Molecule graph reconstruction from atomic force microscope images with machine learning

Niko Oinonen, Lauri Kurki, Alexander Ilin, and Adam S. Foster*

Despite the success of noncontact atomic force microscopy (AFM) in providing atomic-scale insight into the structure and properties of matter on surfaces, the wider applicability of the technique faces challenges in the difficulty of interpreting the measurement data. We tackle this problem by proposing a machine learning model for extracting molecule graphs of samples from AFM images. The predicted graphs contain not only atoms and their bond connections but also their coordinates within the image and elemental identification. The model is shown to be effective on simulated AFM images, but we also highlight some issues with robustness that need to be addressed before generalization to real AFM images.

Impact statement

Developing better techniques for imaging matter at the atomic scale is important for advancing our fundamental understanding of physics and chemistry as well as providing better tools for materials R&D of nanotechnologies. State-of-the-art high-resolution atomic force microscopy experiments are providing such atomic-resolution imaging for many systems of interest. However, greater automation of processing the measurement data is required in order to eliminate the need for subjective evaluation by human operators, which is unreliable and requires specialized expertise. The ability to convert microscope images into graphs would provide an easily understandable and precise view into the structure of the system under study. Furthermore, a graph consisting of a discrete set of objects, rather than an image that describes a continuous domain, is much more amenable to further processing and analysis using symbolic reasoning based on physically motivated rules. This type of image-to-graph conversion is also relevant to other machine learning tasks such as scene understanding.

Introduction

Scanning probe microscopy (SPM) techniques have become an essential tool of surface science and materials research for their ability to visualize, characterize, and manipulate matter at the atomic scale.[1–3] The two main SPM techniques, scanning tunneling microscopy[4] and atomic force microscopy (AFM),[5,6] can be used to measure the tunneling current and forces between the sample and an atomically sharp probe tip, respectively. The added functionalization of the tip apex by a CO molecule enables reliable atomic-resolution imaging.[1] However, despite the successes, wider adoption of the techniques has faced challenges. The difficulties in preparing and operating the devices as well as interpreting the resulting images both require high levels of specialized expertise. The problem of preparing suitable tips for imaging has recently been greatly alleviated by developments in using machine learning in SPM,[7,8] and in particular, using neural networks for automating the conditioning of the metallic tip[9] as well as functionalizing the tip,[10] both essential for reaching atomic resolution.

The focus here is on the image interpretation problem, specifically for noncontact AFM images. For planar systems, the observed contrast in AFM images can often be mapped fairly straightforwardly to the atomic structure of the molecule under the tip.[1] However, for more three-dimensional (3D) structures, the interpretation can become an extremely difficult task even for experts in the field.[11,12] Furthermore, identifying the chemical elements of the atoms is a similarly difficult task, even for planar structures, often requiring additional imaging modes.[13,14]

We previously explored atomic structure recovery from AFM images by using a convolutional neural network (CNN) to translate a set of constant-height AFM images into a two-dimensional descriptor image of the atomic structure.[15] Matching

Niko Oinonen, Department of Applied Physics, Aalto University, Aalto, Helsinki, Finland
Lauri Kurki, Department of Applied Physics, Aalto University, Aalto, Helsinki, Finland
Alexander Ilin, Department of Computer Science, Aalto University, Aalto, Finland
Adam S. Foster, Department of Applied Physics, Aalto University, Aalto, Helsinki, Finland; WPI Nano Life Science Institute (WPI-NanoLSI), Kanazawa University, Kanazawa, Japan; adam.foster@aalto.fi
*Corresponding author

the predicted descriptor images to an existing database of such images allows finding likely candidates for the structure and orientation of the molecule under study. However, despite their usefulness in this task, the image descriptors are not the optimal description of the molecule structure in the sense that the image pixels span a high-dimensional space, whereas the essential information of the molecule structure can be contained in a much lower dimensional space of just the atomic coordinates and their chemical elements. In practice, this can lead to a lack of precision in what the prediction is actually saying about the structure of the molecule. Here we are exploring an alternative method for structure discovery from AFM images that provides a very precise description of the imaged system as a graph where each atom is described by its coordinates and group of chemical elements.
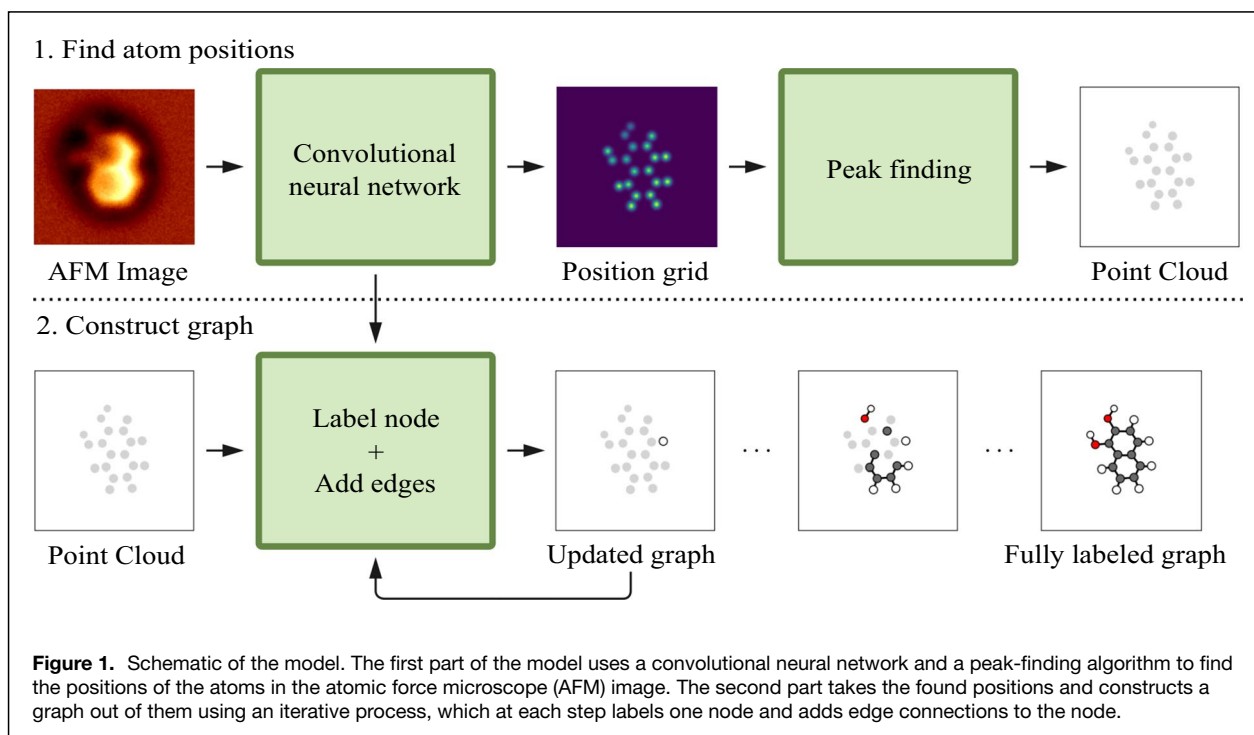
Our work builds on a growing body of literature on graph neural networks (GNNs).[16,17] A GNN operates on graph-structured data, taking into account the relations between the objects in the graphs, and learns a new feature representation of the graphs, much in the same way that a CNN can learn a vector representation of image data. These learned features can be used, for example, for classifying nodes in graphs,[18] and learning similarity metrics between graphs.[19] Molecules are also naturally represented as graphs with atoms as the nodes and covalent bonds as the edges, and indeed, this inherent graph structure in atomic-scale systems has been used to great effect in accurately predicting various chemical properties of molecules without the need of expensive quantum mechanical calculations.[20] In addition to understanding graphs, there is a great interest in generating graphs that fit into a desired distribution or fulfill some condition.[21,22] Examples include generating candidate molecules for synthesis with drug-like properties,[23] and scene-understanding tasks where a graph of relationships between objects is generated conditional to an image of a scene.[24,25] Similar to the last example, we propose here a method for generating a molecule graph conditional to a set of AFM images.

Results

Our method is divided into two parts (see **Figure 1**). First, a CNN[26,27] translates a set of 10 constant-height AFM images obtained at different tip-sample distances into a 3D grid where the positions of atoms are presented as bright peaks (position grid). The positions of these peaks are identified with a peak-finding algorithm in order to obtain a simple list of coordinates for the atoms. Second, the found atoms are connected into a labeled graph in an iterative process, where at each step one of the atoms is labeled based on its chemical element, and its bond connections to other atoms in the graph are identified. This second part is done with a combination of a GNN and multilayer perceptrons. A more detailed description of the model is presented in the section "Materials and methods."

Our training database contains molecules with the elements H, C, N, O, F, Si, P, S, Cl, and Br. For the classification labels, we test a division of the elements into five classes based on the groups in the periodic table: 1: (H), 2: (C, Si), 3: (N, P), 4: (O, S), 5: (F, Cl, Br). This division is used in order to better balance the distribution of the classes compared to having a separate class for each element. We also have the intuition that this division will lead to better learning in the model due to the connection between the groups of the periodic table and the

Figure 1. Schematic of the model. The first part of the model uses a convolutional neural network and a peak-finding algorithm to find the positions of the atoms in the atomic force microscope (AFM) image. The second part takes the found positions and constructs a graph out of them using an iterative process, which at each step labels one node and adds edge connections to the node.

typical number of bonds between atoms, although we do not test this hypothesis here. Additionally, we test three different orderings for the graph construction part of the model: random order, order based on decreasing y-coordinate, and order based on decreasing z-coordinate (increasing depth).

We first present predictions on three example test systems to highlight some of the strengths and weaknesses of the model (**Figure 2**), and then present quantitative metrics on a large test set of samples. The example predictions are presented for the random-order graph construction. All data shown here are based on simulated AFM images.[28]

The first test system is 1-bromo-3,5-dichlorobenzene (Figure 2a), a benzene derivative functionalized by two chlorines and one bromine. This represents a typical case of a small organic molecule, which our training database mostly consists of. The model prediction here is in excellent agreement with the reference (ground truth). The predicted position grid contains all of the atoms as clearly separated peaks, which are all correctly detected by the peak-finding algorithm. In the constructed graph, all of the bond connections are correctly identified and all of the atoms are classified into correct groups.

The second test system is a cluster of seven water molecules relaxed on a Cu(111) surface (Figure 2b). This system represents a more 3D case, where all of the atoms in the system are not clearly seen in the AFM images, making them much harder to interpret. Note also that the reference graph here does not contain all of the atoms in the complete system, because the deepest atoms are blocked by the top atoms, and are therefore not represented in the AFM image in any way. In order to avoid unnecessary noise in the training, all atoms more than 0.8 Å below the top atom are cut off from the reference graph, even though they are present in the AFM simulation. The prediction here correctly finds all of the oxygen atoms as well as two of the four hydrogen atoms present in the reference. The positions of these atoms are not as exact as in the previous example, but are still very close to correct. However, two of the hydrogen atoms located between the top oxygen atoms are missing in the prediction. These atoms, though they are within the cutoff distance, are very difficult to detect due to being very close to the relatively larger oxygen atoms and lower than the hydrogen sticking out in the top water molecule. Additionally, the predicted graph contains three hydrogen atoms not present in the reference graph. A comparison to the complete water cluster reveals that these atoms are at the locations of the three water molecules that were removed from the reference due to below the cutoff distance. The model therefore is not completely incorrect in predicting these atoms, even though in reality they are deeper than the prediction suggests.

Our final example is perylenetetracarboxylic dianhydride (PTCDA, Figure 2c), a commonly used benchmark system in scanning probe microscopy experiments. This molecule presents an example at the upper extreme of graph sizes in our molecule database. The prediction of the positions of the atoms is almost perfect, with only a single extra atom at the
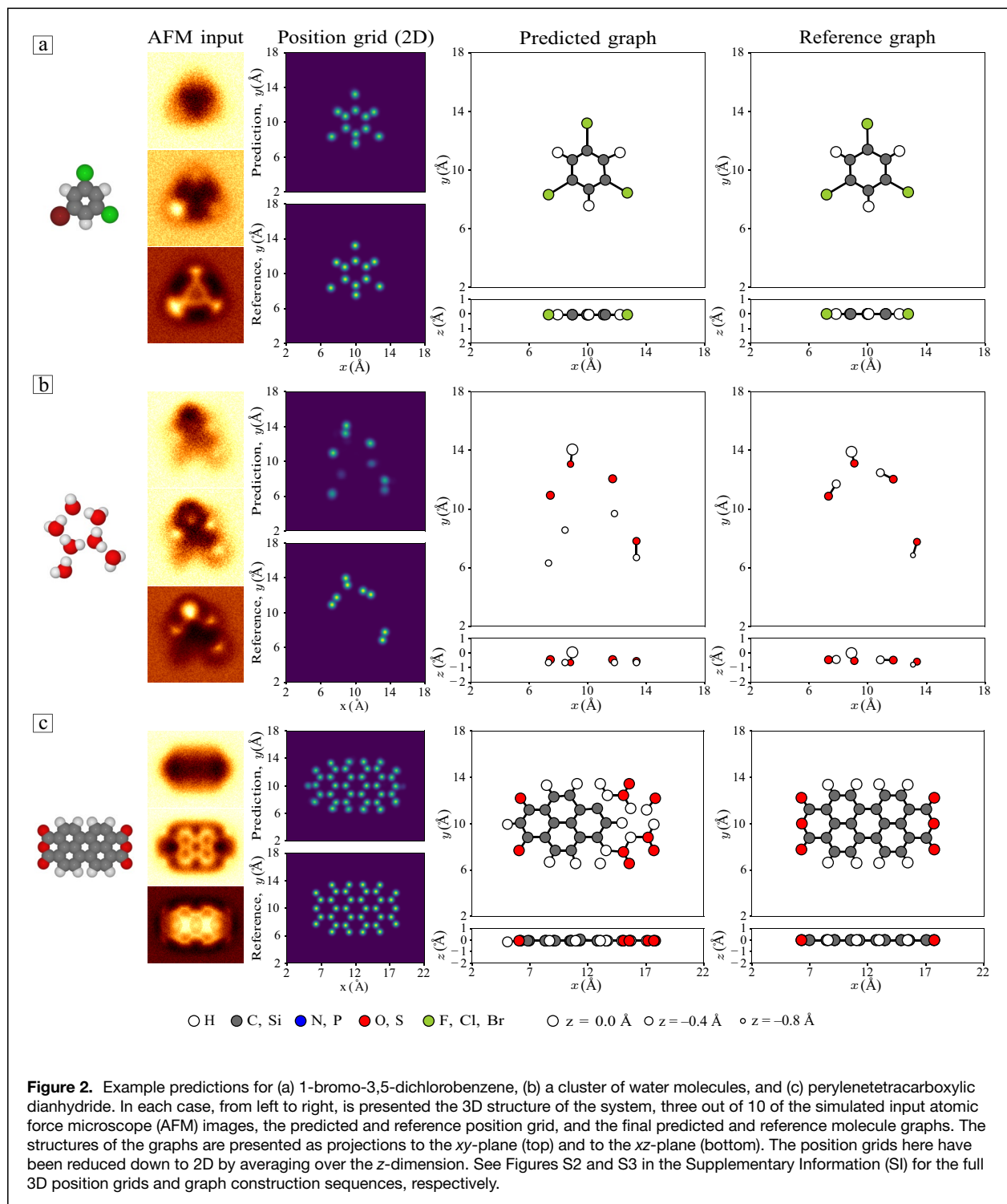
left end of the molecule. The predicted position grid also shows a peak symmetrically at the opposing end of the molecule, but it was too weak to be detected by the peak-finding algorithm. The classes and bond connections of the atoms are nearly perfect in the left half of the molecule apart from the discrepancy around the extra atom. However, the graph on the right half has many mistakes despite the symmetry of the molecule and the AFM image. An explanation for this discrepancy is found in the different coordinate system of this example compared to the other ones. Here the x-axis is wider, going from 2 to 22 Å, whereas in the other examples it is from 2 to 18 Å. The model was trained exclusively on the latter type of samples. If the predictions are done with a shift of -2 Å in the x-axis so that the frame in the PTCDA predictions is centered on the same (10 Å, 10 Å)-coordinate as in the training samples (Figure S4 in the SI), the predicted graph for PTCDA becomes much more symmetric, whereas the predictions for the other two systems are altered in a worse direction.

Additional example predictions on randomly chosen samples from the test set are shown in Figure S5 in the SI.

For more quantitative tests of the accuracy of the predictions, we perform predictions on a test set of 35,554 samples. For each prediction, we first find a mapping between the predicted atom positions and reference atom positions using a threshold distance of 0.35 Å. Atoms within this threshold distance are mapped one-to-one and form matching subgraphs within the prediction and reference. The atom classes and bond connections in the subgraphs can then be compared in confusion matrices and with precision and recall rates. Atoms that fall outside of the threshold distance are counted separately as missing or extra atoms in the prediction. The chosen threshold distance is roughly half of the H–H bonding distance, the smallest possible distance between a pair of atoms in a molecule, so that single atom in the prediction is always uniquely matched to at most one atom in the reference. For more details about the test metrics, see the section "Model training and testing" in the SI.
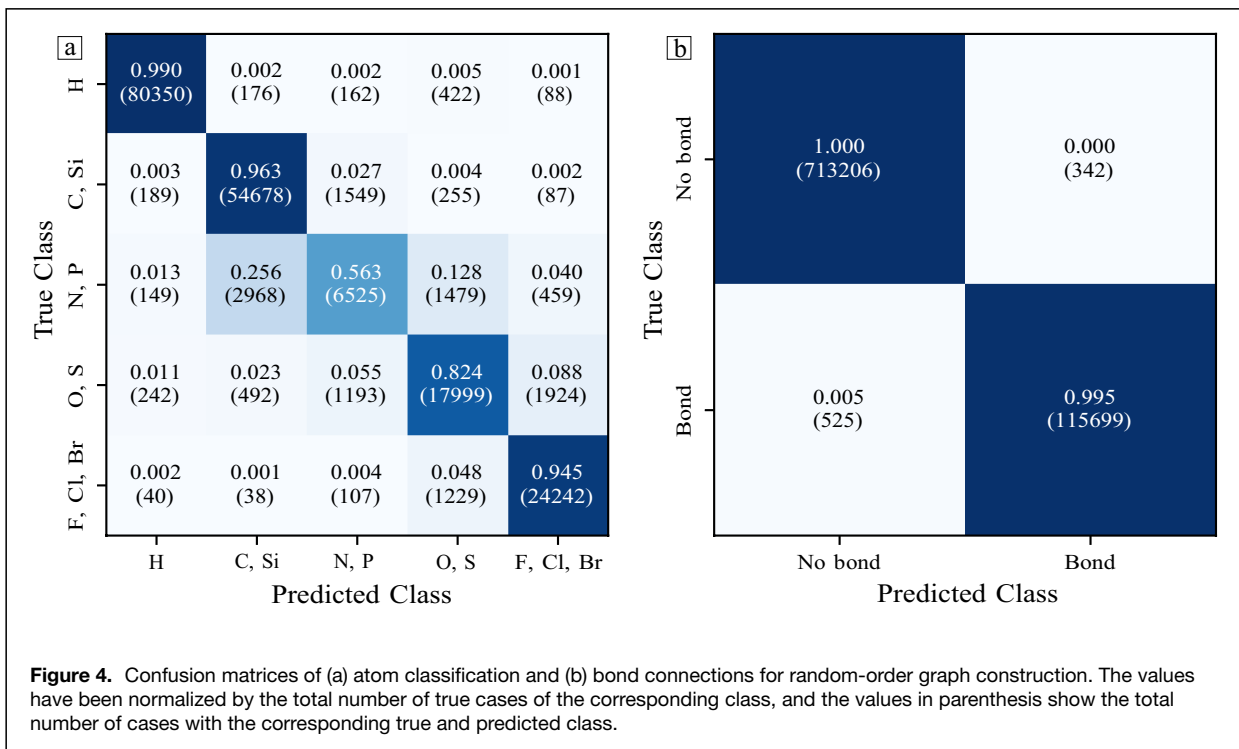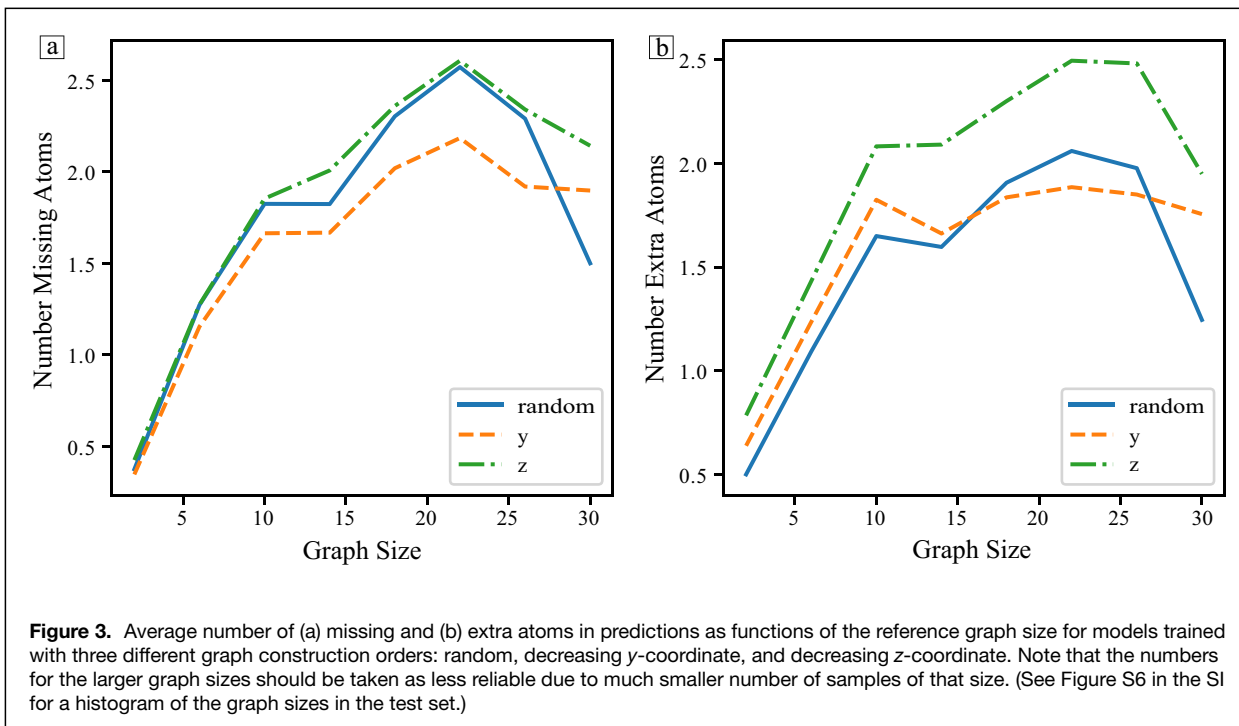
We first turn to the missing and extra atoms in the predictions as a test of the quality of the atom position detection. **Figure 3** shows the number of missing and extra atoms for different graph sizes. For small graph sizes of at most 10 atoms, the numbers increase roughly linearly with ~20% errors relative to the graph size. For larger graphs the curves start to flatten to roughly two missing/extra atoms regardless of graph size. A comparison of the different graph construction orders shows that the y-order results in less missing atoms for large graph sizes, but the random order has less extra atoms for small graph sizes, and the z-order performs the worst in all cases.

Next we look at the accuracy of the graph construction process by considering the precision and recall rates of both the atom type classification (**Table I**) and bond connections (**Table II**). The H-group is clearly the most accurately predicted class with very high precision and recall rates for all construction orders. The

Figure 2. Example predictions for (a) 1-bromo-3,5-dichlorobenzene, (b) a cluster of water molecules, and (c) perylenetetracarboxylic dianhydride. In each case, from left to right, is presented the 3D structure of the system, three out of 10 of the simulated input atomic force microscope (AFM) images, the predicted and reference position grid, and the final predicted and reference molecule graphs. The structures of the graphs are presented as projections to the xy-plane (top) and to the xz-plane (bottom). The position grids here have been reduced down to 2D by averaging over the z-dimension. See Figures S2 and S3 in the Supplementary Information (SI) for the full 3D position grids and graph construction sequences, respectively.

C-group, O-group, and halogen group atoms are also classified with generally high accuracy. Only the N-group has significantly lower precision and recall. A look at the confusion matrix for the random-order model (**Figure 4**a) reveals that most of the mistakes are made between the N-group and the C- and O-groups. The results generally do not differ greatly between the different graph construction orders. The biggest differences are in lower

precision in the halogen group for the z-order, lower recall in the O-group for z-order, and lower recall for both y- and z-orders in the N-group. The precision and recall rates for the bond connection classification are extremely high for every construction order. Only the recall for z-order in the positive bond class is slightly lower. The confusion matrices for the y- and z-order models (Figures S7 and S8 in the SI) show that these models make more

Figure 3. Average number of (a) missing and (b) extra atoms in predictions as functions of the reference graph size for models trained with three different graph construction orders: random, decreasing *y*-coordinate, and decreasing *z*-coordinate. Note that the numbers for the larger graph sizes should be taken as less reliable due to much smaller number of samples of that size. (See Figure S6 in the SI for a histogram of the graph sizes in the test set.)

Figure 4. Confusion matrices of (a) atom classification and (b) bond connections for random-order graph construction. The values have been normalized by the total number of true cases of the corresponding class, and the values in parenthesis show the total number of cases with the corresponding true and predicted class.

mistakes predicting N-group as C-group, but otherwise show a very similar pattern as the random-order model.

We also test combining predictions from 20 different random construction orders by taking the average of the predicted class weights after the graph is fully constructed. The precision and recall rates in this case differ at most by one percentage point from the case of only a single random order per sample.

Table I. Precision and recall for atom type classification for models trained with three different graph construction orders: random, decreasing y-coordinate, and decreasing z-coordinate.

Class	Precision			Recall		
	Random	y	z	Random	y	z
1 (H)	0.992	0.987	0.960	0.990	0.989	0.984
2 (C, Si)	0.937	0.917	0.903	0.963	0.976	0.963
3 (N, P)	0.684	0.733	0.690	0.563	0.451	0.419
4 (O, S)	0.842	0.825	0.873	0.824	0.819	0.674
5 (F, Cl, Br)	0.905	0.907	0.844	0.945	0.934	0.962

Table II. Precision and recall for bond connection classification for models trained with three different graph construction orders: random, decreasing y-coordinate, and decreasing z-coordinate.

Class	Precision			Recall		
	Random	y	z	Random	y	z
Bond	0.997	0.994	0.997	0.996	0.996	0.968
No bond	0.999	0.999	0.995	0.999	0.999	0.999

Discussion and conclusion

There is no clear winner in the overall comparison between the models with different graph construction orders. Whereas the z-order model has overall the worst performance, the random-order and y-order models perform overall close to the same level by our metrics. However, choosing a fixed construction order may be desirable for consistency in the predictions. In our tests we found that the random-order model in some cases can give different answers for the atom classes for the same sample if the prediction is made several times with different random orders.

In our example predictions we saw that the detection of atomic positions works quite reliably even for relatively large molecules such as PTCDA. However, for more 3D systems such as the water cluster, there were still inconsistencies. This was also seen in the relatively high error rate of ~20% for small graph sizes, which mostly consist of these 3D systems where only a few atoms are sticking out. Likely many of the bad predictions can be attributed to inconsistent choice of which atoms to include in the reference graphs. A more sophisticated decision process for the cutoff would take into account not only the depths of the atoms, but also whether they are occluded by other atoms in the vicinity. Additionally, we found that the position detection accuracy was dependent on the graph construction order. In principle the position prediction task is independent from the graph construction, but the model shares a common encoder for the AFM image between the position prediction and the graph construction parts of the model, so that the joint training of the two tasks makes them interdependent in practice. Therefore, it would be useful to find a way of training the two parts of the model in such a way that they do not interfere with one another.

Another difficulty that we faced was the sensitivity of the predictions to the choice of the coordinate system for the atom positions. This problem was observed for the PTCDA example where the coordinate frame was larger than in the training samples. Although the model is constructed in a way that allows for variable sizes of the input AFM images, the current model has not been trained in a way that takes this into account. This problem could be fixed in future possibly by training with samples of variable size and randomizing the origin of the coordinate system or using relative spatial encodings of the atom coordinates in the GNN.[29] Studies related to machine learning material properties and interatomic potentials have also used local geometry descriptors such as the *smooth overlap of atomic positions* to encode the atom coordinates.[30,31] Although GNNs have shown similar or better performance for large data sets,[32,33] these kind of physically motivated encodings of the atomic positions could be used as an additional channel of information inside the model.

We point out that the graph-structured output of the model makes it amenable to further processing based on symbolic logic. One could for example enforce the octet rule on the bond connections or inspect the distances between the atoms to refine the elemental identification based on tables of bond lengths between pairs of elements. Such physically motivated rules could also be incorporated into the loss function in order to encourage the model to respect those rules independent of the specific sample.

The current model was tested only on simulated AFM images, but naturally, the real test will be on experimental AFM images. We recently discussed the importance of an accurate simulation model in generalizing from simulated data to experimental data in the context of extracting the electrostatic contribution from AFM images.[34] Likely, similar issues will come into play here as well, because we use the same point-charge electrostatics for the simulations. Using the more complete Hartree potential could be especially important

for the accurate atom class predictions where we saw significantly lowered accuracy for one of the classes even with the simulated test data. Further difficulties with experimental data include tip-induced relaxation of the sample structure and changes in tip condition during scanning or other sources of artifacts not present in the simulations. The robustness of the model against such artifacts can be improved by adding random noise and other randomized pre-processing steps to the simulated training data.[15,34] In spite of the challenges, our current results demonstrate a promising start for extracting precise atomic and chemical structures of molecules from AFM images. This is further supported by continuing model developments tackling similar challenges.[35–37]

Materials and methods

A detailed rundown of all the parts of the model are presented in the following. See Figure S9 in the SI for a visual presentation of the information flow inside the model.

Position prediction

The task of the first part of the model is to find the positions of the atoms present in the AFM image as a point cloud without a graph structure. The position finding is done in two parts. First, the AFM image is transformed by a CNN into a grid of values that represent the atom positions as Gaussian peaks. Second, the grid is transformed in to a cloud of points by finding the positions of the Gaussian peaks with a more traditional algorithmic approach. The two parts of this process are described in more detail in the following.

Position grid

The input to the CNN is a set of constant-height AFM images separated in space by the distance to the sample. Therefore, the set of images can be seen as a 3D volume of data, but in the following we will refer to the whole volume of data as simply the *AFM image*. The prediction target for the CNN is a 3D grid where the atom positions are represented as normal distributions that are centered on the atomic nuclei. The position grid is constructed such that there is a one-to-one correspondence in the *xy* positions of the voxels of the input AFM image and the output position grid. The *z*-dimension of the position grid is chosen such that there is enough space to capture all the atoms and enough voxels for a good spatial resolution.

For n_a atoms with real-space positions $p_q = (x_q, y_q, z_q)$, the values of the voxels in the position grid, the learning target for the model, are given by

$$v_{ijk} = \sum_{q=1}^{n_a} \mathcal{N}((i \cdot d_x, j \cdot d_y, k \cdot d_z)|p_q, \sigma^2 I) \qquad 1$$

$$\forall (i,j,k) \in \{1, \ldots, v_x\} \times \{1, \ldots, v_y\} \times \{1, \ldots, v_z\}.$$

Here the (i, j, k) coordinates form a 3D grid of voxels with spacing of d_x, d_y, d_z in the x, y, and z dimensions, respectively, and v_x, v_y, v_z are the number of voxels in the respective

dimensions. $\mathcal{N}((x,y,z)|\mu, \Sigma)$ denotes the value at point (x, y, z) of the probability density function of the 3D multivariate normal distribution with mean μ and covariance matrix Σ. For the standard deviation of the peaks, we choose $\sigma = 0.25\,\text{Å}$, so that the peaks are clearly separated even for those atoms that are very close to each other. The atoms are positioned such that their average *xy*-position is at the center of the grid (and the AFM image) and the *z*-positions are chosen such that center of the top atom is 0.5 Å below the top of the grid.

The spatial resolution of the position grid in the *xy*-dimension is fixed by the parameters of the AFM simulations used for the training data, where we choose $d_x = d_y = 0.125\,\text{Å}$. The CNN is constructed such that the total size of the AFM image in *xy*-dimension can be variable, but in practice during training we choose to use a fixed size $v_x = v_y = 128$. In the *z*-dimension we are free to choose any spatial resolution and number of voxels independent of the training data, but they have to be chosen once and then fixed for a given model. Given the high sensitivity of the AFM signal in the *z*-direction, we choose a slightly higher resolution $d_z = 0.1\,\text{Å}$, and we suppose that atoms more than roughly 1 Å below the top atom cannot be detected, so we choose a total size of $v_z = 20$ voxels, which gives a similar amount of headroom for the atoms on both sides in the *z*-dimension.

The type of CNN we use for the position grid prediction is a U-net with attention gates in the skip connections.[26,27] We used a very similar model recently for electrostatic field prediction from AFM images.[34] The main difference here is that due to the 3D nature of both the input and the output of the network, we use here only 3D convolutions, as opposed to switching to 2D convolutions in the middle of the network. The structure of the CNN is shown schematically in the top half of Figure S9 in the SI.

The first part of the CNN encodes the AFM image in four stages into a lower resolution feature space with more channels. Each stage consists of one 3D convolution block with two layers and there is an average pooling operation between each convolution block. The layers have an increasing number of channels, 4, 8, 16, and 32, in each stage, respectively. The second part of the CNN decodes the low-resolution feature maps into the predicted position grid. The decoding also happens in multiple stages that mirror the first three stages of the encoder part. Each stage starts with an upscaling by trilinear interpolation and is followed by a block of four 3D convolutions, which have 32, 16, and 8 channels in the three stages, respectively. Each decoder stage also has an additional channel of information coming through skip connections from the corresponding encoder stage. The skip connection uses an attention-gate mechanism that was detailed in our previous work[34] and attaches as additional channels to the third layer of the corresponding decoder block after an upscale to the correct size.

Peak finding

After the prediction of the position grid from the CNN, we turn the grid into a cloud of points representing the positions of the atoms by finding the positions of the peaks in the predicted grid. This is a post-processing step to the output of the CNN and does not require gradient propagation. Therefore, this step of the process can be tuned or even completely replaced by another algorithm even after the model has been trained. We propose here an algorithm based on template matching because we know the exact shape of the peaks that we are looking for.

A template matching algorithm compares an image or other array of data to a smaller image patch, a template, and looks for regions where the template best matches with the image. In our case, we want to compare the predicted position grid of several Gaussian peaks with a smaller template grid of a single Gaussian peak. The template grid t is constructed using the same parameters as the reference position grid v, such that the peak is centered to the middle of the grid and voxels more than 3σ away from the center along any coordinate axis are cut off. We denote the size of t in the x, y, and z dimensions with t_x, t_y, and t_z, respectively. The sizes are chosen to be odd so that there is a center voxel, and we denote halves of the sizes with $d_x = (t_x - 1)/2$, $d_y = (t_y - 1)/2$, and $d_z = (t_z - 1)/2$, and for cleaner indexing below, we choose to index t symmetrically so that the center voxel has index $(0, 0, 0)$. Then for each voxel $(i, j, k) \in \{1, \ldots, v_x\} \times \{1, \ldots, v_y\} \times \{1, \ldots, v_z\}$ in the position grid, we compute a score for how well the template matches the region around the voxel,

$$S_{ijk} = \frac{\sum_{l=r_i}^{w_i} \sum_{m=r_j}^{w_j} \sum_{n=r_k}^{w_k} (v_{i+l,j+m,k+n} - t_{l,m,n})^2}{\sum_{l=r_i}^{w_i} \sum_{m=r_j}^{w_j} \sum_{n=r_k}^{w_k} t_{l,m,n}^2}. \qquad 2$$

The matching score here is a normalized mean-squared distance between the template and the corresponding patch in the position grid. The normalization makes the score equal exactly 1 when matched against a background of all zeros. The limits of the sums are chosen as

$$r_i = \max(-d_x, -i + 1), \quad w_i = \min(d_x, v_x - i),$$
$$r_j = \max(-d_y, -j + 1), \quad w_j = \min(d_y, v_y - j),$$
$$r_k = \max(-d_z, -k + 1), \quad w_k = \min(d_z, v_z - k),$$

so that the indices for both v and t remain within bounds. The matching is only partial on the edges of the grid, so that, for example, in a corner only 1/8 of the template is matched. Any distance or similarity metric could be chosen here in principle, but we found this particular metric to work well in our experiments.

Next, the map of matching scores is turned into a binary map B by choosing a threshold t_s and setting

$$B_{ijk} = \begin{cases} 1, & S_{ijk} < t_s \\ 0, & \text{otherwise} \end{cases} \qquad 3$$
$$\forall (i, j, k) \in \{1, \ldots, v_x\} \times \{1, \ldots, v_y\} \times \{1, \ldots, v_z\}.$$

The normalization of the matching score makes the choice of the threshold easier, and we found a value of $t_s = 0.7$ to work well in our tests. In the binary map, the peaks correspond to connected sets of voxels with value 1, which can be found by a connected-component labeling algorithm.[38,39] Within each connected component, we choose the position of the peak based on the index of the voxel with best matching score. For a given voxel with index (i, j, k), the corresponding coordinate is set to be

$$p = (i \cdot d_x, j \cdot d_y, k \cdot d_z). \qquad 4$$

This finally yields us the cloud of points P containing the coordinate positions of the atoms. We have made an implementation of the whole peak-finding procedure in CUDA with PyTorch bindings for fast and easy use in the model training.

Graph construction

The second part of the model is tasked with constructing a graph out of the point cloud found by the first part. This is done in an iterative fashion by a GNN, adding one node at a time to the graph. On each iteration the GNN gets as an input the current graph from the previous iteration and new node position, and outputs the class of the new node and the possible edge connections of the new node to all existing nodes in the current graph. Similar graph construction schemes have been used before for generating graphs matching a distribution of graphs in a training set.[21] In our case, we don't want to generate just any graph, but the one that matches the molecule present in the AFM image of interest. For this reason we use inside the GNN an additional channel of information coming from the internal part of the CNN that generated the point cloud.

On each iteration, one new node is selected according to some ordering criterion, and the process of the iteration step can be divided into roughly three parts:

1. Compute a vector encoding of the current graph using a GNN.
2. Compute a vector encoding of the AFM image using a CNN with attention gates utilizing the graph vector encoding and the new node position.
3. Use the vector encodings to predict the class and edge connections of the new node using multilayer perceptrons (MLPs), and add the completed node to the graph.

A schematic of the graph construction process is shown in the bottom half of Figure S9 in the SI, and the details of the whole procedure are presented in the following.

Graph encoding

We represent a molecule as a graph $G = (V, E)$, where $V = \{v_k\}_{k=1}^n$ is a set of n nodes that represent the atoms, and $E \subseteq \{\{v, u\} \mid (v, u) \in V^2, v \neq u\}$ is a set of undirected edges that represent the bond connections between the atoms. Each node $v_k = (p_k, c_k)$ holds two pieces of information, the real-space position $p_k = (x_k, y_k, z_k) \in P$ of the atom and the class c_k of the atom. For the class we also use its associated one-hot encoded vector

$$c'_{k,j} = \begin{cases} 1, & j = c_k \\ 0, & j \neq c_k \end{cases} \quad \forall j \in \{1, \ldots, n_c\}, \qquad 5$$

where n_c is the number of classes. The atoms are divided into classes by their chemical elements, such that each class can hold one or several elements in it.

For a given iteration i in the graph construction, the GNN is given as an input an incomplete graph $G_i = (V_i, E_i)$ and the position p_{i+1} of a new node to add to the graph. The first step in the process is to generate a fixed-size vector encoding h_{G_i} for G_i. On the first iteration ($i = 0$) when the graph is empty, the vector encoding is simply set to a vector of zeros, $h_{G_0} = 0$. On the following iterations the encoding is generated by a message-passing neural network (MPNN) and an aggregator network. An MPNN generates for each node in the graph a hidden vector that encodes the information of the neighborhood of the given node. This is done iteratively, by passing information between neighboring nodes in the graph for a given number of times.

As an initialization step, we generate the initial hidden vector $h_{v_k}^0 \in \mathbb{R}^{n_h}$ for each node $v_k \in V_i$ as

$$h_{v_k}^0 = f_i(p_k, c'_k), \qquad 6$$

where f_i is an MLP, and the size n_h of the vector is a hyperparameter. Here, where there are multiple input vectors to an MLP, the vectors are concatenated before being fed to the MLP. After the initialization, the message-passing scheme for a given iteration $t \in \{1, \ldots, n_t\}$ is performed as

$$m_{v_k}^t = \sum_{u \in \mathcal{N}(v_k)} f_m(h_{v_k}^{t-1}, h_u^{t-1}), \ \forall v_k \in V_i \qquad 7$$

$$h_{v_k}^t = f_u(m_{v_k}^t, h_{v_k}^{t-1}), \ \forall v_k \in V_i, \qquad 8$$

where f_m is an MLP, f_u is a gated recurrent unit,[40] and $\mathcal{N}(v_k) = \{u \mid \{u, v_k\} \in E_i\}$ is the set of neighbors of the node v_k. We additionally denote the complete set of hidden vectors for iteration t as $h_{V_i}^t = \{h_{v_k}^t \mid v_k \in V_i\}$. Here, the hidden vectors of all the neighbors of a node v_k are used for calculating a message m_{v_k} to the node, and this message vector is then used for updating the hidden vector. The iteration completes after n_t iterations, yielding us the final set of hidden vectors $h_{V_i}^{n_t}$. The number of iterations n_t and the size of the message $n_m = |m_{v_k}^t|$ are hyperparameters of the model. In our experiments, we choose $n_h = n_m = 20$ and $n_t = 3$. Additionally, we choose to have no hidden layers in f_i, and two hidden layers of size 64 in f_m.

After obtaining the hidden vector encodings for each node in the graph, we still need to aggregate the information in all the hidden vectors in to a single fixed-size encoding vector for the whole graph. Because the size of the graph is variable, this cannot be done by a simple concatenation. Instead, we use a separate aggregation network that selectively gathers information from the hidden vectors of all the nodes.[21] Given the final hidden vectors $h_{v_k}^{n_t}$, the graph encoding is obtained from

$$u_{v_k} = f_u(h_{v_k}^{n_t}), \ \forall v_k \in V_i \qquad 9$$

$$s_{v_k} = f_s(h_{v_k}^{n_t}), \ \forall v_k \in V_i \qquad 10$$

$$h_{G_i} = \sum_{v_k \in V_i} \sigma(s_{v_k}) u_{v_k}, \qquad 11$$

where f_u and f_s are MLPs and $\sigma(z) = 1/(1 + \exp(-z))$ is the logistic sigmoid function. Here, the hidden vector for each node gets expanded into two intermediate vectors u_{v_k} and s_{v_k} of size $|u_{v_k}| = |s_{v_k}| = |h_{G_i}| = n_G > n_h$. The vector s_{v_k} thresholded by the sigmoid function functions as a *gate* that selects the information from each of the vectors u_{v_k} and the result gets summed together to form the graph encoding vector h_{G_i}. The size n_G of the graph encoding is chosen to be larger than the hidden vector size n_h so that it has enough capacity to capture all the information in the whole graph, as opposed to just the neighborhood of each node as is the case for the hidden vectors. We choose $n_G = 128$, and the MLPs f_u and f_u both have one hidden layer of size 32. Here and in the MPNN we use summation as the aggregation operation, but other operations such as taking the average or the maximum could also be used.

Image feature selection

The second step in the graph construction process is to turn the information in the AFM image into an encoding vector. Information from the AFM image is needed in order to correctly classify the new node and connect it to the existing graph. For this, we will leverage the existing 3D feature map encodings from the encoder part of the U-net that we used for the position grid prediction. We will also utilize the graph encoding vector as an additional channel of information for selecting the most relevant features in the 3D feature maps in a kind of attention-gate mechanism for the feature map channels.

We first turn the graph encoding vector into a *query vector* using an MLP,

$$q = f_q(h_{G_i}, p_{i+1}). \qquad 12$$

Here, f_q has one hidden layer of size 64, and the size $n_q = |q|$ of the query vector is also 64. This is also the first place where we introduce the position p_{i+1} of the new node, which we have picked from the point cloud as the next node to be added to the graph. Knowing the position of the new node should be useful for the network in deciding which parts of the feature maps should be attended to the most.

Next, we consider each of the CNN encoder stages. We denote the outputs of the 3D convolution blocks with X_k, where the index $k \in \{1, 2, 3, 4\}$ enumerates the different stages. We pass each of the X_k feature maps through another 3D convolution block with identical number of layers and channels as in the corresponding CNN encoder stage to get another set of feature maps X'_k. Then we pass these feature maps along with the query vector to the attention-gate network, which works as follows:

$$a_k = f_a(X'_k + f_g(\text{Interp}(X'_g))) \qquad 13$$

$$q' = \sigma'(q) \qquad 14$$

$$A_k = \sum_{c=1}^{n_q} q'_c a_{k,c} \qquad 15$$

$$x_{k,c} = \sum_j (A_k \odot X'_{k,c})_j \quad \forall c \in \{1, \ldots, C_k\}, \qquad 16$$

where $k \in \{1, 2, 3, 4\}$, f_g and f_a are 3D convolutions, $\sigma'(z) = \exp(z)/(\sum_j \exp(z_j))$ is the SoftMax function, and $\odot$ denotes the element-wise product between tensors. Here, $X'_g = X'_4$ comes from the most compressed part of the CNN encoding network and represents a kind of global view of the information in the AFM image. Because X'_g is of smaller size than the other X'_k, it is up-sampled with trilinear interpolation before application of f_g and sum with X'_k. The result is passed to a 3D convolution with number of channels equal to the query vector size n_q, producing the feature map a_k. Next, the query vector q is thresholded with the SoftMax function and the result is used as a gating vector over the channels of a_k, producing a single-channel *attention map* A_k. Lastly, the attention map is multiplied element-wise with each channel in the original feature map X'_k, and the result is summed over every voxel in the 3D feature map, which produces the final encoding vector x_k with size C_k equal to the number of channels in X'_k.

Finally, with the encoding vectors from all of the CNN encoding stages in hand, the final encoding vector for all the information in the AFM image is produced with the application of one more MLP,

$$h_{X_i} = f_X(x_1, x_2, x_3, x_4). \qquad 17$$

Here, f_X has no hidden layers and the size of the output is $|h_{X_i}| = 128$.

Class and edge prediction

Now that we have the encoding vectors for both the current graph and the AFM image, h_{G_i} and h_{X_i}, we are ready to predict the class and the edge connections for the new node. The class is predicted by a simple classification MLP with SoftMax activation after the final layer:

$$c'_{i+1} = \sigma'(f_c(h_{G_i}, h_{X_i}, p_{i+1})) \qquad 18$$

$$c_{i+1} = \text{argmax}(c'_{i+1}). \qquad 19$$

The MLP produces an output vector c'_{i+1} of positive numbers that sum into unity, which can be interpreted as a list of probabilities for each class. The actual class c_{i+1} for the new node is then chosen based on the class with highest predicted probability. The new completed node is now $v_{i+1} = (p_{i+1}, c_{i+1})$.

The prediction of the edge connections is done in one shot by performing a binary classification task for each of the existing nodes in the graph to determine whether or not they should be connected to the new node. To this end, we return to the set of hidden vectors $h_{V_i}^{n_t}$ that we got from the MPNN, and we pass these hidden vectors to the MPNN for another n_t round of message propagation, yielding us an updated set of hidden vectors $h_{V_i}^{2n_t}$. Next, we take the class and the position of the new node and perform the same hidden vector initialization as we did for the existing nodes,

$$h_{v_{i+1}}^0 = f_i(p_{i+1}, c'_{i+1}). \qquad 20$$

Note that here we use the vector c'_{i+1} directly outputted from Equation 18 instead of a one-hot encoded vector. This is so that we can have gradient propagation through this operation as well. Finally, we combine the hidden vectors with the AFM encoding vector to predict the edge connections using an MLP,

$$e'_{v_k} = \sigma(f_e(h_{v_k}^{2n_t}, h_{v_{i+1}}^0, h_{X_i})) \quad \forall v_k \in V_i \qquad 21$$

$$e_{i+1} = \{\{v_{i+1}, v_k\} \mid v_k \in V_i, e'_{v_k} > t_e\}. \qquad 22$$

The MLP f_e with sigmoid activation first produces a probability for each node in the existing graph of whether it should be connected by an edge to the new node. The set of new edges e_{i+1} is then constructed by taking those nodes for which the probability is over a chosen threshold $t_e \in [0, 1]$. We simply choose a value of 0.5 for the threshold, but it could also be chosen to maximize or balance quantitative metrics such as the precision and recall. The MLPs f_c and f_e both have a single hidden layer of size 32.

The updated graph for the next iteration is $G_{i+1} = (V_i \cup \{v_{i+1}\}, E_i \cup e_{i+1})$. The iteration continues until there are no more new nodes to be added to the graph.

Author contributions
N.O., A.I., and A.S.F. conceived the research. N.O. and L.K. developed the software and ran the simulations. All authors were involved in the results analysis and contributed to the manuscript.

Code and data availability

All code and data used for model training and testing are available at https://github.com/SINGROUP/Graph-AFM.

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Open Access

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1557/s43577-022-00324-3.

References

1. L. Gross, F. Mohn, N. Moll, P. Liljeroth, G. Meyer, *Science* **325**, 1110 (2009)
2. A.A. Khajetoorians, D. Wegner, A.F. Otte, I. Swart, *Nat. Rev. Phys.* **1**, 703 (2019)
3. N. Pavliček, L. Gross, *Nat. Rev. Chem.* **1**, 0005 (2017)
4. G. Binnig, H. Rohrer, C. Gerber, E. Weibel, *Phys. Rev. Lett.* **49**, 57 (1982)
5. G. Binnig, C.F. Quate, C. Gerber, *Phys. Rev. Lett.* **56**, 930 (1986)
6. F. Giessibl, *Rev. Mod. Phys.* **75**, 949 (2003)
7. S.V. Kalinin, E. Strelcov, A. Belianinov, S. Somnath, R.K. Vasudevan, E.J. Lingerfelt, R.K. Archibald, C. Chen, R. Proksch, N. Laanait, S. Jesse, *ACS Nano* **10**, 9068 (2016)
8. O.M. Gordon, P.J. Moriarty, *Mach. Learn. Sci. Technol.* **1**, 023001 (2020)
9. M. Rashidi, R. Wolkow, *ACS Nano* **12**, 5185 (2018)
10. B. Alldritt, F. Urtev, N. Oinonen, M. Aapro, J. Kannala, P. Liljeroth, A.S. Foster, *Comput. Phys. Commun.* **273**, 108258 (2022)
11. R. Pawlak, J.G. Vilhena, A. Hinaut, T. Meier, T. Glatzel, A. Baratoff, E. Gnecco, R. Perez, E. Meyer, *Nat. Commun.* **10**, 685 (2019)
12. D. Martin-Jimenez, S. Ahles, D. Mollenhauer, H.A. Wegner, A. Schirmeisen, D. Ebeling, *Phys. Rev. Lett.* **122**, 196101 (2019)
13. F. Schulz, J. Ritala, O. Krejčí, A.P. Seitsonen, A.S. Foster, P. Liljeroth, *ACS Nano* **12**, 5274 (2018)
14. M. Ellner, P. Pou, R. Pérez, *ACS Nano* **13**, 786 (2019)
15. B. Alldritt, P. Hapala, N. Oinonen, F. Urtev, O. Krejci, F.F. Canova, J. Kannala, F. Schulz, P. Liljeroth, A.S. Foster, *Sci. Adv.* **6**, eaay6913 (2020)
16. F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, G. Monfardini, *IEEE Trans. Neural Netw.* **20**, 61 (2009)
17. P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V.F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç.Gülçehre, H.F. Song, A.J. Ballard, J. Gilmer, G.E. Dahl, A. Vaswani, K.R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks. CoRR (2018). http://arxiv.org/1806.01261
18. T. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks (2017). http://arxiv.org/1609.02907
19. Y. Li, C. Gu, T. Dullien, O. Vinyals, P. Kohli, Graph matching networks for learning the similarity of graph structured objects. CoRR (2019). https://arxiv.org/abs/1904.12787
20. J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, in *Proceedings of the 34th International Conference on Machine Learning—Volume 70* (JMLR.org, Sydney, NSW, 2017), p. 1263
21. Y. Li, O. Vinyals, C. Dyer, R. Pascanu, P.W. Battaglia, Learning deep generative models of graphs. CoRR (2018). https://arxiv.org/abs/1803.03324
22. R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D.K. Duvenaud, R. Urtasun, R. Zemel, in *Advances in Neural Information Processing Systems*, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, Eds. (Curran Associates, Red Hook, 2019)
23. J. You, B. Liu, R. Ying, V. Pande, J. Leskovec, Graph convolutional policy network for goal-directed molecular graph generation (2019). https://arxiv.org/abs/1806.02473
24. D. Xu, Y. Zhu, C. B. Choy, L. Fei-Fei, Scene graph generation by iterative message passing (2017). https://arxiv.org/abs/1701.02426
25. J. Yang, J. Lu, S. Lee, D. Batra, D. Parikh, Graph R-CNN for scene graph generation (2018). https://arxiv.org/abs/1808.00191
26. O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional networks for biomedical image segmentation (2015). http://arxiv.org/abs/1505.04597. Accessed 11 June 2021
27. O. Oktay, J. Schlemper, L.L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N.Y. Hammerla, B. Kainz, B. Glocker, D. Rueckert, Attention U-Net: Learning where to look for the pancreas (2020). http://arxiv.org/abs/1804.03999. Accessed 11 June 2021
28. P. Hapala, G. Kichin, C. Wagner, F.S. Tautz, R. Temirov, P. Jelínek, *Phys. Rev. B* **90**, 085421 (2014)
29. J. Ingraham, V. Garg, R. Barzilay, T. Jaakkola, in *Advances in Neural Information Processing Systems*, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, Eds. (Curran Associates, Red Hook, 2019)
30. Y. Zuo, C. Chen, X. Li, Z. Deng, Y. Chen, J. Behler, G. Csányi, A.V. Shapeev, A.P. Thompson, M.A. Wood, S.P. Ong, *J. Phys. Chem. A* **124**, 731 (2020)
31. L. Zhang, J. Han, H. Wang, R. Car, E. Weinan, *Phys. Rev. Lett.* **120**, 143001 (2018)
32. V. Fung, J. Zhang, E. Juarez, B.G. Sumpter, *NPJ Comput. Mater.* **7**, 84 (2021)
33. A. Dunn, Q. Wang, A. Ganose, D. Dopp, A. Jain, *NPJ Comput. Mater.* **6**, 138 (2020)
34. N. Oinonen, C. Xu, B. Alldritt, F.F. Canova, F. Urtev, S. Cai, O. Krejčí, J. Kannala, P. Liljeroth, A.S. Foster, *ACS Nano* **16**, 89 (2022)
35. J. Carracedo-Cosme, C. Romero-Muñiz, R. Pérez, *Nanomaterials* **11**, 1658 (2021)
36. J. Carracedo-Cosme, C. Romero-Muñiz, R. Pérez, *J. Chem. Inf. Model.* **62**, 1214 (2022)
37. J. Carracedo-Cosme, R. Pérez, Molecular identification with atomic force microscopy and conditional generative adversarial networks (2022). http://arxiv.org/abs/2205.00447
38. V. Oliveira, R. Lotufo, *Graphics, Patterns and Images (SIBGRAPI), 2010 23rd SIBGRAPI*, vol. 2010 (2010)
39. S. Allegretti, F. Bolelli, C. Grana, *IEEE Trans. Parallel Distrib. Syst.* **31**, 423 (2020)
40. K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation. CoRR (2014). https://arxiv.org/abs/1406.1078 □