024-08328-6?error=cookies_not_supported&code=833b9421-b45b-4826-8d03-e0101fc8b58f

15 TabPFN: A Revolutionary Transformer for Tabular Classification in ...
https://medium.com/@pns00911/tabpfn-a-revolutionary-transformer-for-tabular-classification-in-seconds-6837c1ac506d

16 Why Feature Scaling in SVM? | Baeldung on Computer Science
https://www.baeldung.com/cs/svm-feature-scaling

19 [PDF] arXiv:1610.02357v3 [cs.CV] 4 Apr 2017
https://arxiv.org/pdf/1610.02357

21 Vision Transformer (ViT)
https://huggingface.co/docs/transformers/en/model_doc/vit