Chapter 9

Making MS Omics Data ML-Ready: SpeCollate Protocols

Muhammad Usman Tariq, Samuel Ebert, and Fahad Saeed

Abstract

The increasing complexity and volume of mass spectrometry (MS) data have presented new challenges and opportunities for proteomics data analysis and interpretation. In this chapter, we provide a comprehensive guide to transforming MS data for machine learning (ML) training, inference, and applications. The chapter is organized into three parts. The first part describes the data analysis needed for MS-based experiments and a general introduction to our deep learning model SpeCollate—which we will use throughout the chapter for illustration. The second part of the chapter explores the transformation of MS data for inference, providing a step-by-step guide for users to deduce peptides from their MS data. This section aims to bridge the gap between data acquisition and practical applications by detailing the necessary steps for data preparation and interpretation. In the final part, we present a demonstrative example of SpeCollate, a deep learning-based peptide database search engine that overcomes the problems of simplistic simulation of theoretical spectra and heuristic scoring functions for peptide-spectrum matches by generating joint embeddings for spectra and peptides. SpeCollate is a user-friendly tool with an intuitive command-line interface to perform the search, showcasing the effectiveness of the techniques and methodologies discussed in the earlier sections and highlighting the potential of machine learning in the context of mass spectrometry data analysis. By offering a comprehensive overview of data transformation, inference, and ML model applications for mass spectrometry, this chapter aims to empower researchers and practitioners in leveraging the power of machine learning to unlock novel insights and drive innovation in the field of mass spectrometry-based omics.

Key words Proteomics, Tandem mass spectrometry, Peptide identification, Database search, Machine learning, Deep learning, Data preprocessing

1 Introduction

Machine learning (AI/ML) is a collection of data-driven technologies which can significantly advance mass spectrometry-based omics research. Open-source repositories such as PRIDE [1], ProteomeXchange [2], NIST [3], and MassIVE [4] make mass spectrometry data available and reusable for research communities to reproduce scientific results. While the availability of the open-source data is a step in the right direction, the MS data made available cannot be efficiently and effectively used for AI/ML

Frédérique Lisacek (ed.), *Protein Bioinformatics*, Methods in Molecular Biology, vol. 2836,
https://doi.org/10.1007/978-1-0716-4007-4_9,

applications. Therefore, specific research and development efforts are needed to make these and other datasets not only FAIR [5], but also ML-ready—which can then be shared through repositories, knowledge bases, and other data-sharing resources.

In contrast to previous approaches needed for computational workflows, for many AI/ML applications, a dataset must be sufficiently large to be considered AI-/ML-ready. Newer workflows that incorporate ML necessitate knowledge of big data management and parallel and distributed computing to ensure that data is partitioned in optimal manner to enable scalable computational feasibility [6–8]. Other expectations that have surfaced because of the development of ML models for MS-based omics include data documentation to ensure reproducibility, data wrangling practices that remove bias, and health and ethical decision that may have been taken to select the data for training, validation, and most importantly testing. Paying attention to ethical, legal, and social implications of these specific decisions such as balancing the data for gender, race, socioeconomic status, and other demographic information for the patients can lead to improved minority health and reduce health disparities. Such practices also ensure that new sciences produced because of these ML-based tools are reliable and reproducible. Other impacts include issues of identifiability and privacy, impact on marginalized groups, health disparities, and unintended consequences for the greater community. While data scientists are still grappling with many aspects discussed above, other aspects of AI/ML readiness are better understood. For example, applications developed using PyTorch and TensorFlow and specific ML models require the data in specific formats, with specific dimensions and parameters. Other works that are specific to MS data such as FAIR principles established for MS data and format are widely known for MS-based omics practitioners.

Like other fields, it is imperative to establish an integrated computational ecosystem that adapts the latest ML/AI technologies using best practice guidelines arising from community consensus. However, many aspects of MS data that are also AI-/ML-ready are established through exploratory testing. This includes how to best transform the given MS data that becomes ML-ready, removal of noise, labels and their specificity, and how much tolerance can be expected for a given data parameter and the effect of those parameters on the performance of the ML model. Further, making data AI-/ML-ready is not simply formulaic but is very specific to the kind of problem that is being solved. We assert that all these practices must be made available as open-source information so that practitioners of ML do not waste time and resources trying to figure out these MS-specific parameters that will result in the best performance for a given problem. Establishment of these guidelines will also ensure that resources are not wasted across teams and labs.

The underlying question that we will try to answer in this chapter is as follows: How can we make our MS data ML-ready and reproducible? Given that much of ML readiness is dependent on the specific problem for which model is being developed, as well as the architecture of the ML model, makes this an interesting and hard problem to solve. Therefore, to illustrate what factors must be considered, we will first present our peptide deduction engine. This will give an idea to the user about the model and what was considered in the design and implementation. Thereafter, we will show how to make the data ML-ready, what design decisions for ML data conversion must be taken, and what are their effects on the performance. We end the chapter by giving a step-by-step tutorial on how to install and run our cross-modal peptide deduction engine SpeCollate.

2 SpeCollate: Deep Cross-Modal Similarity Network for Mass Spectrometry Data-Based Peptide Deductions

The present-day identification of mass spectrometry (MS) based proteomics data primarily relies on database search algorithms utilizing numerical methods (Fig. 1). The operation of these numerical methods involves a comparison between the observed spectra and the simulated spectra, the latter of which is derived from theoretical peptides via a basic simulator [9–12]. Matching between the experimental and theoretical spectra is performed using one of various heuristic scoring functions, including dot product [13], shared peak count [14, 15], and ion matches [16]. Other peptide identification strategies such as de novo algorithms [17–28] try to infer peptides directly from experimental spectra, with varying degrees of success.

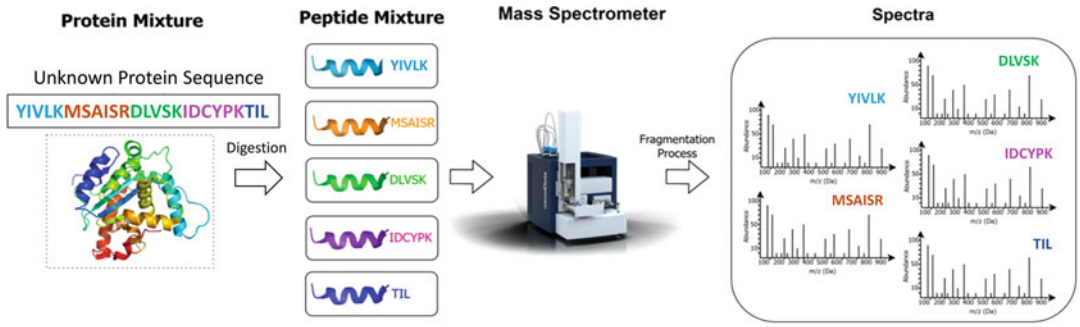

Fig. 1 Process of generating MS/MS spectra from a protein mixture using mass spectrometry analysis. Proteins in the mixture are broken into peptides using the enzyme called trypsin which breaks the protein strings at K and R bases generating peptides of varying sizes. This peptide mixture is then refined, and peptides are moved through a mass spectrometer which generates an MS/MS spectrum for each different peptide

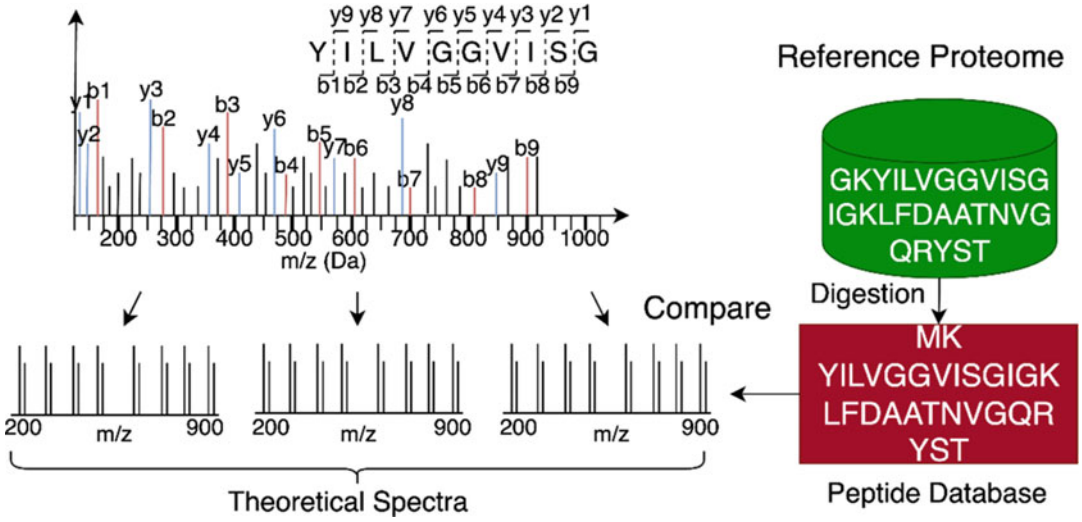

Fig. 2 A generic proteomics flow. In silico digestion of the protein database is performed to generate peptides. These peptides are then converted to the theoretical spectra and compared against the experimental spectra

To this day, no single scoring function from database search techniques stands out as the most precise strategy. Significant efforts have been made toward the development of computational methods for peptide identification through database search [13–16] (refer to Fig. 2 for a generic proteomics database search workflow) and de novo algorithms [17–28]. However, challenges in peptide identification are pervasive and well-documented. These issues include misidentifications or failure to identify peptides, statistical accuracy (FDR), and inconsistencies across different search engines [29]. Given the less than impressive accuracy of de novo algorithms (<35%) [20] and database search algorithms (30–80%) [30], and the absence of quality assessment benchmarks, further in-depth investigation and evaluation of these database search tools are essential.

Two key sources of heuristic errors introduced during numerical database search algorithms are (1) the simulation of spectra (from peptides), i.e., how peptide deduction is conducted, and (2) the peptide-spectrum match scoring function. Earlier studies hint that numerical algorithms and traditional machine learning (ML) algorithms might not effectively capture and integrate the multidimensional aspects of MS data [31]. Contrarily, deep learning methods [20] might provide a superior strategy for identifying peptides in noisy, high-dimensional MS data and peptides bearing a high degree of similarity to one another [11]. Initial exploration of deep learning methods in peptide deduction applied to MS data shows promising results, with an average accuracy of 82–95% on chosen datasets. However, precision (at the amino acid level) and recall (at the peptide level) remained somewhat limited [20].

Deep learning-based models such as Prosit [12] and Slider [32] have shown promise. Prosit incorporates theoretical spectral simulation into the database search pipeline, encoding peptide sequences into a latent space and then decoding the embeddings to predict fragment ion intensities. Slider uses deep convolutional neural networks to score experimental spectra against theoretical spectra. Another study [33] demonstrated how spectra for modified peptides could be embedded close to their nonmodified counterparts in 32-length vectors, facilitating fast lookup of the nonmodified version for a given spectrum. These studies underscore the potential of deep learning models in modeling MS proteomics data and emphasize the need for further exploration into sophisticated machine learning strategies.

SpeCollate [34] represents an example of such strategies, as it learns a fixed-sized embedding of variable length experimental spectra and peptide strings. This technique ensures a given spectrum and its corresponding peptide are projected close to each other (based on L2 distance) in a shared Euclidean subspace. L2 distance is employed as the similarity matrix given its demonstrated effectiveness with similarity ranking loss functions, such as triplet loss, and superior performance to other similarity metrics, such as cosine similarity. The design of SpeCollate takes inspiration from FaceNet [35], with its choice of L2 distance as the similarity metric. Composed of two subnetworks, the Spectrum Sub-Network (SSN) and the Peptide Sub-Network (PSN), SpeCollate is trained on two sets of data points, sparse spectrum vectors and encoded peptide strings, to calculate loss value. The training process employs sextuplets generated after each forward pass and uses online hardest negative mining for selecting the negative spectra and peptides, making the training process more efficient and faster. In the next section, we will discuss the different data parameters that are used for designing and developing SpeCollate so that researchers working in the area can consider these parameters for their own development. The overall ML architecture of SpeCollate is shown in Fig. 3.

3 Design Decisions for ML Data Conversion

To convert MS/MS data to ML-ready, several design decisions are made to ensure compatibility with ML models and optimal performance. Some of the decisions that you want to consider when designing and developing your ML models that can process MS data are as follows.

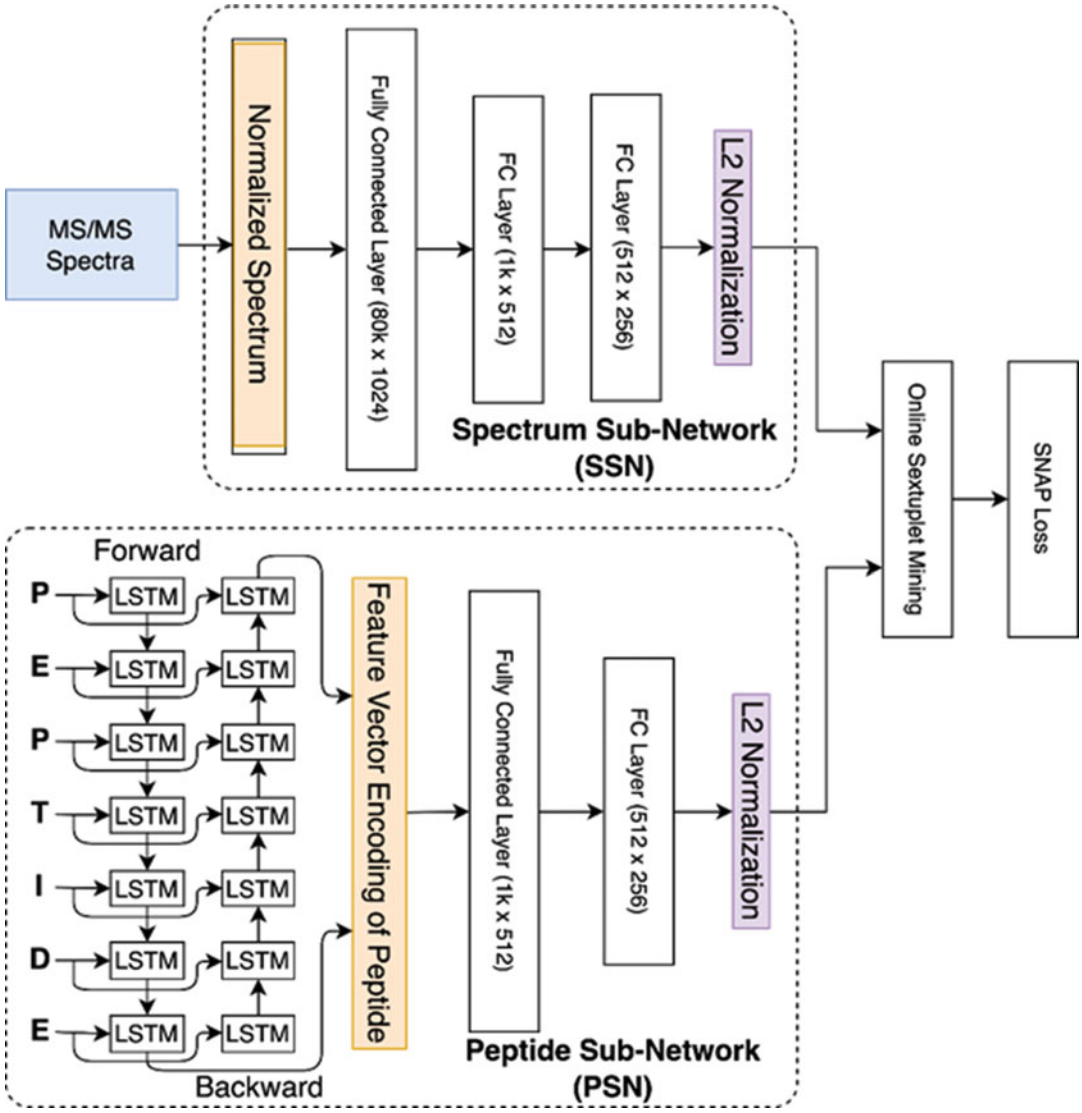

Fig. 3 SpeCollate: Deep Similarity Network for proteomics. The spectra Q are passed to SSN in the form of sparse one-hot normalized representation. The positive and negative peptides (P, N) are passed to PSN one by one in both forward and backward direction

3.1 Vectorization

MS data is typically converted into vectors of fixed length where each element represents a mass-to-charge (m/z) ratio bin. This format is simple as it can be directly input to a fully connected layer. However, it can be memory intensive as the vector size can be upward of 80k in length. SpeCollate and DeepNovo use this representation for spectra which makes the models simpler and quick to train at the expense of memory intensive representation.

Tradeoffs to consider Vectorization is a simple and efficient technique to represent data. However, there are some tradeoffs to be considered: If data is discretized into larger bins, especially for high-precision mass spectrometry data, information can be lost. This can adversely impact a model's performance. On the other hand, using smaller bin sizes may result in too large vector sizes. In addition, MS/MS data are inherently sparse. Converting the data into vectors can exacerbate the sparsity resulting in hyper sparse vectors. This can require specialized techniques and preprocessing layers in the model to deal with hyper sparse data. Moreover, the size of the vector is directly related to the computational and memory complexity. Larger vector sizes provide higher resolution at the cost of the increased complexity of the machine learning model and consuming memory. Conversely, shorter vectors may simplify model layers and be memory efficient but might lose some valuable information due to larger bin sizes.

3.2 Matrix Representation

Spectrum is converted into a vector and then transformed into a two-dimensional matrix where one dimension is the m/z and the other dimension is the normalized right-shifted intensity values vectors. This technique is used by DeepNovo which allows usage of CNN to process spectra. Some versions of DeepNovo use matrix representation when CNN layers are deployed at the input. It allows for information rich representation at the cost of high resource utilization.

Tradeoffs to consider Matrix representation allows for complex information to be captured but suffers from scalability and computation complexity problems. Computation and memory requirements increase at least x26 when compared to the simple vectorization technique, hence making this approach unsuitable for large datasets or where more precise binning of m/z values is needed. Note that all the tradeoffs of vectorization techniques still apply to matrix representation in addition to the scalability issue.

3.3 Sequence Representation

Spectrum is represented as two aligned sequences of discretized m/z and intensity values. Since the sequence is not vectorized, the precision value can be much higher (i.e., bin size can be much smaller). Pairs of m/z and intensity values can be used as tokens. This technique is used whenever a transformer network is employed for processing spectra. Since there is no inherent positional information in transformers, it needs to input using some external mechanism, e.g., sinusoidal positional encoding method. This technique is used by yHydra [36] to allow spectra to be represented efficiently for attention layers.

Tradeoffs to consider Sequence representation allows for much higher precision of m/z values which enables the model to capture more useful information from the spectrum. However, the transformer networks required to properly capture the features can be computationally expensive as the number of trainable parameters increases quadratically (not considering the depth) with respect to the input sequence size.

3.4 Normalization

The intensity values within the vector are normalized to fixed scale such as $[0, 1]$ to reduce the impact of variations due to experimental conditions and facilitate model convergence. Multistep normalization can be employed as well where each spectrum is normalized individually using min-max normalization and then each feature is standardized in the entire dataset to have zero mean and unit variance. Almost each machine learning employs these techniques. Most machine learning models use normalization step as it reduces variance of the data and allows for robust training with faster converging models.

Tradeoffs to consider Normalization is essential for enabling machine learning models to make sense of the data; however, there can be some tradeoffs. The original signal is lost during normalization, and it can affect feature importance which can hurt model's interpretability. Normalization does add to the computational cost; however, it is usually offset by the efficient convergence of the model. Some normalization techniques are not reversible making it difficult to interpret results.

3.5 Peak Picking

This is a critical step used in most MS/MS data processing which aims to identify and extract the most significant peaks (ions) from the raw data. The purpose is to reduce noise, simplify data representation, and retain the most relevant information needed. This can also help in removing outlier peaks and improving overall data quality which helps the training process of machine learning models. SpeCollate and other models employ peak picking to remove noisy data and reduce the number of total peaks in the spectrum.

Tradeoffs to consider Though peak picking is unanimously used across most tools, it can have some tradeoffs. Depending on the peak picking algorithm, information from the original signal might be lost. Moreover, careful parameter tuning is needed to make sure peak picking works properly across different types of spectra and ensure reproducibility. Similar to normalization, peak picking also adds slightly to the computational cost.

3.6 Missing Value Imputation

MS data often contain missing values due to experimental limitations. Different techniques can be used to impute missing values, e.g., zero fill, k-nearest neighbor, etc. Deep learning methods can be used to predict missing peaks, e.g., transformers. Imputing these missing values can help to create a completer and more consistent dataset, which may improve the performance of machine learning models. SpeCollate uses an auxiliary deep learning model to predict missing peaks in spectra; however, the process can be used by any database search tool.

Tradeoffs to consider Data imputation methods can reduce variability in the data leading to overfitting the model. Care must be taken to impute the data in a consistent way which does not result in information loss (reduce variability) in the data. Imputation techniques assume that the missing values have the same distribution as the existing values. This assumption is true for most cases; however, sometimes when this is not the case, the data quality might be affected negatively. If a deep learning model is used for data imputation, it can add significantly to the training time and computational complexity.

Various data regarding wrangling operations that can be performed on the MS data, its utility in various ML models, their effect on the complexity of the model, the effect on training and accuracy performance are summarized in Table 1.

4 Making MS Data ML-Ready: A Practitioner's Approach

4.1 Obtaining the Data: Public Repositories

The first step in any ML pipeline for MS is acquiring the data. Depending on your specific goals, there are a variety of public data repositories. Some examples include the NIST mass spectral library [3], which provides labeled spectra for several species and instrument types in msp format. These can be downloaded directly from the browser.

Another resource is MassIVE [4], which provides researcher-submitted datasets for a variety of species and instrument types. Most MassIVE data comes in mzML or mzXML format. Clicking on the link in the "Title" page of each dataset will take you to a page containing an ftp download link; you can use it to download all files in the dataset as follows:

```
wget -r -p ftp://massive.ucsd.edu/{dataset}/
```

Assuming that the spectra's data type is mzML, after running the above command, the spectra will be in the folder {working_dir}/massive.ucsd.edu/{dataset}/peak/mzMLs. Depending on

Table 1
Comparison of data representation approaches for database search tools

	SpeCollate	*DeepNovo*	yHydra	Effect on model complexity	Effect on training time/ resources	Effect on accuracy
Vectorization	Yes	Yes	No	Model complexity is low as fully connected layers can be used	Training time is usually low with low resource utilization. However, depending on the vector size, the timing and resource utilization can increase	Vectorization can affect accuracy if large bin size is used
Matrix representation	No	Yes	No	Suitable for CNN layers. Can increase model complexity if multiple convolution layers are used	Memory intensive and hence requires more resources to process	Improves accuracy as more meaningful features can be extracted using CNNs at the cost of computational complexity
Sequence representation	No	No	Yes	Model complexity increases as attentional layers are needed	Usage of attention layers increases computation complexity. Moreover, it takes longer to train attention layers	Can improve accuracy significantly as m/z precision is much higher without the extra cost of memory complexity
Normalization	Yes	Yes	Yes	No impact	Improves training time as model converges faster	Improves accuracy as the outlying values are removed and model converges easily
Peak picking	Yes	Yes	Yes	Can allow for use of simpler model as there are a smaller number of peaks	Simpler layers improve time complexity and resource usage	Removing noise improves accuracy by improving training data quality
Missing value imputation	Yes	No	No	Might need a separate model to impute missing values	The imputation model will need separate training adding significant time and resource requirements	Improves accuracy for all tools as the data quality is improved across the board

the size of the dataset, there may be multiple numbered mzML files. For data sources such as the NIST library, the spectra will come in msp format in an archived file. After downloading the library of interest, you can extract the archive as follows:

```
tar -xvzf {nist_filename}.tar.gz
```

4.2 Obtaining the Data: Simulating from a Proteome

Depending on your use case, you may not find the spectra you need in any public library. In this case, it is possible to simulate spectra from any set of peptides using Prosit [12]. This section outlines the steps of an example workflow to simulate theoretical spectra for the entire brown rat (*Rattus norvegicus*) proteome.

4.2.1 Downloading the Proteome

You can download the rat proteome from UniProt [37]. In the search bar, select "proteomes" and search for "rattus norvegicus." This will result in several results; you should select the "reference proteome" with entry ID UP000002494. At the time of this writing (June 2023), this entry contains 47,942 proteins. Download the proteome as "FASTA" canonical; you have the option to download either in compressed or uncompressed format. If you downloaded the compressed file, extract it as follows; otherwise, proceed to the next section.

```
gunzip {uniprot_file}.fasta.gz
```

4.2.2 Converting to Peptides

At this point, you should have a single FASTA file containing proteins. To generate spectra, these must be digested into peptides. You can do this by installing Crux [38] and using the following command (note that at the time of this writing, Prosit only supports simulating spectra for peptides from length 7 to 30, so the length parameters should be in that range):

```
crux generate-peptides -min-length {min_length} --max-length
{max_length} {uniprot_file}.fasta
```

This will result in a file where each line is a tab-delimited list of {peptide} {m+h mass of unmodified} {proteins containing this peptide}.

4.2.3 Simulating with Prosit

Prosit expects its input to be a csv file where the first line is "modified_sequence,collision_energy,precursor_charge" and all subsequent lines are the respective values for each spectrum to simulate. You can convert the peptides from the crux output into this format using any scripting language, with your choice of collision energies and precursor charges for each peptide. Now, you can generate theoretical spectra from this file by navigating to the "Spectral Library" tab on the Prosit site and following the

instructions to upload a peptides file. After submitting the job, you will be provided with a job ID that you must save to access the results later. Once the spectra are generated, you can download the output files (in MSP format) by providing the job ID and unzipping the resulting file. Note that Prosit can fail when given input files with too many lines—the job may appear as "finished" with a download option, but the download itself will not succeed. If this occurs, you should split up the input into smaller files and upload each separately.

4.3 Data Conversion

SpeCollate and many other models require the input spectra to be provided in Mascot generic format (mgf). Depending on how you obtained your data in the previous steps, you are likely to have either mzXML, mzML, or MSP files.

To convert mzXML or mzML files to MGF, you can use the msconvert tool from ProteoWizard [39] as follows:

```
./msconvert {your_mzXML_or_mzML_file} --mgf
```

To convert MSP files, you can use the msp_to_mgf.py script from the MspConverter repository [40], as follows (note that the config file is optional, and both directories must exist):

```
python msp_to_mgf.py path/to/msp/folder/ path/to/output/
folder/ path/to/config.ini
```

4.4 Preprocessing

At this point, you should have a folder containing one or more converted MGF files. You may want to do further preprocessing, such as normalization or filtering, before using the data in your model. You can do this using the Pyteomics library [41], which allows you to load, modify, and save spectra. The exact steps will depend on your specific research problem; a few example workflows are shown below.

4.4.1 Filtering Spectra by Charge

```
from pyteomics import mgf
import glob
import os

mgf_dir = "/path/to/your/mgf/files/"
mgf_files = [f"{mgf_dir}{f}" for f in os.listdir(mgf_dir) if
f.endswith("mgf")]

max_charge = 2
criterion = lambda spec: spec['params']['charge'][0] <= max_-
charge
```

```
def filter_spectra(spectra, criterion):
 for spectrum in spectra:
 if criterion(spectrum):
 yield spectrum

out_file = "/path/to/your_output_file.mgf"

with mgf.chain.from_iterable(mgf_files) as spectra:
 filtered_spectra = filter_spectra(spectra, criterion)
 mgf.write(filtered_spectra, out_file)
```

4.4.2 Filtering Each Spectrum to the Top n Highest-Intensity Peaks

```
from pyteomics import mgf
import glob
import os
import numpy as np

mgf_dir = "/path/to/your/mgf/files/"
mgf_files = [f"{mgf_dir}{f}" for f in os.listdir(mgf_dir) if
f.endswith("mgf")]

peak_count = 50

def filter_peaks(spectrum, n):
 peak_count = len(spectrum["intensity array"])
 if peak_count <= n:
 return spectrum

  peak_indices_to_keep = sorted(np.argpartition(spectrum["in-
tensity array"], -n)[-n:])
  spectrum["intensity array"] = spectrum["intensity array"]
[peak_indices_to_keep]
 spectrum["m/z array"] = spectrum["m/z array"][peak_indices_-
to_keep]
 spectrum["charge array"] = spectrum["charge array"][peak_in-
dices_to_keep]
 return spectrum

def filter_spectra(spectra, peak_count):
 for spectrum in spectra:
 yield filter_peaks(spectrum, peak_count)

out_file = "/path/to/your_output_file.mgf"

with mgf.chain.from_iterable(mgf_files) as spectra:
 filtered_spectra = filter_spectra(spectra, peak_count)
 mgf.write(filtered_spectra, out_file)
```

4.4.3 Normalizing Intensity Values to Zero Mean and Unit Variance

```
from pyteomics import mgf
import glob
import os
import numpy as np

mgf_dir = "/path/to/your/mgf/files/"
mgf_files = [f"{mgf_dir}{f}" for f in os.listdir(mgf_dir) if
f.endswith("mgf")]

all_intensities = []

with mgf.chain.from_iterable(mgf_files) as spectra:
 for spectrum in spectra:
 for intensity in spectrum['intensity array']:
 all_intensities.append(intensity)

intensity_mean = np.mean(all_intensities)
intensity_std = np.std(all_intensities)

def normalize_peaks(spectrum, intensity_mean, intensity_std):
 spectrum["intensity array"] = (spectrum["intensity array"] -
intensity_mean) / intensity_std
 return spectrum

def normalize_spectra(spectra, intensity_mean, intensi-
ty_std):
 for spectrum in spectra:
  yield normalize_peaks(spectrum, intensity_mean, intensi-
ty_std)

out_file = "/path/to/your_output_file.mgf"

with mgf.chain.from_iterable(mgf_files) as spectra:
 normalized_spectra = normalize_spectra(spectra, intensity_-
mean, intensity_std)
 mgf.write(normalized_spectra, out_file)
```

After performing the desired preprocessing, your spectra should now be ready for use with a machine learning model.

5 Installation Instructions for SpeCollate

5.1 Prerequisites

5.1.1 Software

- Python > 3.8, PyTorch > 1.6, NumPy, Pandas, SciKit-Learn
- Cuda Toolkit >= 11.6
- OpenMS for protein database digestion

- Crux (for FDR analysis using its percolator implementation)
- Docker for running msconvert to convert raw spectra files into mgf.

5.1.2 Hardware Requirements

- A computer with Ubuntu >= 16.04 or CentOS >= 8.1
- 120 GBs of system memory and 8 CPU cores
- Cuda enabled GPU with at least 12 GBs of memory

5.1.3 Data Sources for Mass Spectrometry

- Proteome Exchange [2]

5.1.4 Protein Databases

- UniProt [37]

5.2 Installing Conda Distribution of Python

1. Latest version of mini-conda for linux can be downloaded from [42]. If you are using the terminal, the file can be downloaded using the wget command:

   wget https://repo.anaconda.com/miniconda/Miniconda3-py310_23.1.0-1-Linux-x86.sh
2. Once the .sh file is downloaded, execute the file using the command

   ./Miniconda3-py310_23.1.0-1-Linux-x86.sh

   in the directory the file is located. Note that the file name might be different for your download.

5.3 Installing CudaToolkit

Use the following commands to install CudaToolkit on your system:

1. Update the APT repository cache:

   sudo apt-get update
2. Install Cuda SDK:

   sudo apt-get install cuda

   sudo apt-get install nvidia-gds
3. Reboot the system:

   sudo reboot

Detailed instructions can be found at [43].

5.4 Installing PyTorch with Cuda and Other Required Packages

1. Numpy, Pandas, and SciKit-Learn can be installed using the following commands:

   conda install numpy

   conda install pandas

   conda install scikit-learn

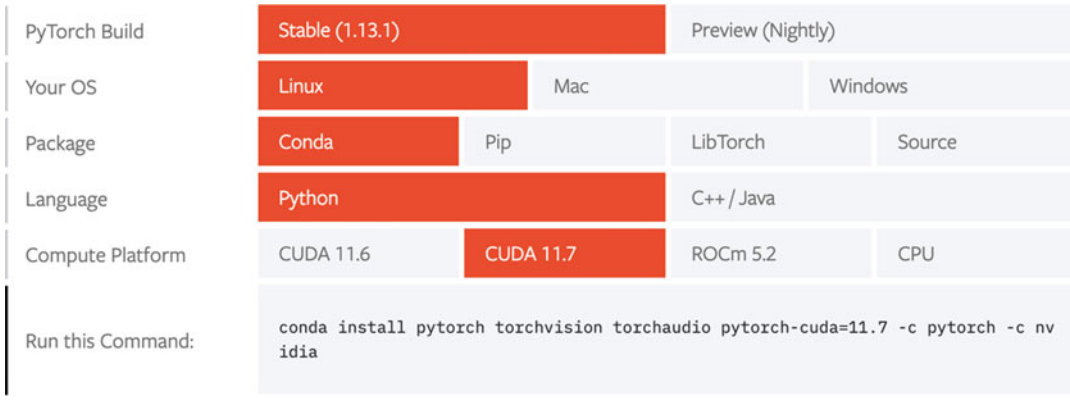

Fig. 4 Screenshot of requirements that are needed for installing PyTorch and related packages

You can download PyTorch from the official site [44].

Select the options as shown in Fig. 4.

For the Compute Platform option, make sure the CUDA version is the same as you installed in the previous step.

Execute the generated command in your terminal.

6 Retraining SpeCollate

6.1 Download Retrainable Executable File for SpeCollate

1. Download the files from the "Training" page at [45].

   The directory contains multiple files, including the following:

   - *specollate-train*: This is the executable for training SpeCollate.
   - *specollate-search*: This is the executable for database search.
   - *config.ini*: parameter file for training and searching.
   - *models (dir):* contains the pretrained model. New models will also be stored here.
   - *percolator (dir):* Percolator input (.pin) files can be placed here after the search is complete.

2. Download the preprocessed data for training [45], and extract using the command

   tar -xzf specollate-training-data.tar.gz.

6.2 Steps to Retrain

1. Open the config.ini file from **step 1** in your favorite text editor, and set the following parameters:

   - in_tensor_dir in [preprocess] section: absolute path of the decompressed file from **step 2** above.
   - model_name in [ml] section: the name by which to wish to save the trained model file.

- other parameters in the [ml] section: You can adjust different hyperparameters in the [ml] section, e.g., learning_rate, dropout, etc.

2. Execute the specollate_train file:

   ./specollate_train

6.3 Performing Database Search

6.3.1 Prerequisites for Database Search

1. Download the files on the "Training" page at [45].

   The extracted directory contains multiple files, including the following:

   - *specollate-train*: This is the executable for training SpeCollate.
   - *specollate-search*: This is the executable for database search.
   - *config.ini:* parameter file for training and searching.
   - *models (dir):* contains the pretrained model. New models will also be stored here.
   - *percolator (dir)*: Percolator input (.pin) files can be placed here after the search is complete.

2. Download the files on the "Search" page at [45]. Or you can use your own spectra files in mgf format.
3. Download the human peptide database on the "Search" page at [45]. You can also provide your own peptide database file.

6.3.2 Steps to Perform Database Search

1. Set the following parameters in the [search] section of the config.ini file:
   - model_name: name of the model to be used. The model should be in the /models directory.
   - mgf_dir: absolute path to the directory containing mgf files to be searched.
   - prep_dir: absolute path to the directory where preprocessed mgf files will be saved.
   - pep_dir: absolute path to the directory containing peptide database.
   - out_pin_dir: absolute path to a directory where percolator pin files will be saved. The directory must exist; otherwise, the process will exit with an error.
   - Set database search parameters, e.g., precursor_mass_tolerance.
2. Execute the specollate_search file:

   ./specollate_search

   If you want to use the preprocessed spectra from a previous run, use the -p False flag:

./specollate_search -p False

3. Once the search is complete, you can analyze the percolator files using the crux percolator tool:

   cd <out_pin_dir>

   crux percolator target.pin decoy.pin --list-of-files T --overwrite T

6.4 Notes

6.4.1 Parameters in Configuration File (config.ini)

- *Ml:batch_size*: batch size of spectra and labeled peptides used when retraining the model. 1024 is the recommended value. However, the user is free to experiment with different values.
- Input:master_port set the port to any five digit number below 65535. If another program is running using a port, the same port number cannot be used. It'll throw an error that the request port is already in use.
- Search:spec_batch_size and Search:pep_batch_size are the batch sizes used for the forward pass of spectra and peptides through the network. Update their value according the available GPU memory on your device.
- Search:search_spec_batch_size is the batch size of spectra used when performing the database search. Normally, a maximum of 512 max batch size is recommended, but it can be reduced depending on the GPU memory.

6.4.2 PyTorch and Cuda Versions

The model is trained on PyTorch version 1.6 with python 3.7.4 and Cuda 11.6. However, the later version should work. User should make sure to use the compatible version of PyTorch and Cuda. The compatibility can be checked on the PyTorch website [44].

6.4.3 Preprocessing Mass Spec Data

- Install docker on your Linux machine. Follow the instructions at [46]. Make sure you have sudo permission to install docker, or ask the system administrator to install it for you.

  Download msconvert docker file using the following command:

  docker pull chambm/pwiz-skyline-i-agree-to-the-vendor-licenses
- Download the required spectra file (at the moment, only Q Exactive instrument is supported) in the raw format from ProteomeExchange.
- Go to the link at [2].
- Scroll down and click Access Data button.
- Search for the dataset you are looking for and click the dataset id.
- Scroll down and click Dataset FTP location.

- Download the raw file(s).
- Once the file is downloaded, convert it to the mgf file using the msconvert docker file:

  docker run -it --rm -e WINEDEBUG=-all -v ${PWD}:/data chambm/pwiz-skyline-i-agree-to-the-vendor-licenses wine msconvert /data/20111219_EXQ5_KiSh_SA_Label-Free_HeLa_Proteome_Control_rep1_pH4.raw --mgf --filter "peakPicking true 2" --filter "msLevel 2" --filter "precursorRefine" --filter "threshold count 100 most-intense" --filter "zeroSamples removeExtra" -o mgfs

  Change the command options depending on your requirements and the dataset. The details of the command option can be found at [39].

Acknowledgments

Research reported in this publication was supported by the National Institute of General Medical Sciences of the National Institutes of Health under award number: R35GM153434. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

1. Perez-Riverol Y, Bai J, Bandla C et al (2022) The PRIDE database resources in 2022: a hub for mass spectrometry-based proteomics evidences. Nucleic Acids Res 50:D543–D552. https://doi.org/10.1093/nar/gkab1038
2. Deutsch EW, Bandeira N, Perez-Riverol Y et al (2023) The ProteomeXchange consortium at 10 years: 2023 update. Nucleic Acids Res 51: D1539–D1548. https://doi.org/10.1093/nar/gkac1040
3. Stein S (2008) NIST Libraries of Peptide Fragmentation Mass Spectra, NIST Standard Reference Database 1 C
4. Welcome to MassIVE. https://massive.ucsd.edu/ProteoSAFe/static/massive.jsp. Accessed 28 Jun 2023
5. Wilkinson MD, Dumontier M, Aalbersberg Ij J et al (2016) The FAIR guiding principles for scientific data management and stewardship. Sci Data 3:160018. https://doi.org/10.1038/sdata.2016.18
6. Saeed F, Haseeb M (2022) High-performance algorithms for mass spectrometry-based omics. Springer International Publishing, Cham
7. Haseeb M, Saeed F (2023) GPU-acceleration of the distributed-memory database peptide search of mass spectrometry data. Sci Rep 13: 18713. https://doi.org/10.1038/s41598-023-43033-w
8. Haseeb M, Saeed F (2021) High performance computing framework for tera-scale database search of mass spectrometry data. Nat Comput Sci 1:550–561. https://doi.org/10.1038/s43588-021-00113-z
9. Gabriels R, Martens L, Degroeve S (2019) Updated MS^2PIP web server delivers fast and accurate MS^2 peak intensity prediction for multiple fragmentation methods, instruments and labeling techniques. Nucleic Acids Res 47: W295–W299. https://doi.org/10.1093/nar/gkz299
10. Tiwary S, Levy R, Gutenbrunner P et al (2019) High-quality MS/MS spectrum prediction for data-dependent and data-independent acquisition data analysis. Nat Methods 16:519–525. https://doi.org/10.1038/s41592-019-0427-6
11. Zhou X-X, Zeng W-F, Chi H et al (2017) pDeep: predicting MS/MS spectra of peptides with deep learning. Anal Chem 89:12690–12697. https://doi.org/10.1021/acs.analchem.7b02566

12. Gessulat S, Schmidt T, Zolg DP et al (2019) Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. Nat Methods 16:509–518. https://doi.org/10.1038/s41592-019-0426-7
13. Diament BJ, Noble WS (2011) Faster SEQUEST searching for peptide identification from tandem mass spectra. J Proteome Res 10:3871–3879. https://doi.org/10.1021/pr101196n
14. Craig R, Beavis RC (2004) TANDEM: matching proteins with tandem mass spectra. Bioinformatics 20:1466–1467. https://doi.org/10.1093/bioinformatics/bth092
15. Kong AT, Leprevost FV, Avtonomov DM et al (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nat Methods 14:513–520. https://doi.org/10.1038/nmeth.4256
16. Zhang J, Xin L, Shan B et al (2012) PEAKS DB: De Novo sequencing assisted database search for sensitive and accurate peptide identification. Mol Cell Proteomics 11(M111):010587. https://doi.org/10.1074/mcp.M111.010587
17. Bandeira N (2007) Spectral networks: a new approach to de novo discovery of protein sequences and posttranslational modifications. BioTechniques 42:687–695. https://doi.org/10.2144/000112487
18. Grossmann J, Roos FF, Cieliebak M et al (2005) AUDENS: a tool for automated peptide de Novo sequencing. J Proteome Res 4:1768–1774. https://doi.org/10.1021/pr050070a
19. Fernandez-de-Cossio J, Gonzalez J, Satomi Y et al (2000) Automated interpretation of low-energy collision-induced dissociation spectra by SeqMS, a software aid forde novo sequencing by tandem mass spectrometry. Electrophoresis 21:1694–1699. https://doi.org/10.1002/(SICI)1522-2683(20000501)21:9<1694::AID-ELPS1694>3.0.CO;2-W
20. Tran NH, Zhang X, Xin L et al (2017) De novo peptide sequencing by deep learning. Proc Natl Acad Sci 114:8247–8252. https://doi.org/10.1073/pnas.1705691114
21. Taylor JA, Johnson RS (2001) Implementation and uses of automated de Novo peptide sequencing by tandem mass spectrometry. Anal Chem 73:2594–2604. https://doi.org/10.1021/ac001196o
22. Mo L, Dutta D, Wan Y, Chen T (2007) MSNovo: a dynamic programming algorithm for de Novo peptide sequencing via tandem mass spectrometry. Anal Chem 79:4870–4878. https://doi.org/10.1021/ac070039n
23. Fischer B, Roth V, Roos F et al (2005) NovoHMM: A Hidden Markov Model for de Novo Peptide Sequencing. Anal Chem 77:7265–7273. https://doi.org/10.1021/ac0508853
24. Yang H, Chi H, Zhou W-J et al (2017) OpenpNovo: De Novo peptide sequencing with thousands of protein modifications. J Proteome Res 16:645–654. https://doi.org/10.1021/acs.jproteome.6b00716
25. Ma B, Zhang K, Hendrie C et al (2003) PEAKS: powerful software for peptidede novo sequencing by tandem mass spectrometry. Rapid Commun Mass Spectrom 17:2337–2342. https://doi.org/10.1002/rcm.1196
26. Frank A, Pevzner P (2005) PepNovo: De Novo peptide sequencing via probabilistic network modeling. Anal Chem 77:964–973. https://doi.org/10.1021/ac048788h
27. Jagannath S, Sabareesh V (2007) Peptide Fragment Ion Analyser (PFIA): a simple and versatile tool for the interpretation of tandem mass spectrometric data andde novo sequencing of peptides. Rapid Commun Mass Spectrom 21:3033–3038. https://doi.org/10.1002/rcm.3179
28. Chi H, Sun R-X, Yang B et al (2010) pNovo: *De novo* peptide sequencing and identification using HCD spectra. J Proteome Res 9:2713–2724. https://doi.org/10.1021/pr100182k
29. Gupta N, Bandeira N, Keich U, Pevzner PA (2011) Target-decoy approach and false discovery rate: when things may go wrong. J Am Soc Mass Spectrom 22:1111–1120. https://doi.org/10.1007/s13361-011-0139-3
30. Chick JM, Kolippakkam D, Nusinow DP et al (2015) A mass-tolerant database search identifies a large proportion of unassigned spectra in shotgun proteomics as modified peptides. Nat Biotechnol 33:743–749. https://doi.org/10.1038/nbt.3267
31. Tran NH, Qiao R, Xin L et al (2019) Deep learning enables de novo peptide sequencing from data-independent-acquisition mass spectrometry. Nat Methods 16:63–66. https://doi.org/10.1038/s41592-018-0260-3
32. Kudriavtseva P, Kashkinov M, Kertész-Farkas A (2021) Deep convolutional neural networks help scoring tandem mass spectrometry data in database-searching approaches. J Proteome Res 20:4708–4717. https://doi.org/10.1021/acs.jproteome.1c00315

33. Qin C, Luo X, Deng C et al (2021) Deep learning embedder method and tool for mass spectra similarity search. J Proteome 232: 104070. https://doi.org/10.1016/j.jprot.2020.104070
34. Tariq MU, Saeed F (2021) SpeCollate: deep cross-modal similarity network for mass spectrometry data based peptide deductions. PLoS One 16:e0259349. https://doi.org/10.1371/journal.pone.0259349
35. Schroff F, Kalenichenko D, Philbin J (2015) FaceNet: a unified embedding for face recognition and clustering. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR). IEEE, Boston, pp 815–823
36. Altenburg T, Muth T, Renard BY (2021) yHydra: Deep Learning enables an Ultra Fast Open Search by Jointly Embedding MS/MS Spectra and Peptides of Mass Spectrometry-based Proteomics. Bioinformatics
37. The UniProt Consortium, Bateman A, Martin M-J et al (2023) UniProt: the universal protein knowledgebase in 2023. Nucleic Acids Res 51: D523–D531. https://doi.org/10.1093/nar/gkac1052
38. McIlwain S, Tamura K, Kertesz-Farkas A et al (2014) Crux: rapid open source protein tandem mass spectrometry analysis. J Proteome Res 13:4488–4491. https://doi.org/10.1021/pr500741y
39. Chambers MC, Maclean B, Burke R et al (2012) A cross-platform toolkit for mass spectrometry and proteomics. Nat Biotechnol 30: 918–920. https://doi.org/10.1038/nbt.2377
40. Tariq MU, Ebert S (2023) MSPConverter. https://github.com/pcdslab/mspconverter
41. Goloborodko AA, Levitsky LI, Ivanov MV, Gorshkov MV (2013) Pyteomics—a Python framework for exploratory data analysis and rapid software prototyping in proteomics. J Am Soc Mass Spectrom 24:301–304. https://doi.org/10.1007/s13361-012-0516-6
42. Miniconda — conda documentation. https://docs.conda.io/en/latest/miniconda.html. Accessed 26 Jun 2023
43. NVIDIA CUDA Installation Guide for Linux. https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html. Accessed 26 Jun 2023
44. PyTorch. https://pytorch.org/. Accessed 26 Jun 2023
45. Tariq MU, Saeed F SpeCollate. https://pcdslab.github.io/specollate-page/
46. Install Docker Engine on Ubuntu | Docker Documentation. https://docs.docker.com/engine/install/ubuntu/. Accessed 26 Jun 2023