Supporting Information

Determining Collision Cross Sections from Differential Ion Mobility Spectrometry

Christian Ieritano,[1,2,3] Arthur Lee,[1,3] Jeff Crouse,[1,2] Zack Bowman,[1] Nour Mashmoushi,[1,3] Paige M. Crossley,[1] Benjamin P. Friebe,[1] J. Larry Campbell, [1,2,4] and W. Scott Hopkins[1,2,3,5]*

1. Department of Chemistry, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, N2L 3G1, Canada. * E-mail: shopkins@uwaterloo.ca.

2. WaterMine Innovation, Inc., Waterloo, Ontario, N0B 2T0, Canada

3. Waterloo Institute for Nanotechnology, University of 200 University Avenue West, Waterloo, Ontario, N2L 3G1, Canada

4. Bedrock Scientific Inc., Milton, Ontario, L6T 6J9, Canada

5. Centre for Eye and Vision Research, Hong Kong Science Park, New Territories, 999077, Hong Kong.

Keywords: ion mobility spectrometry, supervised machine learning, CCS, Neural Networks, Random Forest Regression, MobCal-MPI

Table of Contents

Supporting figures

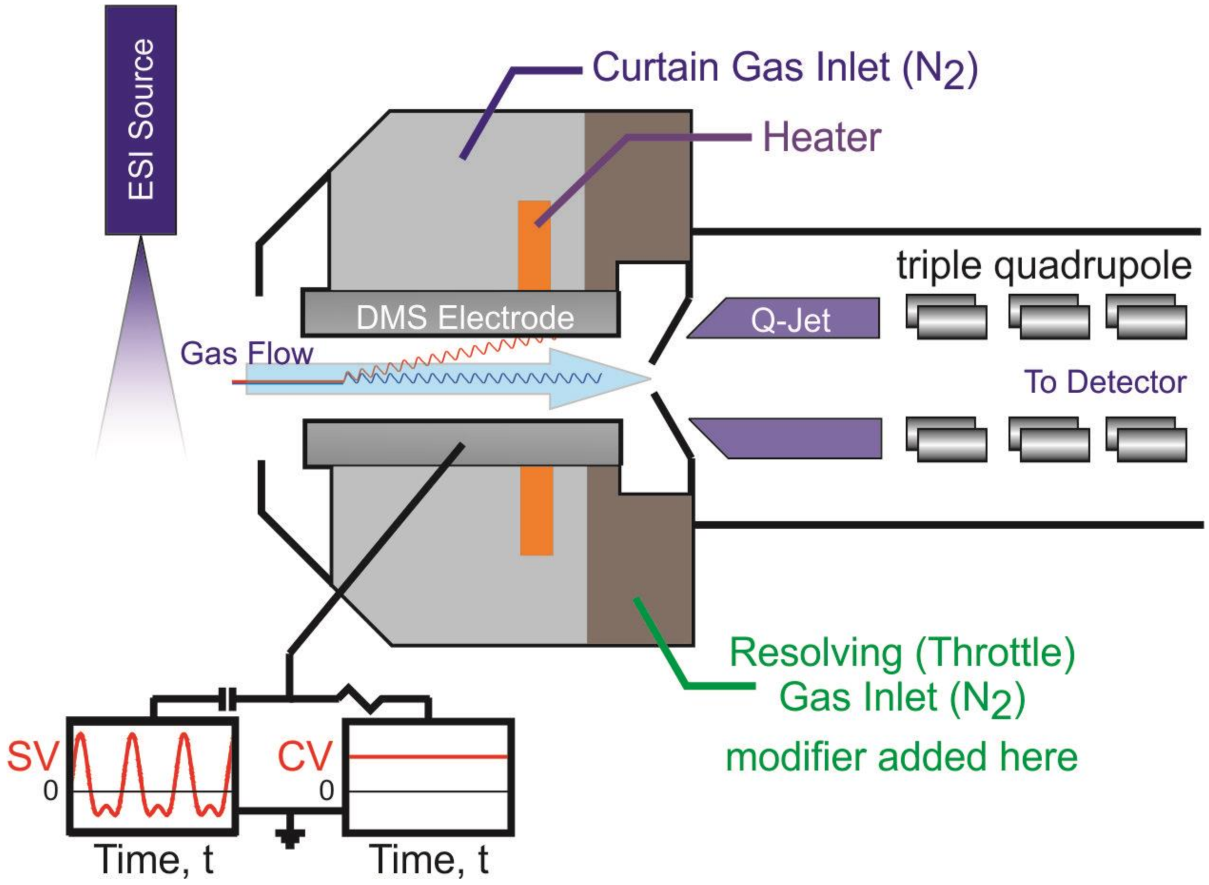

Figure S1. The basic layout of a differential mobility spectrometry (DMS) cell consists of two-planar electrodes separated by a 1 mm gap. Ions are introduced to the DMS cell by ESI and carried through the device with the flow of the N_2 carrier gas. Application of the oscillating separation field, denoted the separation voltage (SV), induces off-axis motion of the analytes that is determined by their field dependent mobility. Ion trajectories are stabilized by application of a static direct-current compensation voltage (CV). The magnitude of the CV applied is correlated with the differential mobility of the ion, enabling spatial resolution of analytes as they are directed towards the exit orifice of the DMS cell to the mass spectrometer.

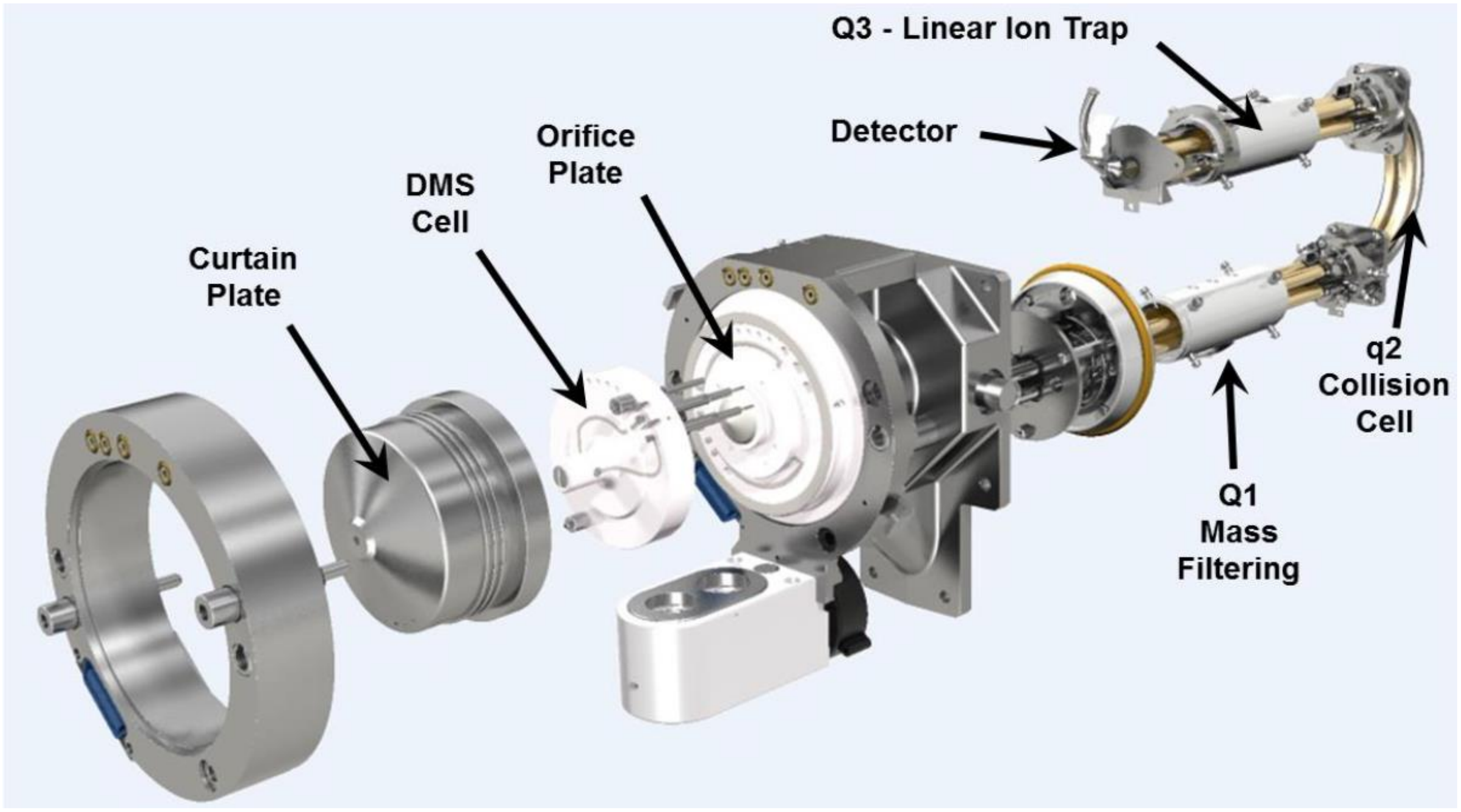

Figure S2. Schematic of the SELEXION system coupled to the QTRAP 5500 (SCIEX) hybrid linear ion trap triple-quadrupole mass spectrometer. The QJet region shown in Figure S1 is not labelled but exists between the orifice plate and Q1.

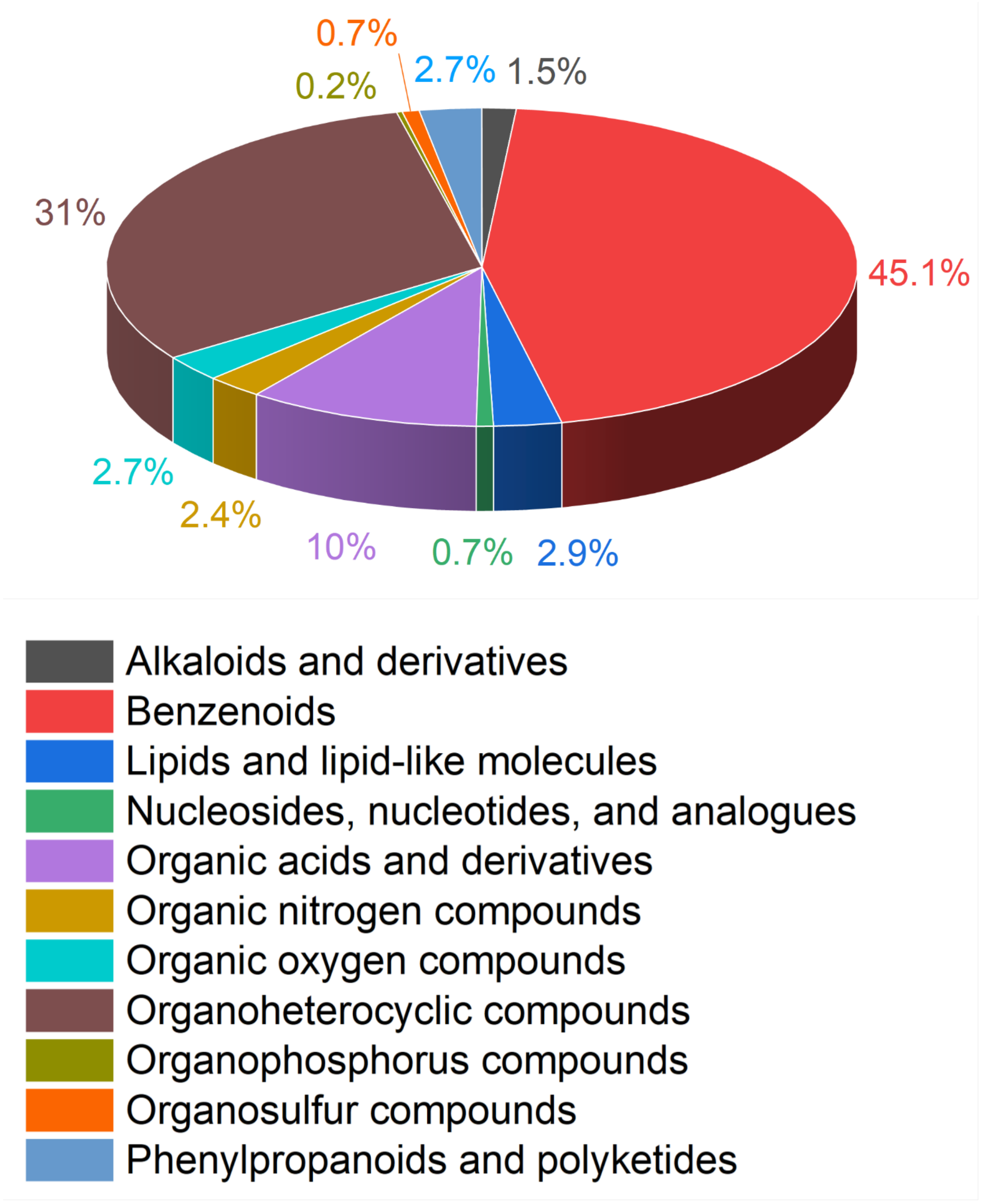

Figure S3. Superclasses, as predicted by ClassyFire, of the 410 analytes used in this study.

Supplementary Sections

S1. Feed-forward Deep Neural Network (DNN) Workflow

To determine CCSs from DMS data, we have implemented a simple feed forward neural network (NN) as depicted in Figure 3. The output of the feed forward network is represented by Y, which is a single scalar value that depends on the inputs to the network, X_i, the network weights, w_i, and the network biases, bi. Note that the inputs, weights, and biases are matrix quantities. The network is implemented with pytorch 1.7.1. Prior to model training, the complete data set (N), which is composed of mass data and DMS data for the 410 analytes, is randomly split such that ca. 70% of the molecules are used to create a model set (M) and ca. 30% of the molecules are used to create a validation set (V). The model set is used to train the NN model and the validation set is used only to assess model performance following training (i.e., V is left "out of the bag" and does not influence model training). The split will be randomized again with weight and biases from the previous iteration retained until program termination. A manual filter of desired mean percent error is added to retain NN parameters that produces prediction with high accuracy due to the random nature in training. Note that lowercase letters are used to specify node indices, activations, and responses, while uppercase letters indicate layers, inputs, and the output, Y.

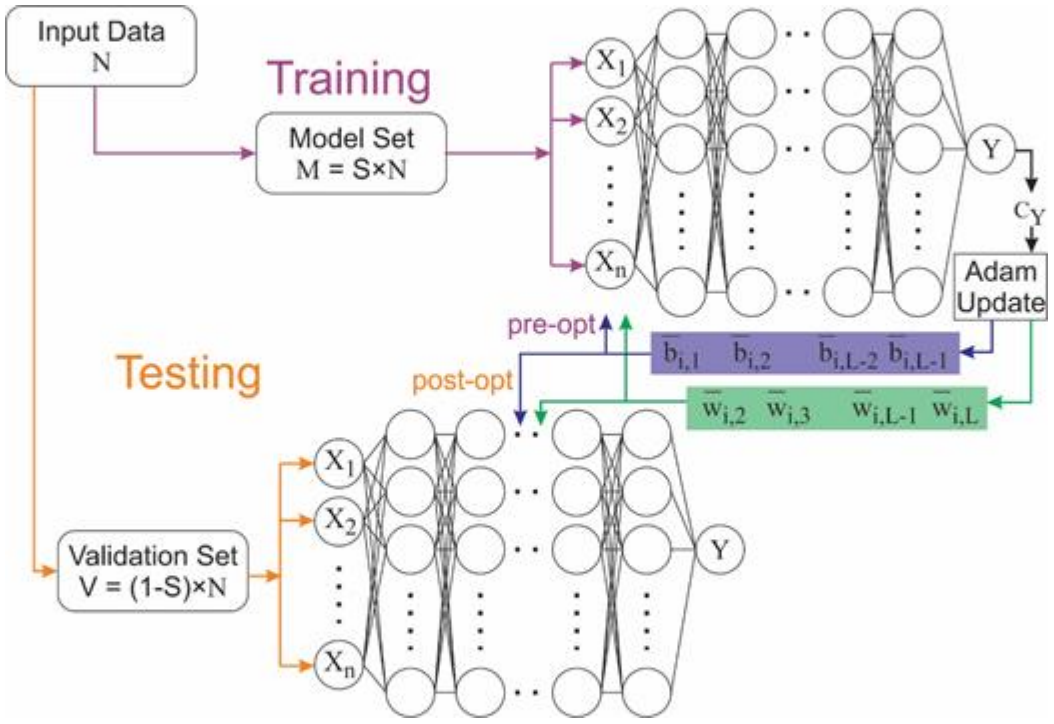
Input Data
N
Training
Model Set
M = S×N
X_1
X_2
X_n
Y
c_Y
Adam Update
pre-opt
post-opt
$\overline{b}_{i,1}$ $\overline{b}_{i,2}$ $\overline{b}_{i,L-2}$ $\overline{b}_{i,L-1}$
$\overline{w}_{i,2}$ $\overline{w}_{i,3}$ $\overline{w}_{i,L-1}$ $\overline{w}_{i,L}$
Testing
Validation Set
V = (1-S)×N
X_1
X_2
X_n
Y

Figure S4. Schematic diagram of the feed forward neural network developed for determining CCS from DMS data. Network inputs (X_i; *i.e.*, *m/z* and DMS data) are mapped to the output (Y; *i.e.*, CCS) by optimization of the biases (b) and weights (w) for the *i* nodes of the *L* layers.

Neural network node activations, a_i^l, for node *i* of layer *l* are computed using the functional form shown in Equation (S1).

$$a_i^l = \sum_{j \epsilon l-1}^{nodes(l-1)} w_{j,l}^i \cdot x_j^{l-1} + b_{i,l} \tag{S1}$$

Note that the weights in Figure S5, shown as $\overline{w}_l$, are sets of rank 2 tensors separated by layer, *l*. Each individual weight as shown in Equation (1), $w_{j,l}^i$, connects a node, *j*, in the previous layer to a node, *i*, in the current layer, *l*. The biases are layer separated vectors, shown in Figure 3 as $\overline{b}_l$ for the biases in the *l*th layer. In Equation S1 the node activations at a particular node, *i*, and in a particular layer, *l*, is denoted by $b_{i,l}$. The sum in Equation S1 connects all nodes in the previous layer to each individual node *i* in layer *l*. The response at each node, x_j^{l-1} in Equation S1, is computed from the activations via the activation function as shown in Equation S2.

$$x_j^l = f\left(a_j^l\right) \tag{S2}$$

During model development, different activation functions are used for testing, however rectified linear unit (ReLU) is the chosen function used to produce results.[1]

Neural network weights and biases are updated based upon the output cost function, C_Y, given in Equation S3.

$$C_Y = \frac{1}{N}\sum_{i=1}^{N}(Y - Y_\alpha)^2 \tag{S3}$$

Calculation of the cost function is performed at the output of the NN using the model output, Y_α, and the expected output, *Y*. The cost function decreases as the neural network outputs converge to the values of the known training set.

Weight and bias updates are handled via momentum, specifically through the Adam routine, implemented as described in the work of Kingma and Ba.[2] Briefly, parameter updates are based upon the gradients of the quadratic cost function and the prototype cost function separately. At every iteration of training, a parameter, Φ, which represents either a weight or bias, is updated according to Equation S4

$$\Phi_{t+1} = \Phi_t - \Delta_t^C \tag{S4}$$

where Φ_{t+1} and Φ_t represent the parameter at the next iteration, *t*+1, and the current iteration, *t*. Updates to the parameters are defined by Δ_t^C, which depends on the gradient of the quadratic cost. Each of the parameter updates, Δ_t, is defined as in Equation S5.

$$\Delta_t = \alpha'_t \cdot \nabla'_t / \sqrt{v_t + \xi} \tag{S5}$$

$$\alpha'_t = \alpha \cdot \frac{\sqrt{1-(\beta_2)^t}}{1-(\beta_1)^t} \tag{S6}$$

$$\nabla'_t = \nabla'_{t-1} \cdot \beta_1 + (1-\beta_1) \cdot \nabla_t \tag{S7}$$

$$v_t = v_{t-1} \cdot \beta_2 + (1-\beta_2) \cdot (\nabla_t)^2 \tag{S8}$$

Equations S6, S7, and S8 follow from the work of Kingma and Ba, and the representation for the step size, Equation S5, follows their suggestions for code simplicity.[2] In Equations S6 and S7, ∇_t represents the parameter gradients back-propagated through the network at each training iteration, *t*. The constants $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are unchanged from the original works and $\xi = 1\cdot10^{-8}$ is a small number to avoid zero division. In Equation S11, α is a constant and determines the initial step size for the optimizations. The quadratic update is given the possibility for different initial step values in our implementation. We have found for this work that an initial step size of α = 0.01 works best for the quadratic momentum updates.

A set of model/validation split of 70:30 to 95:05 is used to assess mean absolute percent errors (MAPEs) across several model/validation split. The NN model is built using pytorch 1.7.1 in accordance with Figure S5 using ReLU as the activation function with 5 layers, 9 input nodes, 14 nodes per hidden layer and 1 output node for CCS prediction. The model set itself is also split into a 90:10 training/test split for internal validation, preventing the NN from over fitting the data. However, due to the limited size of the database, the NN performs, at best, with a 2.92 ± 0.33% MAPE (Table S1). The MAPE decreases as the size of the model set increases, suggesting that a larger database may allow the NN to predict CCSs with a higher accuracy. Performance of each model:Validation split can be found within the Excel sheets accompanying this supplementary material. The excel files contain 2 sets of predictions done by the DNN for Table 1 and Table S1.

Table S1. Mean percent errors (MPEs) of 100 different samples for CCSs predicted by the various percentage of the model-validation split ranging from 70:30 (model:validation) to 90:10 filtered from the middle of the CCS distribution. All data are generated using ReLU activation function with a total of 5 layers and 14 nodes each.

model:Validation split	MAPE ± σ_{MAPE}
70:30	3.09 ± 0.21
75:25	3.01 ± 0.19
80:20	2.99 ± 0.25
85:15	2.97 ± 0.26
90:10	2.92 ± 0.33
95:05	2.95± 0.43