Electronical Supplementary Information

Sigma Profiles in Deep Learning: Towards a Universal Molecular Descriptor

Dinis O. Abranches,[a] Yong Zhang,[a] Edward J. Maginn *[a] and Yamil J. Colón *[a]

[a] Department of Chemical and Biomolecular Engineering, University of Notre Dame, Notre Dame, Indiana 46556, United States.

* Corresponding Authors: Yamil J. Colón (ycolon@nd.edu) and Edward J. Maginn (ed@nd.edu)

GitHub repository: https://github.com/MaginnGroup/SP_ML_CC

Contents

S1. The Sigma Profile

This section briefly examines the concept of the sigma profile (σ-profile) and summarizes the database developed by Mullins et al.,[1] which was used throughout this work. Although some of the properties of the σ-profile are illustrated in this section, a complete and detailed description of the COSMO and COSMO-RS theories is beyond the scope of this work and the interested reader is directed to seminal works in the field.[2,3]

S1.1 Sigma Profiles as Molecular Descriptors

A σ-profile is an unnormalized histogram of the screened surface charge of a molecule. It is defined as:[2,3]

$$\sigma\text{-}Profile = P(\sigma) \cdot A = \frac{n_i(\sigma)}{n} \cdot A \quad \text{(S1)}$$

where $n_i(\sigma)$ is the number of area segments with screened charge σ, n is the total number of area segments of the molecule, and A is the total area of the molecule. Some confusion arises in the literature due to the term *σ-profile* being used interchangeably to mean either the normalized probability distribution $\left(P(\sigma) = \frac{n_i(\sigma)}{n}\right)$ or the unnormalized histogram ($P(\sigma) \cdot A$). Throughout this work, the term *σ-profile* always refers to the unnormalized histogram ($P(\sigma) \cdot A$).

The screened charges mentioned in the previous paragraph are calculated using quantum chemistry methods, in particular density-functional theory (DFT). Briefly, a cavity representing the surface of a molecule is constructed and embedded in a continuum solvent with infinite permittivity (COSMO solvation model). The geometry of this molecule is then optimized and, at each step, the screened charge of each area segment is computed. Because the molecule is optimized in a continuum solvent, rather than vacuum, its σ-profile is a more faithful representation of its geometry, polarity, and potential for non-covalent interactions in condensed phases.

In the main text of this work, some claims were made regarding the advantages of σ-profiles as molecular descriptors in deep learning. These claims are now examined in more detail. To illustrate the type of chemical information encoded in σ-profiles, the σ-profiles of ethane, acetaldehyde, and ethanol, taken from the Mullins et al.[1], are depicted in Figure S1. Note that the three compounds form a series with increasing polarity but similar structural backbones. Also note that P(σ)·A values located at σ values lower than -0.0082 e/Å^2 are usually regarded as the positive polarity of the molecule, whereas those located at σ values larger than 0.0082 e/Å^2 represent the negative polarity of the molecule.[2,4] Consequently, P(σ)·A values for σ values located between -0.0082 and 0.0082 e/Å^2 are representative of the apolar surface of the molecule.

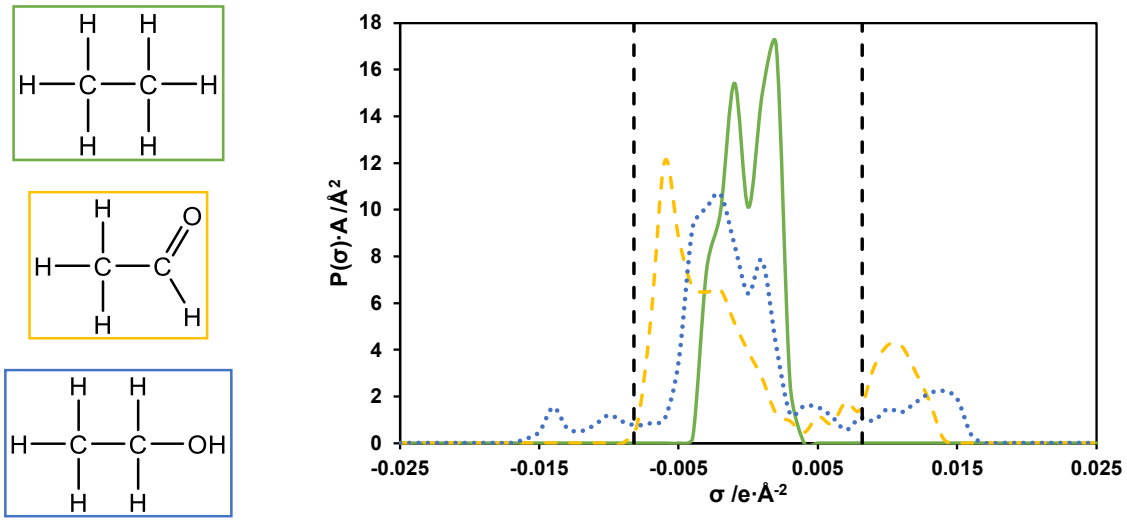

Figure S1. *Chemical structures (left) and σ-profiles (right) of ethane (green, full line), acetaldehyde (yellow, dashed line), and ethanol (blue, dotted line). Vertical dashed lines represent σ = -0.0082 e/Å² and σ = 0.0082 e/Å². Data taken from Mullins et al.*[1]

The σ-profile of ethane, being a fully apolar molecule, is located at σ values around 0 e/Å. Going from ethane to acetaldehyde, which represents the replacement of one proton by an oxygen, a new σ-profile region arises for σ values larger than 0.0082 e/Å². This is to be expected, since the oxygen adds a significant negatively charged surface area (or, in other words, a positive screened charge) to the molecule. Perhaps more interesting is the deviation of the main apolar peak, which for ethane is located around a σ value of 0 e/Å while for acetaldehyde is located around -0.006 e/Å. This is the result of electron density asymmetry in the carbon-oxygen double bond; because oxygen is more electronegative than carbon, a dipole is formed due to electron transfer, with the carbon becoming slightly more positive than its ethane counterpart. Finally, a new peak in the positive area of the molecule (σ < -0.0082 e/Å²) arises for ethanol, as expected, while the apolar peak is located at a position intermediate of that of ethane and acetaldehyde. Note that the oxygen peak is located at a larger σ value, indicating a more pronounced negative polarity, which is due to a pronounced electron density transfer across the O-H bond.

The previous paragraph illustrates some of the chemical properties encoded within the σ-profile, including subtle effects such as electron density asymmetries. Because of this, the σ-profile captures quite well the polarity of each local substructure of the molecule, which is the reason behind its high performance in describing non-covalent interactions. Having demonstrated the type of chemical information captured by the σ-profile, its usefulness with regards to a fixed input size for machine learning methodologies is now discussed. Figure S2 depicts the σ-profiles, taken from the Mullins et al.,[1] of three alkanes with different size: decane (10 carbons), pentadecane (15 carbons), and icosane (20 carbons).

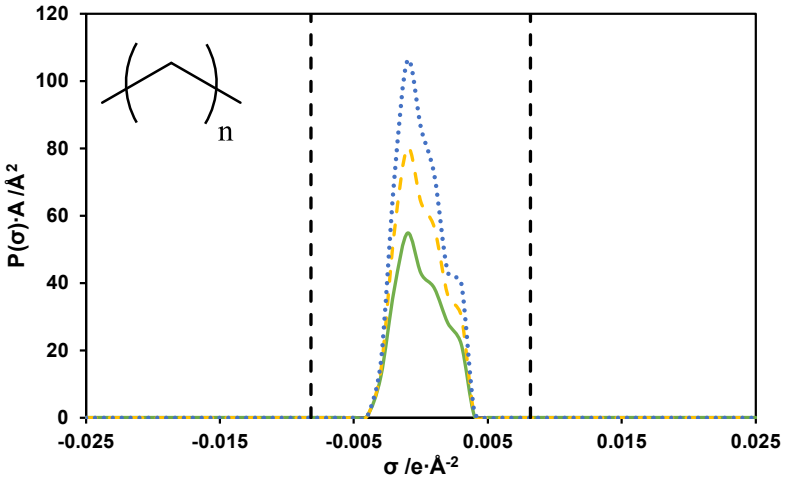

Figure S2. *Sigma profiles of decane (green, full line, n=8), pentadecane (yellow, dashed line, n=13), and icosane (blue, dotted line, n=18). The generical chemical structure of these alkanes is provided as inset. Vertical dashed lines represent σ = -0.0082 e/Å² and σ = 0.0082 e/Å².Data taken from Mullins et al.*[1]

Figure S2 shows that the main difference between the σ-profiles of decane, pentadecane, and icosane is the size of their apolar peak (the σ-profile peak around a σ value of 0 e/Å). This is quite important for machine learning methodologies. The size of most of the molecular descriptors mentioned in the main text (SMILES and SELFIES strings, fingerprints, graph representations, etc.) increases with the size of the molecule.[5] Due to the inherent architecture of neural networks (and other machine learning methodologies), the size of their input must be fixed. Thus, if the size of the molecular descriptor increases with increasing size of the molecule, the neural network input size must be fixed and equal to the largest molecule available. However, if the σ-profile is used as a feature set in deep learning, the size constraint of this input is no longer the structural size of the molecule, as this is captured by the value of the σ-profile at each σ value. In other words, because the screened charges of virtually all neutral molecules are contained in the range between σ = -0.025 e/Å² and σ = 0.025 e/Å², the size of a σ-profile input to a neural network depends only on the resolution chosen for the σ-profile (i.e., number of points selected from the σ-profile plot).

S1.2 Sigma Profile Database

All σ-profiles used in this work were obtained from the database developed by Mullins et al.[1] This database contains the σ-profiles for 1432 different compounds, including, among others, alkanes, alkenes, alkynes, alcohols, ketones, amines, carboxylic acids, esters, aromatics, and halogenated compounds. While most of the database consists of organic compounds, some inorganic acids, bases, and gases are also included, as well as water. In their work, Mullins et al.[1] computed all σ-profiles using the software package DMoL. In particular, DFT was used with the DNP v4.0.0 basis set, GGA/VWN−BP functional, and the COSMO continuum solvent model with infinite permittivity. This procedure can be carried out, at present, using fully open-source software.

The data contained in the Mullins et al.[1] database was read using an in-house Python script and converted into a single csv file, which is included in the GitHub repository associated with this work. This Python script calculates the molar mass of each compound and validates its database entry by cross-checking with the Chemical Identifier Resolver (CIR) database, through its Python wrapper (CIRpy).[6] All changes to the original database were registered and are listed in the GitHub repository. In particular, the following typos were identified:

Index 78: CAS number changed from "4390-04-09 00:00:00" to "4390-04-9"
Index 158: CAS number changed from "6094-06-2" to "6094-02-6"
Index 162: CAS number changed from "7642-10-06 00:00:00" to "7642-10-6"
Index 426: CAS number changed from "15798-64-8" to "4170-30-3"
Index 1000: CAS number changed from "288761" to "2690-08-6"
Index 1057: CAS number changed from "136911" to "2274-11-5"
Index 1077: CAS number changed from "2148878" to "7783-06-4"
Index 1080: CAS number changed from "2125683" to "7719-12-2"
Index 1106: CAS number changed from "2125597" to "7719-09-7"
Index 1240: CAS number changed from "2998-08-05 00:00:00" to "2998-08-5"
Index 1259: CAS number changed from "4445-07-02 00:00:00" to "4445-07-2"
Index 1692: CAS number changed from "91352" to "2150-02-9"

Each σ-profile extracted from the database developed by Mullins et al.[1] is composed of 51 data points. These represent the σ-profile values at 51 different σ values, from σ = -0.025 e/Å^2 to σ = 0.025 e/Å^2, in intervals of 0.001 e/Å^2. All 51 σ-profile values were used as the input for the convolutional neural networks developed in this work. This is illustrated in Figure S3 for three assorted compounds (6-aminohexanol, chlorosulfonic acid, and metylamine).

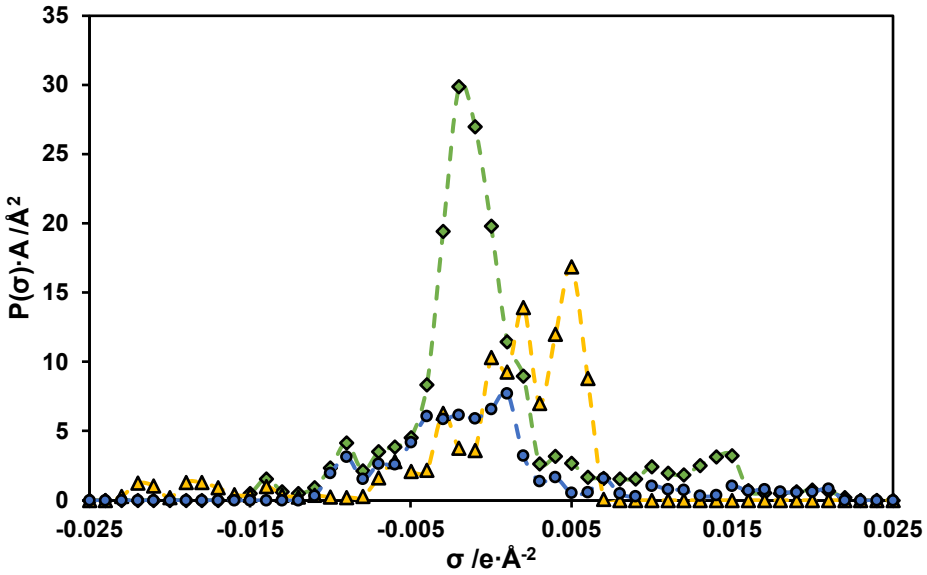

Figure S3. *Sigma profiles of 6-aminohexanol (green diamonds), chlorosulfonic acid (yellow triangles), and methylamine (blue circles), highlighting the 51 data points of each σ-profile, which are used in this work as the input to convolutional neural networks. Data taken from Mullins et al.*[1]

As usual in the machine learning literature, the input data to a neural network should be normalized, as it enhances the convergence smoothness of the training of the network and allows for the use of advanced fitting techniques such as regularization. This means that the σ-profile data at each σ value must be normalized. However, these data do not follow a typical normal distribution, as illustrated in Figure S4.

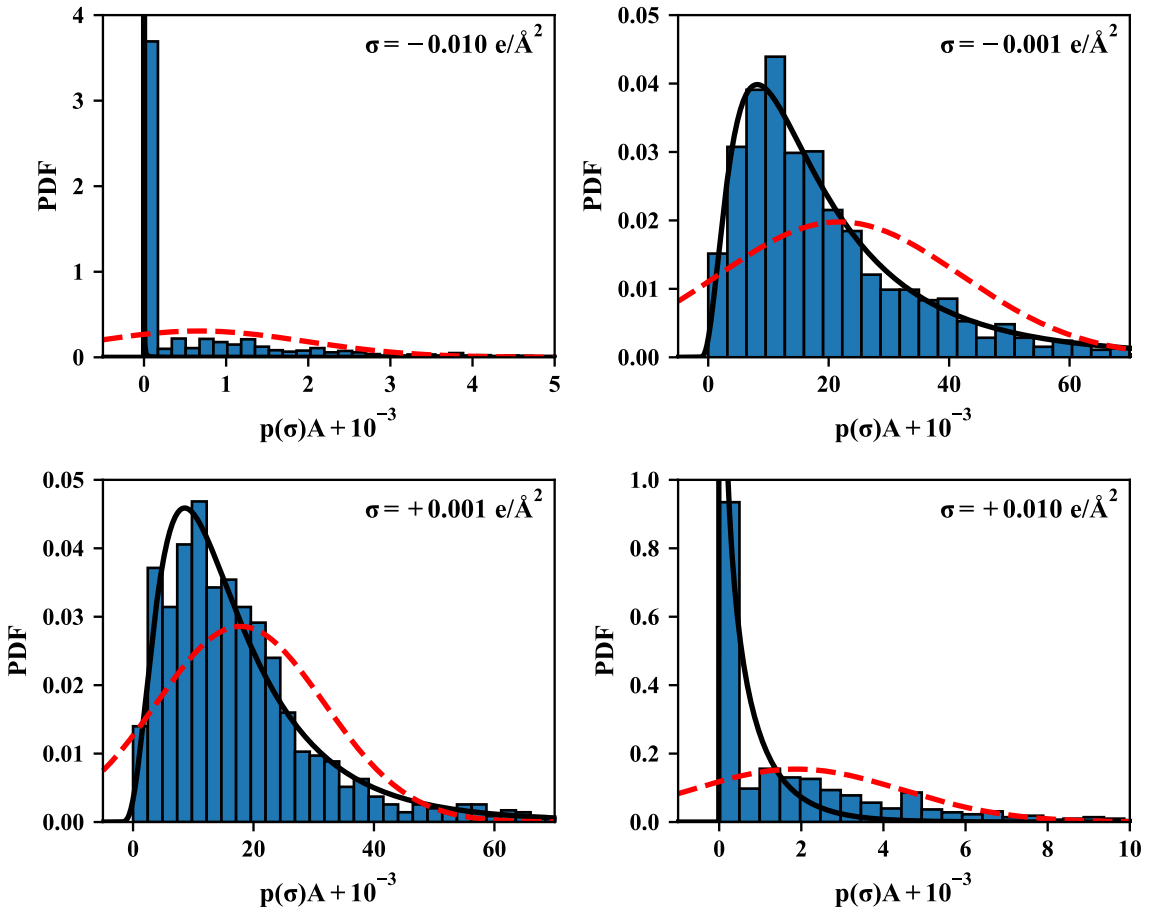

Figure S4. *Histograms, in the form of probability density functions (PDF), for the σ-profile at σ values of -0.010 e/Å² (top left corner), -0.001 e/Å² (top right corner), +0.001 e/Å² (bottom left corner), and +0.010 e/Å² (bottom right corner). Lines represent fitted log-normal (black, full) or normal (red, dashed) probability distributions.*

Figure S4 illustrates the distribution of the σ-profile at different σ values, showing that the data follows more closely log-normal probability distributions than normal distributions. This means that the logarithm of the data, not the data itself, follows a normal distribution. Thus, the data are normalized in this work using log-standardization:

$$p(\sigma)A' = \frac{\ln[p(\sigma)A+10^{-3}]-\langle\ln[p(\sigma)A+10^{-3}]\rangle}{s_{\ln[p(\sigma)A+10^{-3}]}} \quad \text{(S2)}$$

where $p(\sigma)A'$ represents a normalized σ-profile series for a specific σ, whereas $\langle\ln[p(\sigma)A+10^{-3}]\rangle$ and $s_{\ln[p(\sigma)A+10^{-3}]}$ are the average and standard deviation of the $\ln[p(\sigma)A+10^{-3}]$ data for a

specific σ value. Note that the σ-profile can take the value of zero at any σ value; as such, a small buffer (10^{-3}) is added to allow for the logarithmization of the data. The data depicted in Figure S4 are now presented in Figure S5 after being normalized using Equation S2.

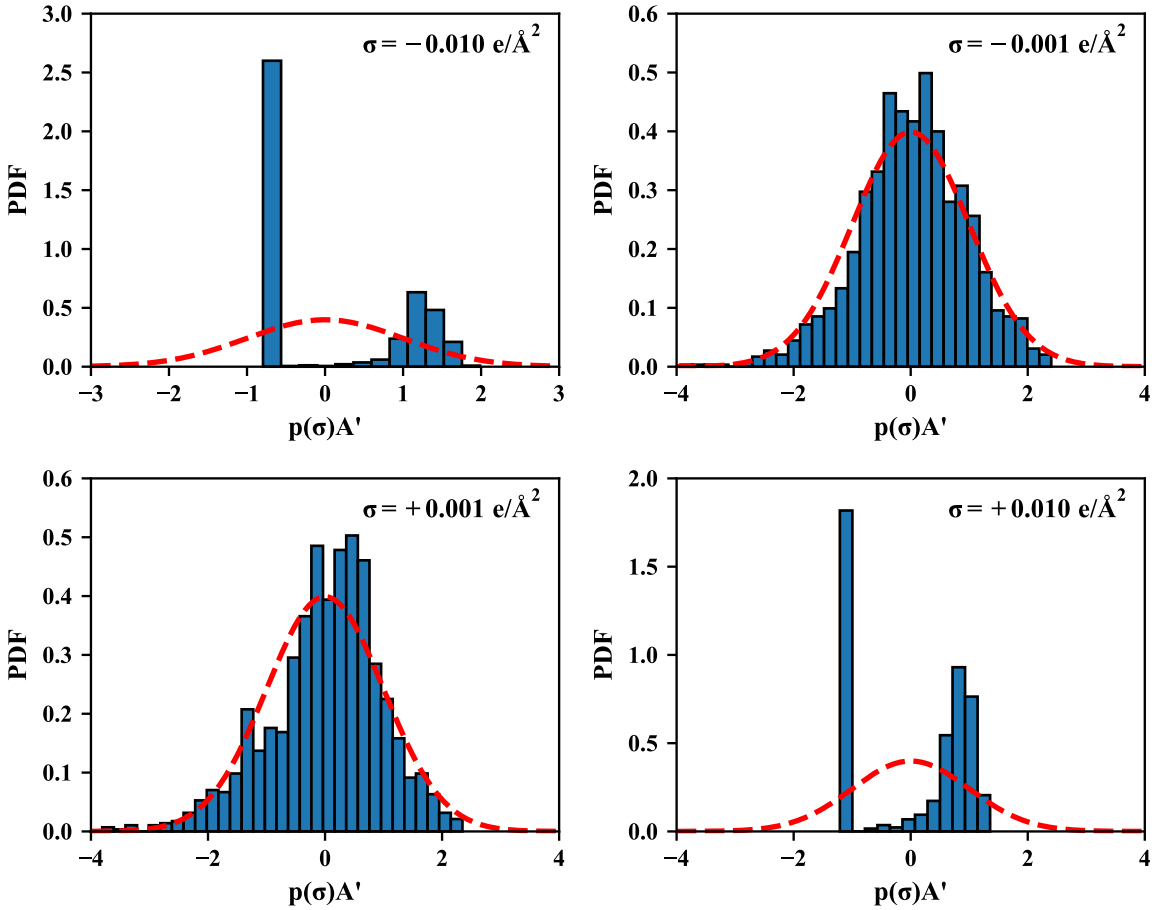

Figure S5. *Histograms, in the form of probability density functions (PDF), for the normalized σ-profiles, as per Equation S2, at σ values of -0.010 e/Å² (top left corner), -0.001 e/Å² (top right corner), +0.001 e/Å² (bottom left corner), and +0.010 e/Å² (bottom right corner). Red dashed lines represent fitted standard probability distributions.*

As Figure S5 clearly shows, the normalization procedure represented by Equation S2 works well for σ-profile series at small σ values but not for larger σ values. However, this normalization procedure is still more accurate than standardization alone and was, thus, employed throughout this work.

S2. Physicochemical Properties

This section examines the physicochemical properties used in this work, particularly their distributions and normalization procedures. The properties studied in this work are:

- Molar mass
- Normal boiling temperature
- Vapor pressure at 25 ºC
- Density (at 20 ºC or temperature-dependent)
- Refractive index at wavelength 589 nm (at 20 ºC or temperature-dependent)
- Aqueous solubility (at 25 ºC or temperature-dependent)

Except for molar mass, which was calculated from the molecular structure of each compound, the data for all properties were taken, when available, from the CRC Handbook of Chemistry and Physics (Internet version).[7] Note that density, refractive index, and aqueous solubility are available at different temperatures. In a first instance of this work, these three properties were selected at a specific temperature (20 ºC and 25 ºC). Then, for these three properties, the entire dataset was used, and temperature was explored as a feature in the neural networks developed in this work.

S2.1 No Temperature Input

The properties listed above were chosen to demonstrate the ability of σ-profiles to encode the necessary information to accurately fit and predict several different material properties. In particular, molar mass was chosen because i) it is available for all compounds included in the σ-profile database and ii) it shows that σ-profiles also encode molecule structure information that can be retrieved (this is not a trivial result, given how the position of different polarity peaks changes with the local structural environment of the molecule, as demonstrated in Figure S2). The sizes and data ranges of the datasets for each fixed-temperature property are reported in Table S1. The datasets are depicted as histograms in Figure S6, in a similar fashion to the analysis performed in the previous section for the σ-profile dataset.

Table S1. *Summary (size and range) of the datasets used in this work to train convolutional neural networks (no temperature input).*

Property	Size	Range
Molar Mass	1432	2 – 447 g/mol
Normal Boiling Temperature	1208	-192 – 463 ºC
Vapor Pressure (25 ºC)	594	10^{-6} – 10^{3} kPa
Density (20 ºC)	711	0.6 – 3.3 g/mL
Refractive Index (20 ºC)	834	1.26 – 1.87
Aqueous Solubility (25 ºC)	327	10^{-7} - 10^{4} g/kg

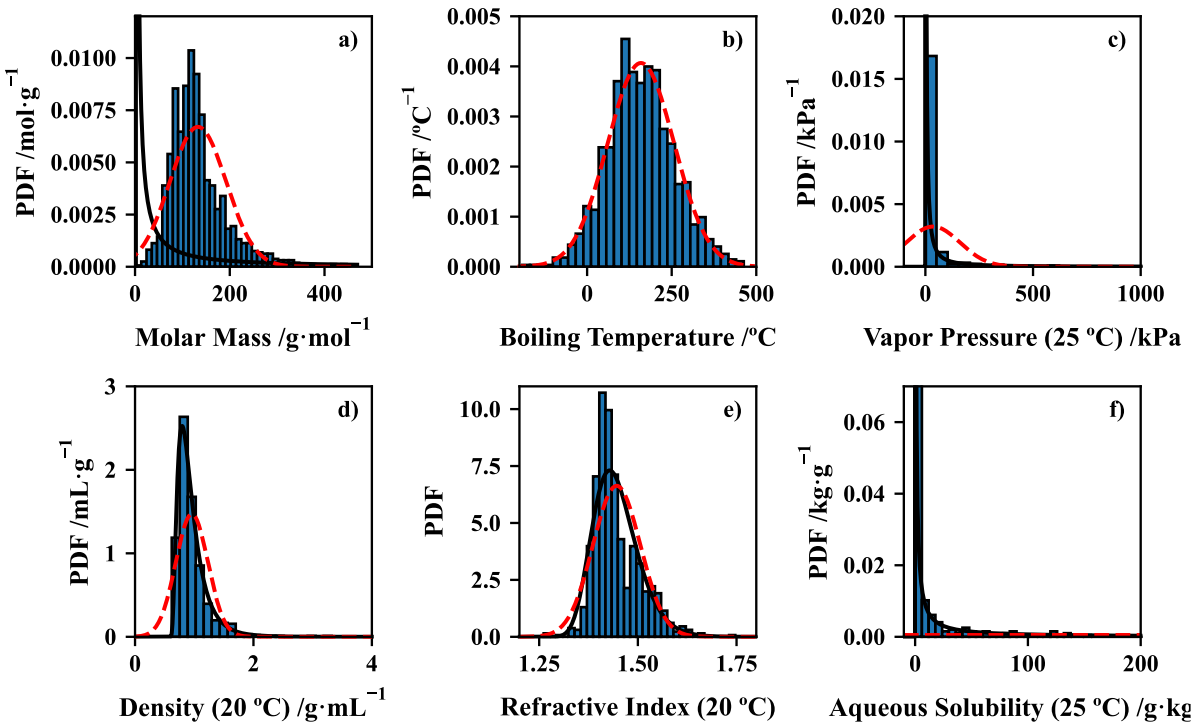

Figure S6. *Histograms, in the form of probability density functions (PDF), for the temperature-independent properties studied in this work, namely molar mass (a), normal boiling temperature (b), vapor pressure at 25 ºC (c), density at 20 ºC (d), refractive index at 20 ºC (e), and aqueous solubility at 25 ºC (f). Lines represent fitted log-normal (black, full) or normal (red, dashed) probability distributions.*

Besides summarizing the structure of the datasets available, Figure S6 sheds light on the best normalization procedures for each property. Because these properties are well described by normal distributions, with the exceptions of vapor pressure and aqueous solubility, they were normalized using standardization:

$$X' = \frac{X - \langle X \rangle}{s_X} \quad \text{(S3)}$$

where X and X' represent a generic property and its normalized version, respectively, while $\langle X \rangle$ and s_X are the average and standard deviation of the dataset of X. Because vapor pressure and aqueous solubility are more accurately described by a log-normal probability distribution, they were normalized using log-standardization, akin to the procedure carried out for the σ-profile datasets but without using a non-zero buffer:

$$X' = \frac{\ln(X) - \langle \ln(X) \rangle}{s_{\ln(X)}} \quad \text{(S4)}$$

where $\langle \ln(X) \rangle$ and $s_{\ln(X)}$ represent the average and standard deviation of the logarithmized dataset of X. The normalized data using these procedures (Equations S3 and S4) are depicted in Figure S7.

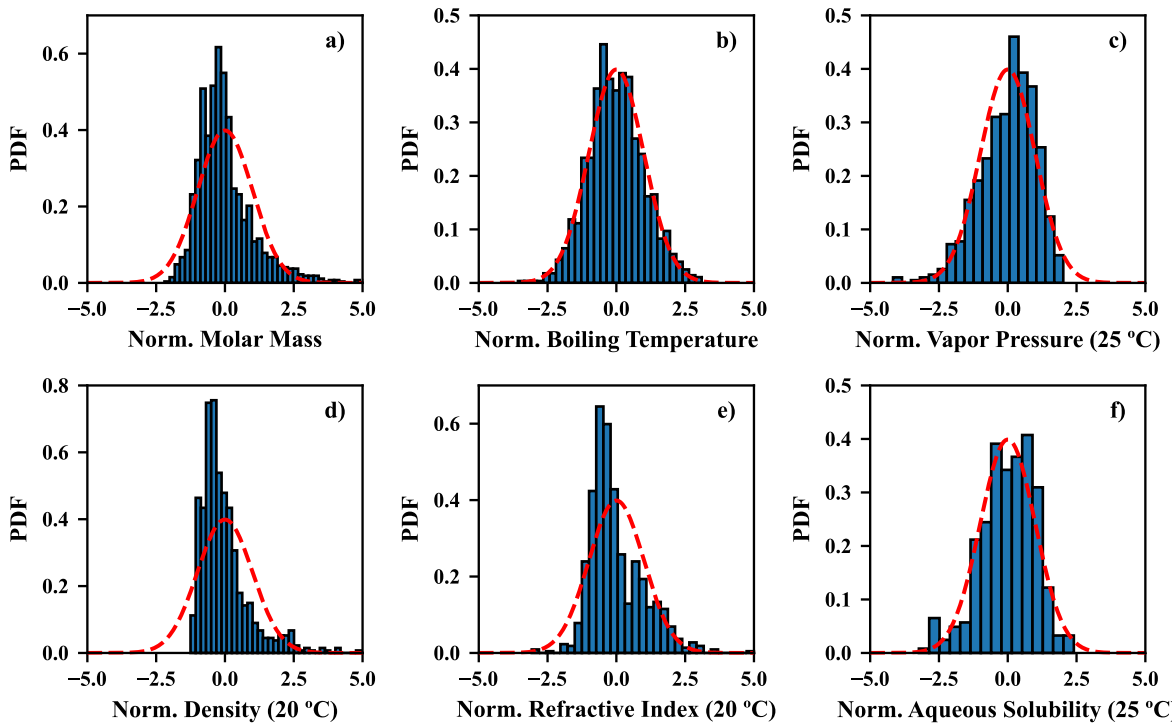

Figure S7. *Histograms, in the form of probability density functions (PDF), for the normalized, temperature-independent properties studied in this work, namely molar mass (a), normal boiling temperature (b), vapor pressure at 25 ºC (c), density at 20 ºC (d), refractive index at 20 ºC (e), and aqueous solubility at 25 ºC (f). Red dashed lines represent fitted standard probability distributions.*

Figure S7 shows that the normalization methods used on the datasets available are appropriate, as they are all well described by the standard probability distribution. It is particularly interesting to note the diversity of vapor pressure and aqueous solubility data, which was not as easy to see in Figure S6 given the fact that their values span across several orders of magnitude.

S2.2 Temperature as an Additional Feature

As mentioned in the previous section, some physicochemical properties (density, refractive index, and aqueous solubility) were also studied at all temperatures available in the CRC Handbook of Chemistry and Physics (Internet version).[7] The data are summarized in Table S2 and Figure S8, while the results of the normalization procedure are depicted in Figure S9. Not surprisingly, the data depicted in Figure S8 follow the same pattern as the data in Figure S6.

Table S2. *Summary (size and ranges) of the datasets used in this work to train convolutional neural networks using temperature as an additional feature.*

Property	Size	Property Range	Temperature Range
Density	1146	0.4 – 3.3 g/mL	-200 – 240 ºC
Refractive Index	1053	1.17 – 1.87	-100 – 134 ºC
Aqueous Solubility	518	10^{-7} – 10^{4} g/kg	-3 – 40 ºC

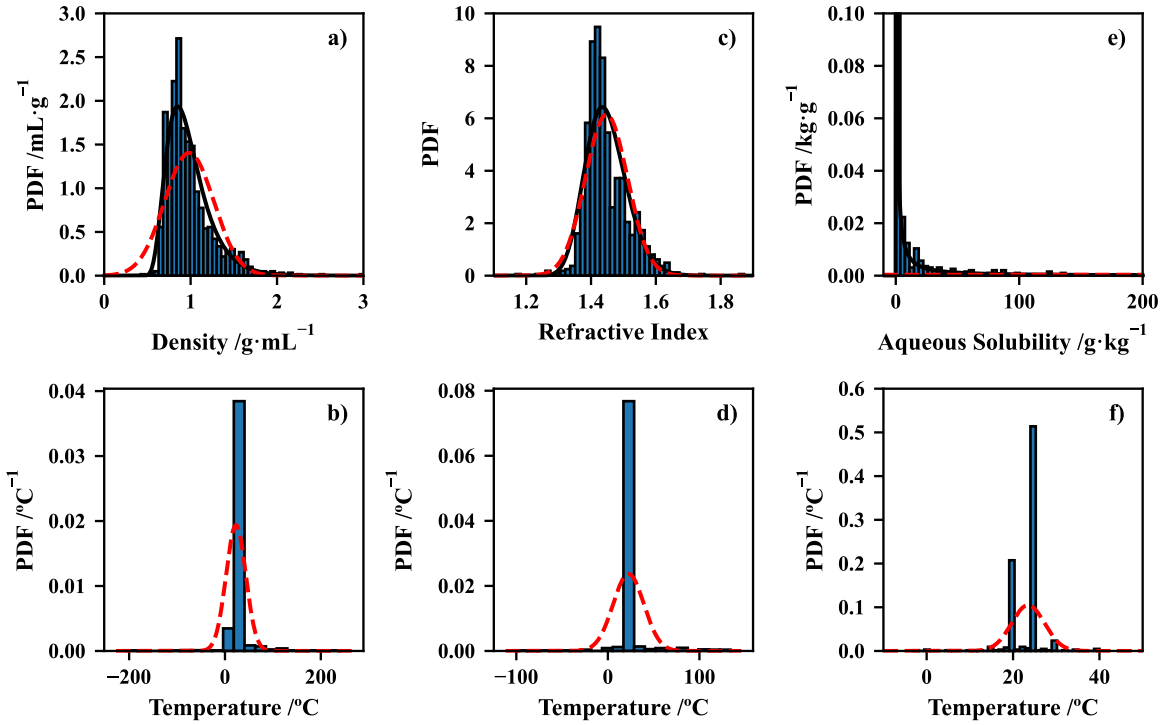

Figure S8. *Histograms, in the form of probability density functions (PDF), for the temperature-dependent properties studied in this work and their corresponding temperatures, namely density (a and b), refractive index (c and d), and aqueous solubility (e and f). Lines represent fitted log-normal (black, full) or normal (red, dashed) probability distributions.*

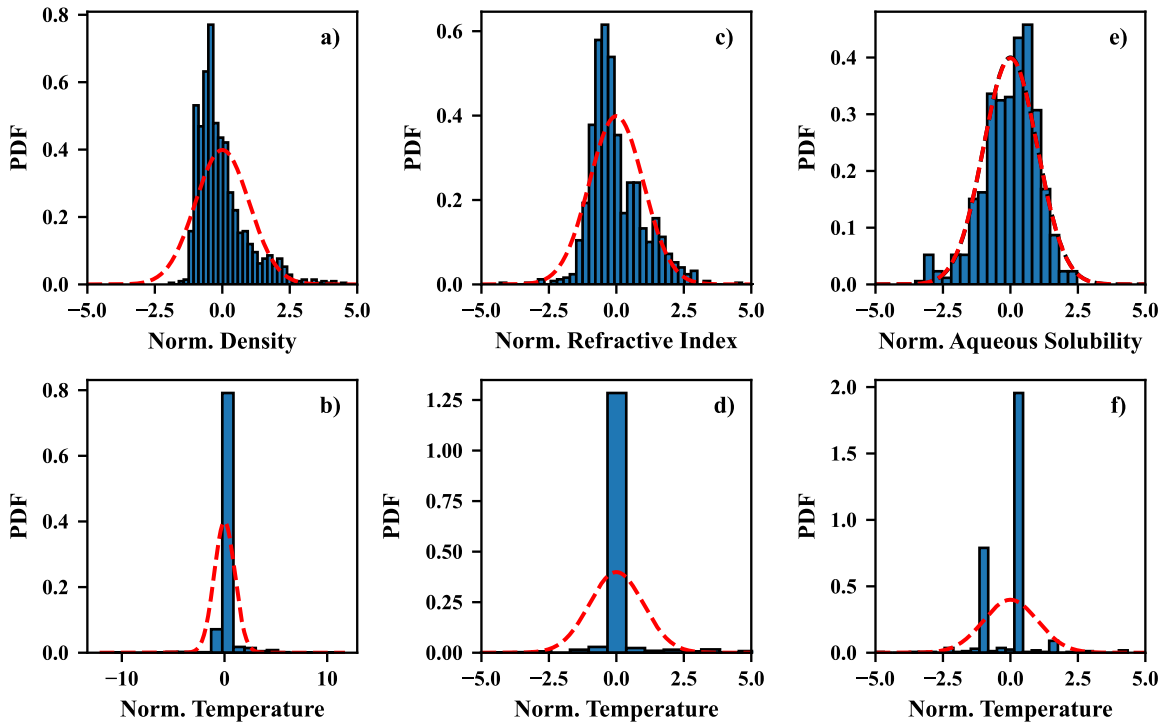

Figure S9. *Histograms, in the form of probability density functions (PDF), for the normalized, temperature-dependent properties studied in this work and their corresponding normalized temperatures, namely density (a and b), refractive index (c and d), and aqueous solubility (e and f). Red dashed lines represent fitted standard probability distributions.*

S3. Computational Details

This section provides details regarding the deep learning methodologies used throughout this work. The Python code mentioned in this section can be accessed through the GitHub repository. The Keras module of the open source Python package TensorFlow (V. 2.6.0)[8] is the main machine learning engine used in this work.

S3.1 Neural Network Architecture

As mentioned in the main text, given the relatively small size of the datasets used in this work, a typical densely connected deep neural network could not be used, as this would lead to neural networks with an enormous number of fitting parameters. Thus, convolutional neural networks were used instead. Convolution is a technique useful for large inputs where data are somewhat ordered, such as temporal plots, where the values around a given time instance are highly correlated, or images, where pixels in each region of the image are also highly correlated.[9] The generic architecture of each convolutional neural network (CNN) developed is depicted in Figure 1 of the main text. Note that a different CNN was developed to fit and predict each different property. This subsection provides details about their general architecture.

The main input of each CNN is a 1-dimensional vector composed of 51 entries (the 51 σ-profile data points of each molecule). This input is passed through two sets of convolution (TensorFlow module *Conv1D*) and pooling layers (TensorFlow module *AveragePooling1D* or *MaxPool1D*). The number of filters, kernel size, and number of strides in each convolution layer are hyperparameters to be tuned, as will be discussed in section S3.3. The pool size of the pooling layers also is set as a hyperparameter to be tuned and the number of strides is set to be the same as the pool size. In both convolution and pooling layers, padding is allowed (TensorFlow keyword *same*). Thus, if the ratio between the input size of the layer and its kernel or pool size is not an integer, zeros are evenly added at the beginning and end of the input until this ratio becomes a whole number. After the two sets of convolution and pooling layers, the output is flattened and passed through two densely connected layers (TensorFlow module *Dense*). The number of nodes on each of these layers is a hyperparameter to be tuned.

The activation function of all nodes, be it in a convolution or dense layer, is set as a hyperparameter to be tuned. Because the work here developed is regression deep learning, rather than classification, two types of activation functions were examined: ReLU and Swish. Swish is a novel activation function that mitigates some of the well-known disadvantages of the ReLU activation function, outperforming it in most deep learning methodologies.[10]

S3.2 Neural Network Fitting

To ensure an efficient fitting of the networks and to maximize their generalization capability, several techniques were used, namely stratified data splitting, regularization, and early stopping, as detailed below.

The dataset of each property was initially split into a fitting dataset (90%) and a testing dataset (10%). This testing dataset was never used in any of the tuning or fitting procedures described below. Instead, it was employed to independently evaluate the final model obtained for each property. The fitting dataset of each property was further split into training (80%) and validation (20%) sets using stratified splitting. The stratification was based on the value of the labels. In each data split performed, the labels of the dataset were divided into equidistant bins. The number of bins was selected as the minimum number which ensured that each bin possessed at least five data points. There were no restrictions regarding the maximum number of data points per bin. Finally, the splitting was performed such that the training/validation ratio for each bin corresponds approximately to the same overall ratio (80/20). This procedure ensures that data lying in the edges of each dataset range is still well represented in the validation set.

L2 regularization was employed on all layers that contain trainable parameters (convolution and dense layers) for both the weights and biases. The lambda parameter of the regularization was left as a hyperparameter to be tuned. The use of L2 regularization is a great asset to prevent overfitting of the neural networks and to maximize their generalization capability. In particular, L2 regularization, as opposed to L1 regularization, is better suited to feature sets that present a high degree of multicollinearity (local correlation), such as those used in this work (σ-profiles).

The Adam optimizer[11] with a mean-squared-error loss function was used to fit all neural networks developed in this work. The learning rate and its exponential decay rates (β_1 and β_2) were left as hyperparameters to be tuned. Each neural network was fitted for an arbitrary number of epochs, and early-stopping was used with a patience of 500 epochs (number of epochs with no improvement after which training is stopped) to halt the fitting when the mean-absolute-error of the validation set stopped improving. Note that the batch size during fitting was also left as an hyperparameter to be tuned. Finally, the initial values of the biases were set to zero and the initial guesses for all weights of each CNN were obtained using the methodology developed by He et al.[12] (TensorFlow keyword *HeUniform*), which is particularly efficient at initializing the weights of convolutional neural networks.

S3.3 Hyperparameter Optimization

The hyperparameters of the convolutional neural networks to be optimized, which were mentioned throughout the previous two sections, are listed in Table S3. The hyperparameter tuning was performed using the open source Python package Sherpa (V. 1.0.6).[13] Because of the tremendous number of possible hyperparameter combinations, which renders any attempt to test each combination computationally infeasible, a two-fold tuning strategy was adopted by first using Bayesian optimization and then a local search algorithm, as will be detailed below. Regardless of the tuning algorithm, the objective function (f) was defined as follows:

$$\begin{cases} f = -\ln(R^2)\ , if\ R^2 > 0.1\ and\ N < \frac{N_D}{3} \\ f = -\ln(0.1)\ , if\ R^2 \leq 0.1\ or\ N > \frac{N_D}{3} \end{cases} \quad \text{(S5)}$$

where R^2 is the coefficient of determination of the validation set, N is the total number of trainable parameters of the CNN (not to be confused with hyperparameters), and N_D is total number of data points in the training set. Note that this objective function greatly limits the number of trainable parameters of the CNN, which is a technique here employed to prevent overfitting and maximize generalization capability.

Table S3. *List of the hyperparameters, segmented into architecture and fitting hyperparameters, optimized for each convolutional neural network developed in this work, along with their possible values.*

Hyperparameter	Range
Architecture Hyperparameters	
Activation Function	*ReLU* | *Swish*
Conv. Layer 1: Number of Filters	1 – 10
Conv. Layer 1: Kernel Size	1 – 10
Conv. Layer 1: Number of Strides	1 – 10
Pooling Layer 1: Type	*Average* | *Maximum*
Pooling Layer 1: Pool Size	1 – 10
Conv. Layer 2: Number of Filters	0 – 10[a)]
Conv. Layer 2: Kernel Size	1 – 10
Conv. Layer 2: Number of Strides	1 – 10
Pooling Layer 2: Type	*Average* | *Maximum*
Pooling Layer 2: Pool Size	1 – 10
Dense Layer 1: Number of Nodes	1 – 10
Dense Layer 2: Number of Nodes	0 – 10[b)]
Fitting Hyperparameters	
Adam Optimizer: Learning Rate	10^{-5} – 10^{-1}
Adam Optimizer: β_1	0.9 – 0.999
Adam Optimizer: β_2	0.9 – 0.9999
L2 Regularization: λ	10^{-5} – 10^{-1}
Batch Size	8 – 64

a) If the number of filters is zero, the 2nd convolution layer is not used.
b) If the number of nodes is zero, the 2nd dense layer is not used.

As stated above, the first stage of the hyperparameter tuning is Bayesian optimization. During this step, the fitting hyperparameters were fixed (a learning rate of 10^{-3}, β_1=0.99, β_2=0.999, λ=10-3, and a batch size of 16) and the architecture hyperparameters were tuned. A schematic representation of this procedure is depicted in Figure S10.

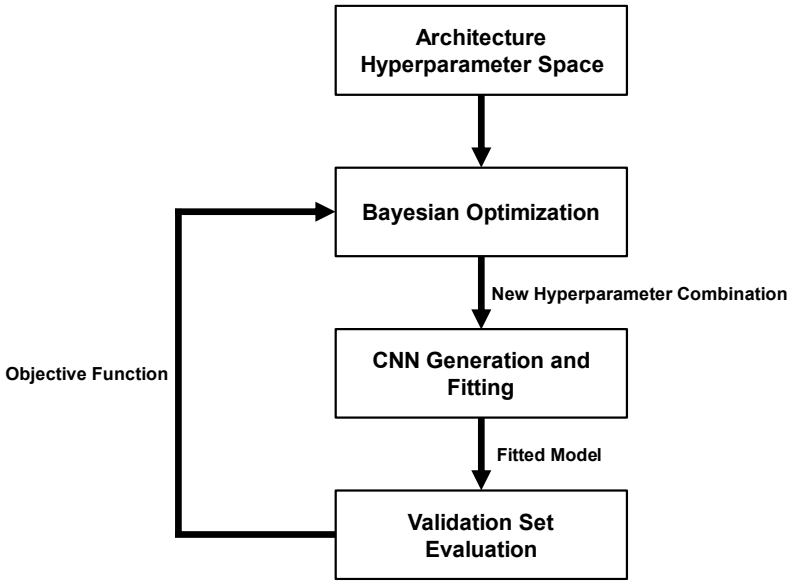

Figure S10. *Schematic representation of the Bayesian optimization procedure employed to tune the architecture hyperparameters of the convolutional neural networks developed in this work.*

The Bayesian optimization procedure depicted in Figure S10 utilizes roughly 2000 trials (2000 different hyperparameter combinations) and repeats each trial three times. Because this is an initial hyperparameter tuning, no effort is made to control the inherent randomness of the process. That is, the splitting of the fitting dataset into training and validation sets as well as the weight initialization were performed at random for each trial repetition. The objective function returned to the algorithm is the average of the two best trials. The Bayesian optimization, as implemented in the Sherpa Python package,[13] employs a Gaussian process model and an expected improvement acquisition function. Finally, to reduce computation time, the algorithm was run using parallel computation, and up to 30 trials are run in parallel with local penalization.

A Local Search algorithm was employed after the initial Bayesian optimization step. This algorithm takes a configuration seed (a hyperparameter set that has been pre-optimized) and makes small, stepwise changes to its parameters, trying to find a slightly different configuration that performs better. This algorithm was first applied to the architecture hyperparameter space using the best Bayesian optimization hyperparameter set as the configuration seed. Then, the results of this first iteration were used as the next configuration seed, and the fitting hyperparameters were tuned. This process was repeated back and forth five times, as illustrated in Figure S11. During this entire process, all random processes were fixed using seeds.

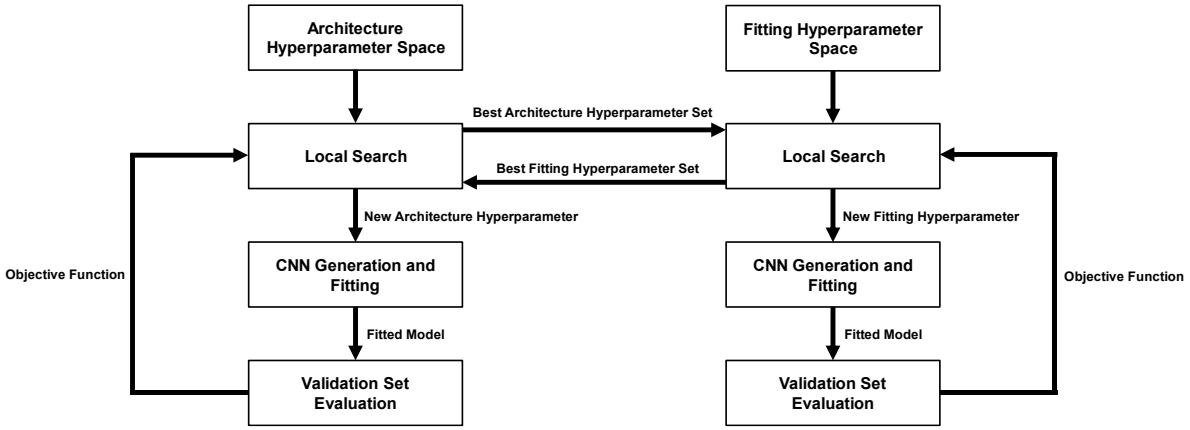

Figure S11. *Schematic representation of the Local Search optimization procedure employed to tune the architecture hyperparameters of the convolutional neural networks developed in this work.*

Finally, once the best architecture and fitting parameters were found, the random seed was allowed to change, and several attempts were performed to fit the final network.

S4. Additional Results

This section presents the intermediate results of this work, including the results of the hyperparameter tuning and the performance of the convolutional neural networks. The results are divided between those relating to the temperature-independent datasets and those relating to the networks where temperature is used as a feature.

S4.1 No Temperature Input

The results of the Bayesian optimization (the first step in the hyperparameter optimization) for the temperature independent datasets are reported in Figure S12.

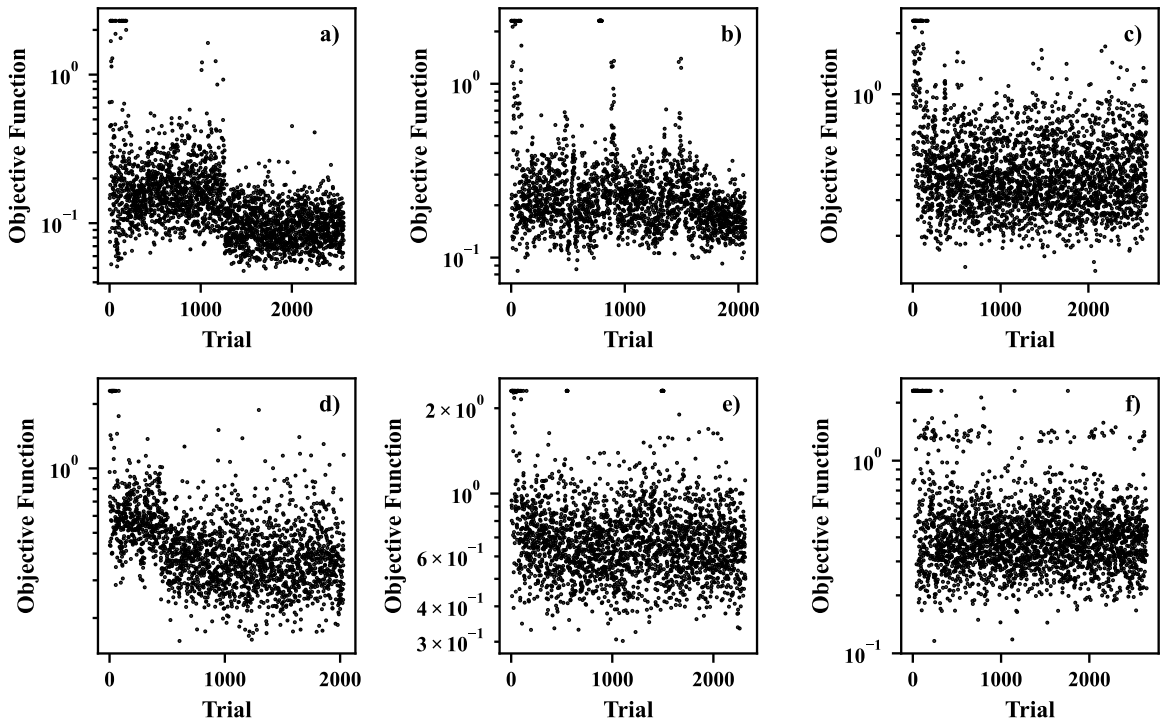

Figure S12. *Convergence plots of the Bayesian optimization (Metric vs Trial) performed to optimize the hyperparameters of the convolutional neural networks developed for the temperature-independent datasets studied in this work, namely molar mass (a), normal boiling temperature (b), vapor pressure at 25 ºC (c), density at 20 ºC (d), refractive index at 20 ºC (e), and aqueous solubility at 25 ºC (f).*

As explained in section S3.3, the best result of the Bayesian optimization for each dataset was used as the initial configuration seed for the Local Search algorithm. The results of this hyperparameter tuning step are reported in Figure S13.

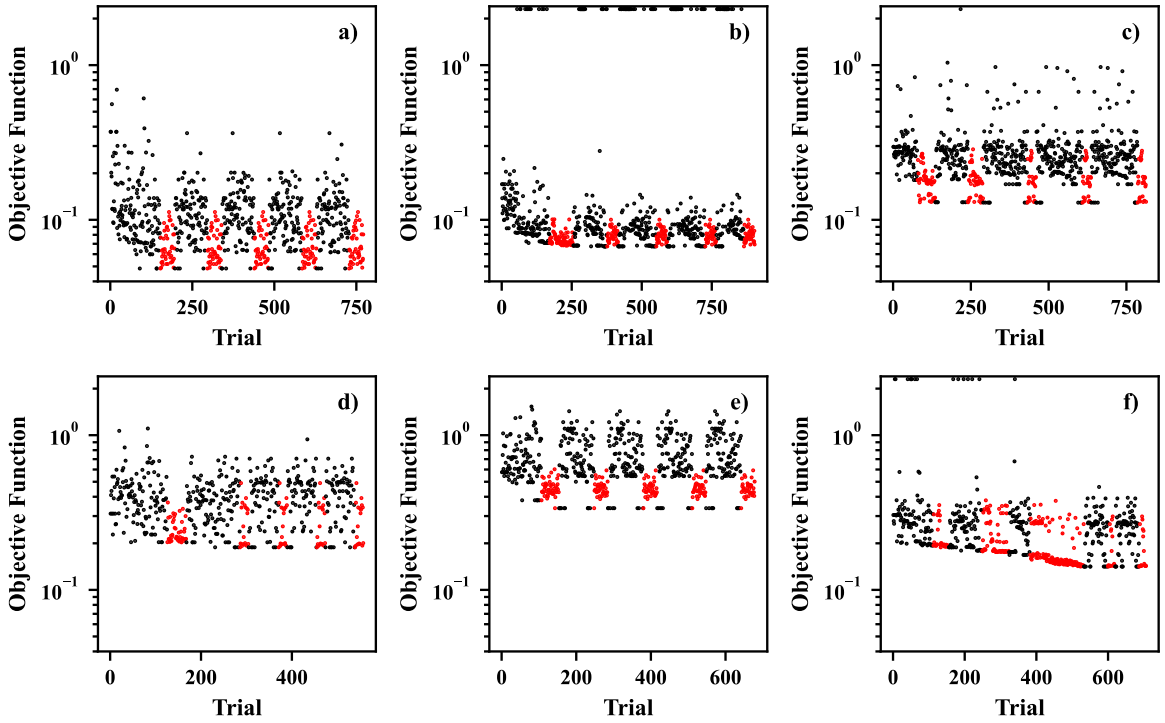

Figure S13. *Convergence plots of the Local Search Algorithm (Metric vs Trial) performed to optimize the hyperparameters of the convolutional neural networks developed for the temperature-independent datasets studied in this work, namely molar mass (a), normal boiling temperature (b), vapor pressure at 25 ºC (c), density at 20 ºC (d), refractive index at 20 ºC (e), and aqueous solubility at 25 ºC (f). The Architecture Local Search and Fitting Local Search steps are highlighted in black and red, respectively.*

The best hyperparameter set, obtained using the aforementioned procedures, are reported in Table S4 for each CNN developed. The performances of each CNN are depicted in Figure 2 of the main text and Figure S14. Note that the performance of the CNN for density in the testing set is particularly biased due to an outlier. If this outlier is removed, the coefficient of determination changes from 0.68 to 0.79 and the mean absolute error changes from 0.09 g/mL to 0.07.

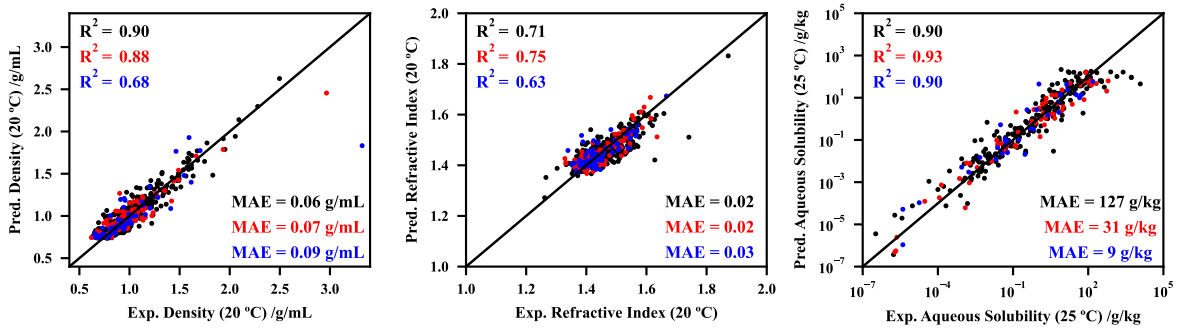

Figure S14. *Performance (predicted property vs experimental property) of the convolutional neural networks developed in this work for density at 20 ºC (left), refractive index at 20 ºC (middle), and aqueous solubility at 25 ºC (right). Black, red, and blue colors represent the results for the training, validation, and testing sets, respectively. The coefficient of determination (R^2) and mean absolute error (MAE) are also included.*

Table S4. *List of the final optimized hyperparameters, segmented into architecture and fitting hyperparameters, of each convolutional neural network developed in this work for the temperature-independent datasets studied in this work, namely molar mass (a), normal boiling temperature (b), vapor pressure at 25 ºC (c), density at 20 ºC (d), refractive index at 20 ºC (e), and aqueous solubility at 25 ºC (f).*

Hyperparameter	a)	b)	c)	d)	e)	f)
Number of Trainable Parameters	175	359	87	152	145	63
Architecture Hyperparameters						
Activation Function	Swish	Swish	Swish	Swish	Swish	Swish
Conv. Layer 1: Number of Filters	7	7	2	10	1	1
Conv. Layer 1: Kernel Size	8	7	8	4	3	10
Conv. Layer 1: Number of Strides	4	5	2	3	4	5
Pooling Layer 1: Type	*avg*	*avg*	*max*	*avg*	*max*	*avg*
Pooling Layer 1: Pool Size	2	2	5	8	1	8
Conv. Layer 2: Number of Filters	2	0[b]	4	4	4	3
Conv. Layer 2: Kernel Size	8	——[b]	7	10	5	8
Conv. Layer 2: Number of Strides	1	——[b]	4	6	1	4
Pooling Layer 2: Type	*avg*	——[b]	*avg*	*avg*	*avg*	*avg*
Pooling Layer 2: Pool Size	8	——[b]	10	9	10	5
Dense Layer 1: Number of Nodes	3	6	4	3	6	2
Dense Layer 2: Number of Nodes	0[a]	4	0[a]	1	4	4
Fitting Hyperparameters						
Adam Optimizer: Learning Rate	0.001	0.00144	0.0008	0.0012	0.00096	0.00077
Adam Optimizer: β_1	0.99	0.99	0.99	0.99	0.99	0.9
Adam Optimizer: β_2	0.999	0.999	0.9	0.999	0.999	0.9
L2 Regularization: λ	0.001	0.00096	0.001	0.001	0.001	0.00020
Batch Size	16	16	8	8	16	8

a) 2nd dense layer is not used.
b) 2nd convolution layer is not used.

S4.2 Temperature as an Additional Feature

The results of the Bayesian optimization (the first step in the hyperparameter optimization) for the convolutional neural networks that use temperature as an additional input are reported in Figure S15.

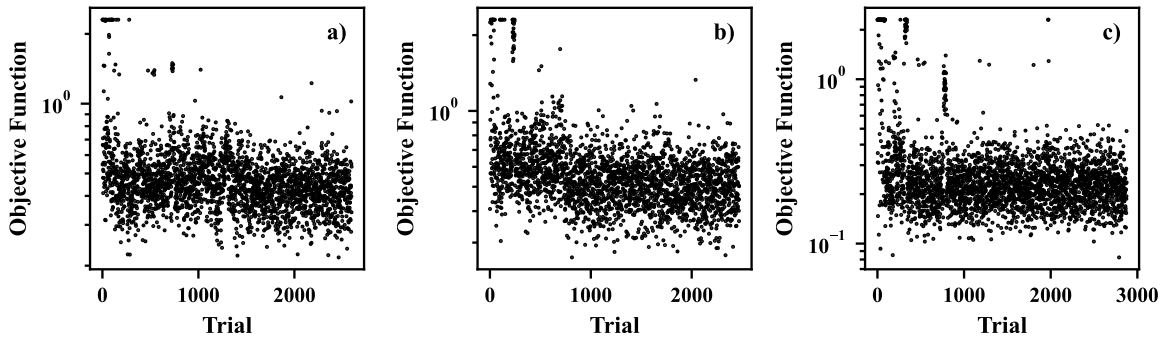

Figure S15. *Convergence plots of the Bayesian optimization (Metric vs Trial) performed to optimize the hyperparameters of the convolutional neural networks developed for the temperature-dependent datasets studied in this work, namely density (a), refractive index (b), and aqueous solubility (c).*

Akin to the procedure carried out in the previous section, the best result of the Bayesian optimization for each dataset was used as the initial configuration seed for the Local Search algorithm. The results of this hyperparameter tuning step are reported in Figure S16.

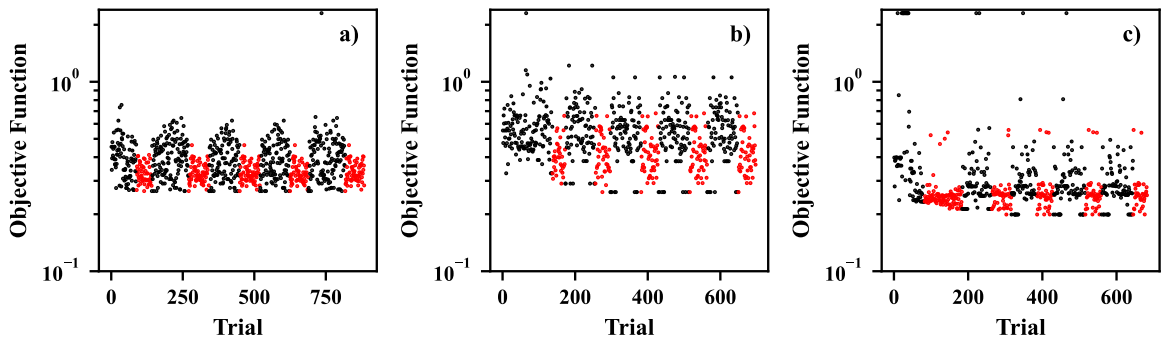

Figure S16. *Convergence plots of the Local Search Algorithm (Metric vs Trial) performed to optimize the hyperparameters of the convolutional neural networks developed for the temperature-dependent datasets studied in this work, namely density (a), refractive index (b), and aqueous solubility (c). The Architecture Local Search and Fitting Local Search steps are highlighted in black and red, respectively.*

The best hyperparameter set, obtained using the aforementioned procedures, are reported in Table S5 for each CNN developed.

Table S5. *List of the final optimized hyperparameters, segmented into architecture and fitting hyperparameters, of each convolutional neural network developed in this work for the temperature-dependent datasets studied in this work, namely density (a), refractive index (b), and aqueous solubility (c).*

Hyperparameter	a)	b)	c)
Number of Trainable Parameters	273	211	96
Architecture Hyperparameters			
Activation Function	Swish	Swish	Swish
Conv. Layer 1: Number of Filters	10	2	3
Conv. Layer 1: Kernel Size	9	4	4
Conv. Layer 1: Number of Strides	4	3	3
Pooling Layer 1: Type	*avg*	*avg*	*avg*
Pooling Layer 1: Pool Size	9	9	10
Conv. Layer 2: Number of Filters	4	10	0[a)]
Conv. Layer 2: Kernel Size	6	8	——[a)]
Conv. Layer 1: Number of Strides	4	7	——[a)]
Pooling Layer 2: Type	*avg*	*avg*	——[a)]
Pooling Layer 2: Pool Size	2	7	——[a)]
Dense Layer 1: Number of Nodes	5	7	4
Dense Layer 2: Number of Nodes	10	6	0[b)]
Fitting Hyperparameters			
Adam Optimizer: Learning Rate	0.0008	0.0012	0.000922
Adam Optimizer: β_1	0.99	0.9	0.99
Adam Optimizer: β_2	0.999	0.999	0.99999
L2 Regularization: λ	0.001	0.0012	0.002074
Batch Size	32	16	32

a) 2nd convolution layer is not used.
b) 2nd dense layer is not used.

S5. References

1 E. Mullins, R. Oldland, Y. A. Liu, S. Wang, S. I. Sandler, C.-C. Chen, M. Zwolak and K. C. Seavey, *Ind. Eng. Chem. Res.*, 2006, **45**, 4389–4415.

2 A. Klamt, *COSMO-RS: from quantum chemistry to fluid phase thermodynamics and drug design*, Elsevier, 2005.

3 A. Klamt, *WIREs Comput. Mol. Sci.*, 2011, **1**, 699–709.

4 J. Palomar, J. S. Torrecilla, J. Lemus, V. R. Ferro and F. Rodríguez, *Phys. Chem. Chem. Phys.*, 2010, **12**, 1991.

5 B. Sanchez-Lengeling and A. Aspuru-Guzik, *Science (80-.).*, 2018, **361**, 360–365.

6 M. Swain, CIRpy 1.0.2, https://github.com/mcs07/CIRpy.

7 J. R. Rumble, in *CRC Handbook of Chemistry and Physics*, CRC Press/Taylor & Francis, Boca Raton, United States, 102nd Edit., 2021.

8 M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org.

9 S. Albawi, T. A. Mohammed and S. Al-Zawi, in *2017 International Conference on Engineering and Technology (ICET)*, IEEE, 2017, pp. 1–6.

10 P. Ramachandran, B. Zoph and Q. V. Le, *arXiv Prepr., 2017, arXiv:1710.05941.*

11 D. P. Kingma and J. Ba, *arXiv Prepr., 2014, arXiv:1412.6980*.

12 K. He, X. Zhang, S. Ren and J. Sun, *arXiv Prepr., 2015, arXiv:1502.01852*.

13 L. Hertel, J. Collado, P. Sadowski, J. Ott and P. Baldi, *SoftwareX*, 2020, **12**, 100591.