

Electronic Supplementary Information for

An Automated Protocol to Construct Flexibility Parameters for Classical Forcefields: Applications to Metal-Organic Frameworks

Reza Ghanavati⁺, Alma C. Escobosa⁺, and Thomas A. Manz*

Chemical & Materials Engineering, New Mexico State University, Las Cruces, NM 88001

*corresponding author email: tmanz@nmsu.edu. ⁺These authors contributed equally to this work.

Contents

[S1. Torsion model potential for dihedrals with linear equilibrium bond angle](#)

[S2. Python function to determine which dihedral middle bonds are contained in rings](#)

[S3. Detailed linear equations for flexibility parameters](#)

[S4. Python function to compute lambda best](#)

[S5. Test of training datasets not including AIMD data](#)

[S6. Test of torsion scans for all instances of a rotatable dihedral type](#)

[S7. Histograms showing the number of instances per type](#)

[S8. Additional LASSO results for MOFs without rotatable dihedrals](#)

[S.9 Images of MOF crystal structures](#)

S1. Torsion model potential for dihedrals with linear equilibrium bond angle

A dihedral in which one or both of the equilibrium bond angles is linear (or close to linear) will be referred to here as a ‘linear dihedral’. For linear dihedrals, the leading angle-damped dihedral torsion (ADDT) modes can be derived as follows. First, we start with the allowed angle-damped dihedral torsion potential forms as given in ref [S1]:

$$U_{\text{form}_1}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = \frac{k_{\text{form}_1}}{2} \left(\frac{f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}]}{f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}^{\text{eq}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}^{\text{eq}}]} h_{\phi}[\phi_{\text{ABCD}}] - h_{\phi}[\phi_{\text{ABCD}}^{\text{eq}}] \right)^2 \quad (1)$$

$$U_{\text{form}_2}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = k_{\text{form}_2} \frac{f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}]}{f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}^{\text{eq}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}^{\text{eq}}]} \left(g_{\phi}[\phi_{\text{ABCD}}] - g_{\phi}[\phi_{\text{ABCD}}^{\text{eq}}] \right) \quad (2)$$

(Here, $f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}]$ is a generic placeholder for some angle-damping function(s) that depends on the bond angle's value. Moreover, we could use different specific functions for $f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}]$ in the form_1 and form_2 potentials.) Because the torsion barrier becomes zero when the bond angle is linear, we have

$$\lim_{\theta_{\text{ABC}}^{\text{eq}} \rightarrow \pi} f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}^{\text{eq}}] = 0 \quad (S3)$$

To avoid division by zero for linear dihedrals, we must modify the two forms above as

$$\tilde{k}_{\text{form}_1} = \frac{k_{\text{form}_1}}{\left(f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}^{\text{eq}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}^{\text{eq}}] \right)^2} \quad (S4)$$

$$U_{\text{form}_1}^{\text{linear_dihedral}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = \frac{\tilde{k}_{\text{form}_1}}{2} \left(f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}] h_{\phi}[\phi_{\text{ABCD}}] \right)^2 \quad (S5)$$

$$\tilde{k}_{\text{form}_2} = \frac{k_{\text{form}_2}}{f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}^{\text{eq}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}^{\text{eq}}]} \quad (S6)$$

$$U_{\text{form}_2}^{\text{linear_dihedral}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = \tilde{k}_{\text{form}_2} f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}] f_{\theta_{\text{BCD}}}[\theta_{\text{BCD}}] g_{\phi}[\phi_{\text{ABCD}}] \quad (S7)$$

Since $\phi_{\text{ABCD}}^{\text{eq}}$ is undefined for linear dihedrals, form_2 for linear dihedrals omits the $g_{\phi}[\phi_{\text{ABCD}}^{\text{eq}}]$ term.

In a procedure analogous to that used for the nonlinear dihedrals in ref [S1], for the linear dihedrals we first insert $h_{\phi}[\phi_{\text{ABCD}}] = \sin[m\phi_{\text{ABCD}}]$ and $f_{\theta_{\text{ABC}}}[\theta_{\text{ABC}}] = f_m^{\text{ABC}}[\mathcal{K}_{\text{ABC}}]$ into eqn (S5) to get

$$U_{\text{sin_form}_1}^{\text{linear_dihedral}}[\mathcal{K}_{\text{ABC}}, \mathcal{K}_{\text{BCD}}, \phi_{\text{ABCD}}] = \frac{\tilde{k}_{\text{form}_1}^{\text{m, sin}}}{2} \left(f_m^{\text{ABC}}[\mathcal{K}_{\text{ABC}}] \right)^2 \left(f_m^{\text{BCD}}[\mathcal{K}_{\text{BCD}}] \right)^2 \sin^2[m\phi_{\text{ABCD}}] \quad (S8)$$

where the kangal is

$$\mathcal{K}_{\text{ABC}} = \cos[\theta_{\text{ABC}}/2] \quad (S9)$$

This procedure is repeated for $h_{\phi}[\phi_{\text{ABCD}}] = \cos[m\phi_{\text{ABCD}}]$ to give:

$$U_{\text{cos_form}_1}^{\text{linear_dihedral}}[\mathcal{K}_{\text{ABC}}, \mathcal{K}_{\text{BCD}}, \phi_{\text{ABCD}}] = \frac{\tilde{k}_{\text{form}_1}^{\text{m, cos}}}{2} \left(f_m^{\text{ABC}}[\mathcal{K}_{\text{ABC}}] \right)^2 \left(f_m^{\text{BCD}}[\mathcal{K}_{\text{BCD}}] \right)^2 \cos^2[m\phi_{\text{ABCD}}] \quad (S10)$$

Clearly,

$$U_{\text{sin_form}_1}^{\text{linear_dihedral}}[\mathcal{K}_{\text{ABC}}^{\text{eq}}, \mathcal{K}_{\text{BCD}}^{\text{eq}}, \phi_{\text{ABCD}}] = U_{\text{cos_form}_1}^{\text{linear_dihedral}}[\mathcal{K}_{\text{ABC}}^{\text{eq}}, \mathcal{K}_{\text{BCD}}^{\text{eq}}, \phi_{\text{ABCD}}] \approx 0 \quad (S11)$$

Since the potential energy increases whenever the system's geometry is moved away from its minimum energy (i.e., the optimized equilibrium ground-state geometry), this requires the constraints

$$\tilde{k}_{\text{form}_1}^{\text{m, sin}} \geq 0 \quad (S12)$$

$$\tilde{k}_{\text{form}_1}^{\text{m, cos}} \geq 0 \quad (S13)$$

Substituting

$$\sin^2[m\phi_{ABCD}] = \frac{1 - \cos[2m\phi_{ABCD}]}{2} \quad (\text{S14})$$

$$\cos^2[m\phi_{ABCD}] = \frac{1 + \cos[2m\phi_{ABCD}]}{2} \quad (\text{S15})$$

into eqn (S8) and (S10) gives

$$U_{\sin_form_1}^{\text{linear_dihedral}}[\mathcal{K}_{ABC}, \mathcal{K}_{BCD}, \phi_{ABCD}] = \frac{\tilde{k}_{form_1}^{m,\sin}}{2} (f_m^{ABC}[\mathcal{K}_{ABC}])^2 (f_m^{BCD}[\mathcal{K}_{BCD}])^2 \left(\frac{1 - \cos[2m\phi_{ABCD}]}{2} \right) \quad (\text{S16})$$

$$U_{\cos_form_1}^{\text{linear_dihedral}}[\mathcal{K}_{ABC}, \mathcal{K}_{BCD}, \phi_{ABCD}] = \frac{\tilde{k}_{form_1}^{m,\cos}}{2} (f_m^{ABC}[\mathcal{K}_{ABC}])^2 (f_m^{BCD}[\mathcal{K}_{BCD}])^2 \left(\frac{1 + \cos[2m\phi_{ABCD}]}{2} \right) \quad (\text{S17})$$

Since $\sin^2[m\phi_{ABCD}] + \cos^2[m\phi_{ABCD}] = 1$, $U_{\sin_form_1}^{\text{linear_dihedral}}$ and $U_{\cos_form_1}^{\text{linear_dihedral}}$ are not fully linearly independent. Consequently, to span the entire possible space of dihedral potential energy functions, it is necessary to also include dihedral potentials of form_2. This is obvious, because $U_{\sin_form_1}^{\text{linear_dihedral}}[\mathcal{K}_{ABC}, \mathcal{K}_{BCD}, \phi_{ABCD}]$ and $U_{\cos_form_1}^{\text{linear_dihedral}}[\mathcal{K}_{ABC}, \mathcal{K}_{BCD}, \phi_{ABCD}]$ are both even functions of ϕ_{ABCD} . The odd functions of ϕ_{ABCD} can be included by choosing

$$U_{form_2}^{\text{linear_dihedral}}[\theta_{ABC}, \theta_{BCD}, \phi_{ABCD}] = \tilde{k}_{form_2} (f_m^{ABC}[\mathcal{K}_{ABC}])^2 (f_m^{BCD}[\mathcal{K}_{BCD}])^2 S_{\text{instance}} \sin[2m\phi_{ABCD}] \quad (\text{S18})$$

$$S_{\text{instance}} = \pm 1 \quad (\text{S19})$$

The form of eqn (S18) was specifically chosen so that the angle-damping prefactor $(f_m^{ABC}[\mathcal{K}_{ABC}])^2 (f_m^{BCD}[\mathcal{K}_{BCD}])^2$ has the same form as in eqn (S16)–(S17) above, and also so that the $\sin[2m\phi_{ABCD}]$ function in eqn (S18) complements the $\cos[2m\phi_{ABCD}]$ function appearing in eqn (S16)–(S17) above. In eqn (S18), the lowest allowed power of \mathcal{K}_{ABC} and \mathcal{K}_{BCD} must be ≥ 2 (i.e., strictly greater than one) to ensure the torsion force goes to zero when the bond angle reaches its equilibrium linear value. To ensure combined angle-dihedral coordinate branch equivalency (see ref [S1]), the sin function must contain an even multiple of ϕ_{ABCD} to match the even power of \mathcal{K}_{ABC} in $(f_m^{ABC}[\mathcal{K}_{ABC}])^2$. These observations show that eqn (S18) has the proper form.

$U_{form_2}^{\text{linear_dihedral}}[\theta_{ABC}, \theta_{BCD}, \phi_{ABCD}]$ involves the sign S_{instance} , because $U_{form_2}^{\text{linear_dihedral}}[\theta_{ABC}, \theta_{BCD}, \phi_{ABCD}]$ is an odd function of ϕ_{ABCD} . In general, different dihedral instances of the same dihedral type can have different values of S_{instance} . Because ϕ_{eq} is undefined for linear dihedrals, the value of S_{instance} cannot be assigned using the sign of ϕ_{eq} for linear dihedrals. For linear dihedrals, the value of S_{instance} must be determined by detecting local mirror-image environments (i.e., local chiral enantiomer environments) for different dihedral instances of the same dihedral type. Owing to this complication, we did not include $U_{form_2}^{\text{linear_dihedral}}[\theta_{ABC}, \theta_{BCD}, \phi_{ABCD}]$ in the flexibility models parameterized in this article.

In general, the ADDT_linear model potential may contain contributions from $m = 1, 2, 3$, etc. In practice, this will be truncated at some order. For calculations presented in this article, our ADDT_linear model included the following two modes:

$$G_{\text{mod e}_8}^{\text{ADDT_linear}}[\theta_{ABC}, \theta_{BCD}, \phi_{ABCD}] = (f_1^{ABC})^2 (f_1^{BCD})^2 \left(\frac{1 - \cos[2\phi_{ABCD}]}{2} \right) \quad (\text{S20})$$

$$G_{\text{mode}_9}^{\text{ADDT_linear}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = (f_1^{\text{ABC}})^2 (f_1^{\text{BCD}})^2 \left(\frac{1 + \cos[2\phi_{\text{ABCD}}]}{2} \right) \quad (\text{S21})$$

$$U_{\text{mode}_8}^{\text{ADDT_linear}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = k_{\phi}^8 G_{\text{mode}_8}^{\text{ADDT_linear}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] \quad (\text{S22})$$

$$U_{\text{mode}_9}^{\text{ADDT_linear}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] = k_{\phi}^9 G_{\text{mode}_9}^{\text{ADDT_linear}}[\theta_{\text{ABC}}, \theta_{\text{BCD}}, \phi_{\text{ABCD}}] \quad (\text{S23})$$

The ADDT_linear potential modes were numbered as modes ‘8’ and ‘9’, to keep them separate from the seven modes used in the ADDT_rotatable model potential. Comparing eqn (S20)–(S23) to eqn (S16)–(S17) shows that

$$k_{\phi}^8 = \tilde{k}_{\text{form}_1}^{1,\sin} / 2 \quad (\text{S24})$$

$$k_{\phi}^9 = \tilde{k}_{\text{form}_1}^{1,\cos} / 2 \quad (\text{S25})$$

In this work, we treated MOFs according to two cases. *Case # 1*: When the after-pruning dihedral types for a particular MOF contained no linear dihedral types, then we did not include any ADDT_linear model potential in the parameterized flexibility model for this MOF. In this case, when we generated a before-pruning flexibility model for comparison purposes (to quantify the effects of dihedral pruning on the R-squared and RMSE values) then we also did not include any linear dihedral types (or ADDT_linear model potential) for this MOF in the before-pruning parameterization. *Case # 2*: When the after-pruning dihedral types for a particular MOF contained one or more linear dihedral types, then we included the ADDT_linear model potential in the parameterized flexibility model for this MOF. In this case, when we generated a before-pruning flexibility model for comparison purposes (to quantify the effects of dihedral pruning on the R-squared and RMSE values) then we also included all linear dihedral types (and the ADDT_linear model potential) for this MOF in the before-pruning parameterization.

S2. Python function to determine which dihedral middle bonds are contained in rings

Figure S1 shows a rigorously correct and complete Python function we wrote that determines which middle bonds are parts of rings (i.e., bond path cycles) and which are not. The algorithm we used is briefly described as follows. The function takes three inputs:

- (1) The list of bonds that comprise middle bonds in (active) dihedral instances; this list is called ‘middle_bonds_dihedrals’.
- (2) The list of first neighbor atom images (aka ‘first_neighbor_atoms’) for every atom in the reference unit cell. For example, suppose that atom 43 in the reference (i.e., (0,0,0)) unit cell is directly bonded to the (1,0,0) image of atom 10; then this would be stored as an entry [10,(1,0,0)] in the list of first neighbors to atom 43.
- (3) The number of atoms (N_{atoms}) in the reference unit cell.

Here the term ‘atom image’ refers to an entry such as [10,(1,0,0)] that contains the atom index (in this case = 10) and the unit cell translation indices (in this case = (1,0,0)). The middle bond of a dihedral instance is read and the two corresponding atom images forming this bond are ‘atom_image_A’ and ‘atom_image_B’. Next, we initialized four working lists:

- (a) ‘visited_A’ and ‘visited_B’ lists that are each initialized to [atom_image_A, atom_image_B],
- (b) A ‘not_searched_A’ list that is initialized to [atom_B], and a ‘not_searched_B’ list that is initialized to [atom_A].

The algorithm contains an outer, middle, and inner layer of nested loops. The outer loop is a FOR loop that runs over the middle bonds in the dihedral instances. The middle loop is a WHILE loop that determines whether a particular middle bond is contained within a ring or not. The two inner loops run over the first neighbor atoms of `current_atom_A` and `current_atom_B`, which are the current atoms along the bond paths being searched and the first elements of `not_searched_A` and `not_searched_B`, respectively.

`Current_atom_image_A` is set to the first entry in `not_searched_A`. The inner loop for `current_atom_A` works as follows. Using the `first_neighbor_atoms` list, the algorithm identifies all of the first neighbors of `current_atom_image_A`. For example, if `current_atom_image_A = [43,(0,3,1)]` then one of its first neighbors is identified as $[10,(1,0,0) + (0,3,1)] = [10,(1,3,1)]$, where the image indices in `current_atom_image_A` and `first_neighbor_atoms` are added together to get ‘atom image’ of the first neighbor (aka ‘adjusted_neighbor_atom_image’) of `current_atom_image_A`. If neither `current_atom_image_A` nor `adjusted_neighbor_atom_image` equals `atom_image_B`, then if `adjusted_neighbor_atom_image` is in `visited_B` then a ring has been located and the inner loop iterations break. Otherwise, if `adjusted_neighbor_atom_image` is not in `visited_A`, then it is appended to `visited_A` and also appended to `not_searched_A` if the edge depth is $\leq 2N_{\text{atoms}}$. After looping over all of the first neighbor atom images of `current_atom_A`, the first entry in `not_searched_A` is removed because it has now been searched.

`Current_atom_image_B` is set to the first entry in `not_searched_B`. The algorithm then proceeds to the inner loop for `current_atom_B`. This works analogously to the inner loop iterations for `current_atom_A` described in the previous paragraph, except the roles of ‘A’ and ‘B’ are now swapped.

Here, the ‘edge_depth’ refers to the number of bonds in the shortest bond path connecting `current_atom_image_A` to `atom_image_A` along the search path, or in the shortest bond path connecting `current_atom_image_B` to `atom_image_B` along the search path. The algorithm searches paths emanating from each end of the middle bond to an edge_depth of $2N_{\text{atoms}}$. By adding the edge depths from each end, a total edge depth of $2N_{\text{atoms}} + 2N_{\text{atoms}} = 4N_{\text{atoms}}$ is searched. This guarantees the algorithm searches for all bond path cycles containing up to $4N_{\text{atoms}}$, which guarantees that a bond path cycle (aka ring) will be discovered if one passes through the middle bond being investigated. The algorithm finds a ring if any bond path emanating from the A end of the middle bond joins up with any bond path emanating from the B end of the middle bond. The algorithm determines a middle bond is not part of any ring if it exhausts all search paths on either side (i.e., either `not_searched_A` or `not_searched_B` becomes empty) or it reaches a searched edge depth of $2N_{\text{atoms}}$ on both sides.

```

def finding_rings(middle_bonds_dihedrals, first_neighbor_atoms, Natoms)
    rings_found = []

    for i in range(len(middle_bonds_dihedrals)):
        visited_A = []
        visited_B = []
        rings_found.append(False)
        not_searched_A = []
        not_searched_B = []
        atom_image_A = middle_bonds_dihedrals[i][0]
        atom_image_B = middle_bonds_dihedrals[i][1]
        visited_A.append(atom_image_A)
        visited_A.append(atom_image_B)
        not_searched_A.append([atom_image_B,1])
        visited_B.append(atom_image_B)
        visited_B.append(atom_image_A)
        not_searched_B.append([atom_image_A,1])

    while len(not_searched_A) > 0 and len(not_searched_B) > 0:
        current_atom_image_A = not_searched_A[0][0]
        edge_depth_A = not_searched_A[0][1]
        current_atom_A = current_atom_image_A[0]
        current_image_A = current_atom_image_A[1]
        current_atom_image_B = not_searched_B[0][0]
        edge_depth_B = not_searched_B[0][1]
        current_atom_B = current_atom_image_B[0]
        current_image_B = current_atom_image_B[1]

    for j in range(len(first_neighbor_atoms[current_atom_A-1])):
        neighboring_atom_image = first_neighbor_atoms[current_atom_A-1][j]
        neighboring_atom = neighboring_atom_image[0]
        neighboring_image = neighboring_atom_image[1]
        adjusted_neighboring_image = [neighboring_image[0] + current_image_A[0], neighboring_image[1] + current_image_A[1], neighboring_image[2] + current_image_A[2]]
        adjusted_neighbor_atom_image = [neighboring_atom,adjusted_neighboring_image]
        if current_atom_image_A != atom_image_B:
            if (adjusted_neighbor_atom_image in visited_B) and (adjusted_neighbor_atom_image != atom_image_B):
                rings_found[i] = True
                not_searched_A = []
                not_searched_B = []
                break
            if adjusted_neighbor_atom_image not in visited_A:
                visited_A.append(adjusted_neighbor_atom_image)
                if edge_depth_A <= 2*Natoms:
                    not_searched_A.append([adjusted_neighbor_atom_image,edge_depth_A+1])

    if len(not_searched_A) > 0:
        not_searched_A.pop(0)
    else:
        break

    for j in range(len(first_neighbor_atoms[current_atom_B-1])):
        neighboring_atom_image = first_neighbor_atoms[current_atom_B-1][j]
        neighboring_atom = neighboring_atom_image[0]
        neighboring_image = neighboring_atom_image[1]
        adjusted_neighboring_image = [neighboring_image[0] + current_image_B[0], neighboring_image[1] + current_image_B[1], neighboring_image[2] + current_image_B[2]]
        adjusted_neighbor_atom_image = [neighboring_atom,adjusted_neighboring_image]
        if current_atom_image_B != atom_image_A:
            if (adjusted_neighbor_atom_image in visited_A) and (adjusted_neighbor_atom_image != atom_image_A):
                rings_found[i] = True
                not_searched_B = []
                not_searched_A = []
                break
            if adjusted_neighbor_atom_image not in visited_B:
                visited_B.append(adjusted_neighbor_atom_image)
                if edge_depth_B <= 2*Natoms:
                    not_searched_B.append([adjusted_neighbor_atom_image,edge_depth_B+1])

    if len(not_searched_B) > 0:
        not_searched_B.pop(0)

    return rings_found

```

Figure S1: Python function to determine which dihedral middle bonds are contained in rings.

S3. Detailed linear equations for flexibility parameters

First, we need to read the files containing lists of internal coordinate types and instances, which were generated using the procedure described in Section 5 of the main text. Specifically, files containing lists of the following are read: atom types; bond types; bond instances; angle types; angle instances; dihedral types; dihedral instances; classification of each dihedral type as non-rotatable, hindered, rotatable, or linear; training and validation geometries and their quantum-mechanically-computed atom-in-material forces; the (experimental) lattice vectors; rigid torsion scan geometries (and quantum-mechanically-computed single-point energies) for one instance of each rotatable dihedral type. Then, the internal coordinates (i.e., bond lengths, UB lengths, angles, and dihedrals) are parsed into nested arrays containing all the interactions grouped by type. The angles list contains a flag indicating whether the angle is part of a 3-membered or 4-membered ring or not. The bond-bond cross terms use the full list of angles, while the angle-bending potential skips any angles that are part of a 3-membered or 4-membered ring.

Our linear regression problem contains two kinds of observation variables in the combined training dataset: (a) quantum-mechanically-computed atom-in-material forces and (b) quantum-mechanically-computed total energies. The quantum-mechanically-computed atom-in-material forces are from the QM-optimized geometry, AIMD geometries, and finite-displacement ‘Hessian’ geometries. There are a total of

$$f_rows = 3N_{atoms}N_{force_geoms} \quad (S26)$$

force components in the forces training dataset. The quantum-mechanically-computed total energies are from the rigid torsion scan (RTS) geometries and comprise the rotatable dihedrals training dataset.

Since the forces are zero at the equilibrium geometry, the no-intercept linear regression model is used. Therefore, to be able to use a no-intercept model, we centered the observation variable (i.e., the QM-computed energy) for rotatable dihedral torsions by subtracting the average value:

$$E_{avg} = \frac{1}{TN_{rdt}} \sum_{i=1}^{N_{rdt}} \sum_{j=1}^T E_{i,j}^{SPG} \quad (S27)$$

$T=36$ is the number of equally spaced dihedral values along each rigid torsion scan. N_{rdt} is the number of rotatable dihedral types. Here, $E_{i,j}^{SPG}$ is the quantum-mechanically-computed total energy of the single-point geometry (SPG) corresponding to the j^{th} displaced geometry of the i^{th} rotatable dihedral type.

This gives the following observation variables for the combined training dataset \vec{Y}^{QM} comprised of the forces training dataset \vec{Y}^{QM_forces} and the rotatable dihedrals training dataset $\vec{Y}^{QM_energies}$:

$$\vec{Y}^{QM} = \begin{bmatrix} \vec{Y}^{QM_forces} \\ \vec{Y}^{QM_energies} \end{bmatrix} \quad (S28)$$

$$\vec{Y}^{QM_forces} = \begin{bmatrix} F_{atom1,x}^{QM} \\ F_{atom1,y}^{QM} \\ F_{atom1,z}^{QM} \\ F_{atom2,x}^{QM} \\ \vdots \\ F_{N_{atoms},z}^{QM} \end{bmatrix} \quad (S29)$$

$$\vec{Y}^{QM_energies} = \begin{bmatrix} (E_{1,1}^{SPG} - E_{avg}) \\ (E_{1,2}^{SPG} - E_{avg}) \\ (E_{1,3}^{SPG} - E_{avg}) \\ \vdots \\ (E_{N_{rd},T}^{SPG} - E_{avg}) \end{bmatrix} \quad (S30)$$

N_{atoms} is the number of atoms in the material's unit cell. As a result, \vec{Y}^{QM_forces} is a column vector with $\text{length}[\vec{Y}^{QM_forces}] = 3(N_{atoms})(N_{forces_geoms})$, and $\vec{Y}^{QM_energies}$ is a column vector with $\text{length}[\vec{Y}^{QM_energies}] = TN_{rd}$.

The predictor variables also contain two sets of data: one for forces fitting and the other for the rotatable dihedrals fitting. Let \mathbf{M} be a matrix containing values of the predictor variables. This leads to the following linear model:

$$Y_i^{pred} = \sum_{j=1}^p (M_{ij}\beta_j) \quad (S31)$$

where i is the observation index and j is the model parameter index. In our case, each model parameter is a force constant for a flexibility term:

$$\beta_j = k_j \quad (S32)$$

The total number of attempted force constants in the model is p . Here, the term 'total number of attempted force constants' refers to the number of flexibility terms (i.e., number of force constants) that were 'attempted' before any of these were zeroed out by the bounds or regularization constraints.

Because the atom-in-material forces for the AIMD-generated geometries depend on all of the force constants while the RTS energies depend only on the rotatable dihedral force constants, it follows that the predictor variables matrix \mathbf{M} has the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{M1}(1:f_rows, 1:(N_{AFCs} - N_{rd_AFCs})) & \mathbf{M1}(1:f_rows, (1 + N_{AFCs} - N_{rd_AFCs}):N_{AFCs}) \\ 0 & \mathbf{M2} \end{bmatrix} \quad (S33)$$

It directly follows that

$$\text{size}[\mathbf{M1}] = (f_rows, N_{AFCs}) \quad (S34)$$

$$\text{size}[\mathbf{M2}] = (TN_{rd}, N_{rd_AFCs}) \quad (S35)$$

N_{AFCs} is the total number of attempted force constants. N_{rd_AFCs} is the number of rotatable dihedral attempted force constants.

To define the $\mathbf{M1}$, we need to start from the following relation:

$$\vec{F}_A^{\text{bonded}} = -\vec{\nabla}_A \mathbf{U}_{\text{cluster}}^{\text{bonded,new}} \left[\left\{ \vec{R}_B \right\} \right] = -\sum_{j=1}^p k_j \vec{\nabla}_A G_j \left[\left\{ \vec{R}_B \right\}, \left\{ \alpha_h^{\text{eq}} \right\} \right] \quad (\text{S36})$$

Comparing eqn (S31) and (S32) to (S36) reveals that

$$\mathbf{M1}_{(3N_{\text{atoms}}(\mu-1)+3(\gamma-1)+\xi),j} = -\vec{\nabla}_\gamma G_j \left[\left\{ \vec{R}_B^\mu \right\}, \left\{ \alpha_h^{\text{eq}} \right\} \right] \quad (\text{S37})$$

where μ is the geometry number. $\xi = 1, 2, 3$ for x, y, z force components of atom γ , respectively. $\left\{ \alpha_h^{\text{eq}} \right\}$ are the equilibrium values of the internal coordinates. The derivatives in eqn (S37) can be computed either numerically (using finite difference approximation) or analytically. In this article, we used finite difference approximation with 0.0001 Å displacement of an atom along x, y, or z to compute these derivatives:

$$\frac{\partial G_j}{\partial R_\gamma^\xi} = \frac{G_j \left[\left\{ \vec{R}_{B \neq \gamma}^\mu, \vec{R}_\gamma^\mu + (\Delta d) \hat{x}_\gamma^\xi \right\}, \left\{ \alpha_h^{\text{eq}} \right\} \right] - G_j \left[\left\{ \vec{R}_{B \neq \gamma}^\mu, \vec{R}_\gamma^\mu \right\}, \left\{ \alpha_h^{\text{eq}} \right\} \right]}{\Delta d} \quad (\text{S38})$$

where j is the interaction index; μ is the geometry index; γ is the atom index; $\xi = 1, 2, 3$ for x, y, z components, respectively; Δd is the displacement distance; \hat{x}_γ^ξ is a unit vector.

For the predictor variables in rotatable dihedrals fitting, as discussed in Sections 2.2 and 7.1, we can have up to seven active modes for each rotatable dihedral. We first determine which dihedral modes are active for each rotatable dihedral type using the method described in Section 7.1. Then let $G_{\mu,i,9,m_\zeta}^{\text{SPG}}$ be the k-free flexibility term for the 9^{th} dihedral instance's ζ^{th} smart-selected torsion mode $m_\zeta \in \{1, 2, \dots, 7\}$ of the i^{th} rotatable dihedral type of the μ^{th} geometry in the rotatable dihedrals training dataset. We sum these over all instances belonging to the same rotatable dihedral type to get

$$G_{\mu,i,m_\zeta}^{\text{SPG}} = \sum_9 G_{\mu,i,9,m_\zeta}^{\text{SPG}} \quad (\text{S39})$$

For a rigid torsion scan along an instance of a particular rotatable dihedral type, we still have to compute $G_{\mu,i,9,m_\zeta}^{\text{SPG}}$ for every instance of every rotatable dihedral type, because the dihedral displacements of different instances and/or different types are not necessarily independent of each other. Next, we average over all of the rigid torsion scan geometries:

$$G_{i,m_\zeta}^{\text{avg}} = \frac{1}{\text{TN}_{\text{rdt}}} \sum_{\mu=1}^{\text{TN}_{\text{rdt}}} G_{\mu,i,m_\zeta}^{\text{SPG}} \quad (\text{S40})$$

This allows us to construct a zero-intercept model by defining the matrix **M2** as follows:

$$\mathbf{M2} = \begin{bmatrix} \left(G_{1,1,m_1}^{\text{SPG}} - G_{1,m_1}^{\text{avg}} \right) & \left(G_{1,1,m_2}^{\text{SPG}} - G_{1,m_2}^{\text{avg}} \right) & \cdots & \left(G_{1,\text{N}_{\text{rdt}},m_{\text{last}}}^{\text{SPG}} - G_{\text{N}_{\text{rdt}},m_{\text{last}}}^{\text{avg}} \right) \\ \left(G_{2,1,m_1}^{\text{SPG}} - G_{1,m_1}^{\text{avg}} \right) & & & \\ \vdots & \ddots & \ddots & \vdots \\ \left(G_{\text{TN}_{\text{rdt}},1,m_1}^{\text{SPG}} - G_{1,m_1}^{\text{avg}} \right) & \cdots & \cdots & \left(G_{\text{TN}_{\text{rdt}},\text{N}_{\text{rdt}},m_{\text{last}}}^{\text{SPG}} - G_{\text{N}_{\text{rdt}},m_{\text{last}}}^{\text{avg}} \right) \end{bmatrix} \quad (\text{S41})$$

(Note: In the software code, the columns for the dihedral potential modes (and associated dihedral torsion force constants) could be presented in a different order than that shown in eqn (S41). However, regardless of the particular ordering, the set of equations solved would be functionally equivalent.)

S4. Python function to compute lambda best

```
def finding_lambda_best(lambda_sequence, R2_calculated_sequence, number_of_parameters_sequence, Natoms):
    temp = 0
    lambda_a = lambda_sequence[-1]
    R_sq_a = R2_calculated_sequence[-1]
    n_params_a = number_of_parameters_sequence[-1]
    lambda_temp = lambda_a

    for i in range((len(lambda_sequence)-2),-1,-1):
        lambda_b = lambda_sequence[i]
        R_sq_b = R2_calculated_sequence[i]
        n_params_b = number_of_parameters_sequence[i]
        if n_params_b < n_params_a:
            temp = ((3*Natoms)/(1-R_sq_b))*((R_sq_a - R_sq_b)/(n_params_a - n_params_b))
            if temp > 0.5:
                lambda_temp = lambda_a
                n_parameters_temp = n_params_a
                break
            else:
                lambda_a = lambda_b
                R_sq_a = R_sq_b
                n_params_a = n_params_b

    lambda_best = lambda_temp
    n_parameters_best = n_parameters_temp

    return [lambda_best, n_parameters_best]
```

Figure S2: Python function to compute lambda_best.

S5. Test of training datasets not including AIMD data

For one MOF without rotatable dihedrals and one MOF with rotatable dihedrals, we tested the impact of omitting AIMD data from the training dataset. In this case, the training dataset was comprised of only finite-displaced ‘Hessian’ geometries, the optimized geometry, and single-point displaced dihedral geometries (if any rotatable dihedrals were present in the MOF). The validation datasets were comprised of the optimized geometry plus AIMD data from runs independent of AIMD data (if any) used in the training dataset. For each MOF, the same validation dataset was used irrespective of the training dataset used. As shown in Table S1, including AIMD data in the training dataset resulted in a better flexibility model, because it significantly increased the R-squared validation and significantly lowered the RMSE validation.

Table S1: Effects of excluding AIMD data from the training dataset. Values are listed with using individual [average] equilibrium values of internal coordinates. These calculations were performed after dihedral pruning with no bond-bond cross terms.

	R-squared validation	RMSE (eV/Å) validation
PEYPIU with AIMD in training	0.9317 [0.9231]	0.1779 [0.1887]
PEYPIU without AIMD in training	0.8951 [0.8536]	0.2204 [0.2603]
PEGBEK01 with AIMD in training	0.9464 [0.9394]	0.1486 [0.1581]
PEGBEK01 without AIMD in training	0.9323 [0.9181]	0.1670 [0.1838]

S6. Test of torsion scans for all instances of a rotatable dihedral type

Among the MOFs from quadrants 3 and 4 (i.e., having at least one rotatable dihedral type), we used a random number generator to randomly select one MOF. Then we used a random number generator to randomly select one of this MOF's rotatable dihedral types that had more than one instance. As it happened, this process randomly selected a rotatable dihedral type with six instances in the MOF BEPMEQ. Then we quantum-mechanically-computed torsion scan curves for every instance of this rotatable dihedral type. As shown in Figure S3, all of these torsion scan curves were identical. More generally, if two instances of the same type have different signs for ϕ_{eq} , then one would have torsion scan curves for these two instances that are mirror images of each other; since this case is automatically handled by Manz's torsion model potentials, it does not require generating separate torsion scan curves for these two instances. This validates the method of generating a torsion scan curve for one instance of each rotatable dihedral type.

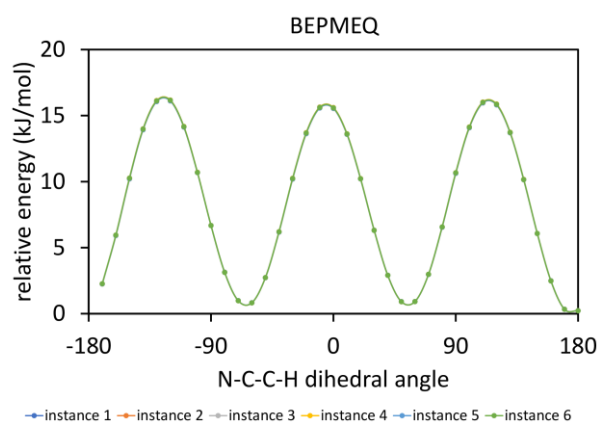


Figure S3: Torsion scan curves for six instances of the same rotatable dihedral type in the MOF BEPMEQ. All of the curves are directly on top of each other.

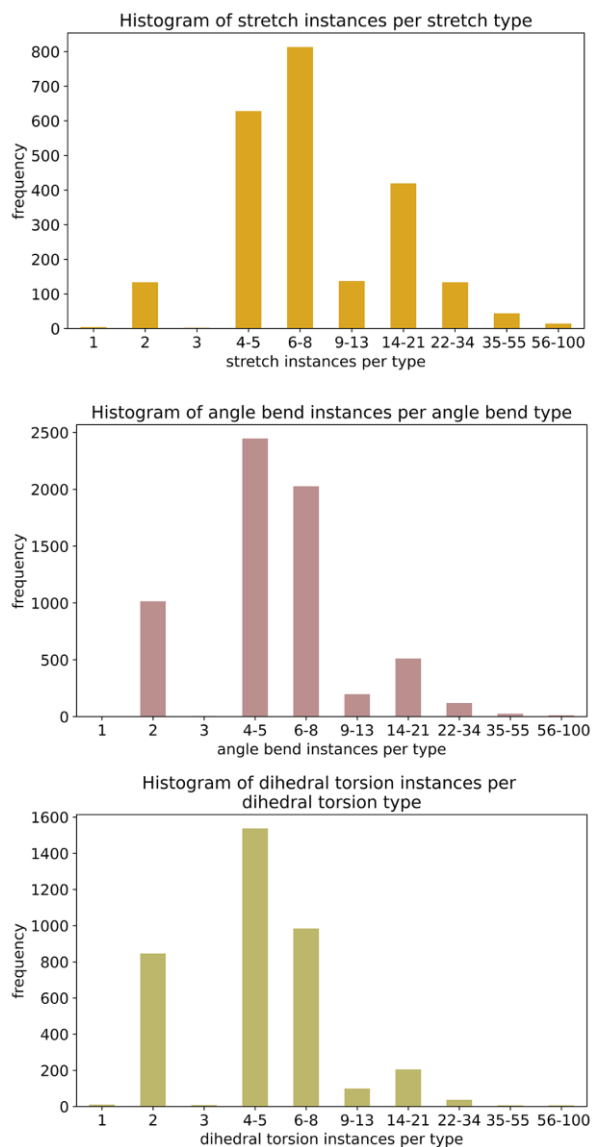
S7. Histograms showing the number of instances per type

Figure S4: Histograms showing the number of stretch instances per stretch type (top panel), the number of angle bend instances per angle bend type (middle panel), and the number of dihedral torsion instances per dihedral torsion type (bottom panel). The data includes all MOFs in quadrants 1, 2, 3, and 4.

S8. Additional LASSO results for MOFs without rotatable dihedrals

All results in this section are for calculations following dihedral pruning. In Figure S5, we present a histogram illustrating the difference between the R-squared value for $\lambda \rightarrow 0$ compared to $\lambda = \lambda_{\text{best}}$, on forces fitting within the training dataset of MOFs in quadrants 1 and 2. This R-squared difference peaked within the range 0 to 0.0001. Hence, the effect on R-squared was almost negligible. Notably, the subsequent range, spanning from 0.0001 to 0.001, gave the second-highest bar in the plot.

Figure S6 contains raincloud plots showing the distribution of R-squared and RMSE (eV/Å) values for forces in the training and validation datasets for MOFs in quadrants 1 and 2 using λ_{best} . As shown in these raincloud plots, R-squared > 0.81 for each MOF in these quadrants. Such high R-squared values indicate strong agreement between the model-predicted and quantum-mechanically-computed forces. Even when using new geometries (i.e., validation dataset) that were not part of the model's training, the model accurately reproduced the ab-initio forces.

Figure S7 shows a stacked histogram of the R-squared difference when bond-bond cross terms are included in the potential model versus when they are not. The plotted data is for MOFs in quadrants 1 and 2 after dihedral pruning and using λ_{best} . When using either average equilibrium values or individual equilibrium values, the distribution peaked at a R-squared difference of 0.015–0.02. This shows that including bond-bond cross terms slightly improves the flexibility model's accuracy; however, this improvement in accuracy comes with some increased computational costs due to the added bond-bond cross terms.

Figure S8 shows the number of force constants eliminated by the LASSO method for $\lambda \rightarrow 0$ and the additional number that were eliminated when increasing λ to λ_{best} . The results plotted in this figure included bond-bond cross terms. Results without bond-bond cross terms were already plotted in Section 8.6.1 of the main text. Some of the k's eliminated for $\lambda \rightarrow 0$ may have been eliminated by the non-negative bounds placed on stretches, bends, single-mode torsions, and linear-dihedral torsions, while others may have been eliminated due to linear dependencies between the flexibility terms. The k's eliminated when increasing λ to λ_{best} also correspond to unimportant flexibility terms that contribute negligibly to the model's accuracy. Examining Figure S8, the percentage of k's eliminated was relatively small but nonzero.

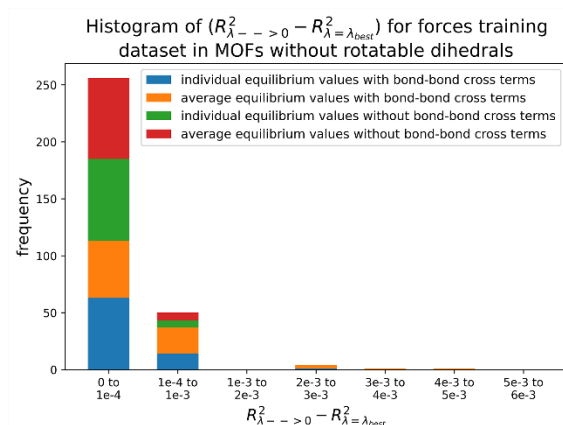


Figure S5: Histogram of difference between R-squared for $\lambda \rightarrow 0$ and R-squared for $\lambda = \lambda_{\text{best}}$ for forces training dataset in MOFs belonging to quadrants 1 and 2.

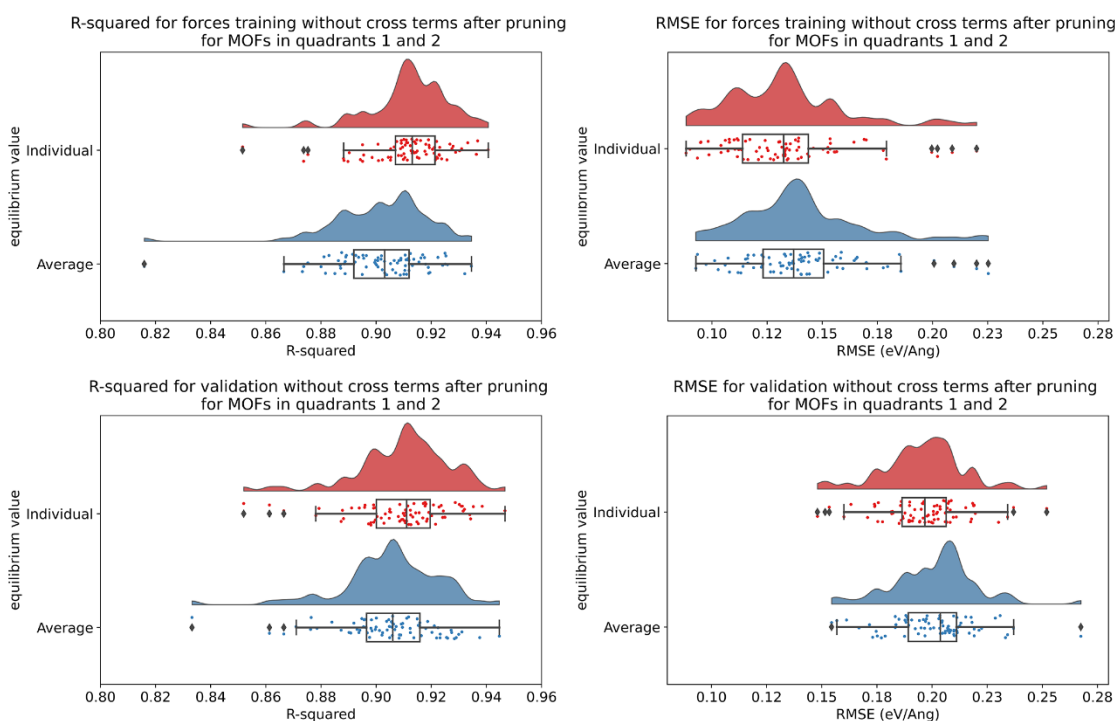


Figure S6: Raincloud plots of R-squared (left panels) and RMSE (right panels) for forces training datasets (top panels) and validation datasets (bottom panels) for MOFs in quadrants 1 and 2 without cross terms and after pruning. The red distributions represent the values for individual equilibrium values, while the blue distributions represent the values for average equilibrium values.

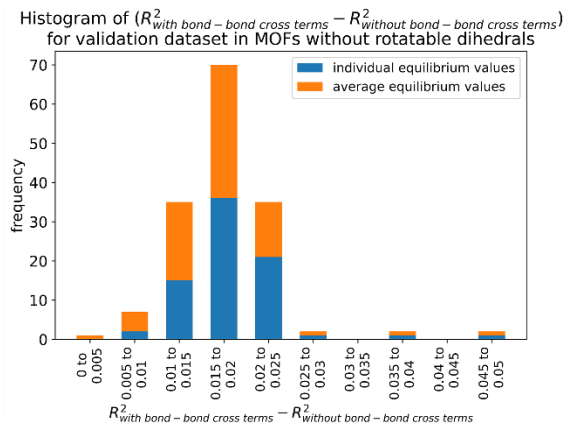


Figure S7: Histogram of difference between R-squared with bond-bond cross terms and R-squared without bond-bond cross terms for MOFs without rotatable dihedrals.

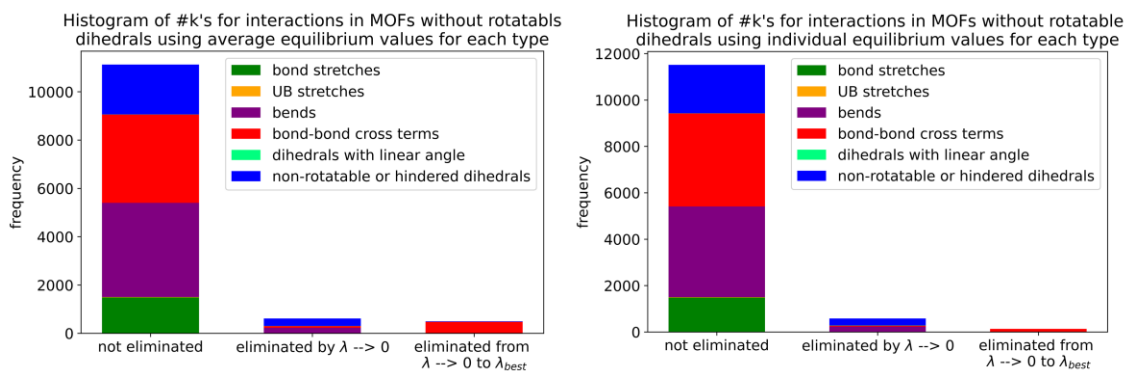


Figure S8: Histograms of force constants eliminated by the bounds or regularization constraints in the LASSO method applied to MOFs in quadrants 1 and 2 for flexibility models including bond-bond cross terms. Data shown in the left (right) panel used average (individual) equilibrium values.

S.9 Images of MOF crystal structures

In the below images, the gold lines indicate the unit cell boundary within the displayed $2 \times 2 \times 2$ supercell. The three panels display views along each of the three lattice vectors.

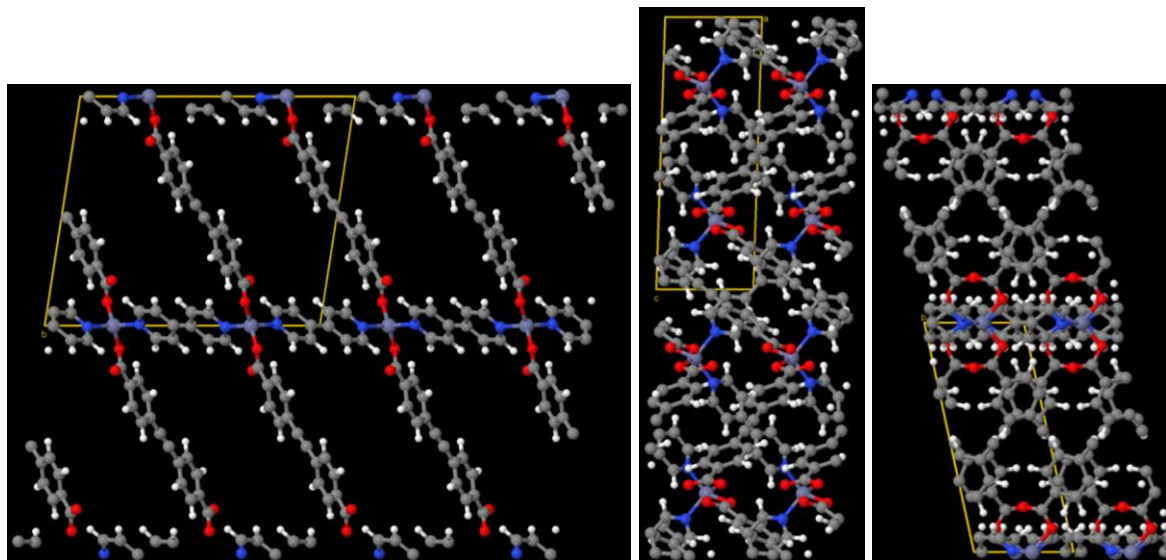


Figure S9: AFITEP contains the chemical elements Zn, H, C, N, and O.

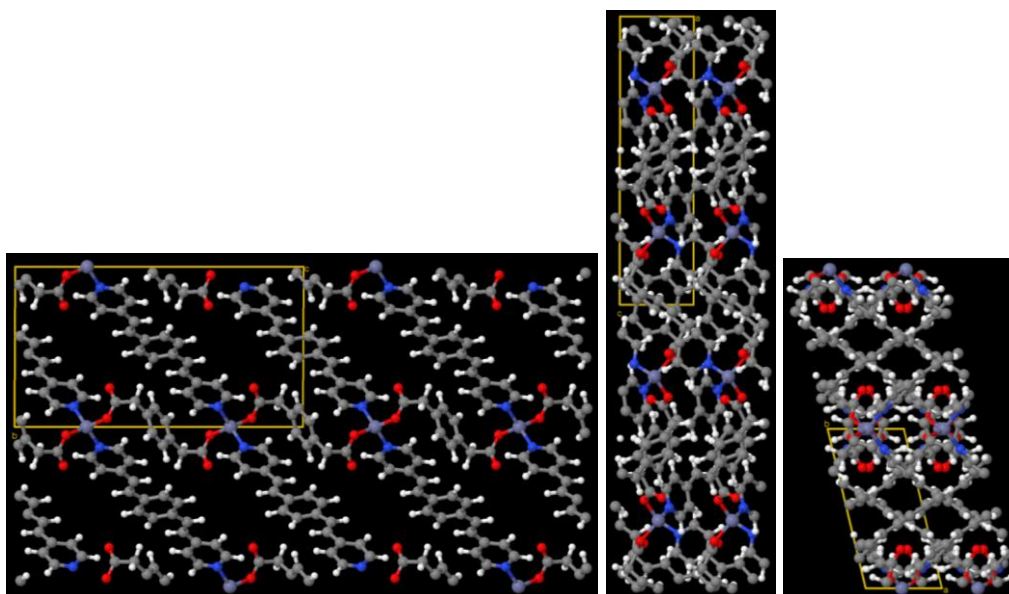


Figure S10: AMOYOR contains the chemical elements Zn, H, C, N, and O.

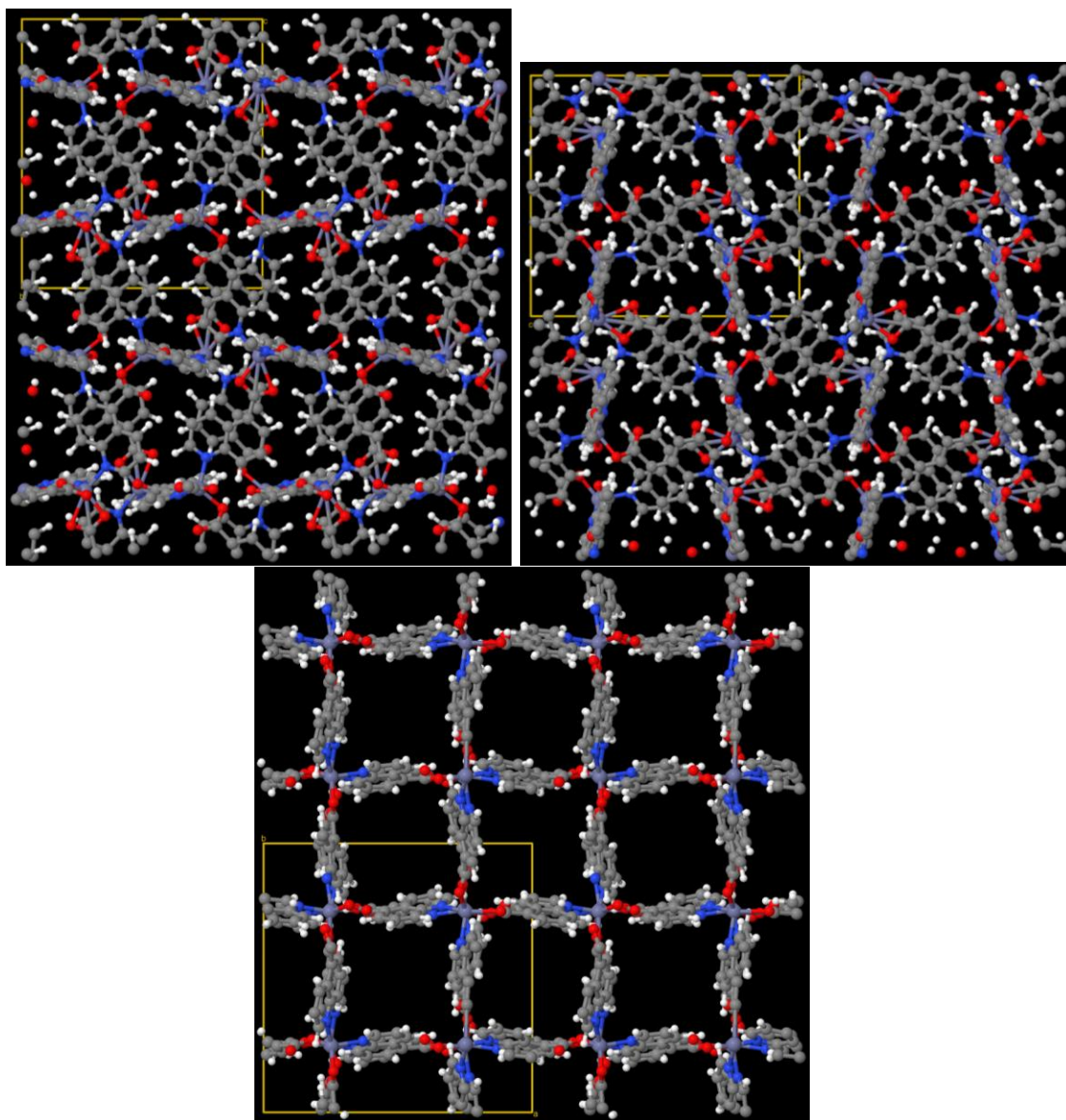


Figure S11: ARAKAH contains the chemical elements O, C, H, Zn, and N.

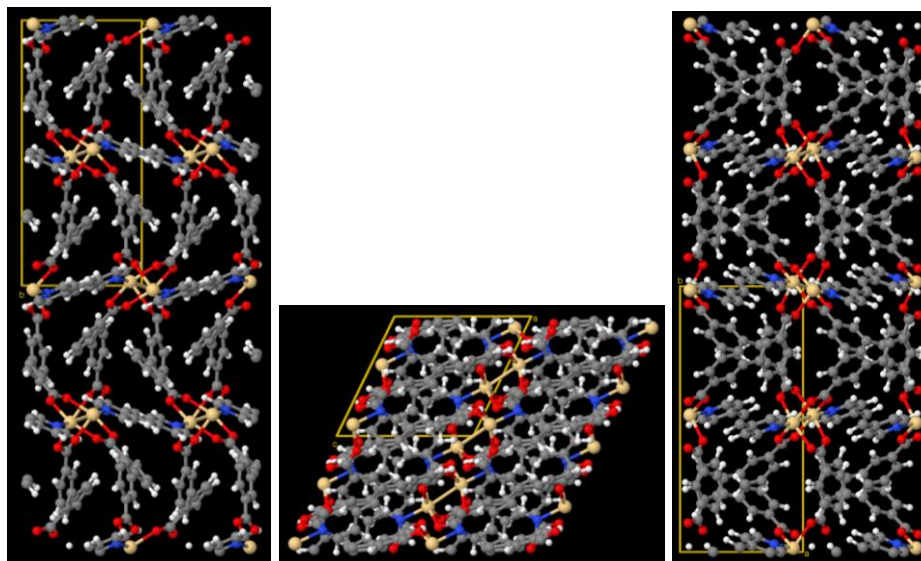


Figure S12: ARIBOS contains the chemical elements Cd, O, C, H, and N.

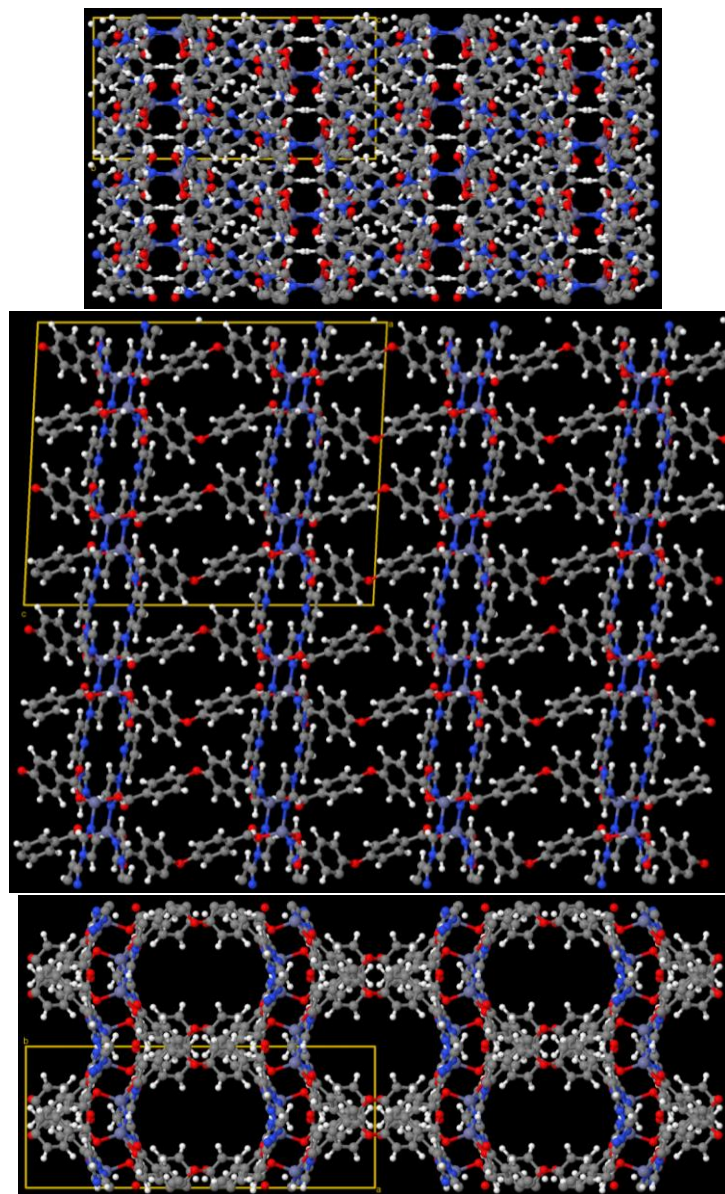


Figure S13: ATOB IW contains the chemical elements C, H, O, Zn, and N.

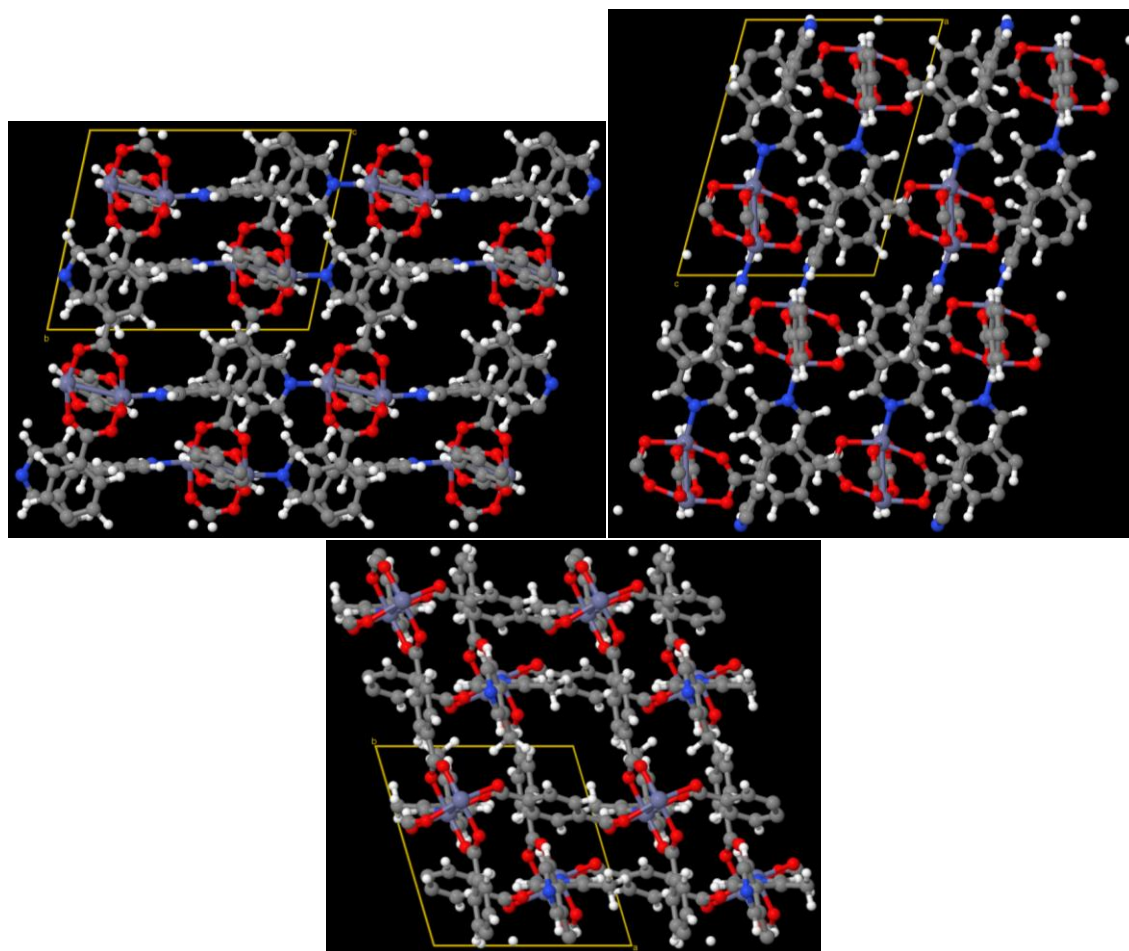


Figure S14: ATOXEN contains the chemical elements C, H, O, Zn, and N.

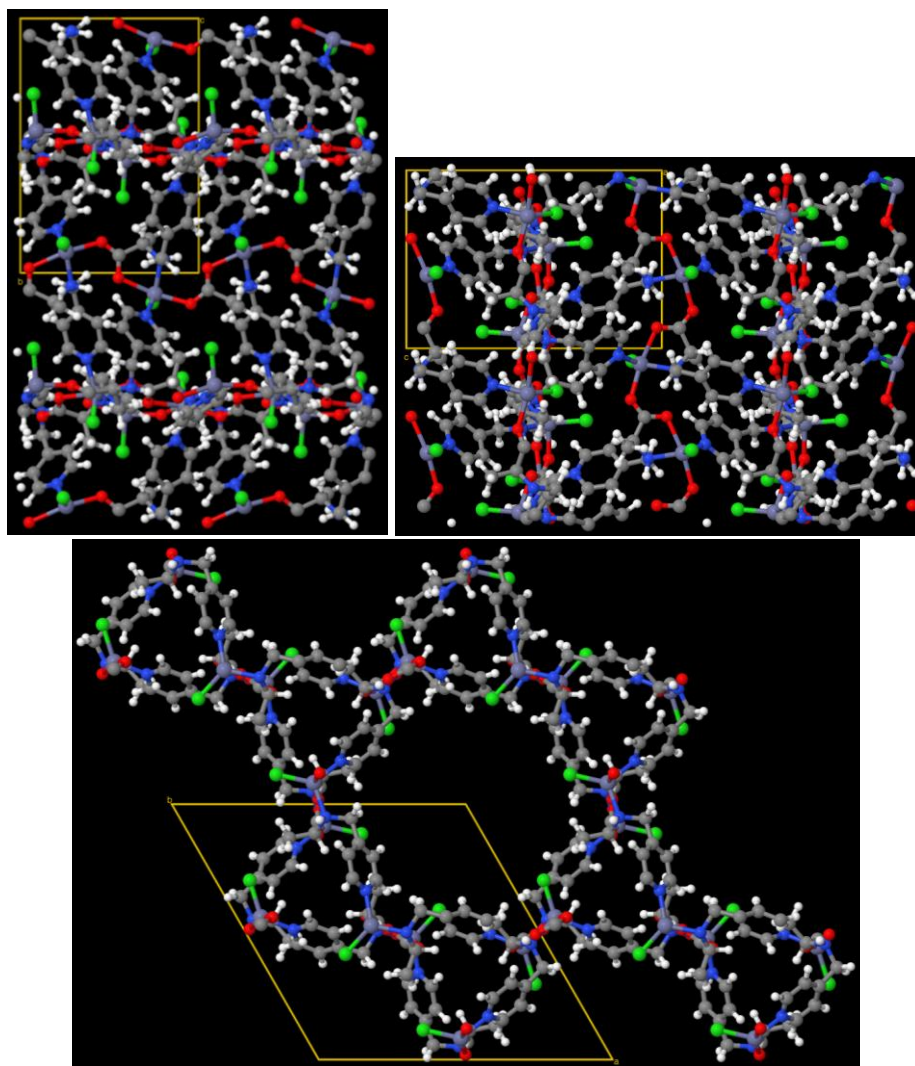


Figure S15: BEPMEQ contains the chemical elements C, H, N, Zn, O, and Cl.

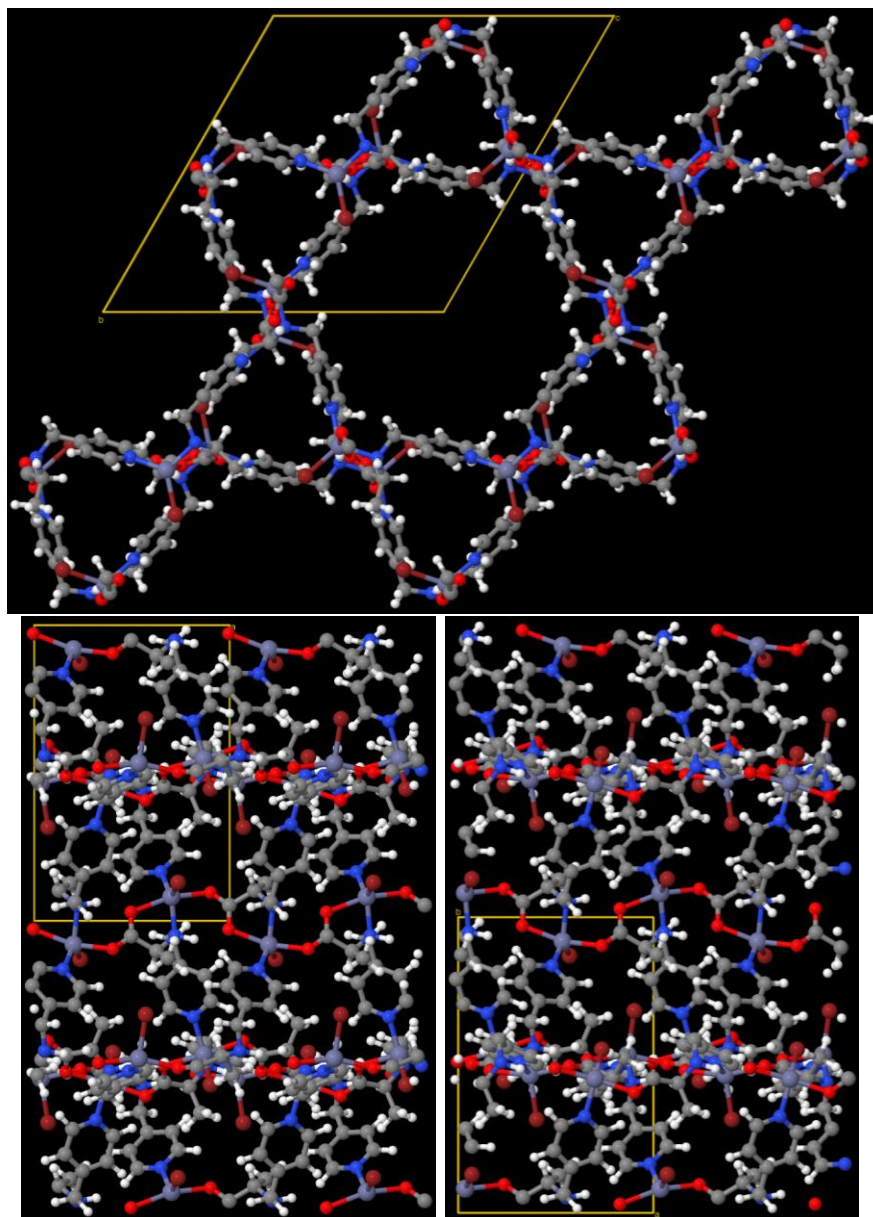


Figure S16: BEPNUH contains the chemical elements Zn, H, C, Br, N, and O.

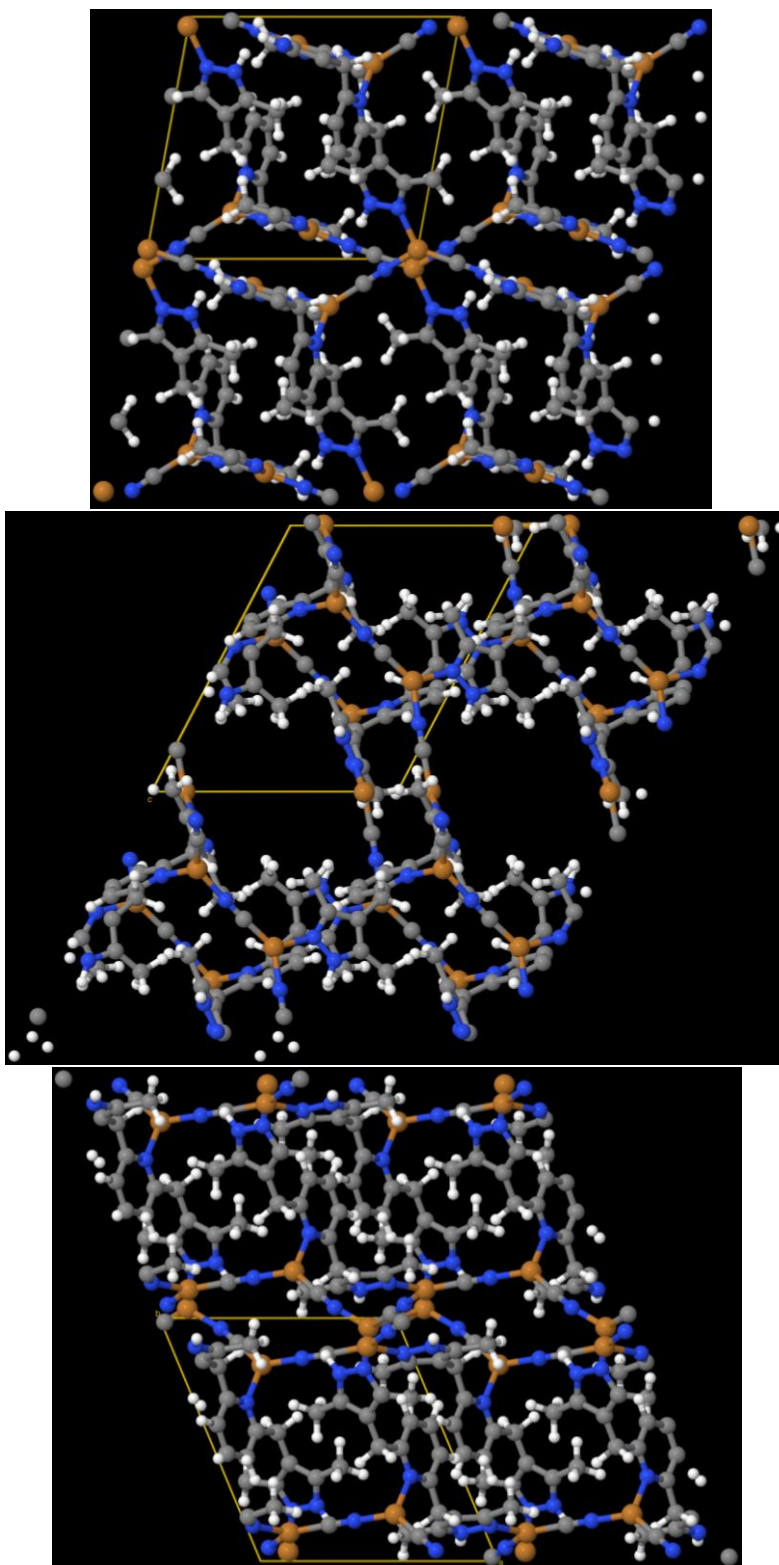


Figure S17: BEPVID contains the chemical elements Cu, H, C, and N.

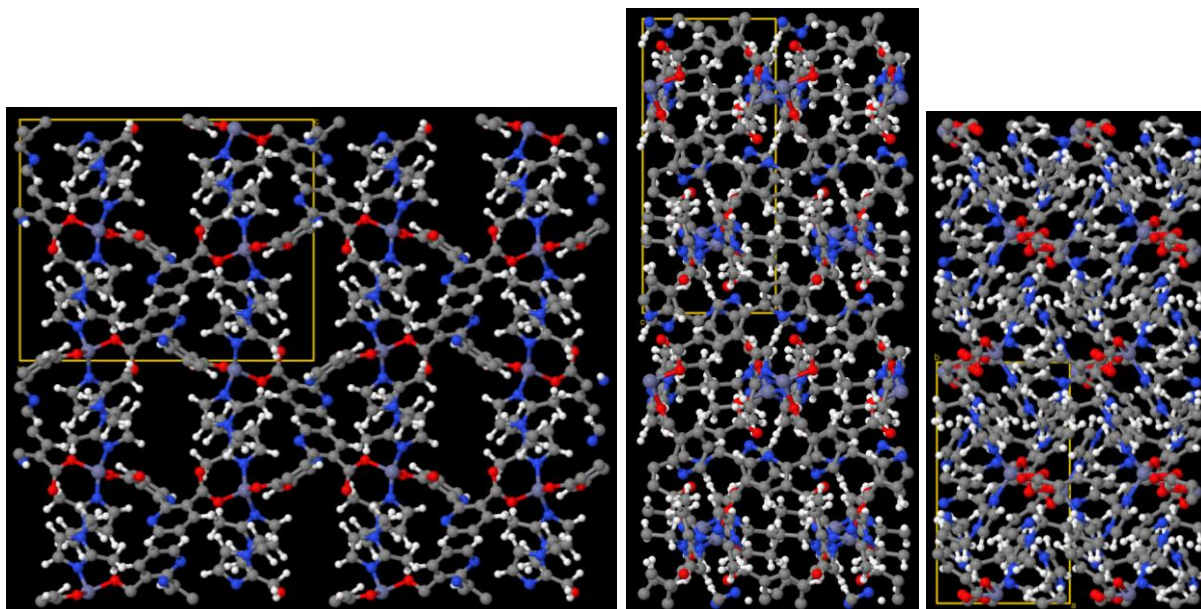


Figure S18: BIBUL contains the chemical elements Zn, H, C, N, and O.

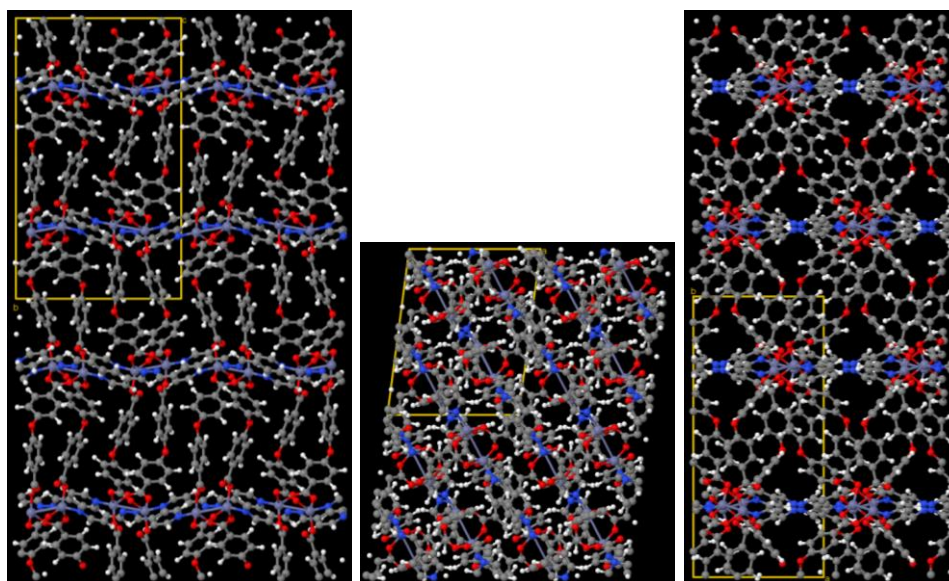


Figure S19: BOMCOX contains the chemical elements Zn, O, N, C, and H.

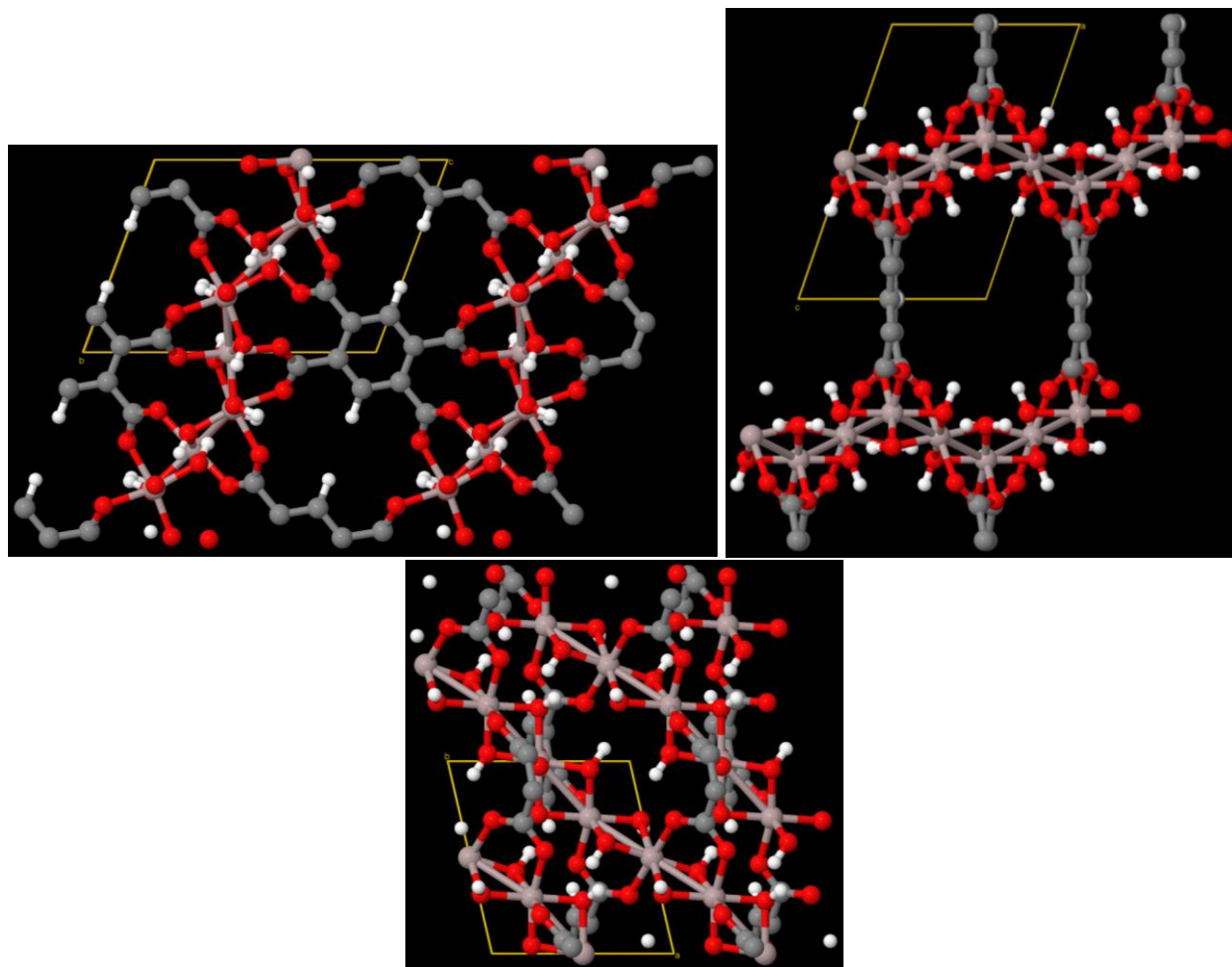


Figure S20: BUSQIQ contains the chemical elements Al, H, C, and O.

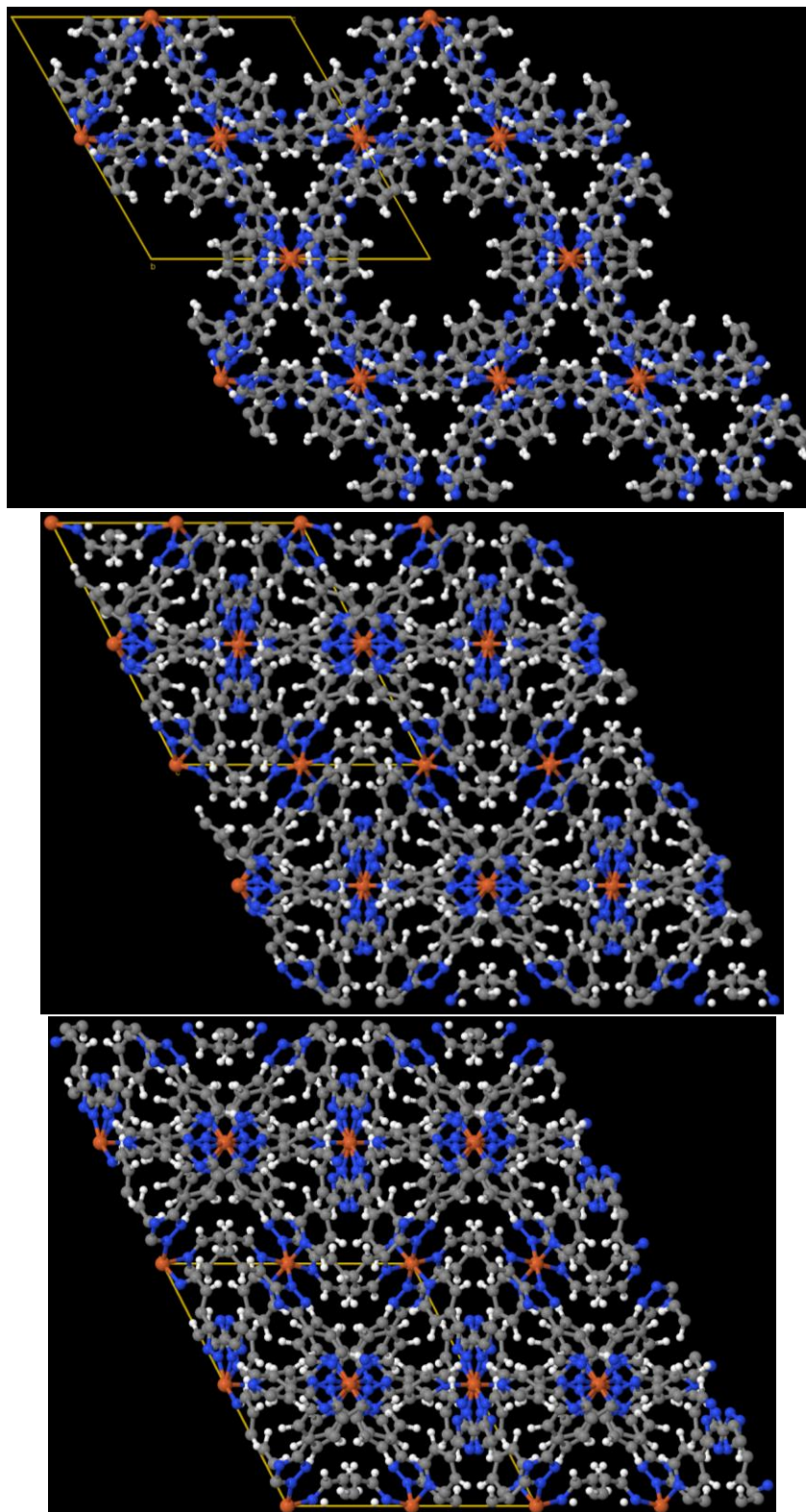


Figure S21: BUWOF03 contains the chemical elements Fe, H, C, and N.

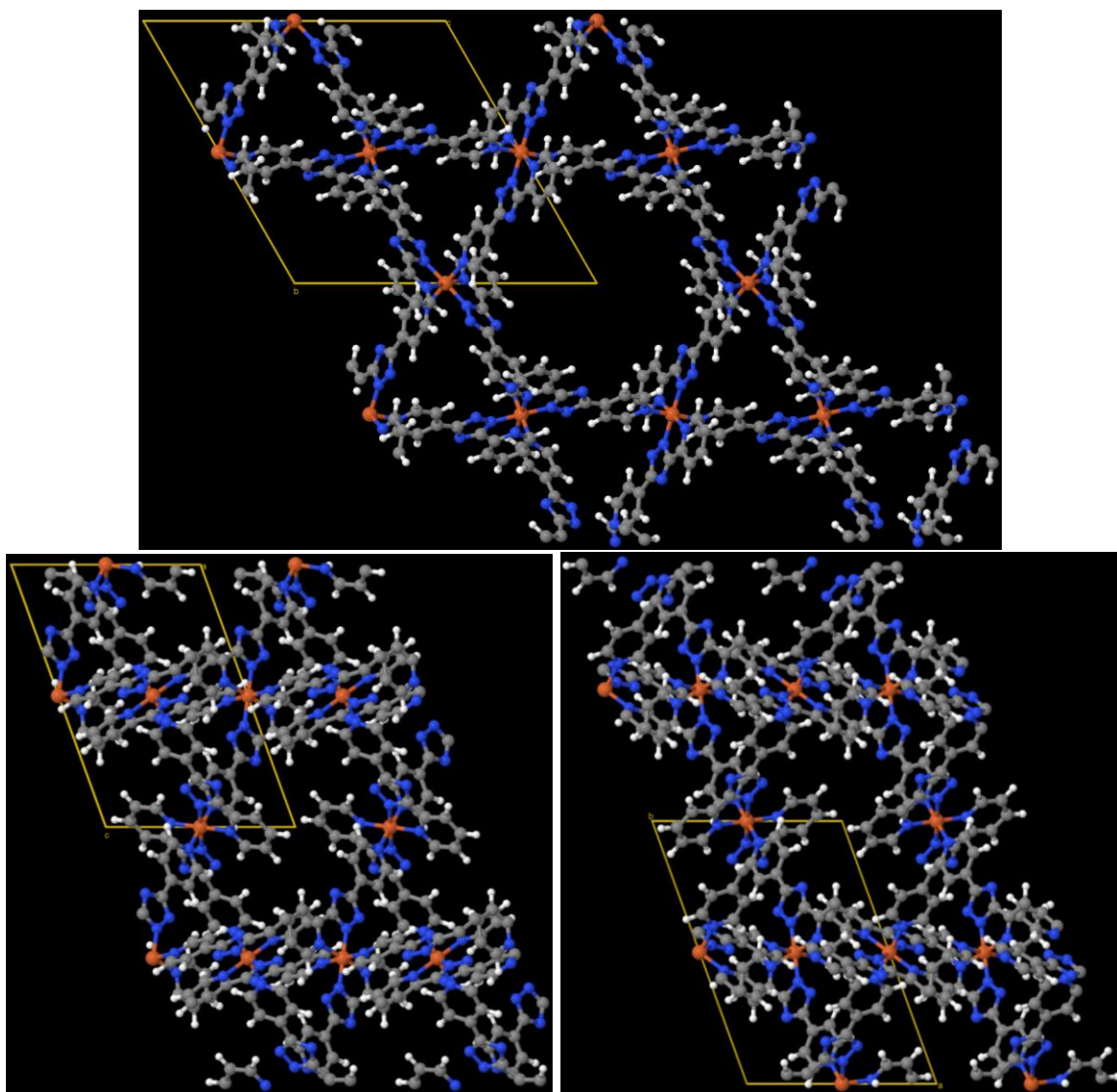


Figure S22: BUZYIB contains the chemical elements Fe, H, C, and N.

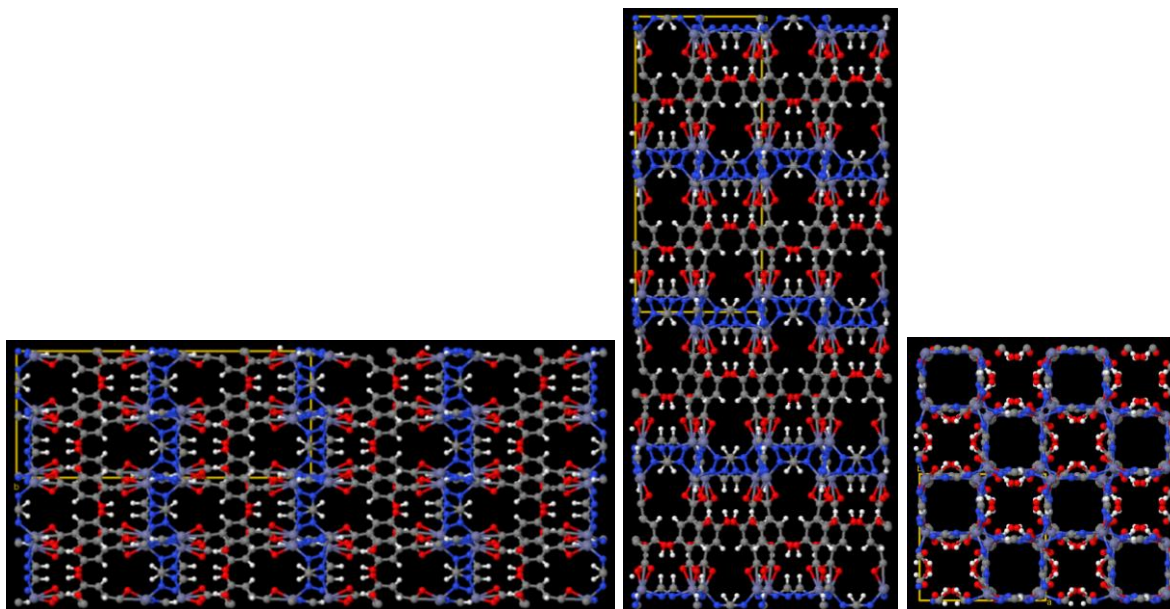


Figure S23: BUYDEG contains the chemical elements C, H, N, O, and Zn.

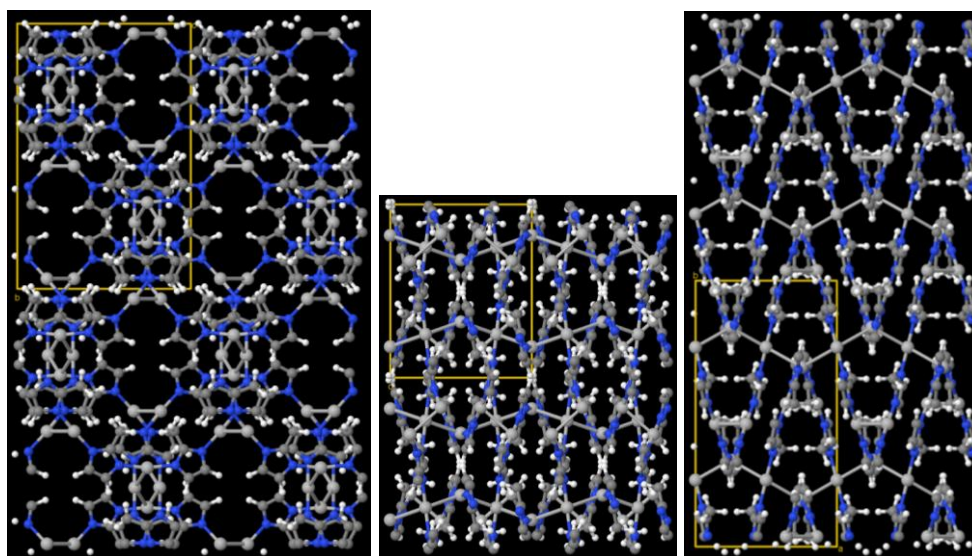


Figure S24: CEGDUO contains the chemical elements Ag, H, C, and N.

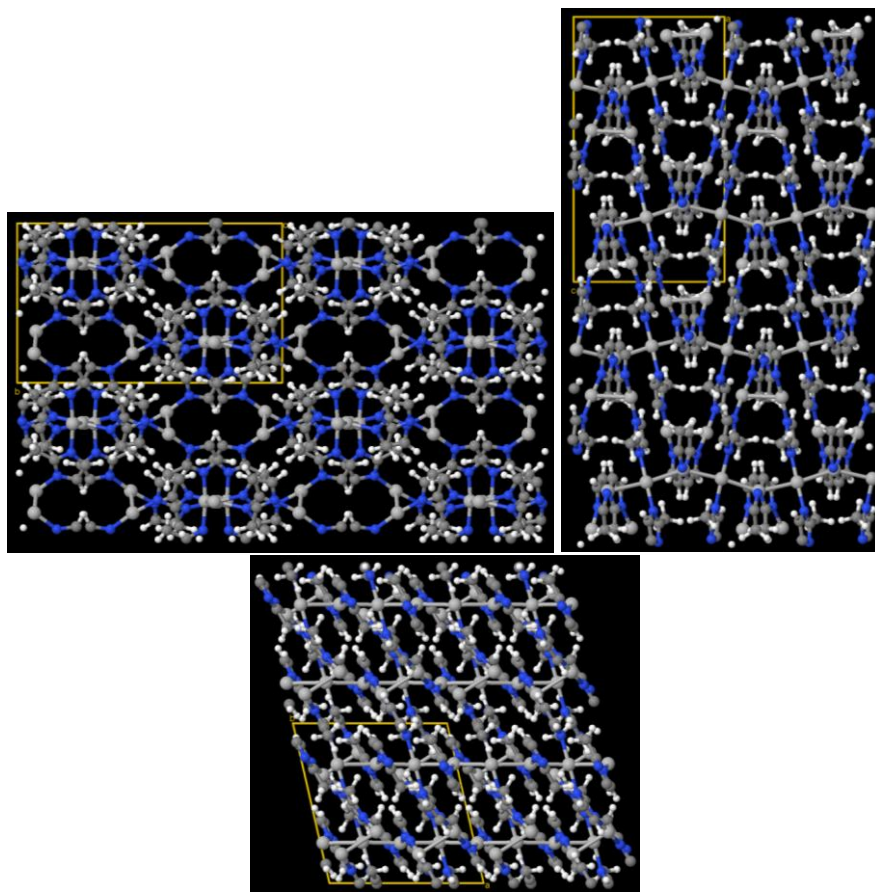


Figure S25: CEGFAW contains the chemical elements Ag, H, C, and N.

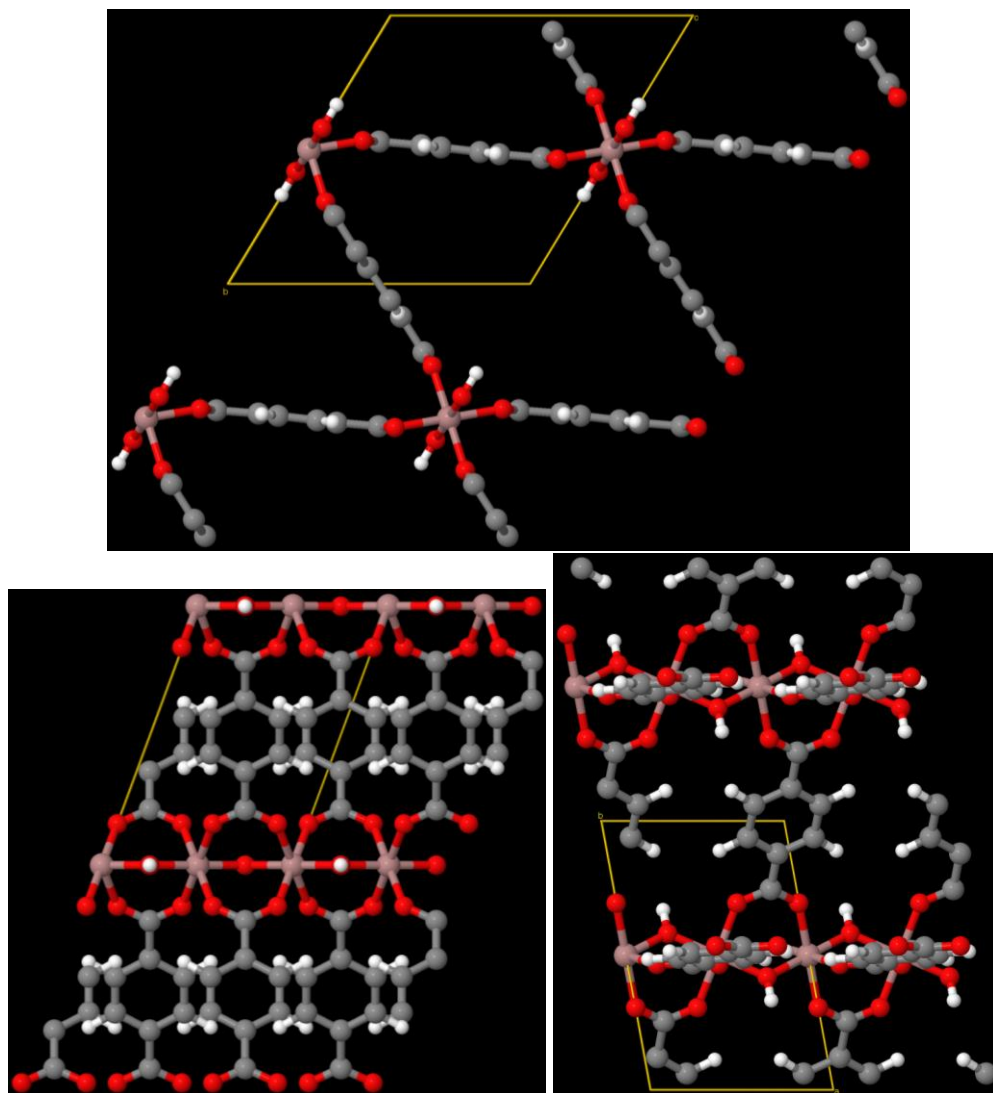


Figure S26: COMDOY contains the chemical elements Ga, H, C, and O.

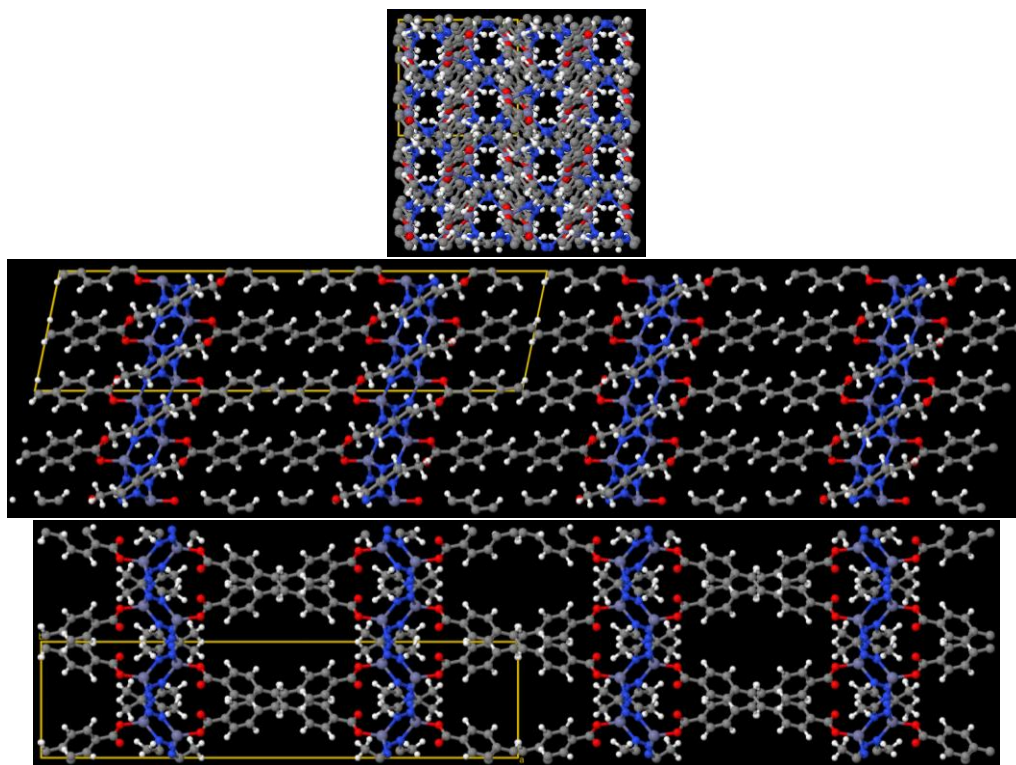


Figure S27: CUQRIR contains the chemical elements Zn, N, O, C, and H.

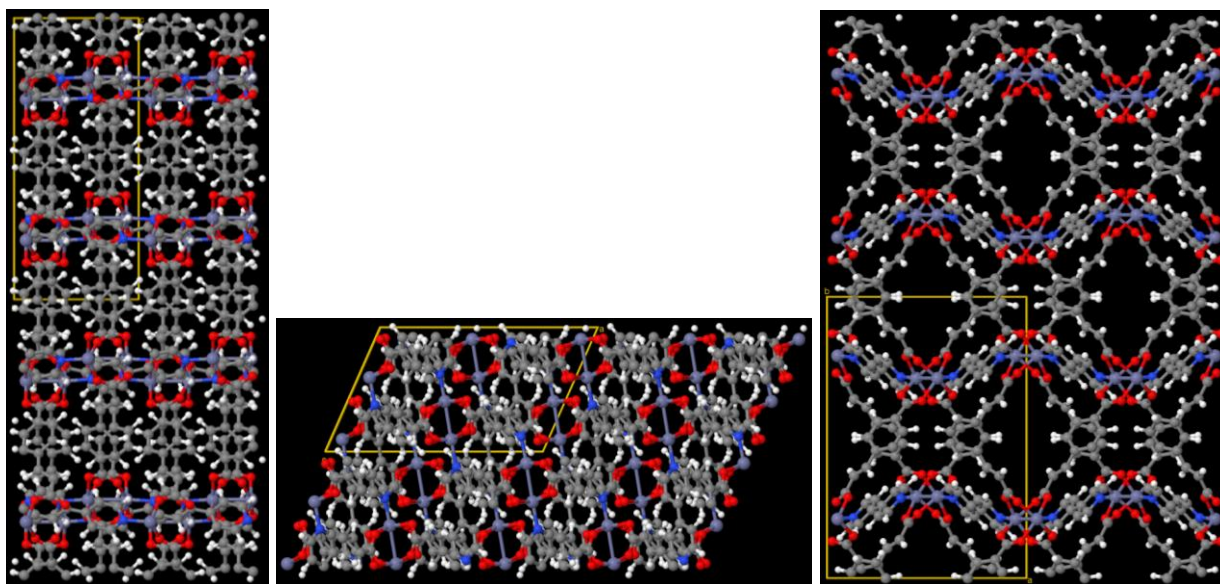


Figure S28: DIXJIF contains the chemical elements Zn, O, C, H, and N.

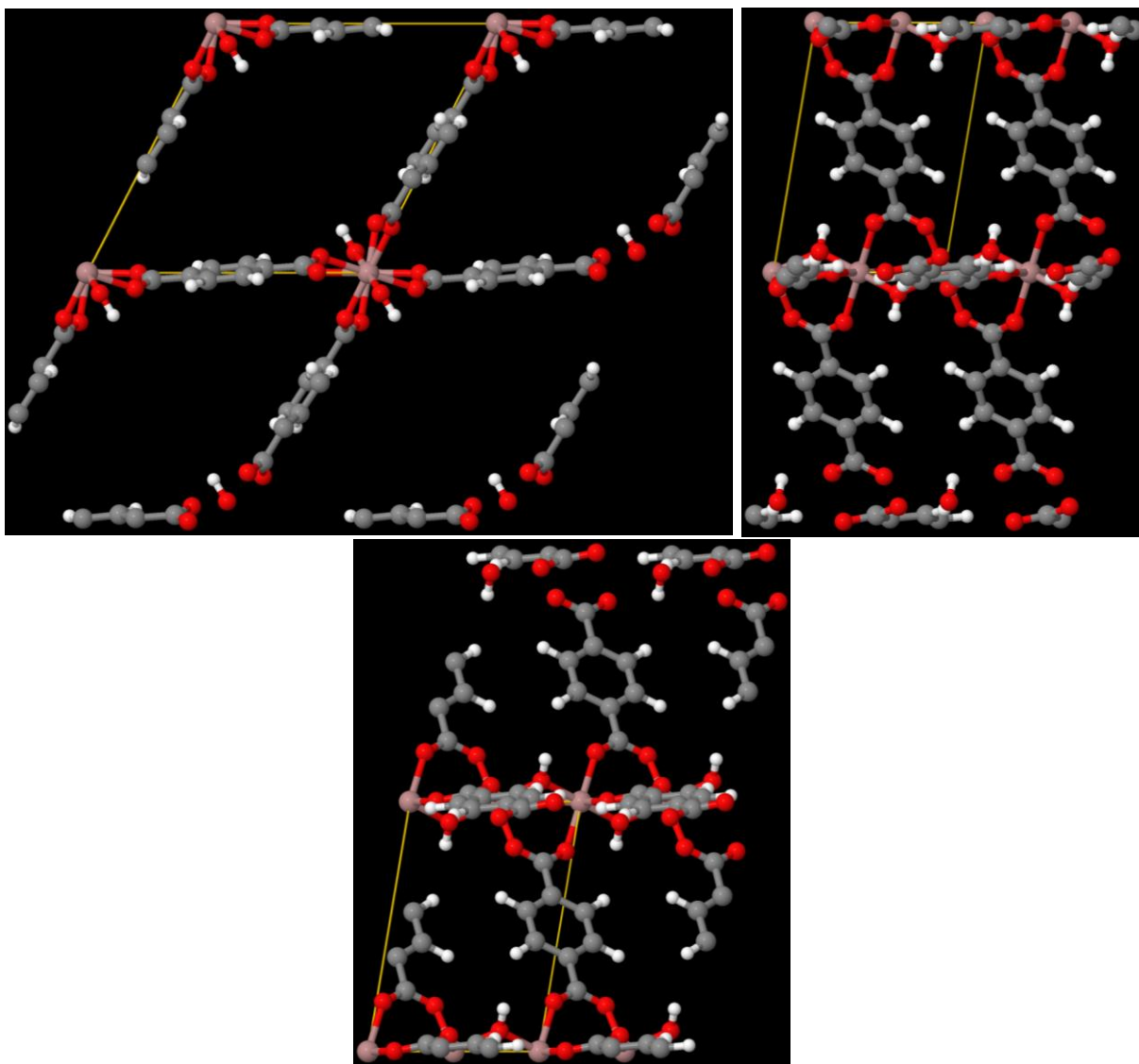


Figure S29: DONNEA contains the chemical elements Ga, H, C, and O.

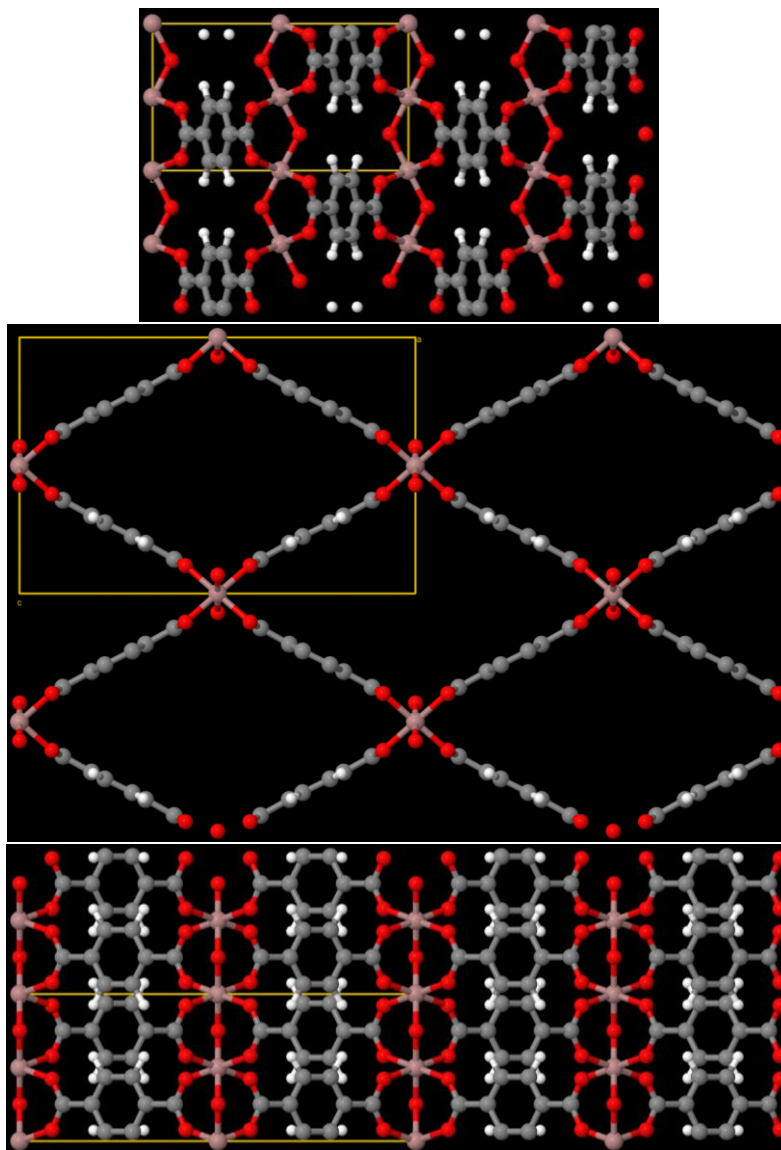


Figure S30: DONNIE contains the chemical elements O, C, H, and Ga.

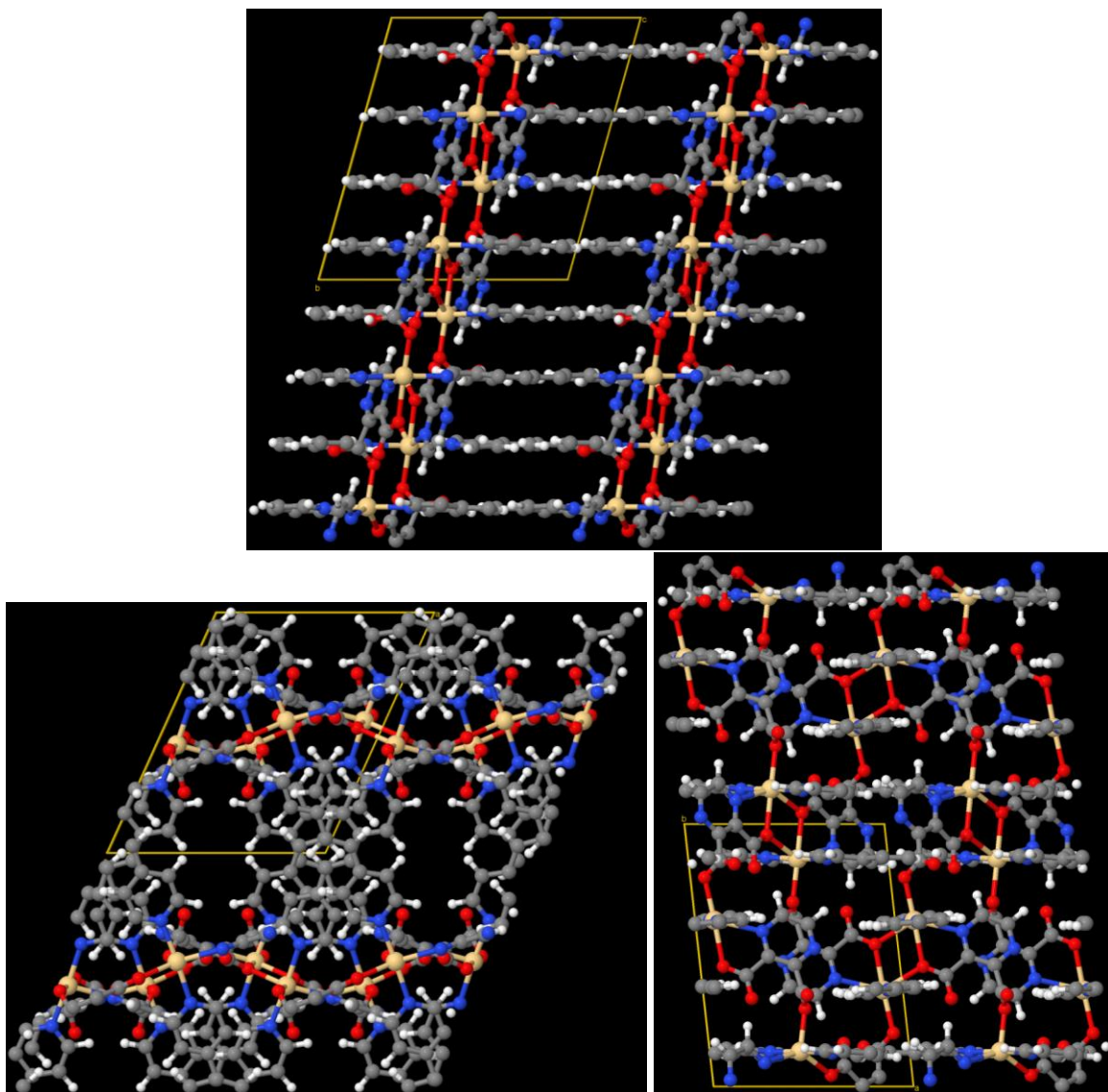


Figure S31: EBEMII contains the chemical elements Cd, H, C, N, and O.

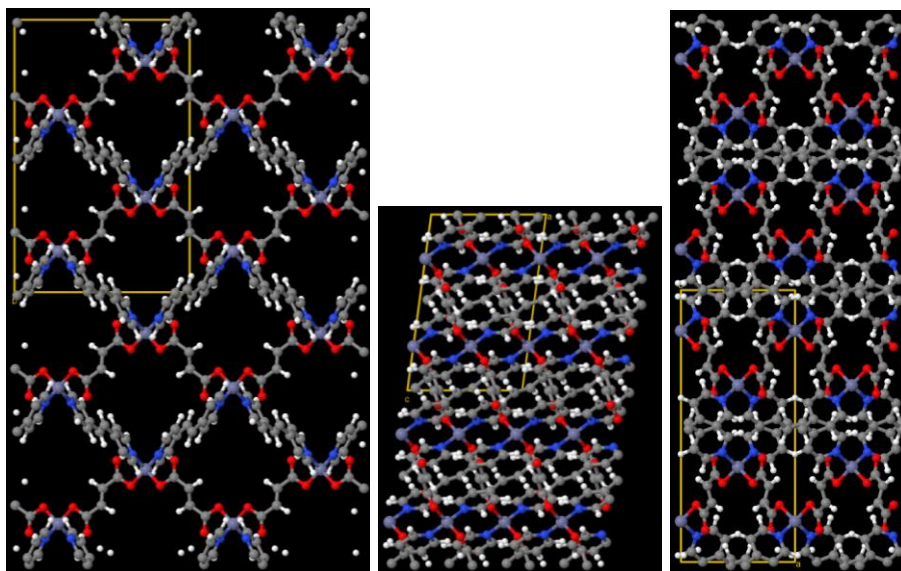


Figure S32: EBOBIJ contains the chemical elements O, N, C, H, and Zn.

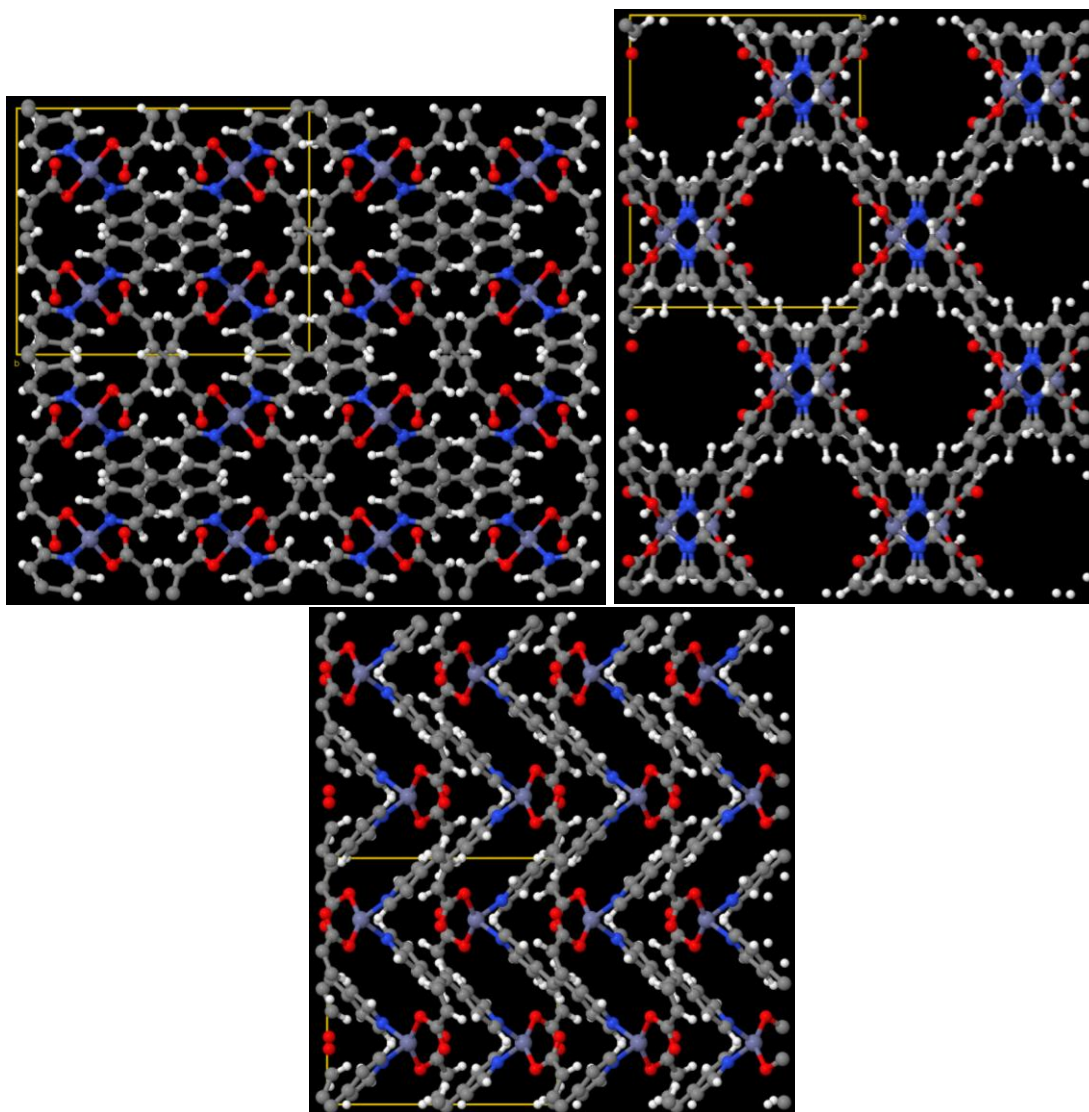


Figure S33: EBOBUV contains the chemical elements C, H, O, Zn, and N.

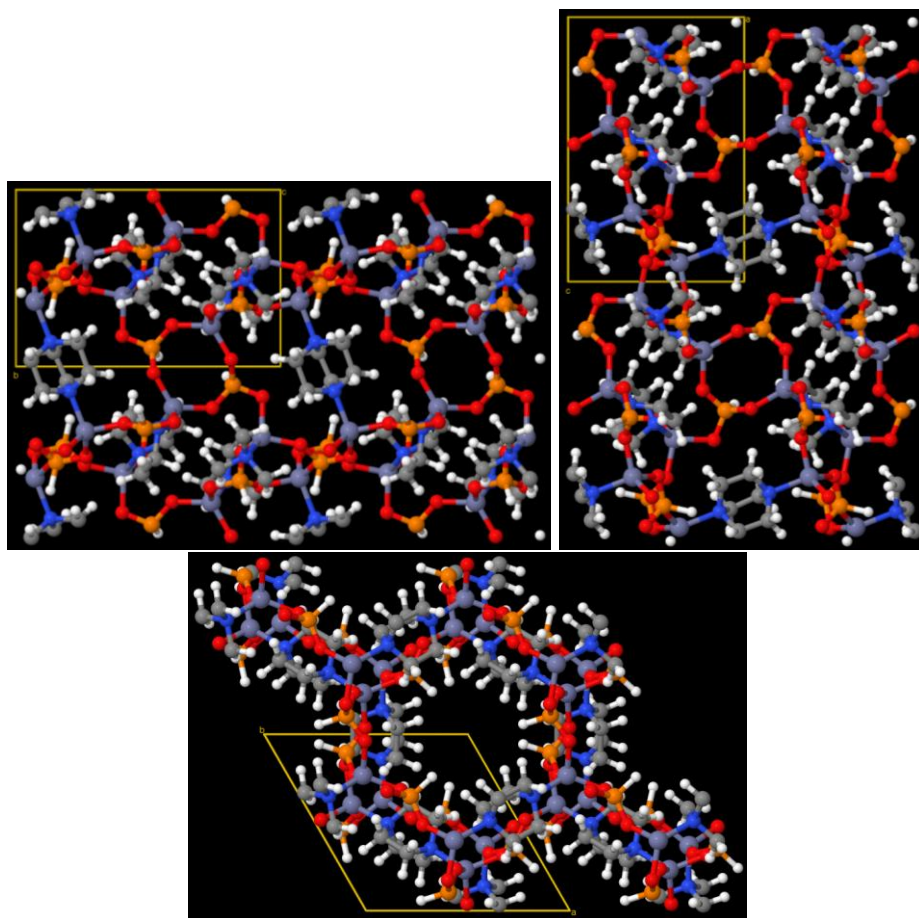


Figure S34: EKEHUZ contains the chemical elements Zn, P, H, O, N, and C.

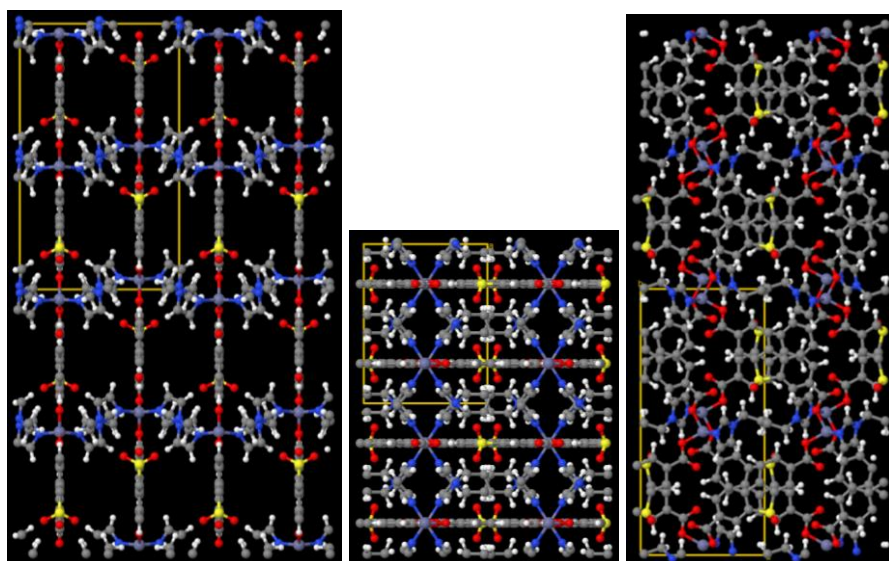


Figure S35: EMIVAY contains the chemical elements Zn, H, N, C, S, and O.

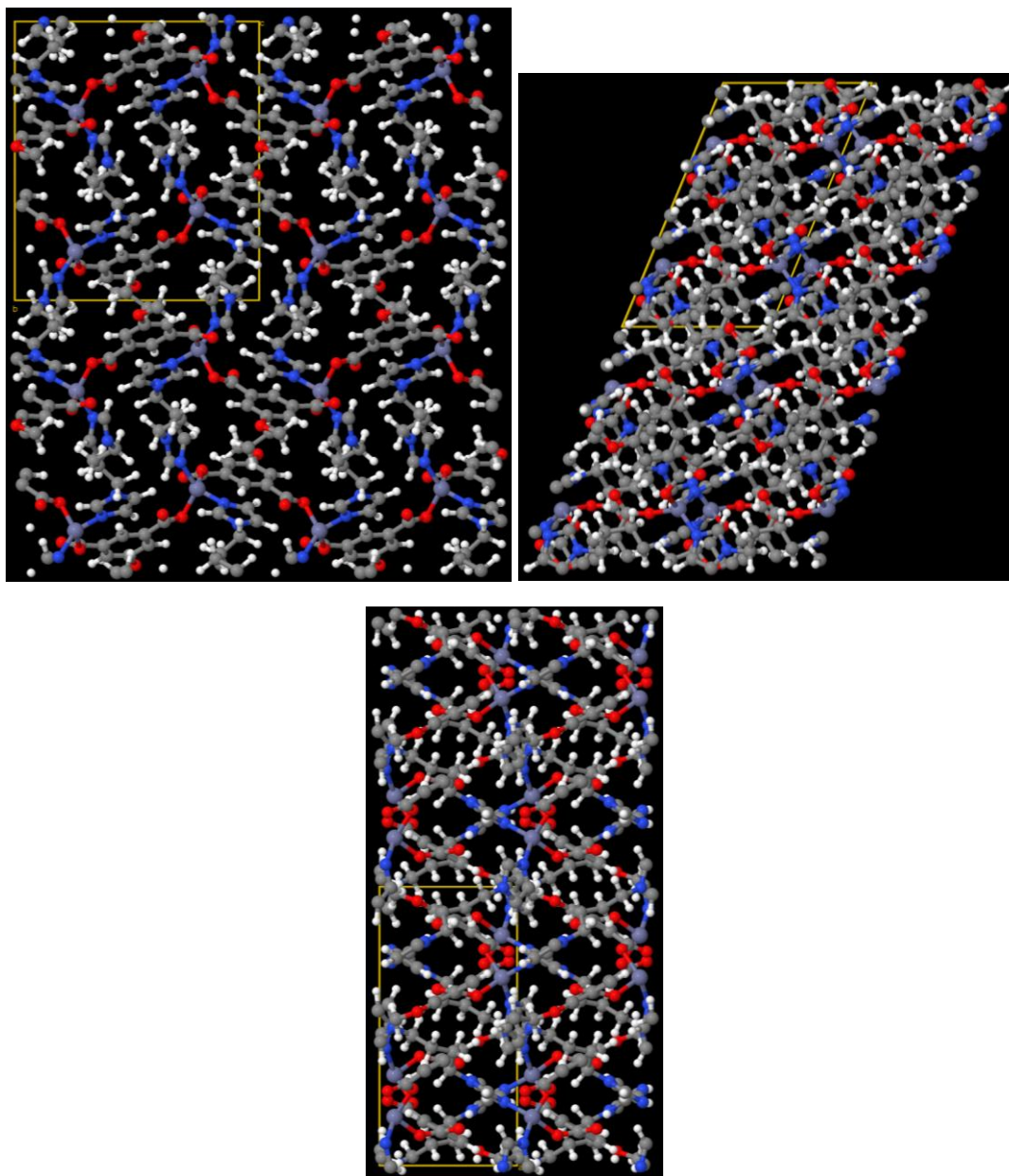


Figure S36: EMOFIX contains the chemical elements Zn, O, N, C, and H.

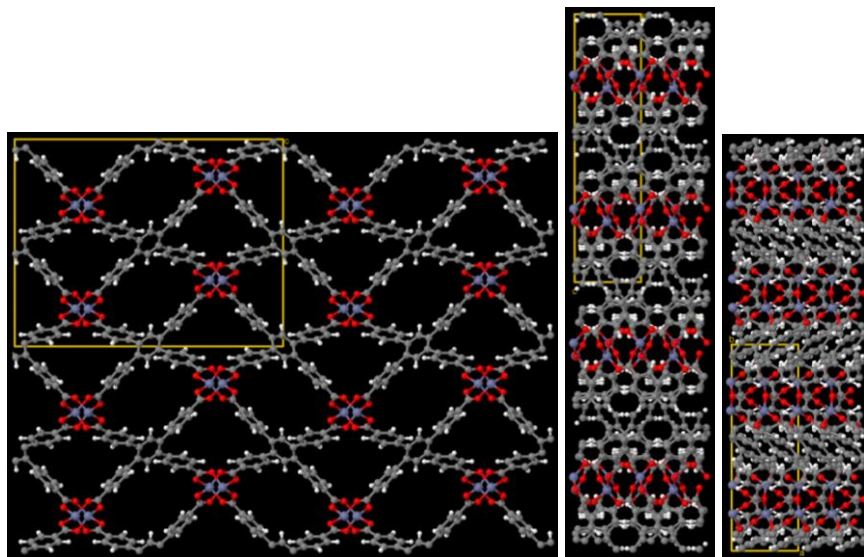


Figure S37: ESEDAJ contains the chemical elements C, H, O, and Zn.

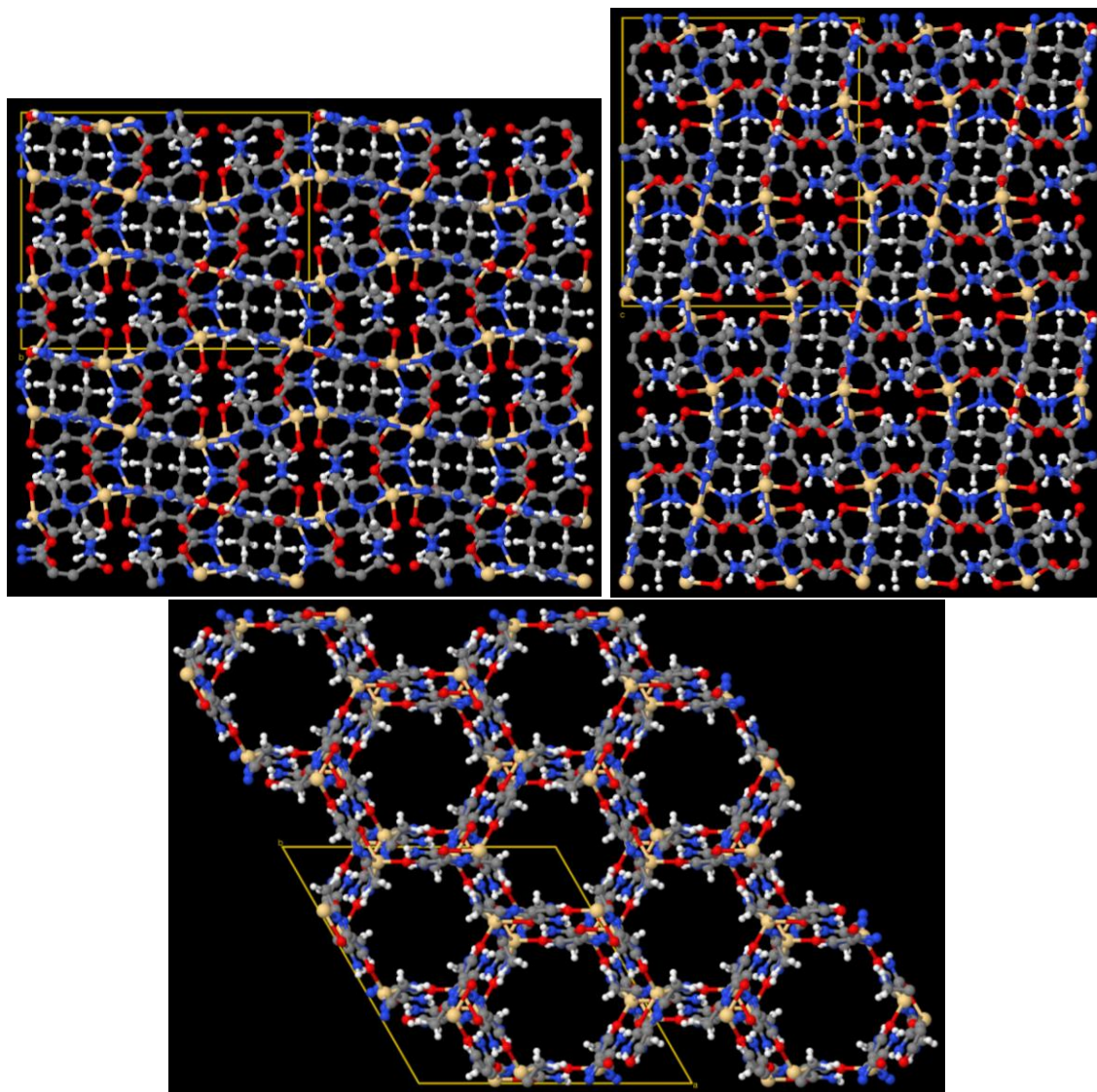


Figure S38: ESIFIX contains the chemical elements C, H, Cd, N, and O.

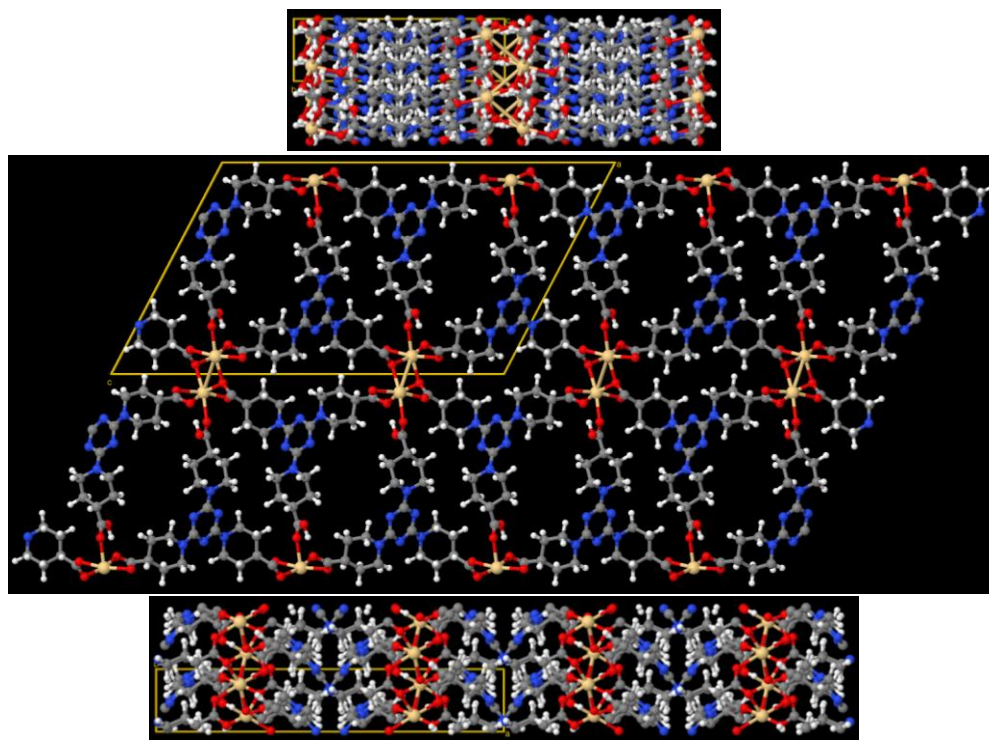


Figure S39: EWUGEK contains the chemical elements C, H, N, O, and Cd.

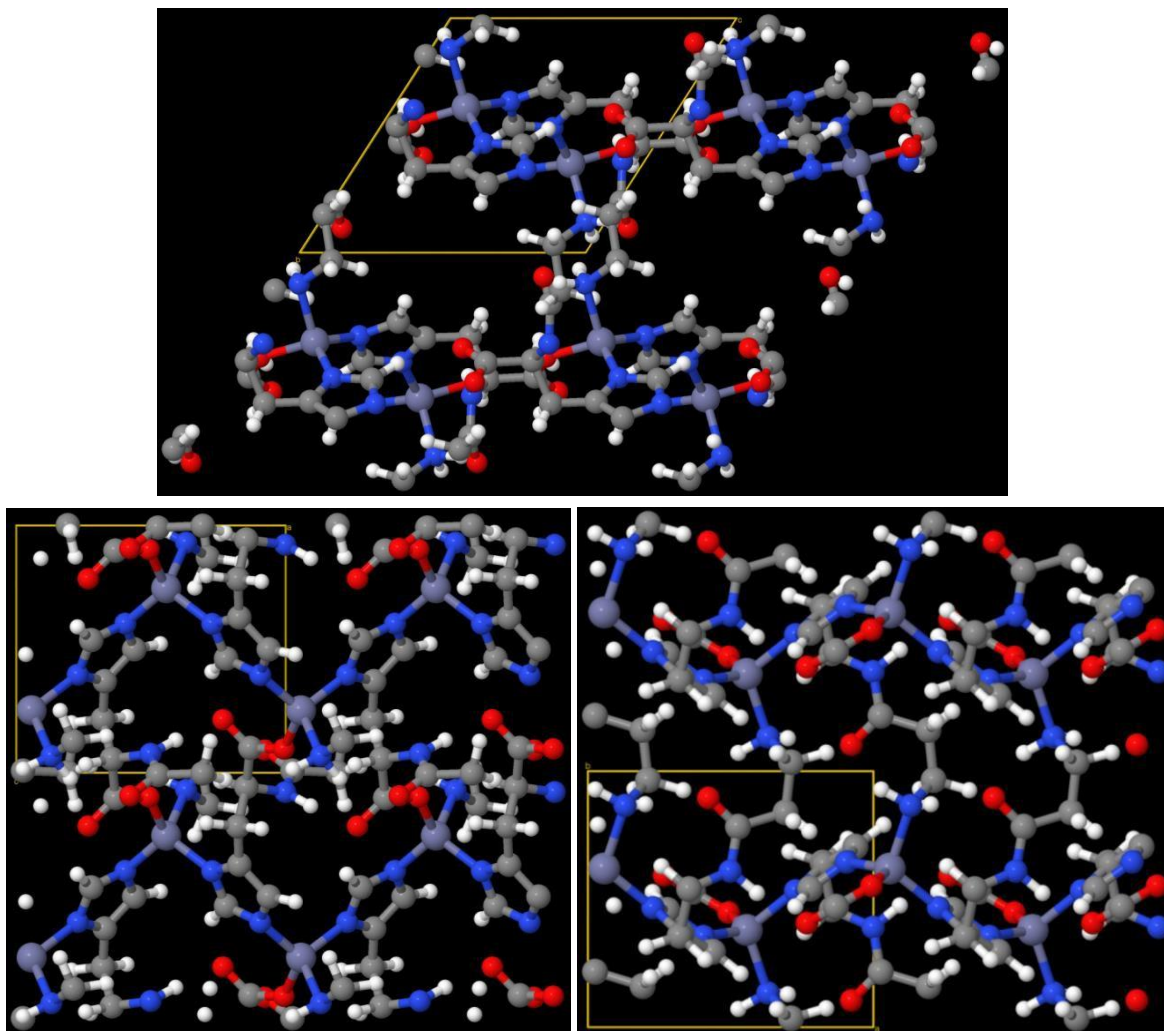


Figure S40: FIPWOS contains the chemical elements Zn, H, C, N, and O.

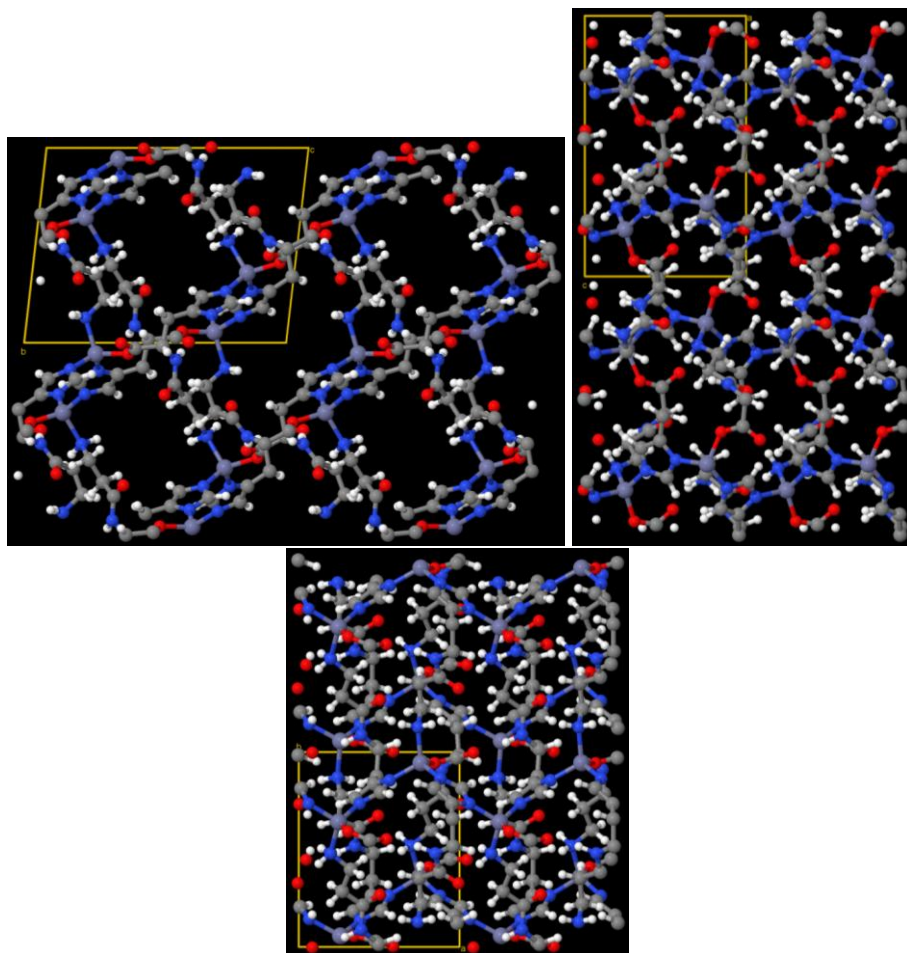


Figure S41: FIPXEJ contains the chemical elements Zn, H, C, N, and O.

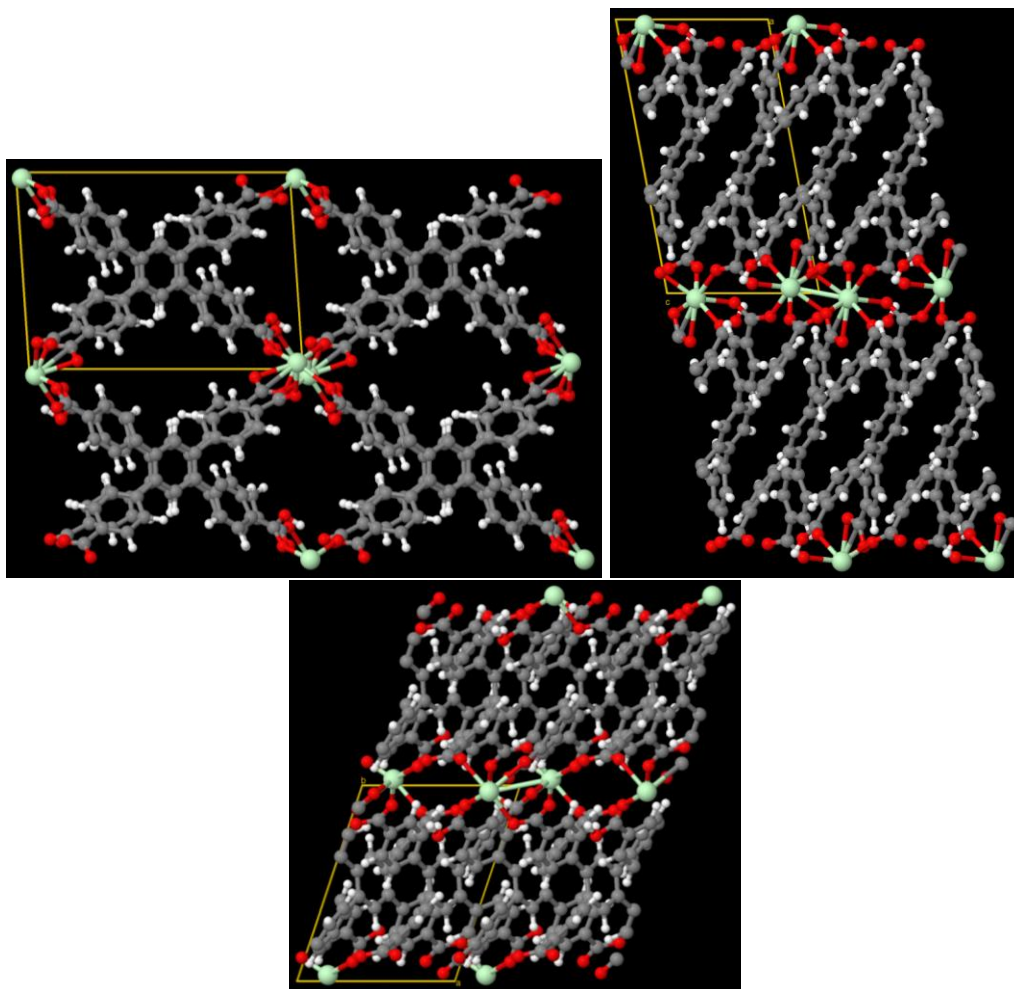


Figure S42: FOCYUT contains the chemical elements Nd, O, H, and C.

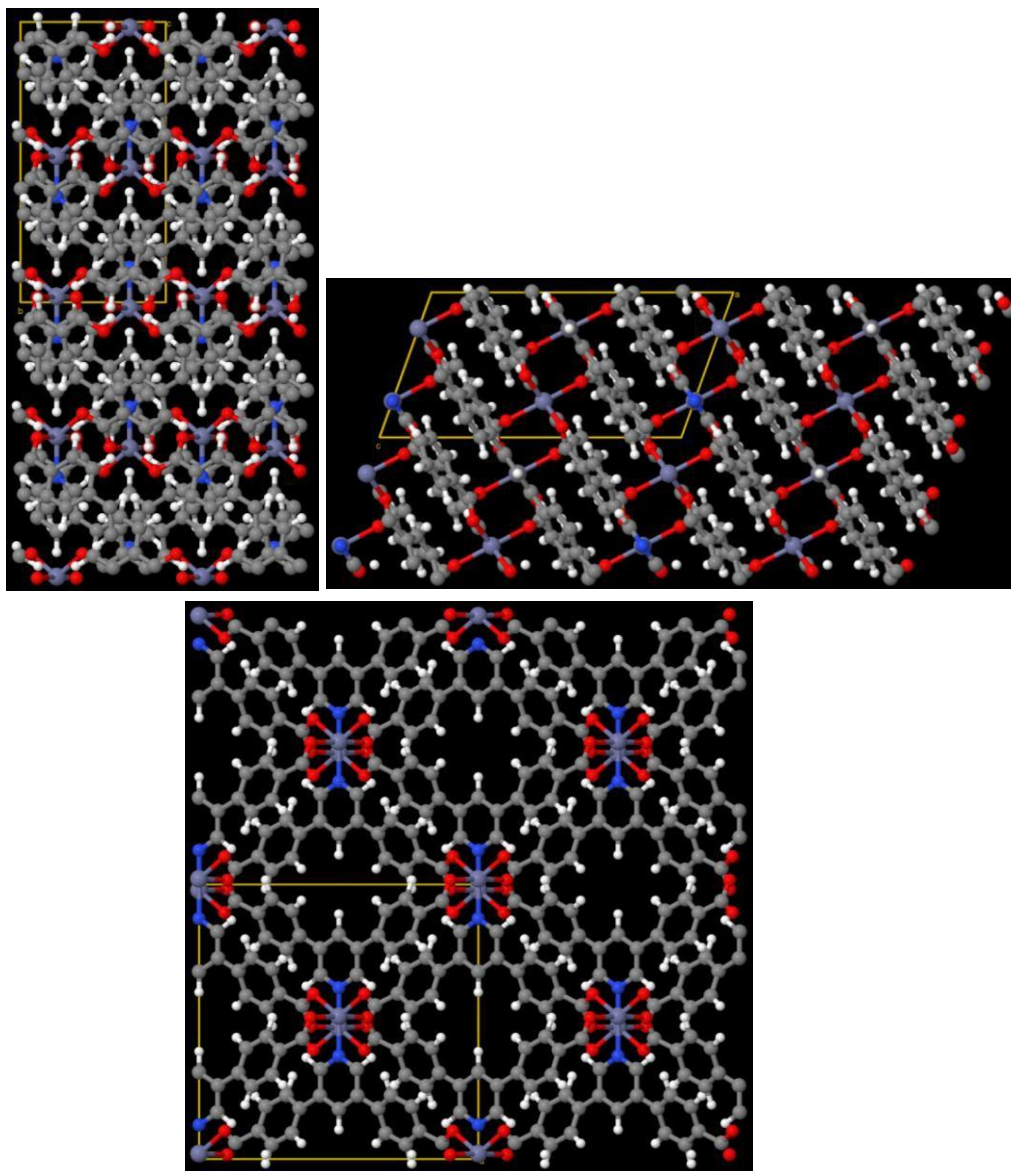


Figure S43: FUBNOH contains the chemical elements C, H, O, Zn, and N.

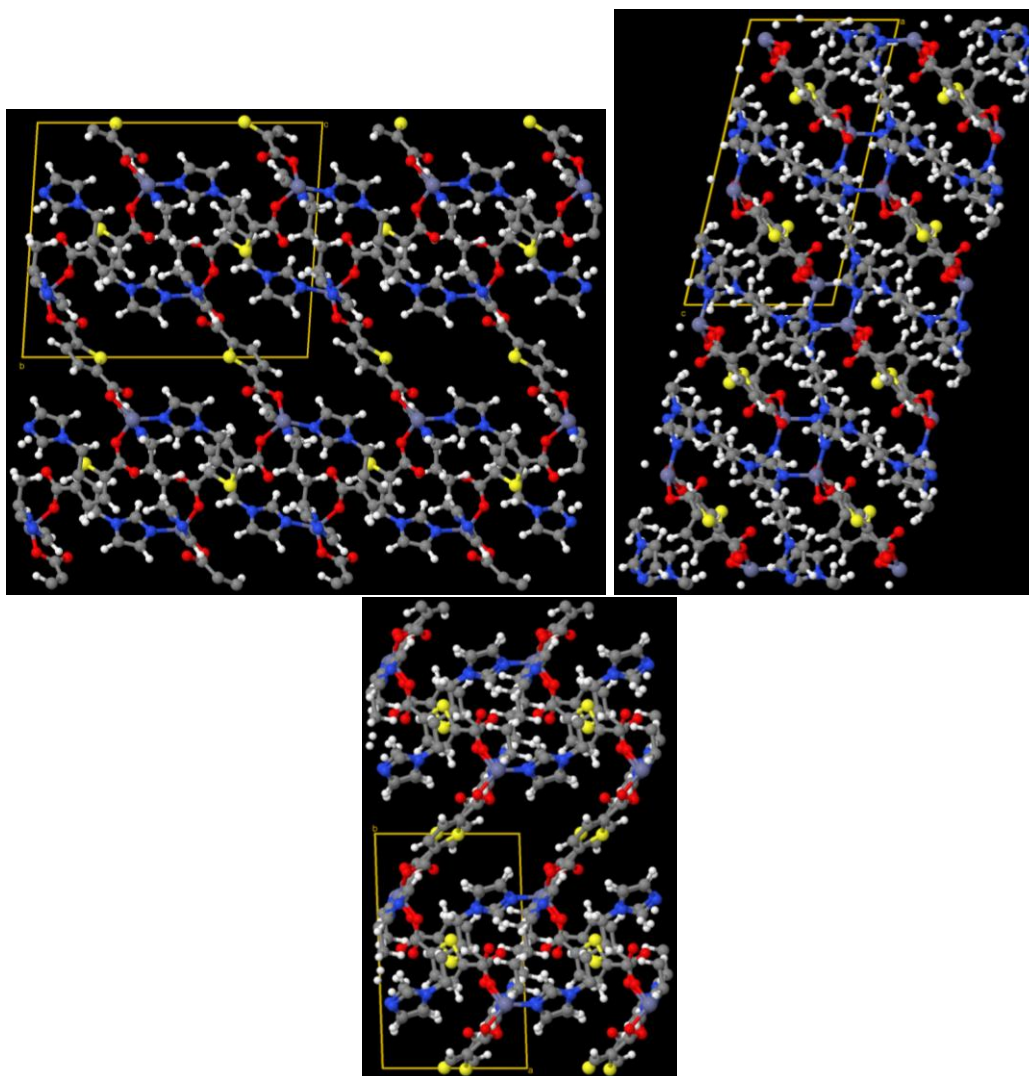


Figure S44: FURFOP contains the chemical elements Zn, C, H, O, N, and S.

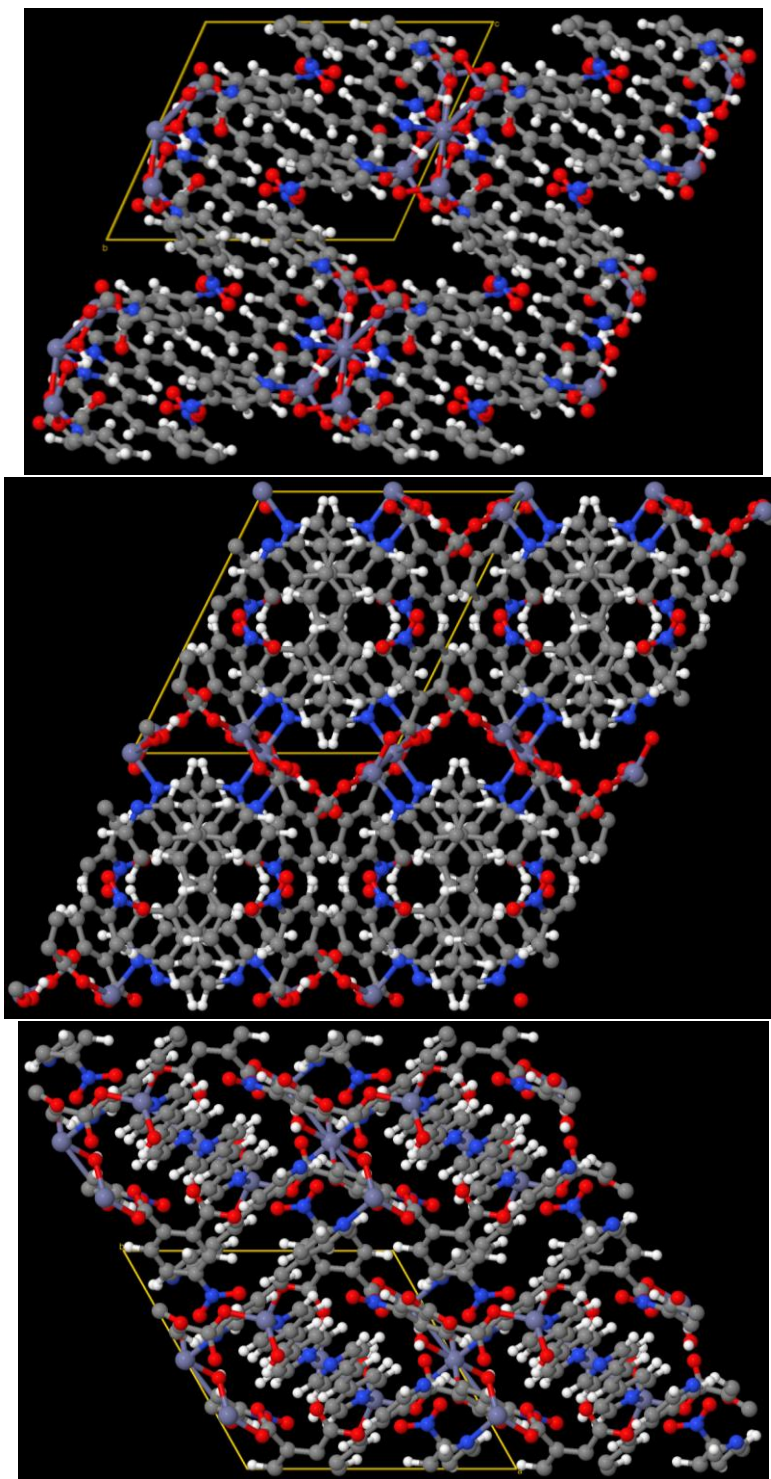


Figure S45: GIRYIR contains the chemical elements Zn, H, C, N, and O.

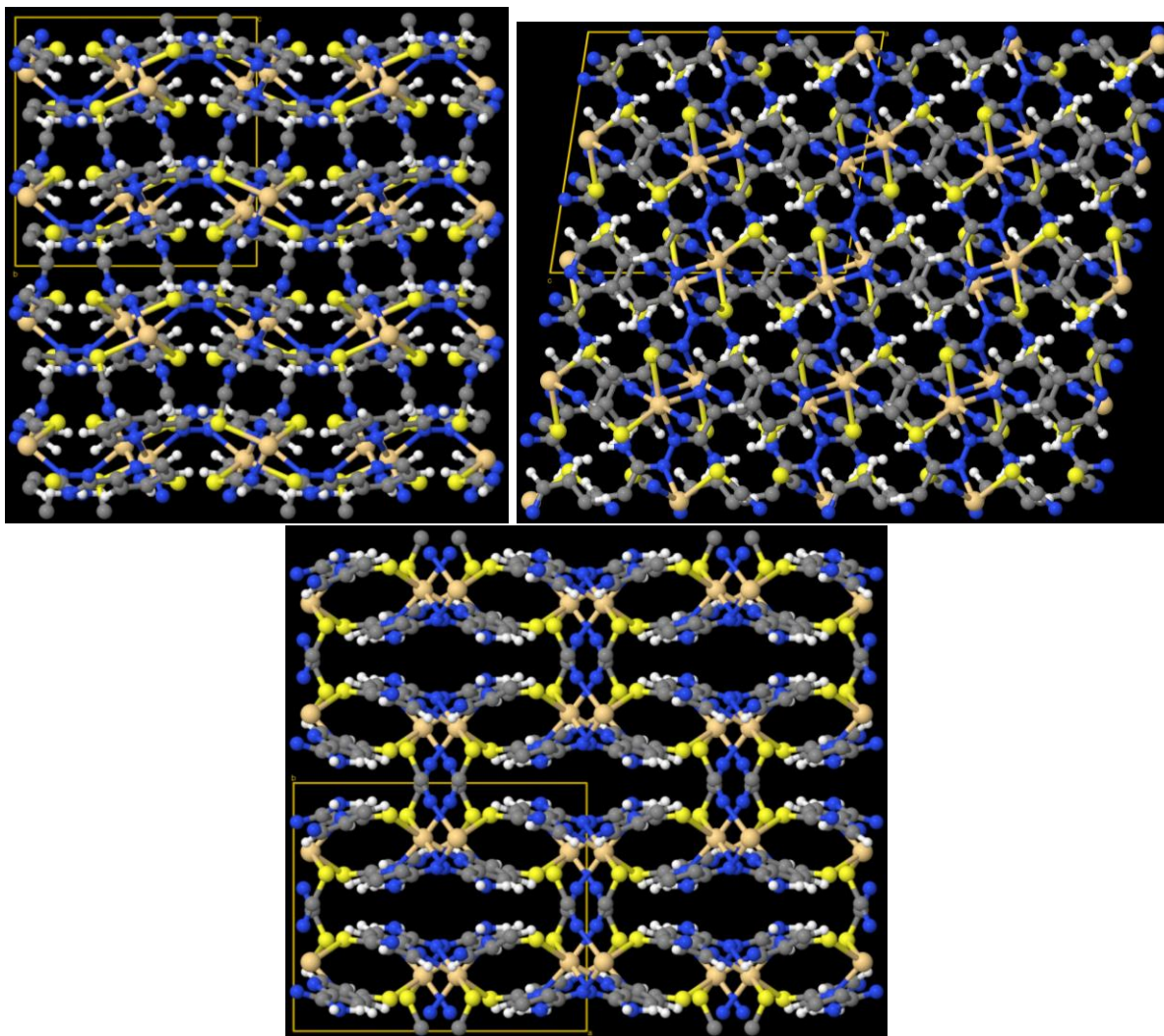


Figure S46: GIWMOP contains the chemical elements Cd, C, H, N, and S.

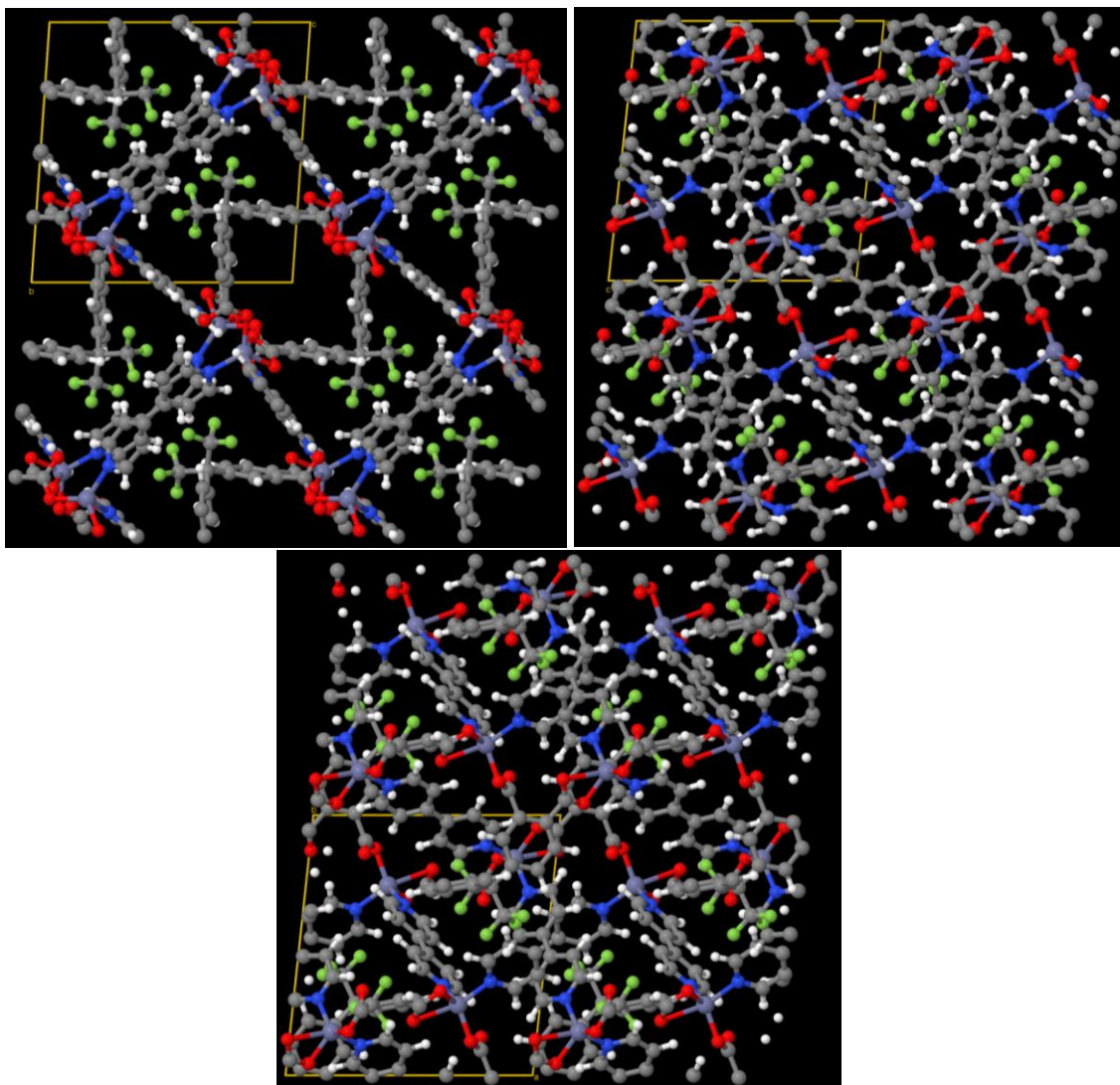


Figure S47: GULPIN contains the chemical elements Zn, H, C, N, O, and F.

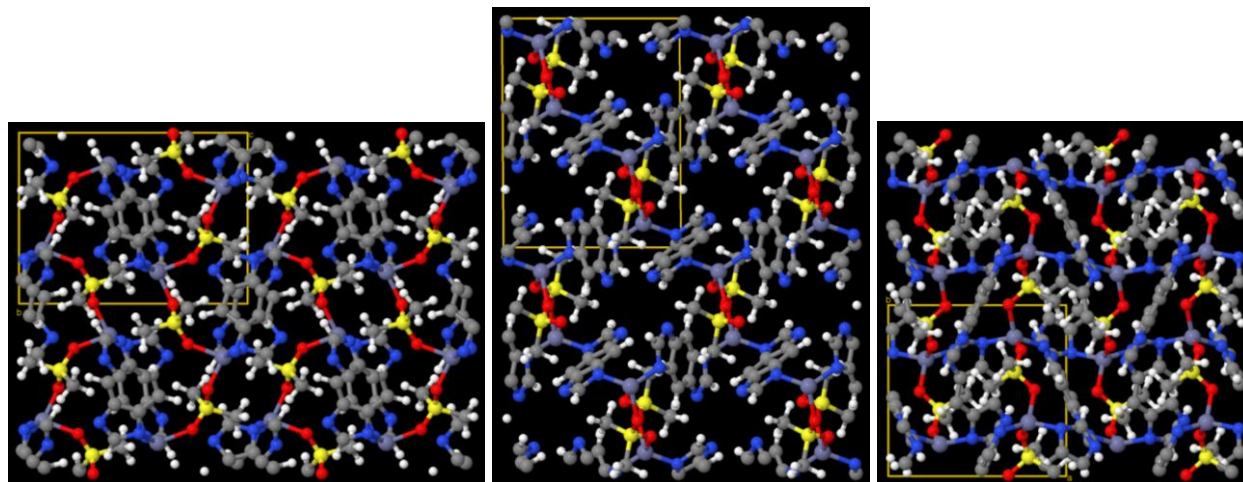


Figure S48: GUMDEZ contains the chemical elements Zn, S, C, H, N, and O.

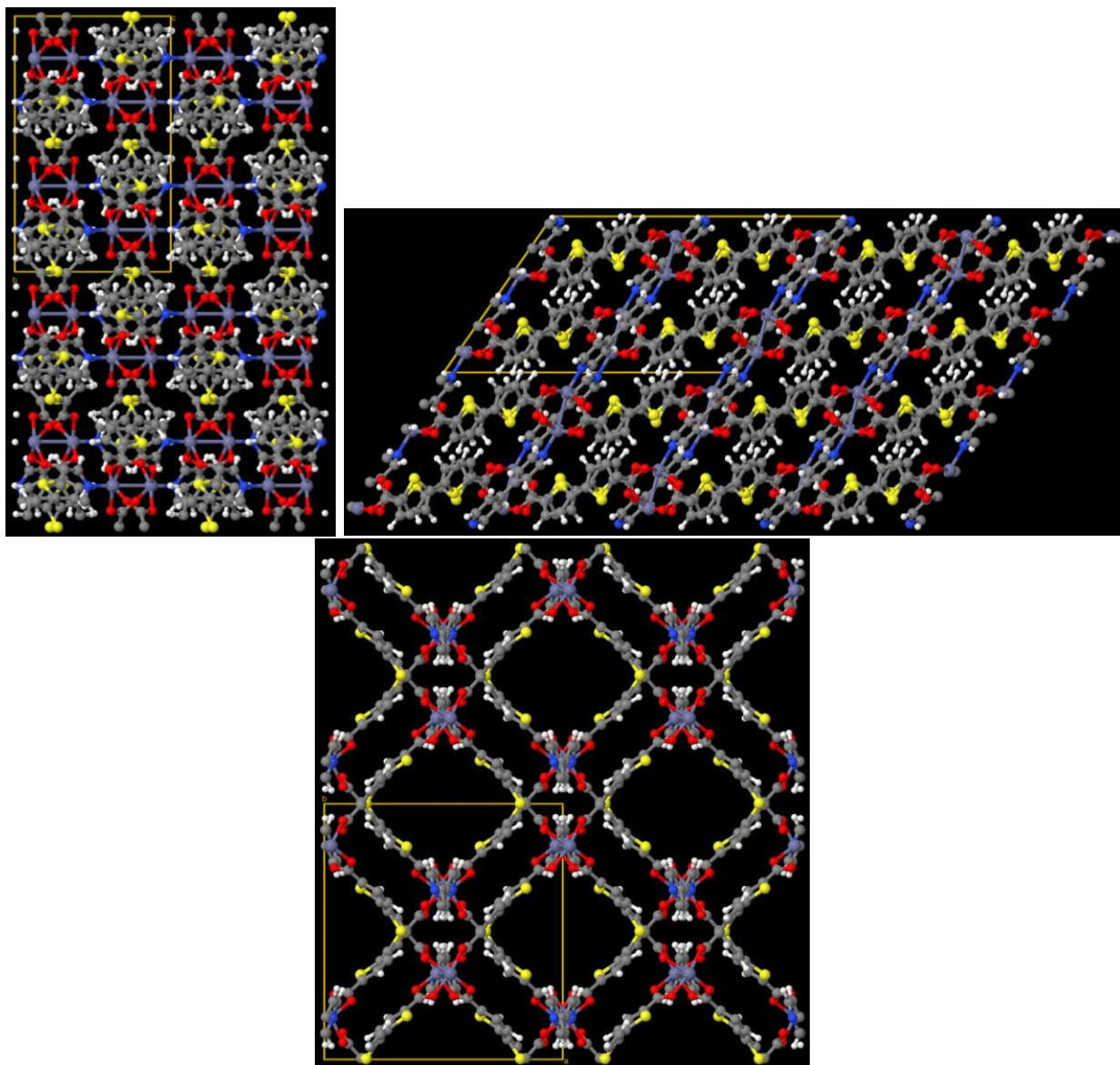


Figure S49: GUYLOC contains the chemical elements Zn, O, C, H, N, and S.

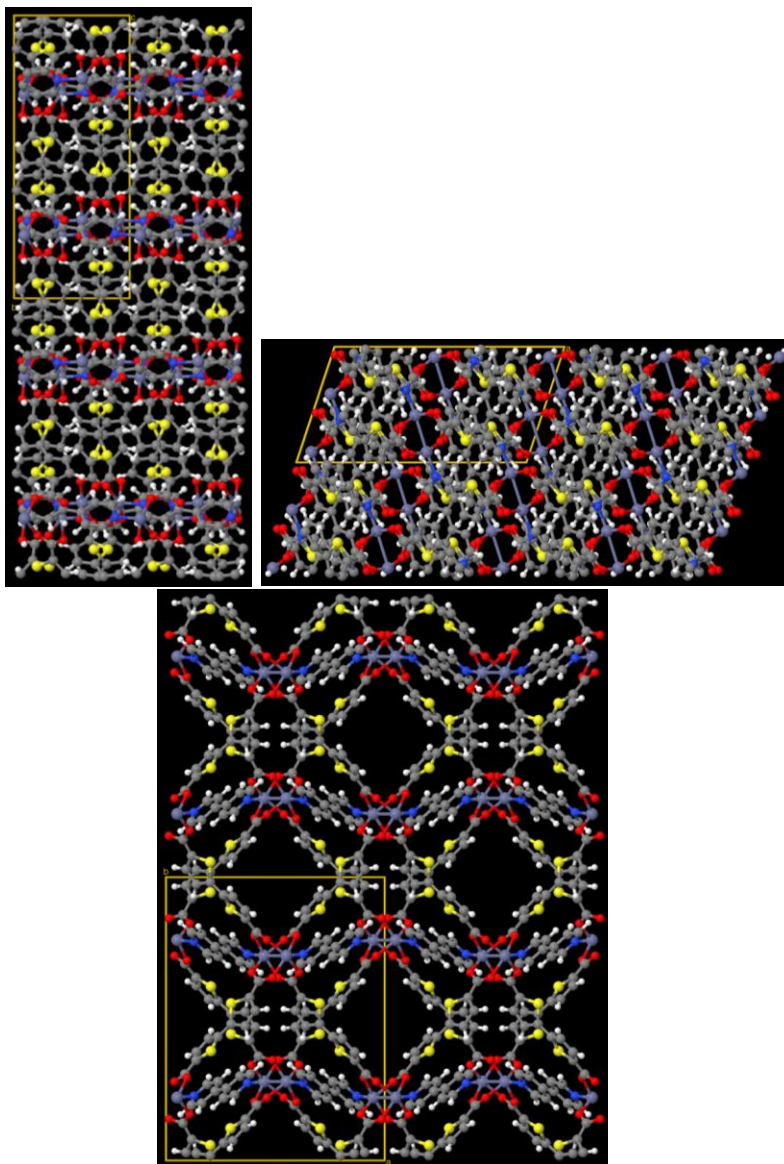


Figure S50: GUYLUI contains the chemical elements O, C, S, H, Zn, and N.

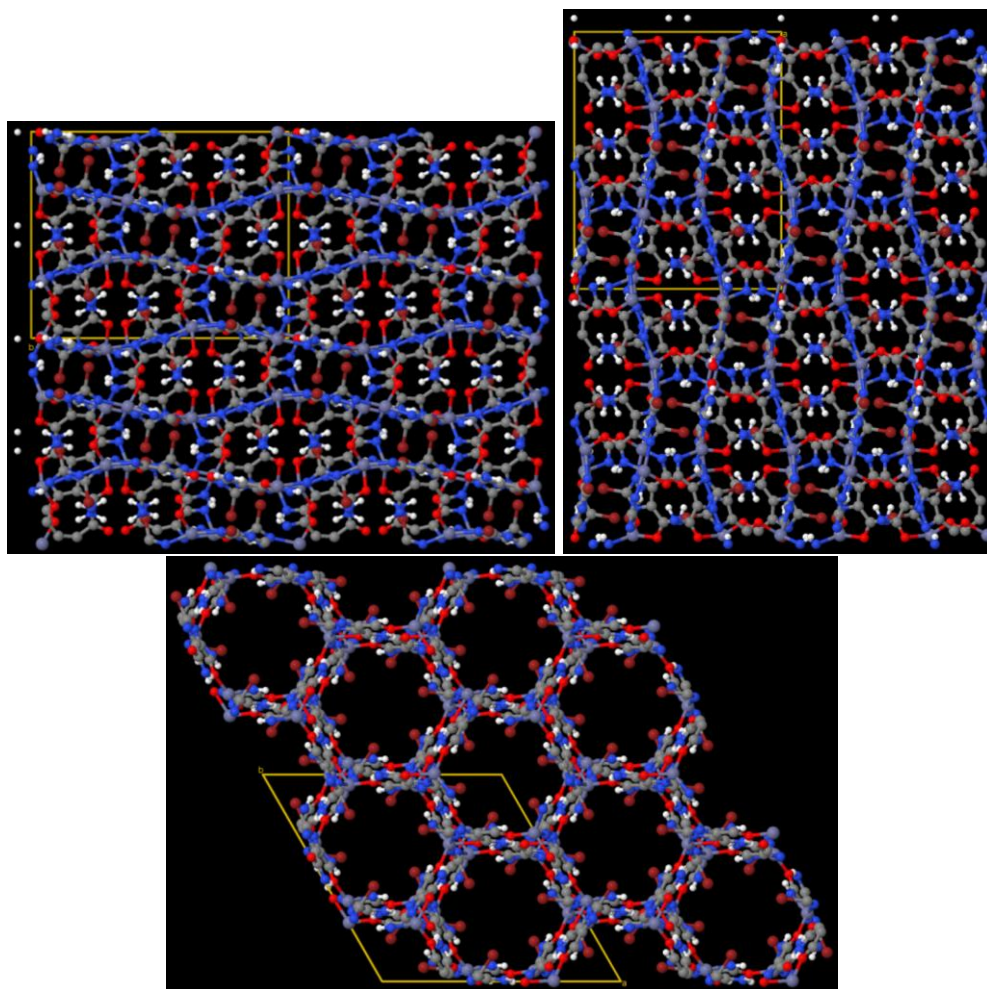


Figure S51: HEBZAR contains the chemical elements C, Br, N, Zn, H, and O.

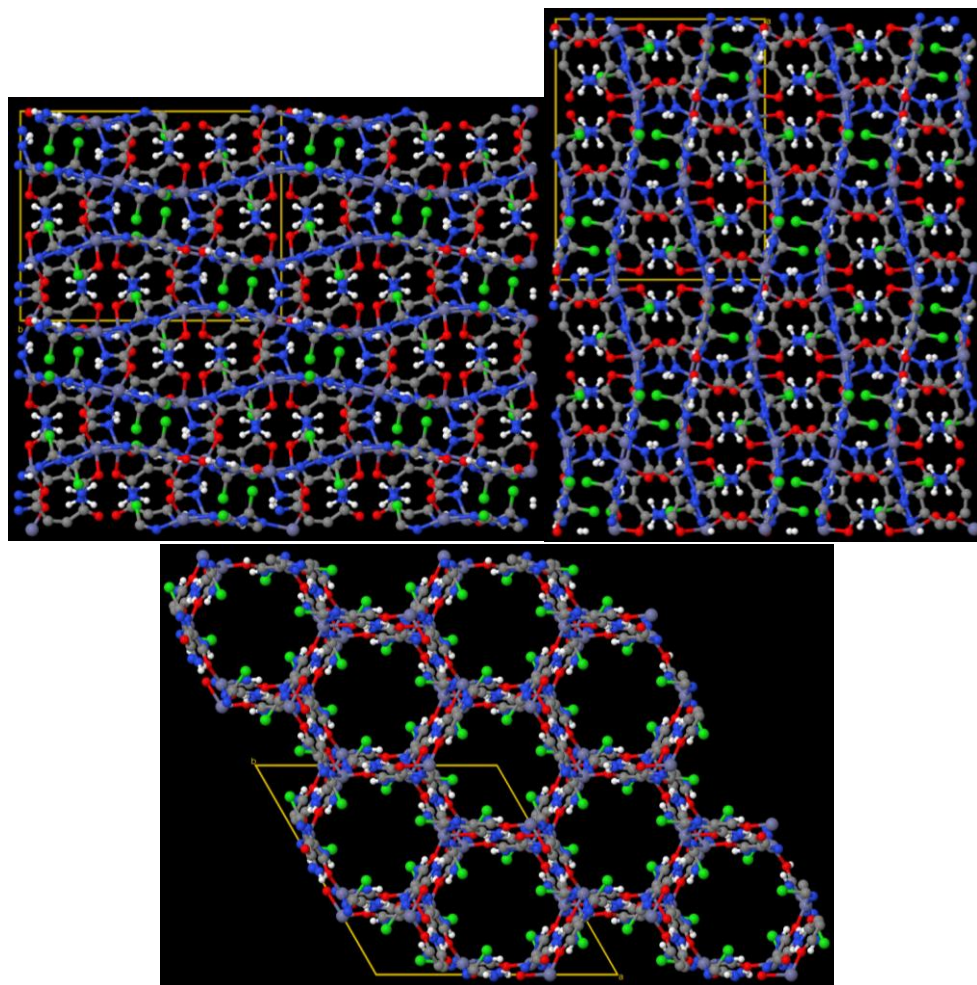


Figure S52: HEBZEV contains the chemical elements C, Cl, N, Zn, H, and O.

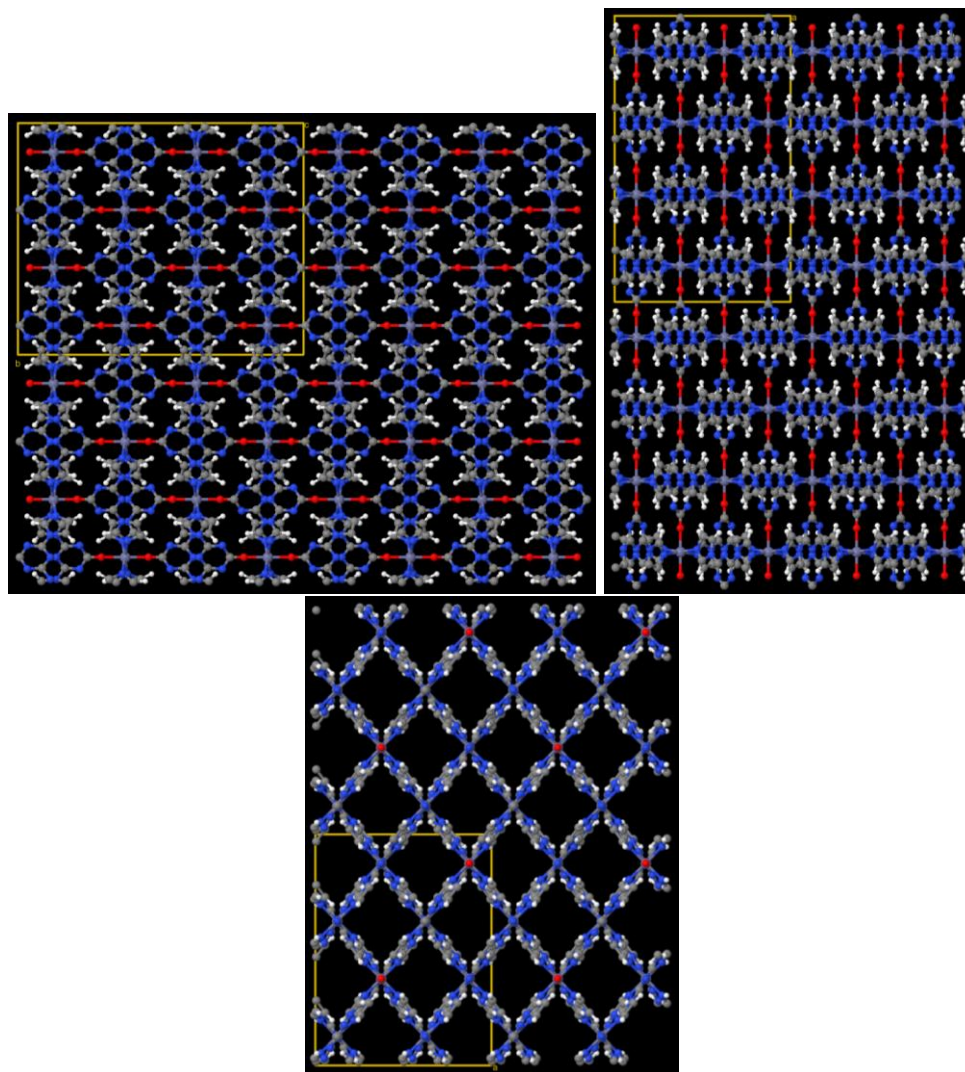


Figure S53: HECQUB contains the chemical elements N, C, H, Zn, and O.

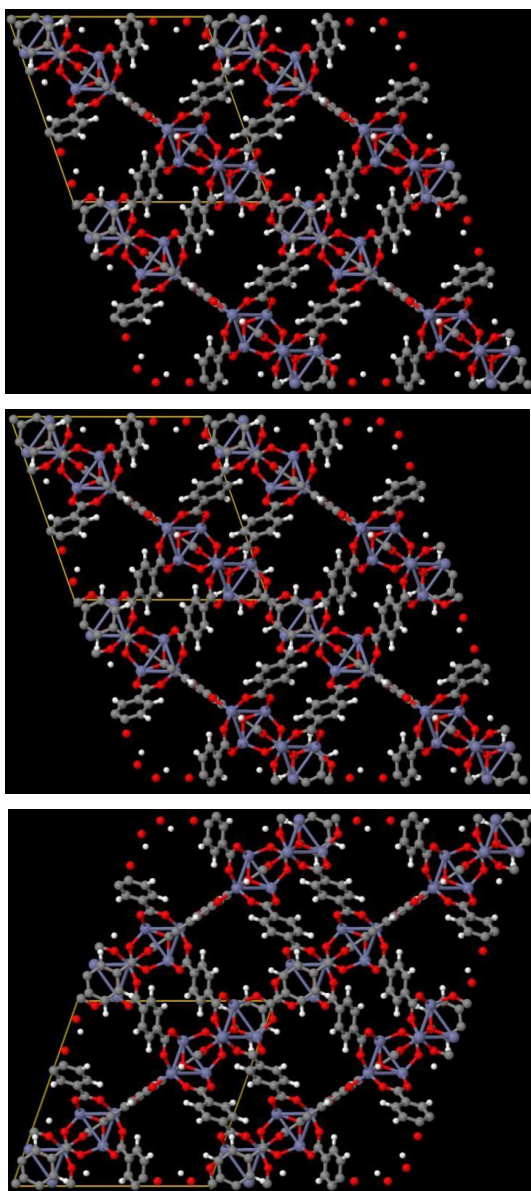


Figure S54: HIFTOG02 contains the chemical elements Zn, H, C, and O.

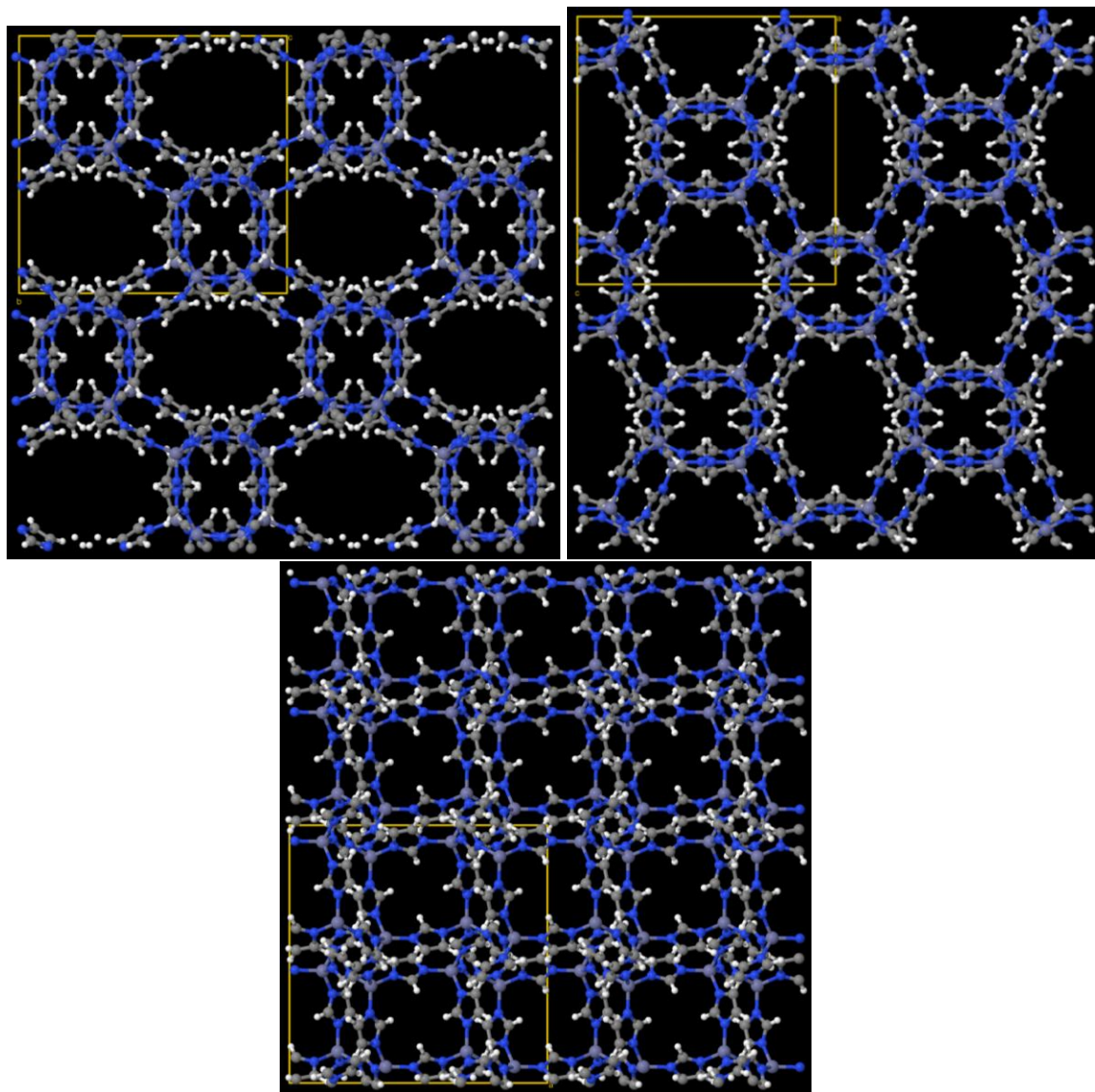


Figure S55: HIFVUO contains the chemical elements Zn, N, C, and H.

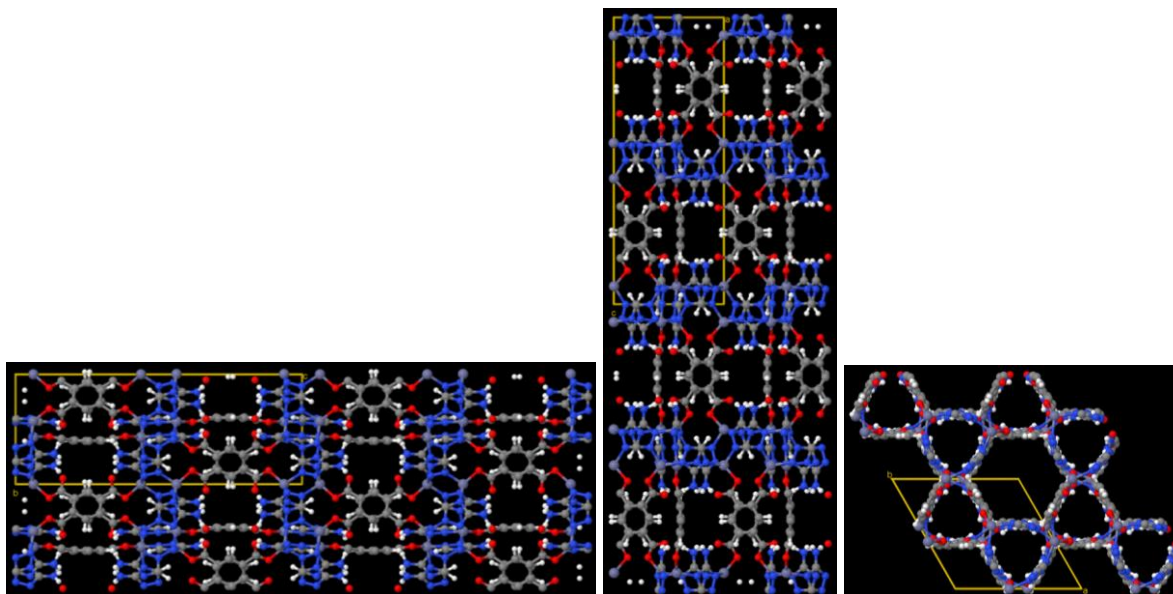


Figure S56: HOXGEH contains the chemical elements O, N, C, H, and Zn.

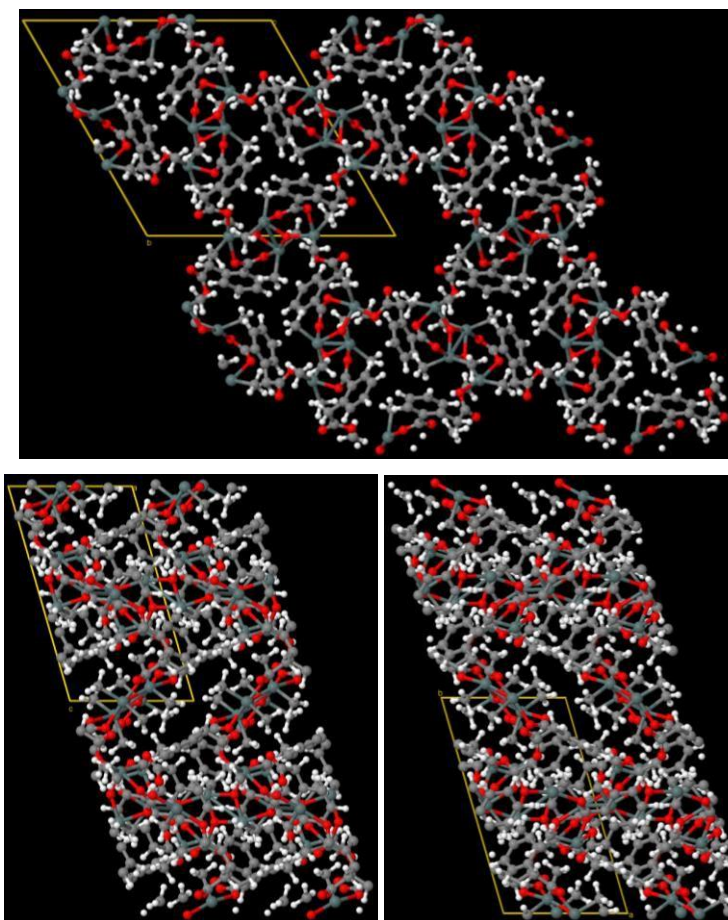


Figure S57: HOXGEH contains the chemical elements Sn, H, C, and O.

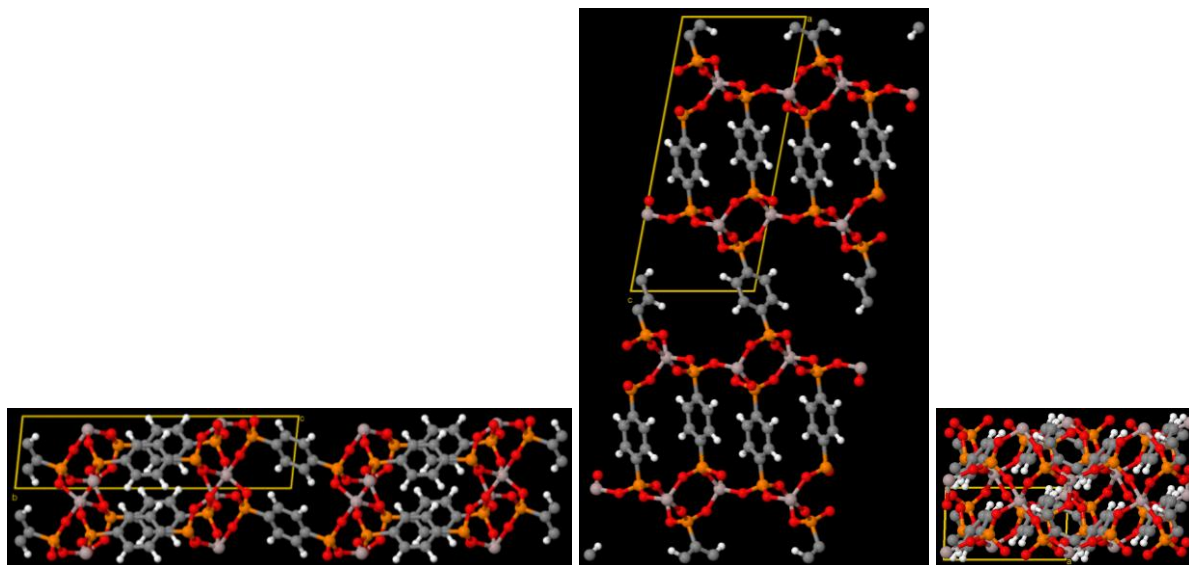


Figure S58: HOZDOR contains the chemical elements Al, P, O, C, and H.

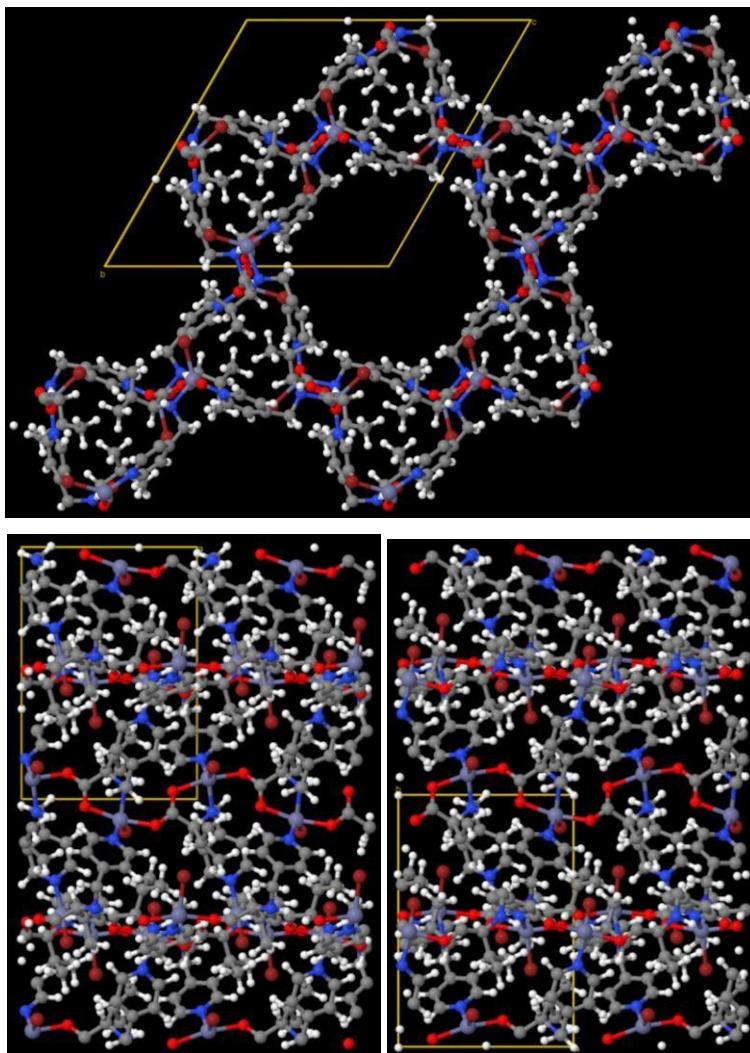


Figure S59: IBICED contains the chemical elements Zn, H, C, Br, N, and O.

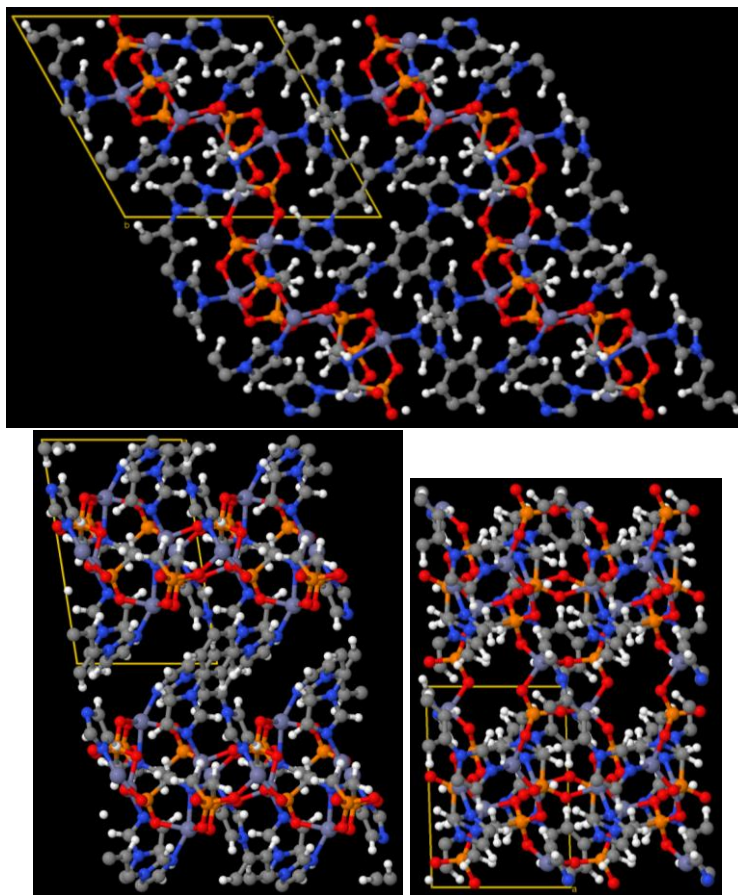


Figure S60: IBOCIO contains the chemical elements H, C, O, N, Zn, and P.

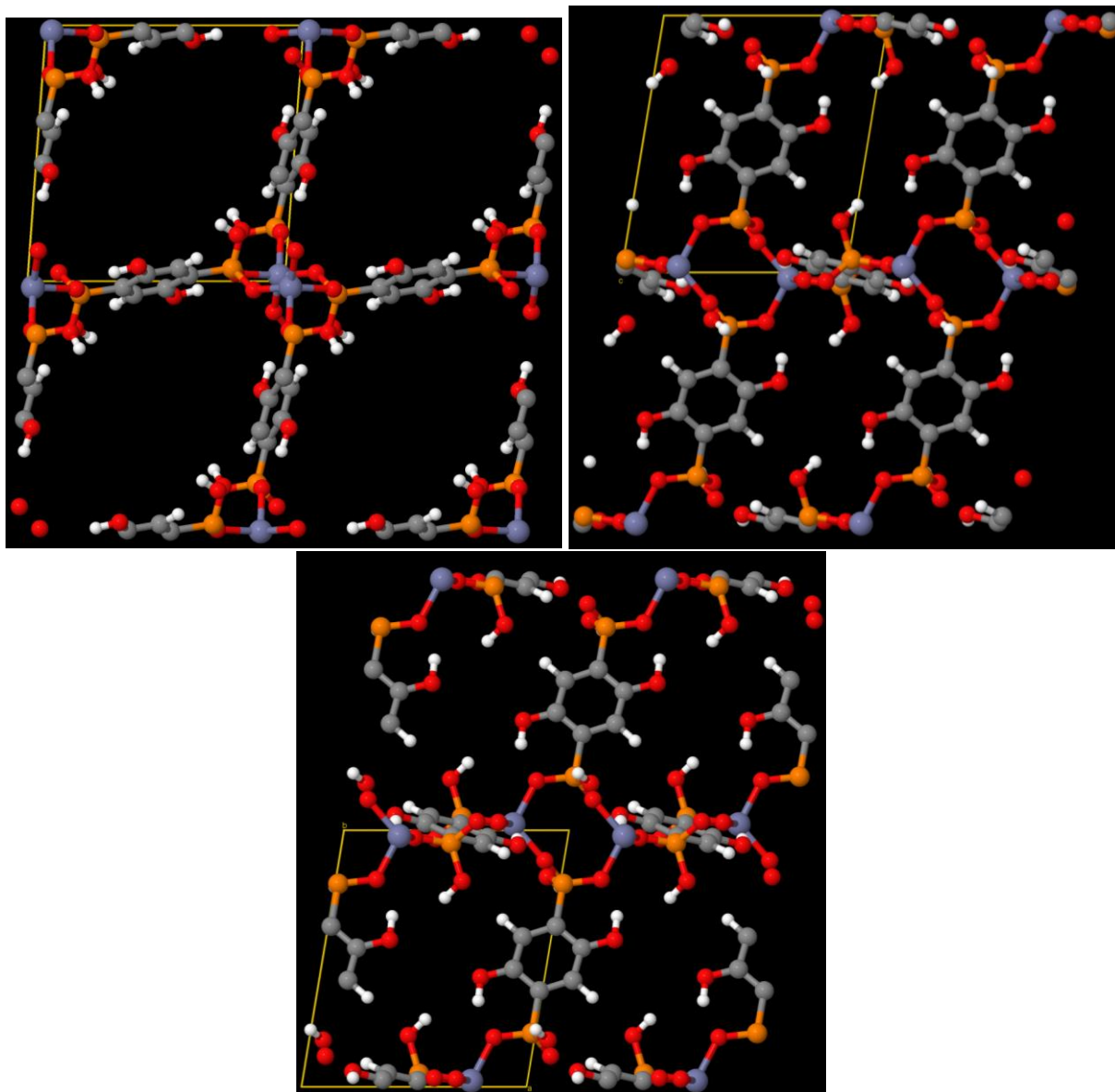


Figure S61: JIVFUQ contains the chemical elements Zn, P, H, C, and O.

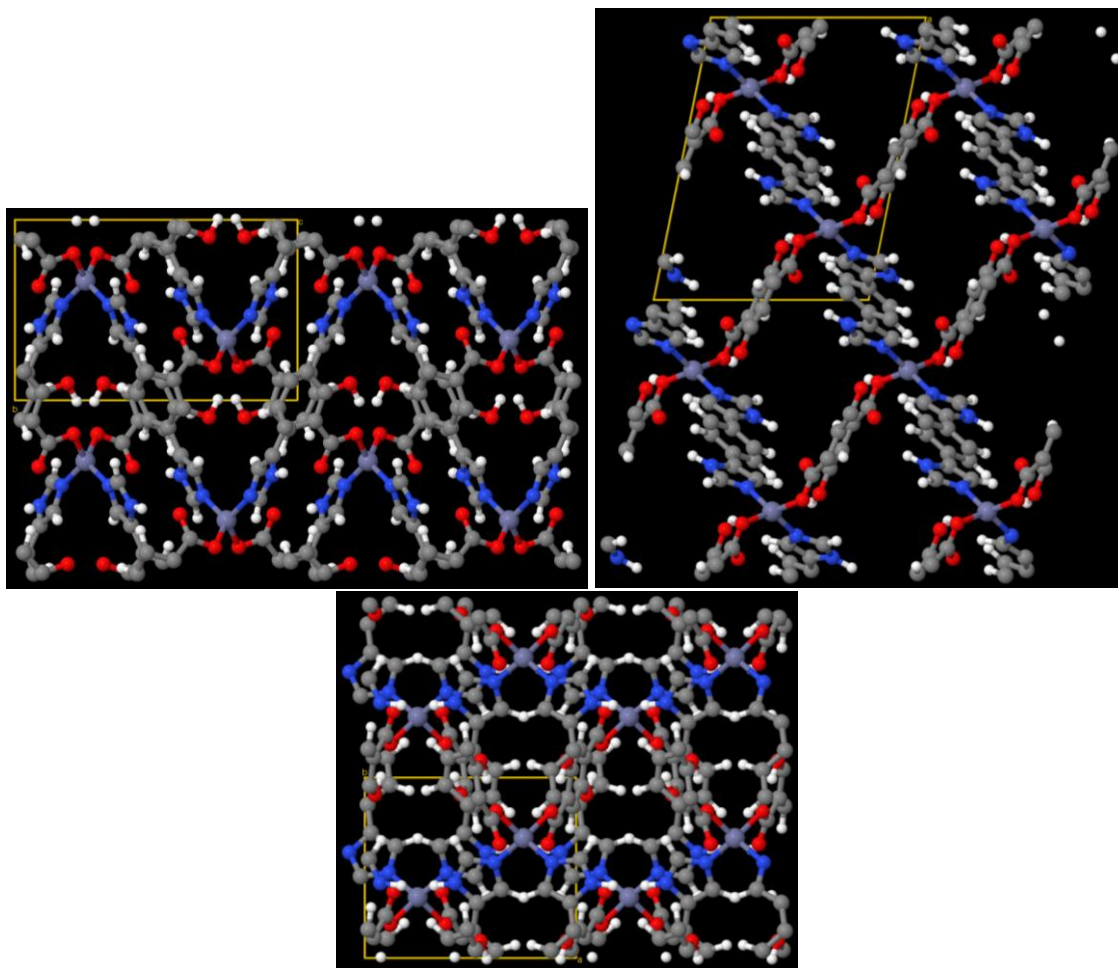


Figure S62: KACZUM contains the chemical elements O, C, H, Zn, and N.

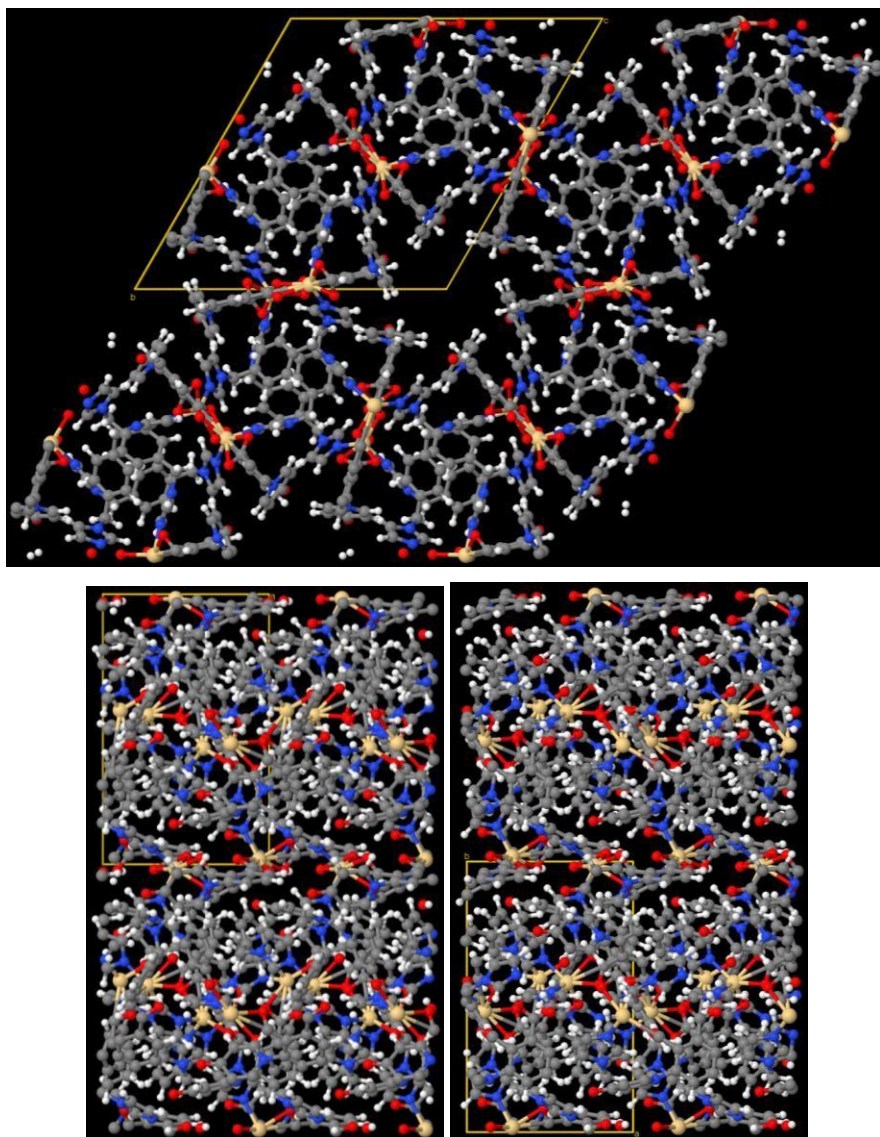


Figure S63: KATDAM contains the chemical elements Cd, H, C, N, and O.

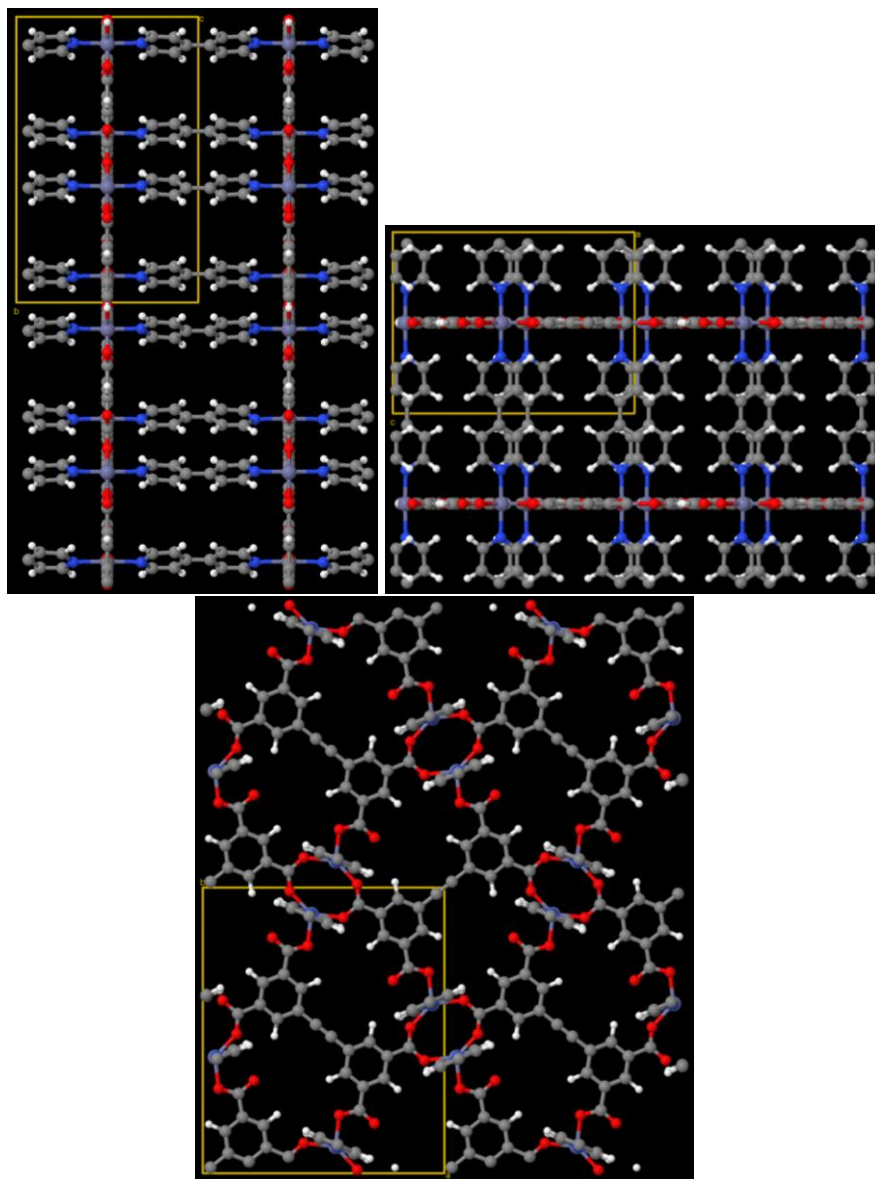


Figure S64: KEWZOD contains the chemical elements O, C, H, Zn, and N.

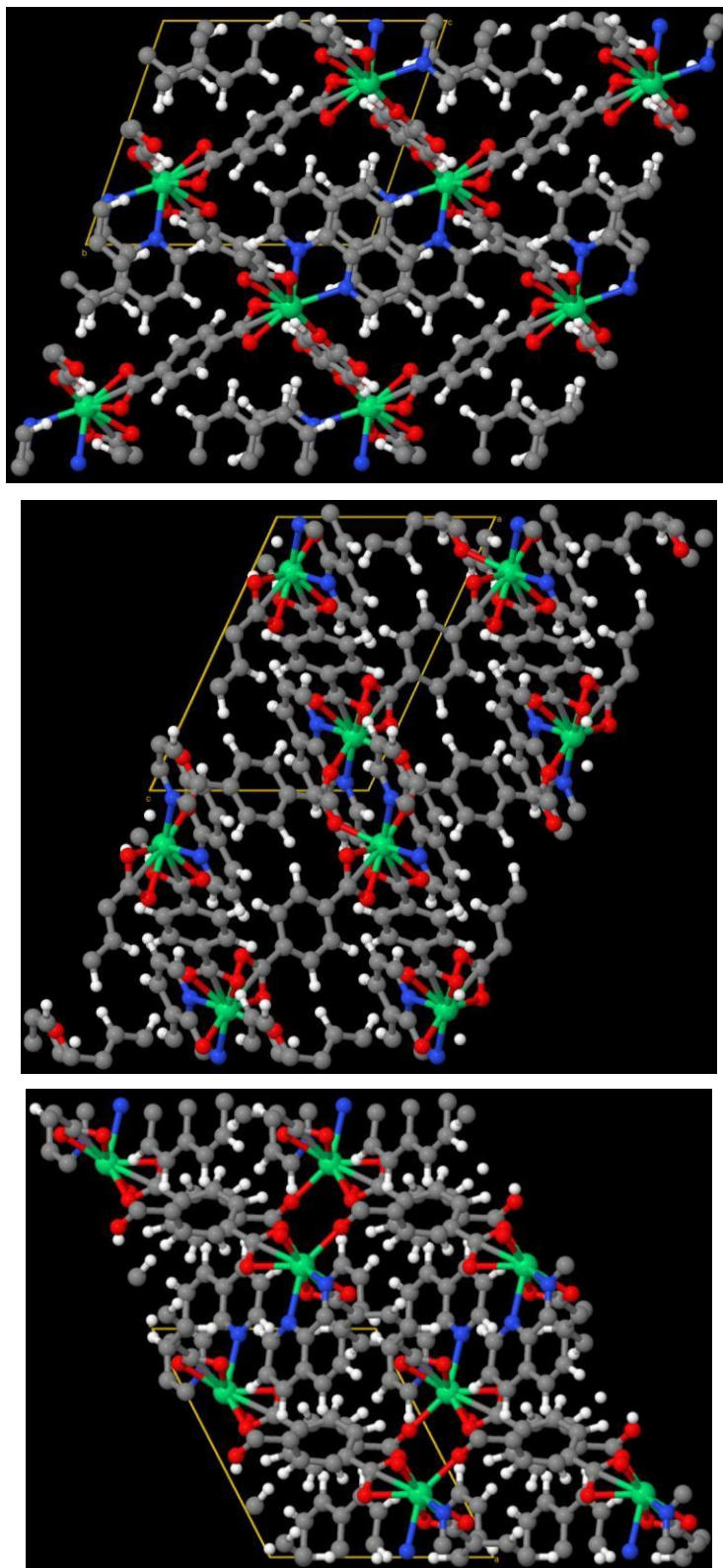


Figure S65: KINDIW contains the chemical elements Er, H, C, N, and O.

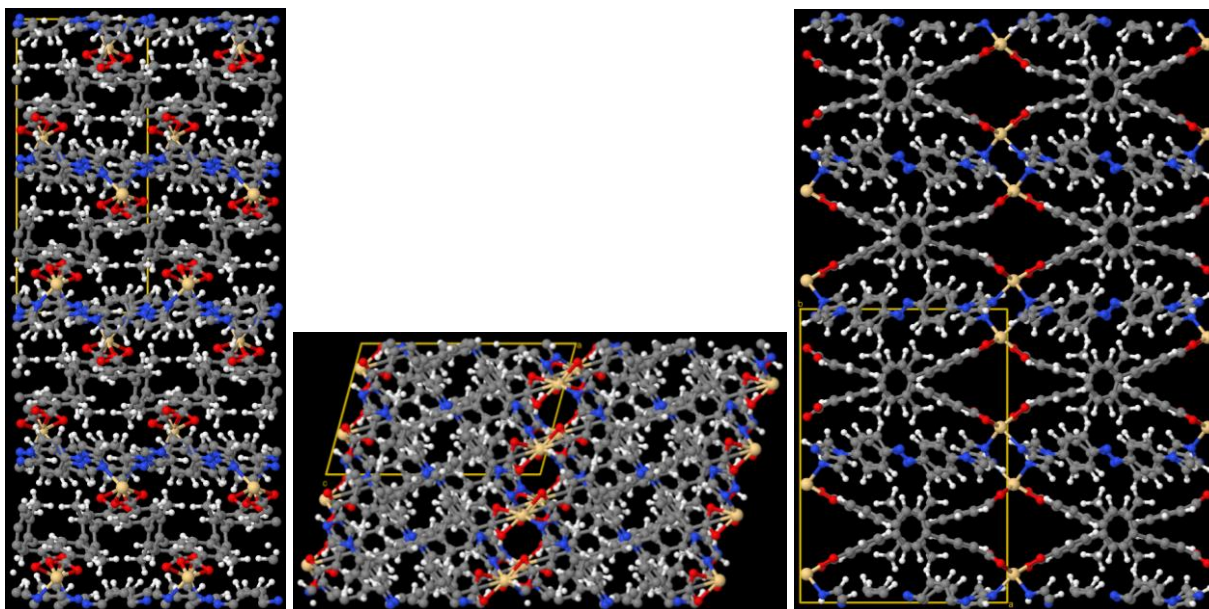


Figure S66: KINJUO contains the chemical elements C, H, N, Cd, and O.

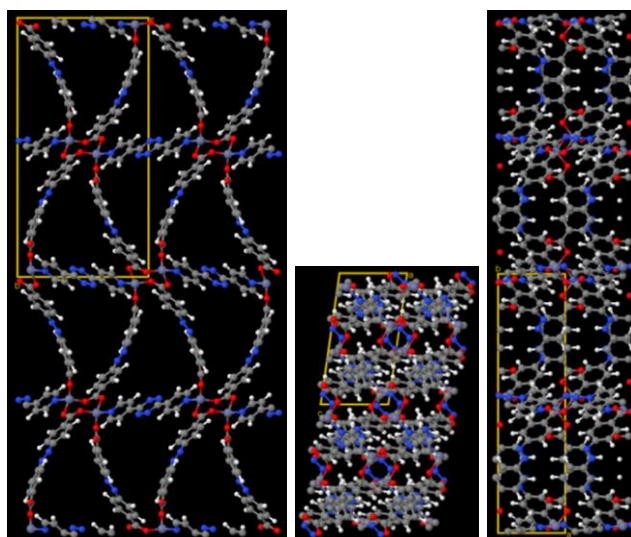


Figure S67: KURSOH contains the chemical elements Zn, O, C, H, and N.

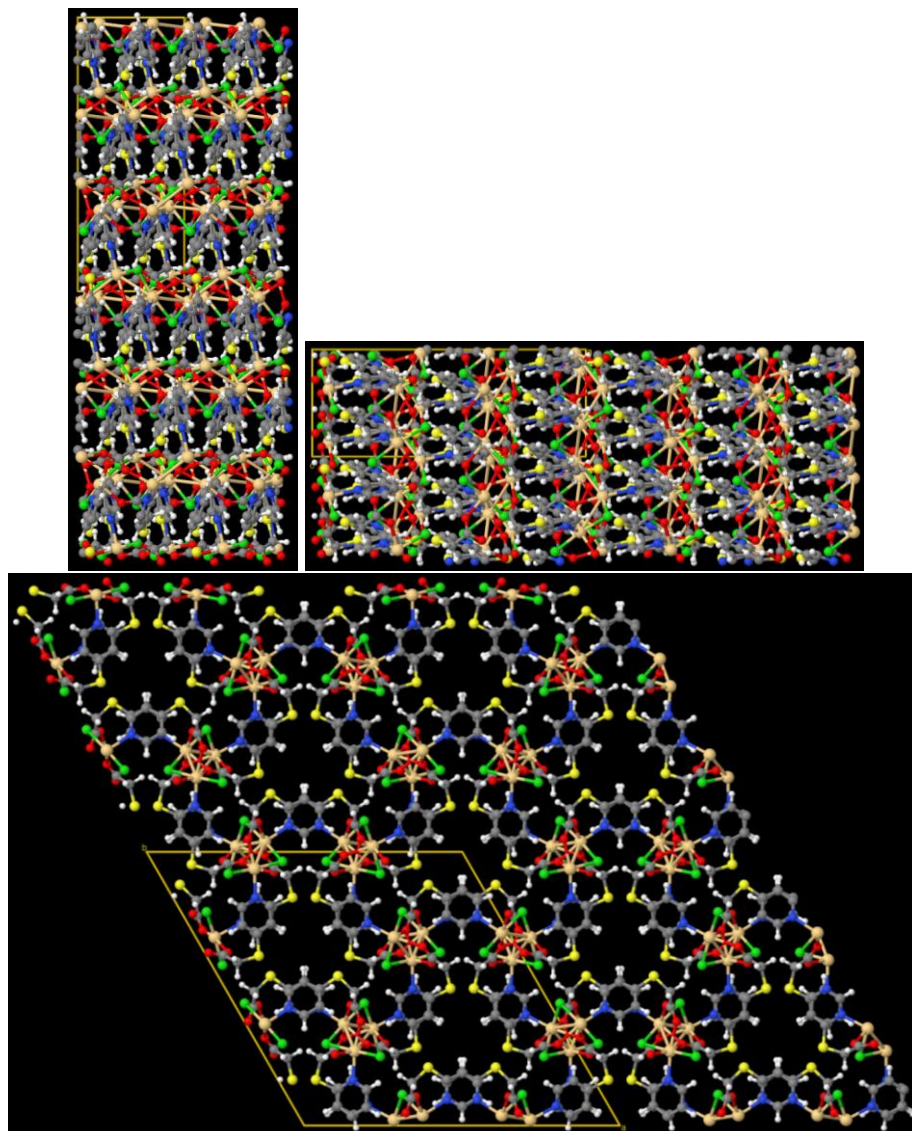


Figure S68: LALYOO contains the chemical elements Cd, Cl, O, S, N, C, and H.

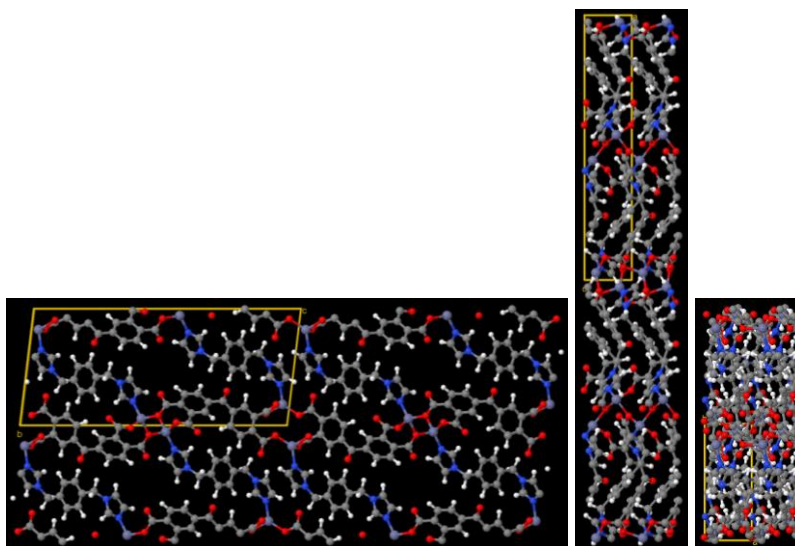


Figure S69: LIWXIZ contains the chemical elements Zn, H, C, N, and O.

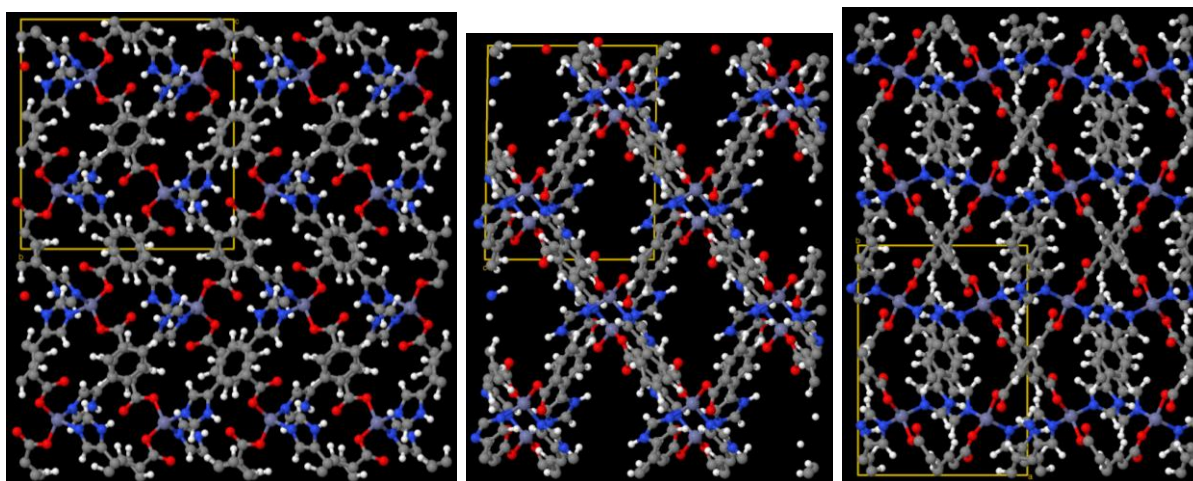


Figure S70: LODPOL contains the chemical elements N, H, C, O, and Zn.

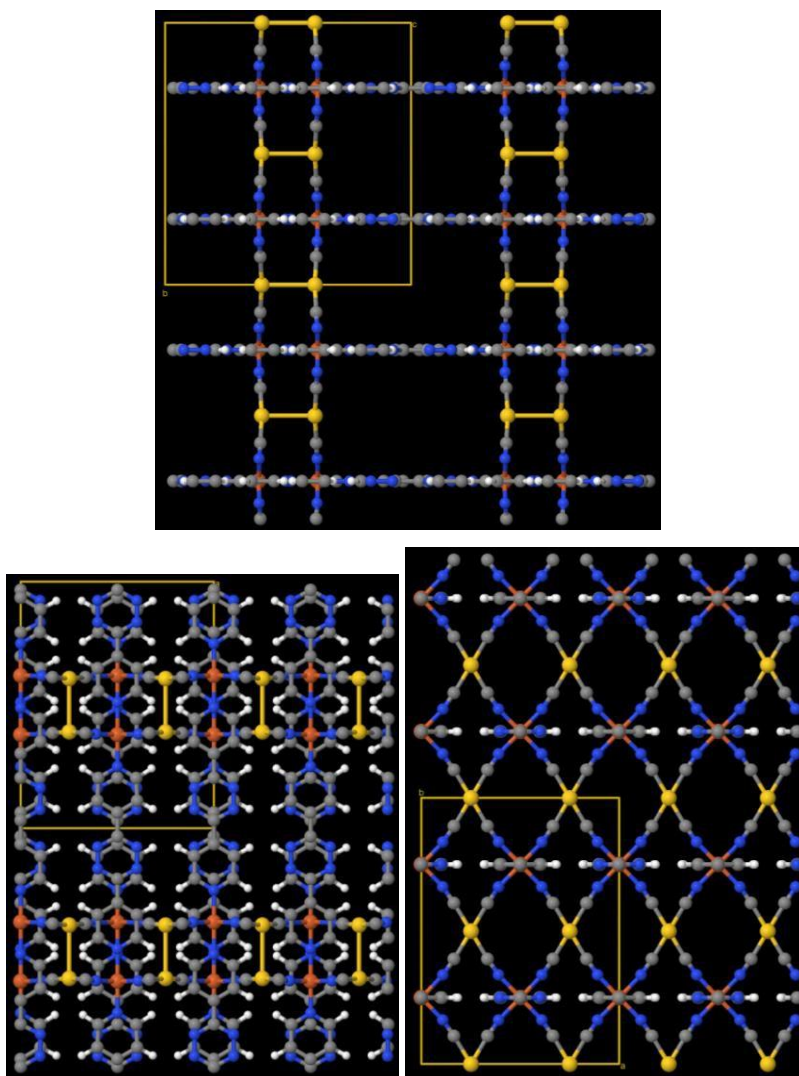


Figure S71: LOJLAZ contains the chemical elements N, C, Au, Fe, and H.

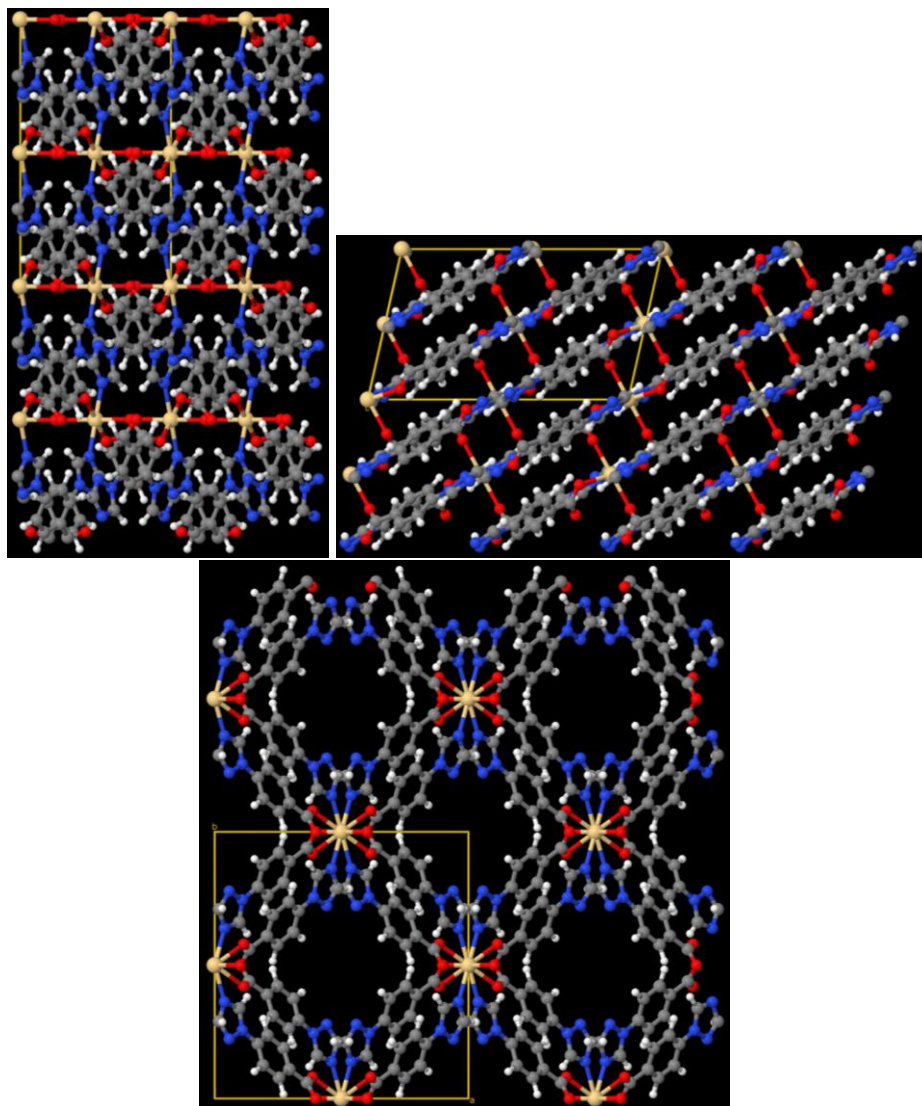


Figure S72: LUFCAS contains the chemical elements O, N, C, H, and Cd.

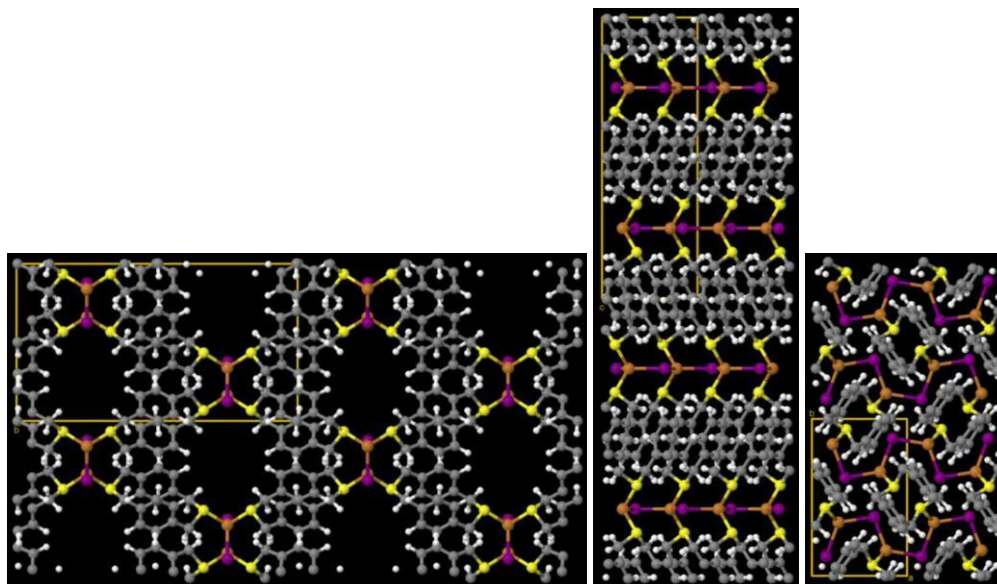


Figure S73: MADVUJ contains the chemical elements Cu, H, C, S, and I.

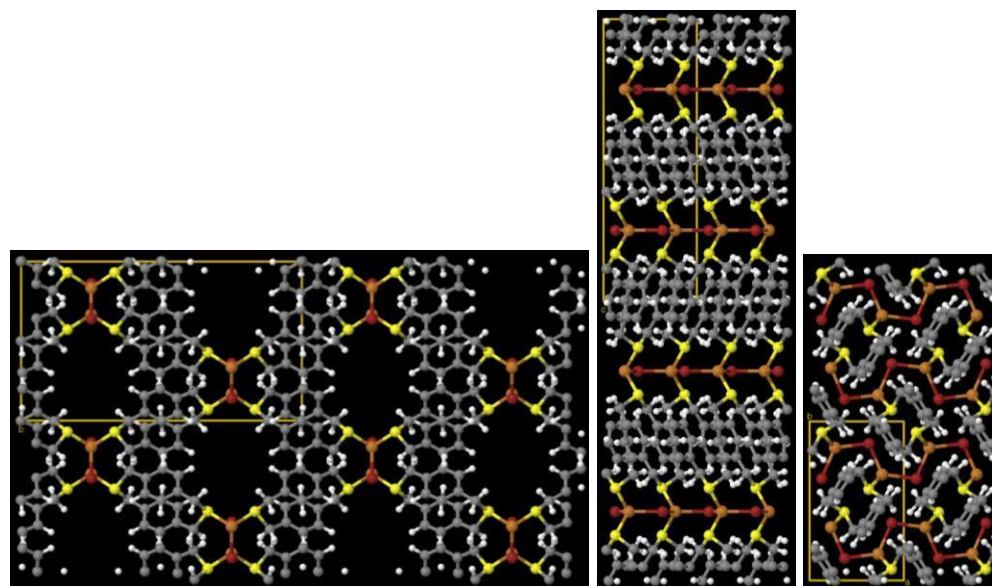


Figure S74: MADWAQ contains the chemical elements Cu, H, C, S, and Br.

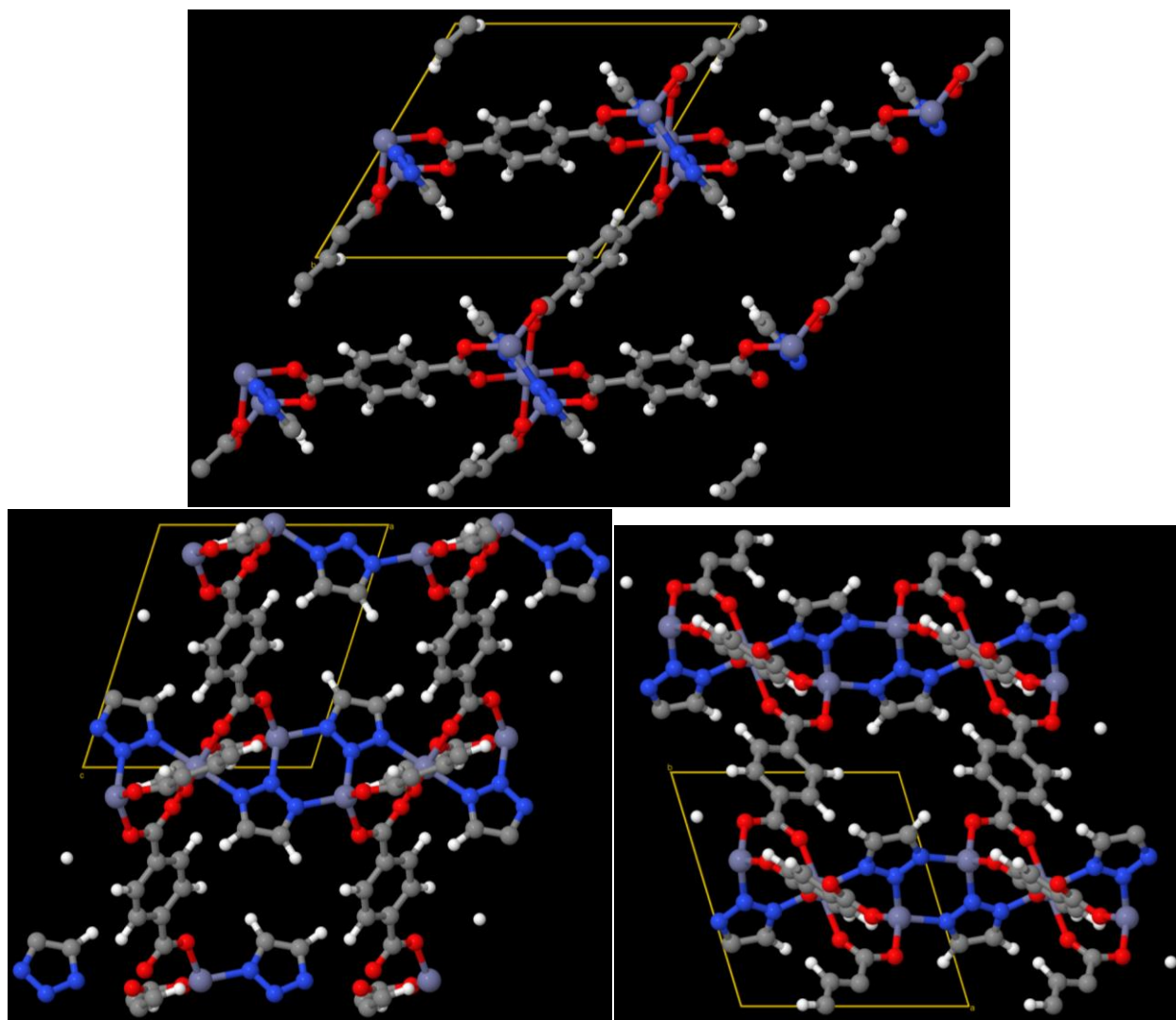


Figure S75: MAZSUD contains the chemical elements Zn, H, C, N, and O.

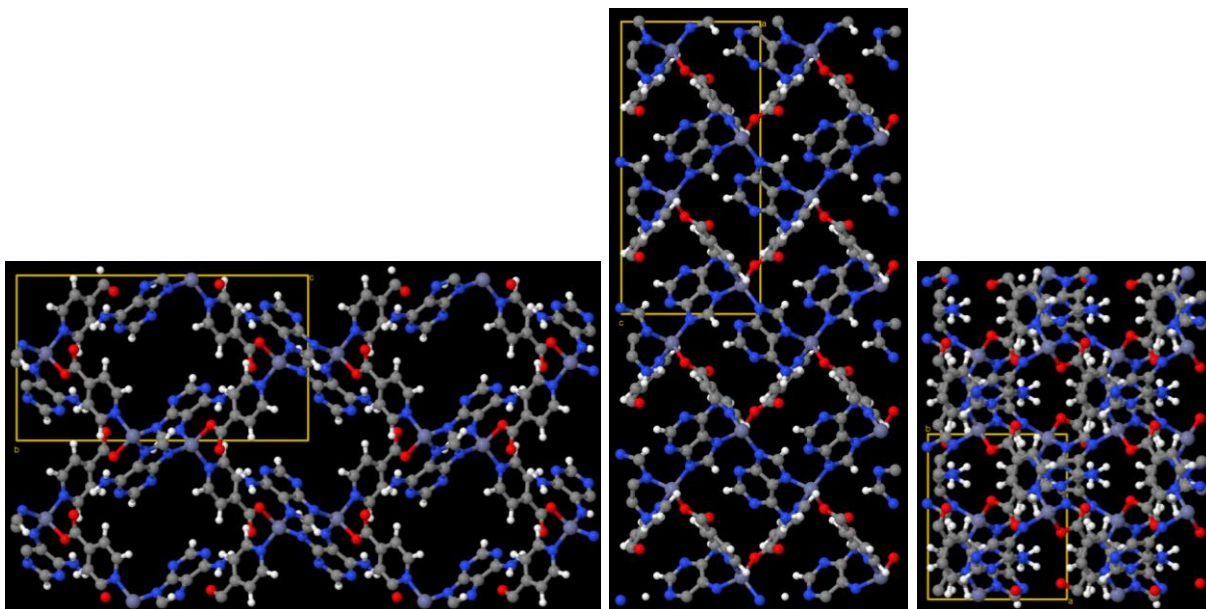


Figure S76: MEJQID contains the chemical elements Zn, H, C, N, and O.

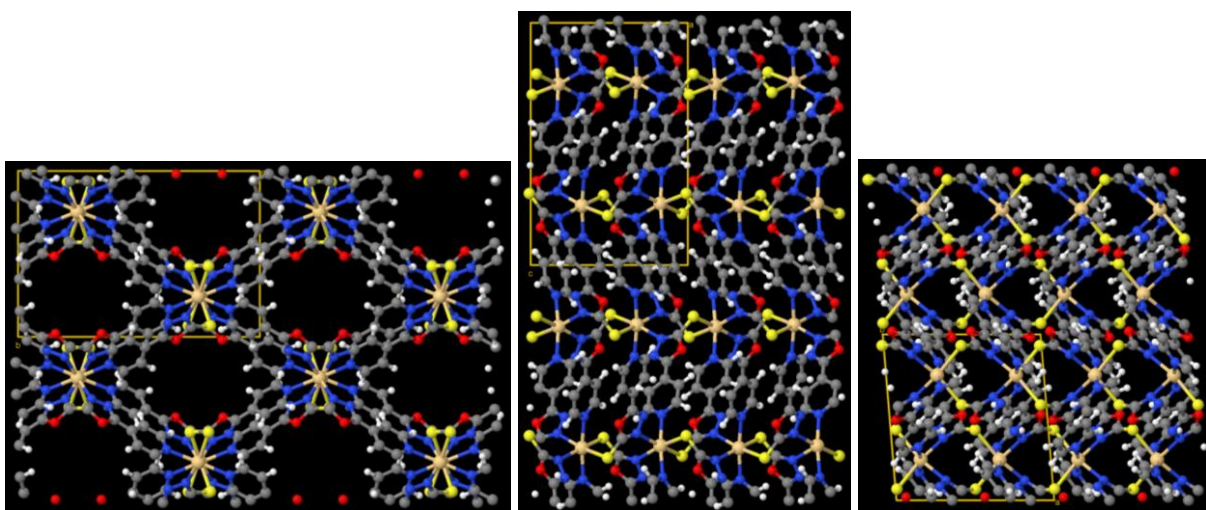


Figure S77: MEKDUC contains the chemical elements Cd, H, C, S, N, and O.

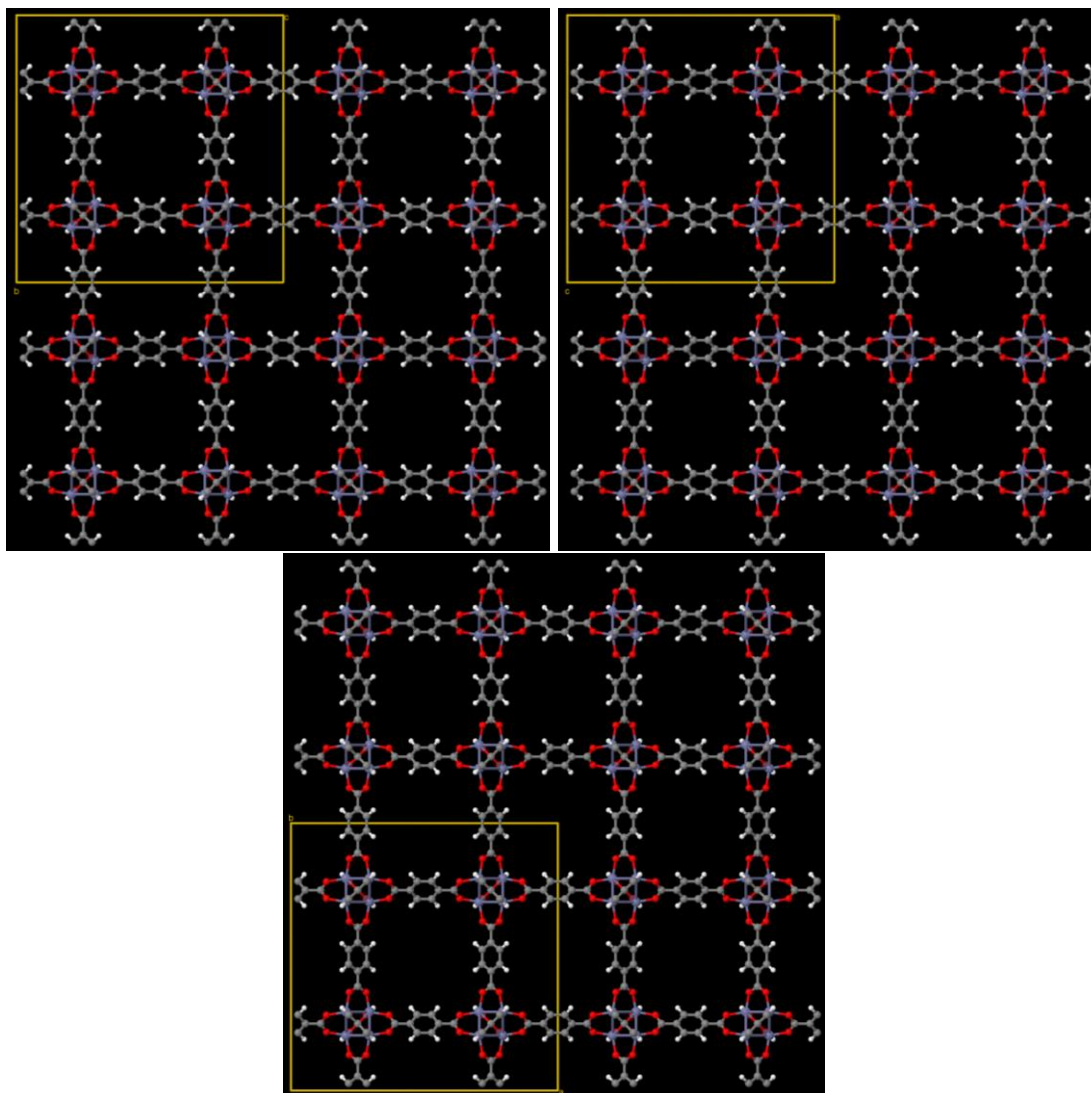


Figure S78: MIBQAR01 contains the chemical elements Zn, O, C, and H.

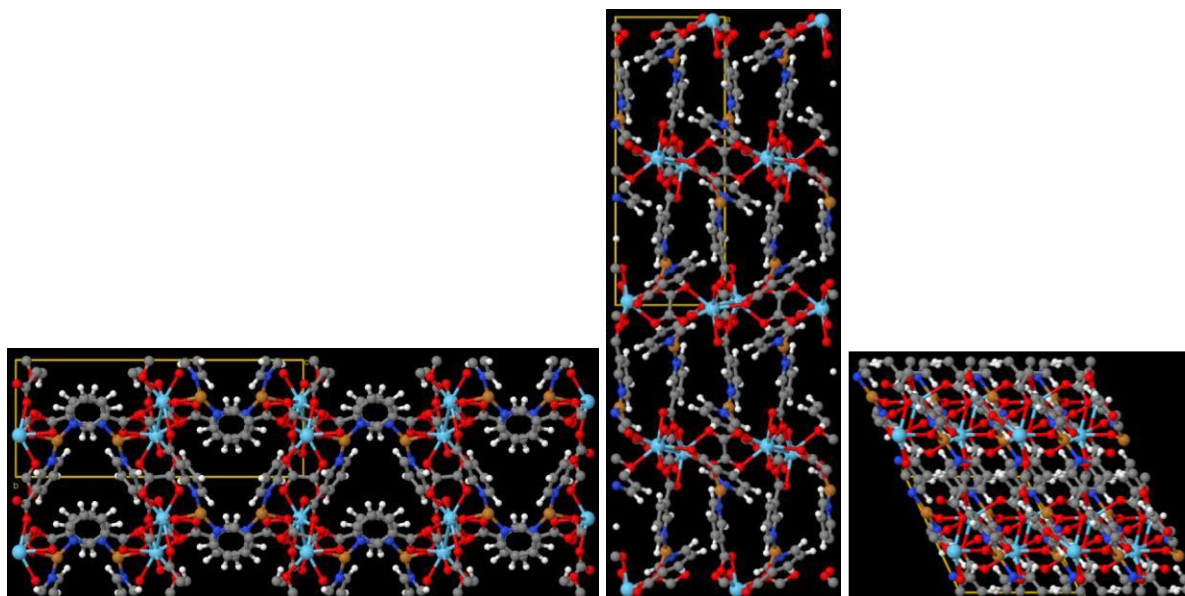


Figure S79: MIZKOW contains the chemical elements La, Cu, H, C, N, and O.

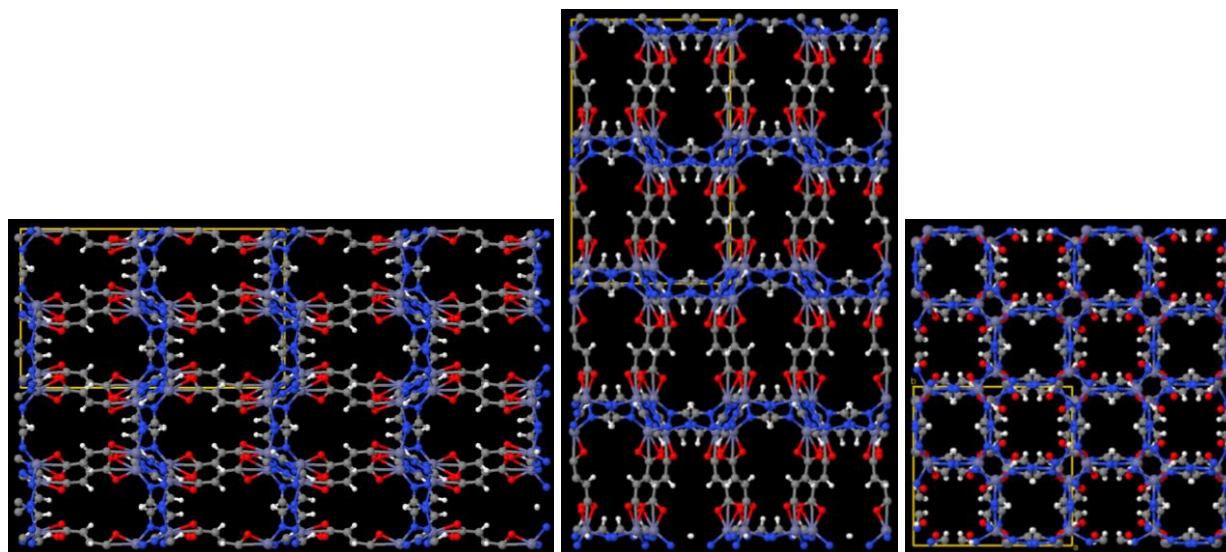


Figure S80: MOPFOO contains the chemical elements O, H, N, C, and Zn.

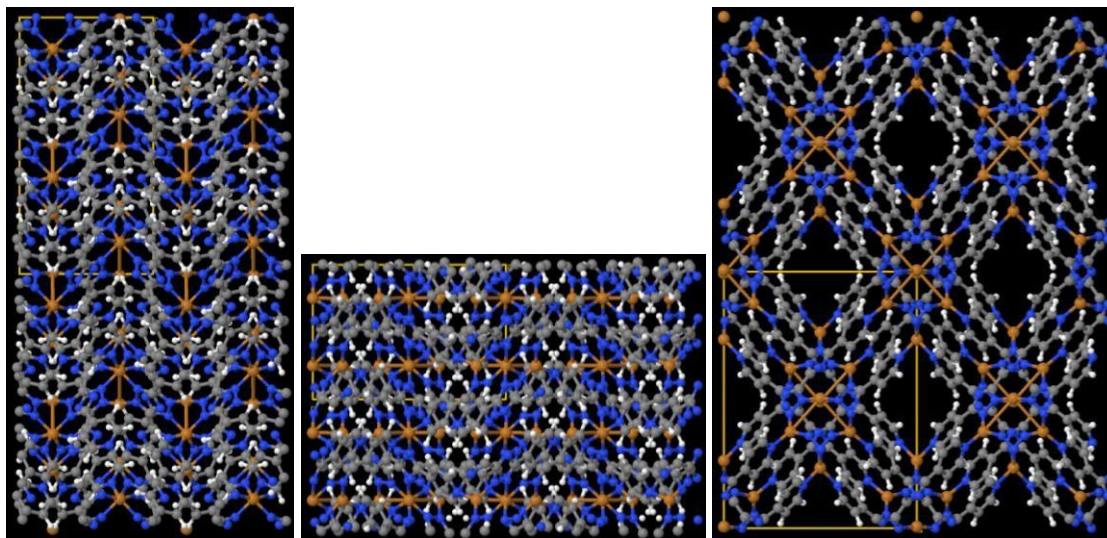


Figure S81: MUGQAI contains the chemical elements Cu, N, C, and H.

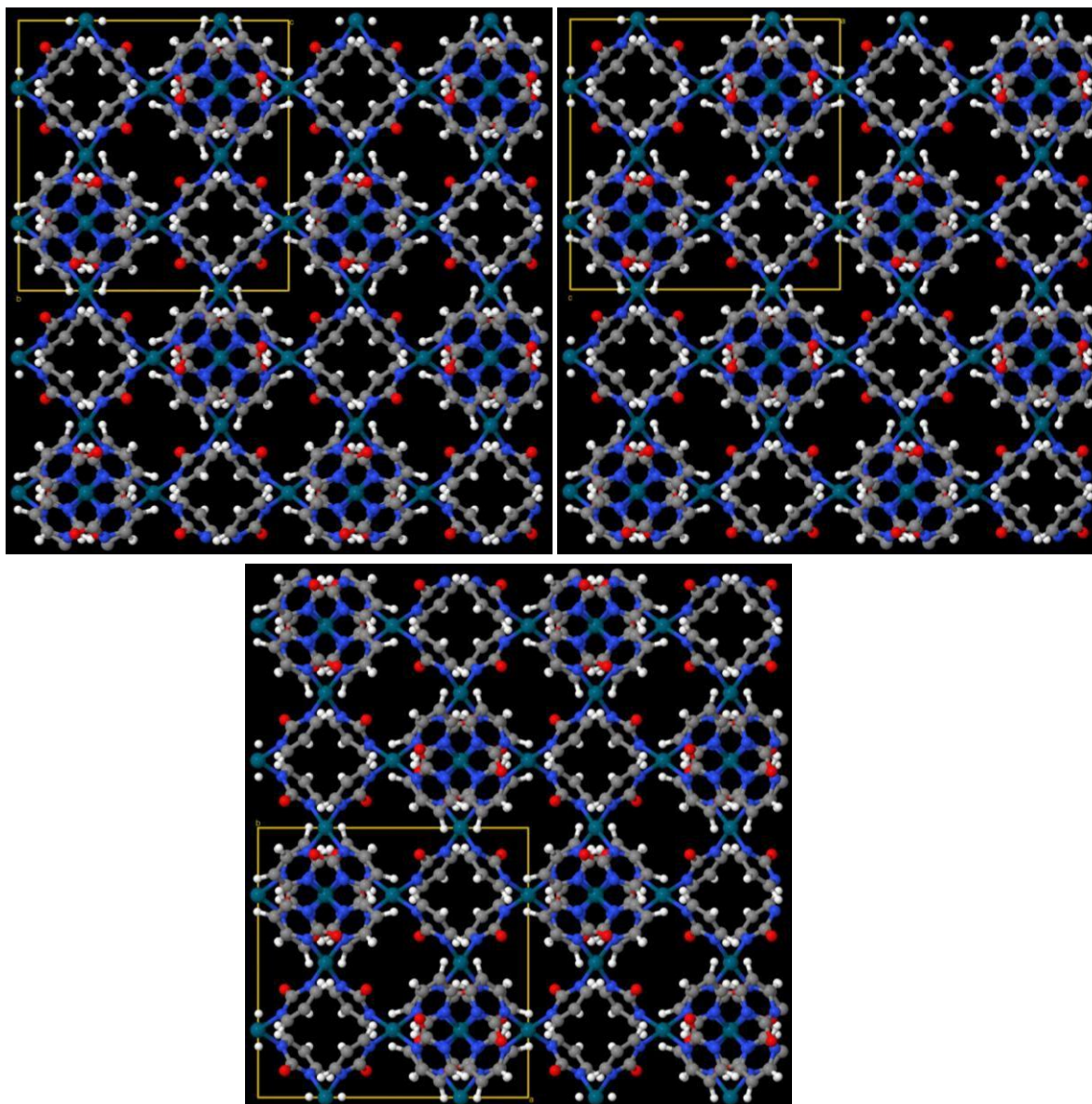


Figure S82: NEKREB contains the chemical elements N, C, H, Pd, and O.

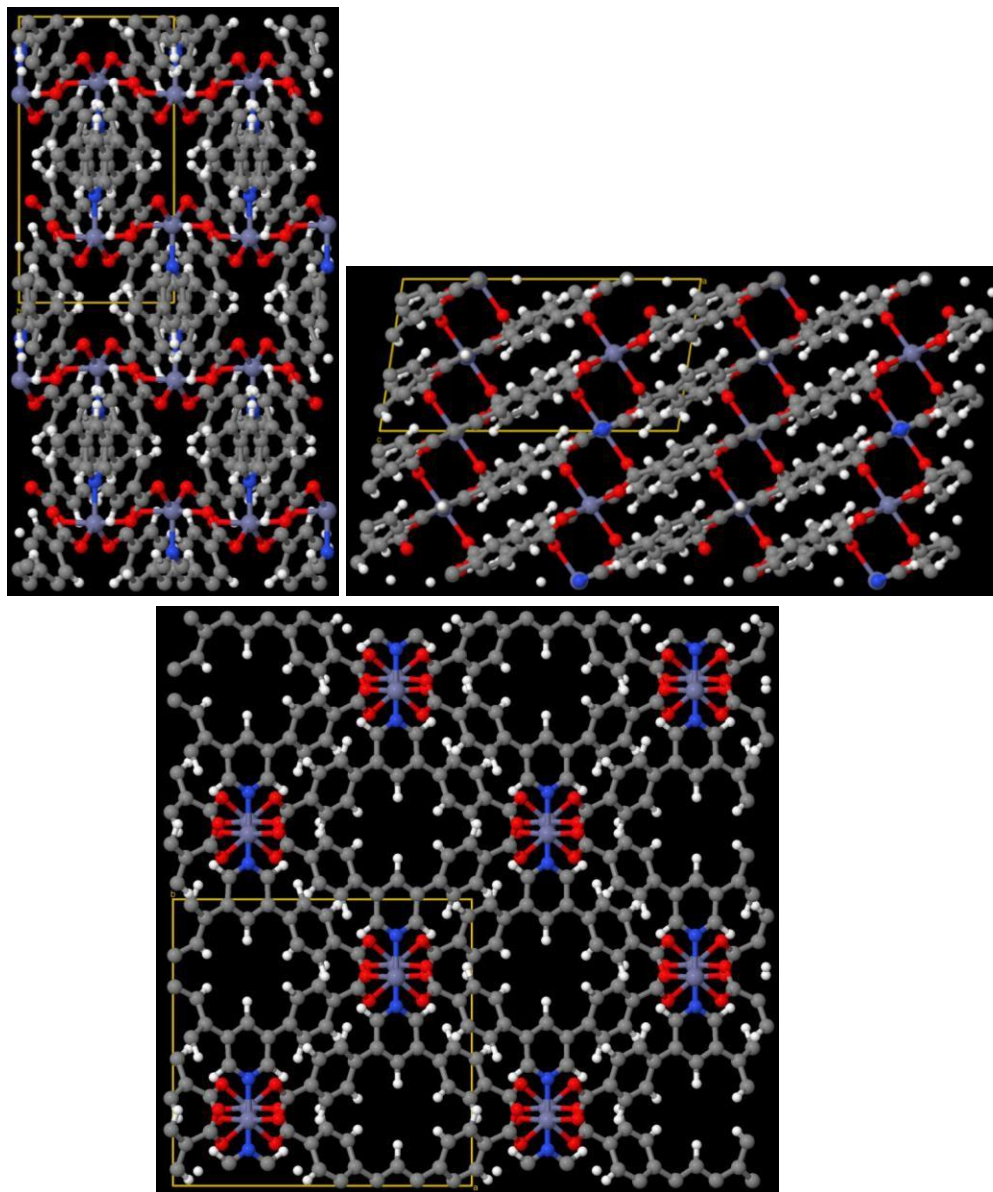


Figure S83: NOMQAJ01 contains the chemical elements C, H, N, O, and Zn.

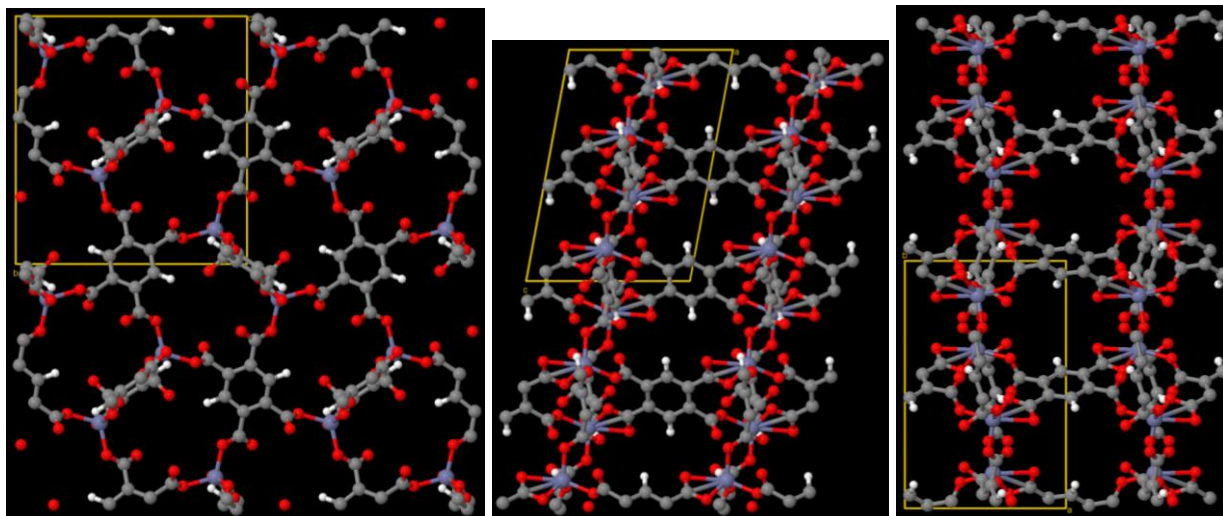


Figure S84: OGIBUD contains the chemical elements Zn, O, C, and H.

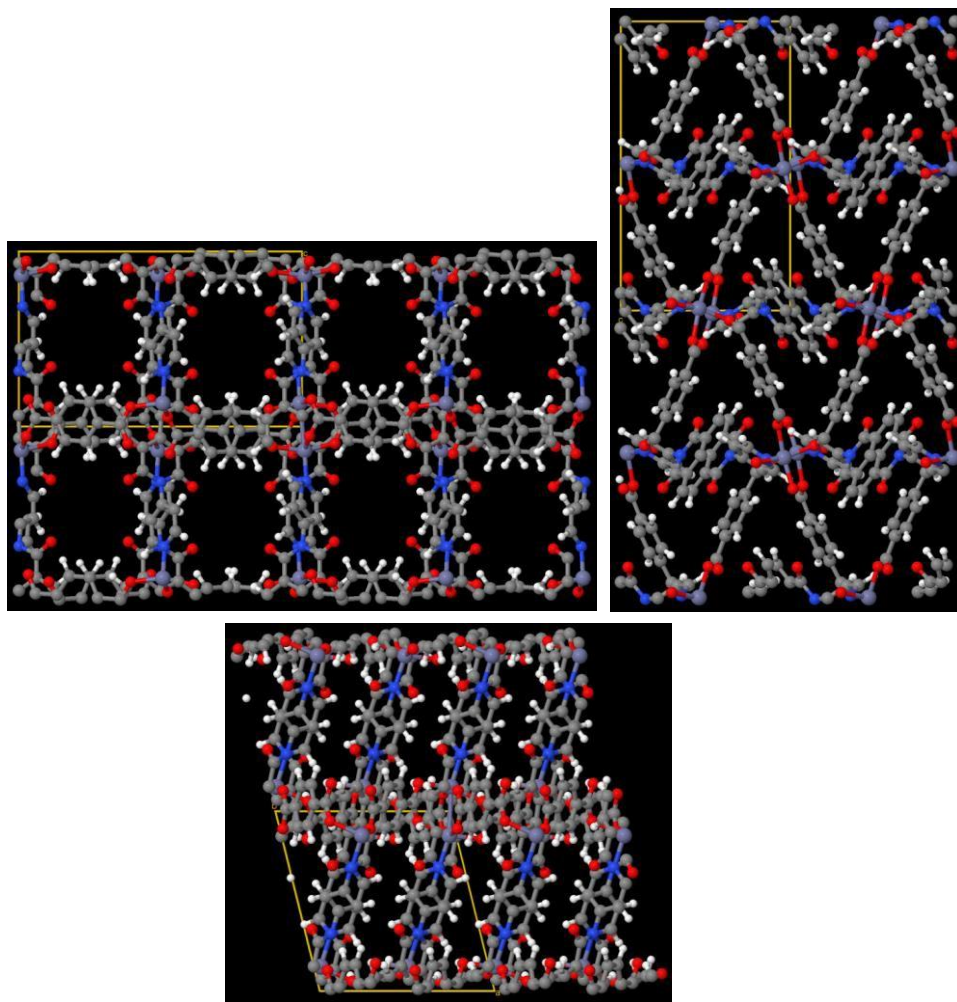


Figure S85: OJIWIO contains the chemical elements Zn, H, C, N, and O.

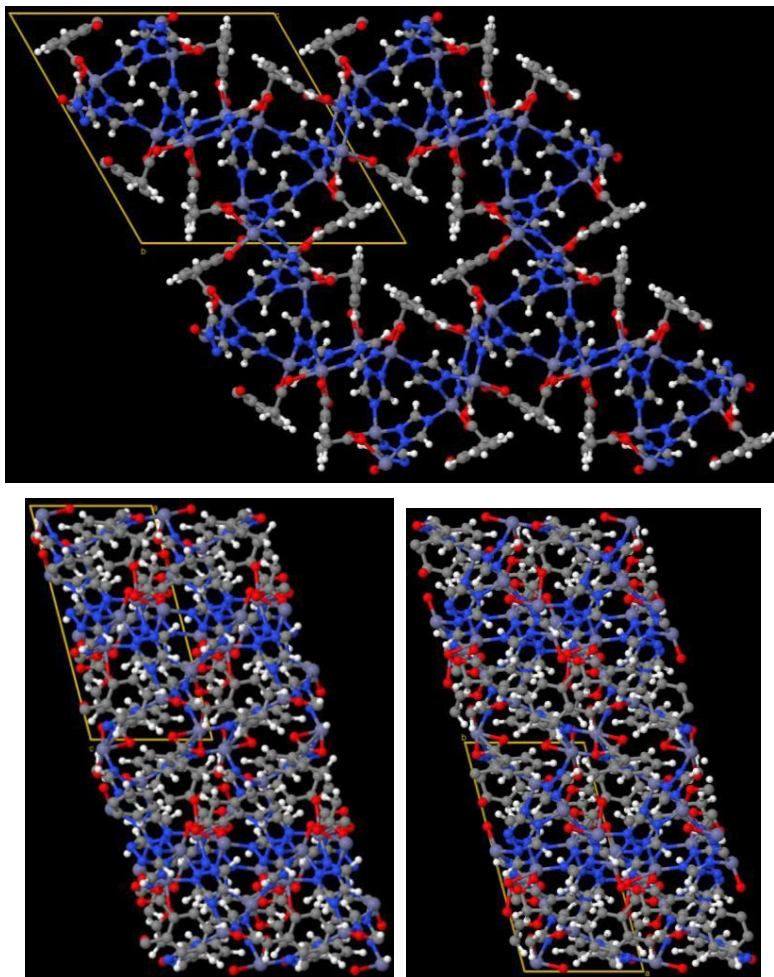


Figure S86: OPOBIF contains the chemical elements Zn, H, C, N, and O.

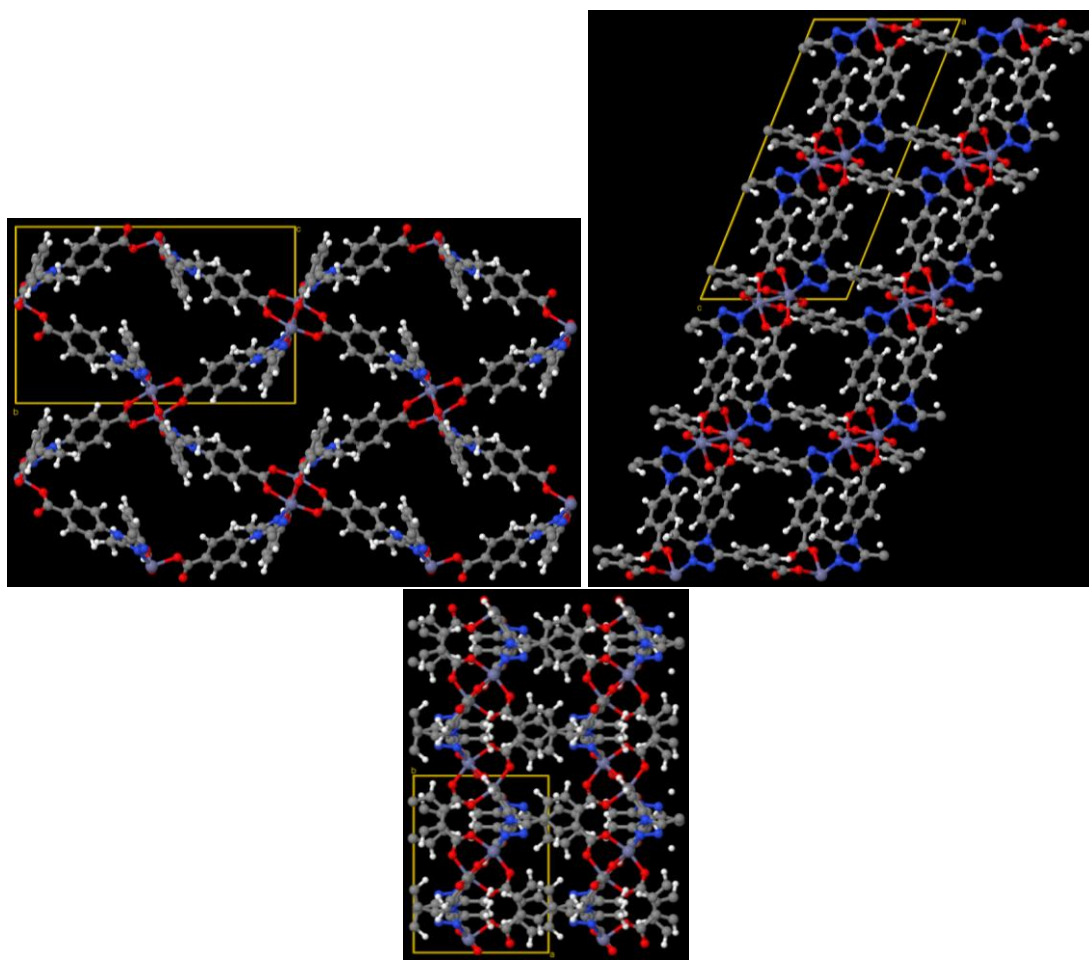


Figure S87: OTOZOO contains the chemical elements Zn, C, H, O, and N.

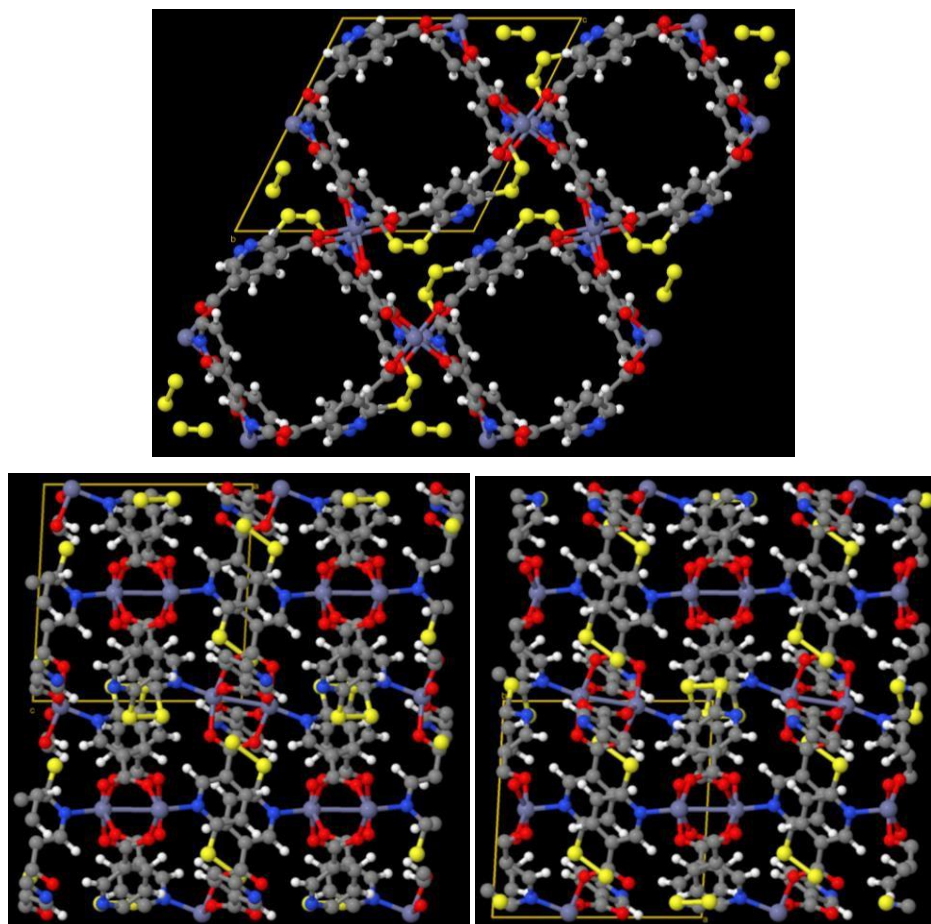


Figure S88: OVICUS contains the chemical elements Zn, H, C, S, N, and O.

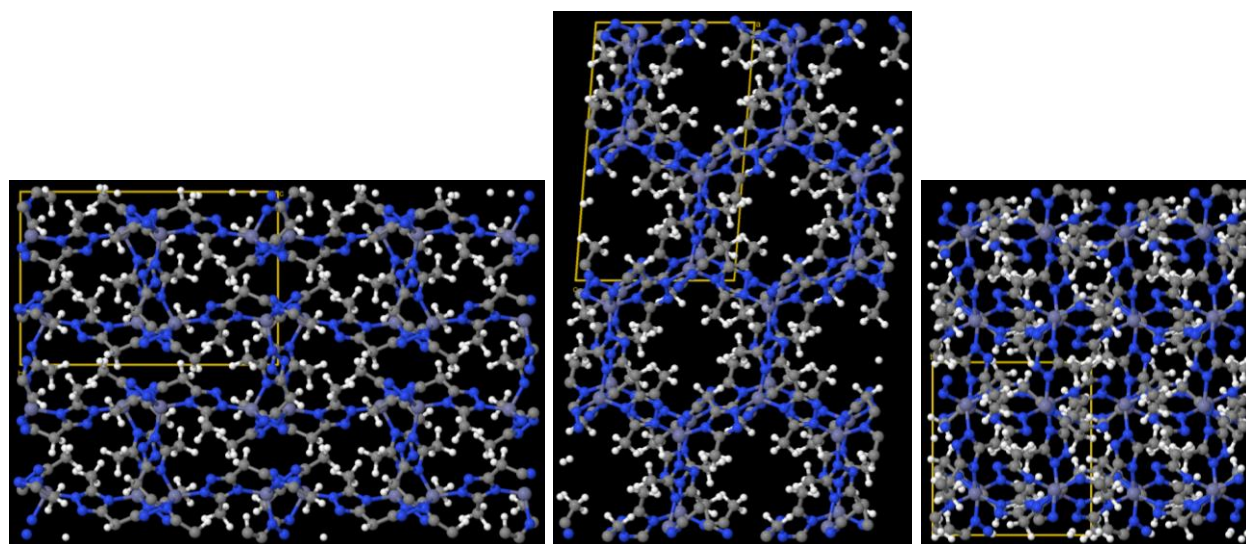


Figure S89: PEGBEK01 contains the chemical elements N, H, Zn, and C.

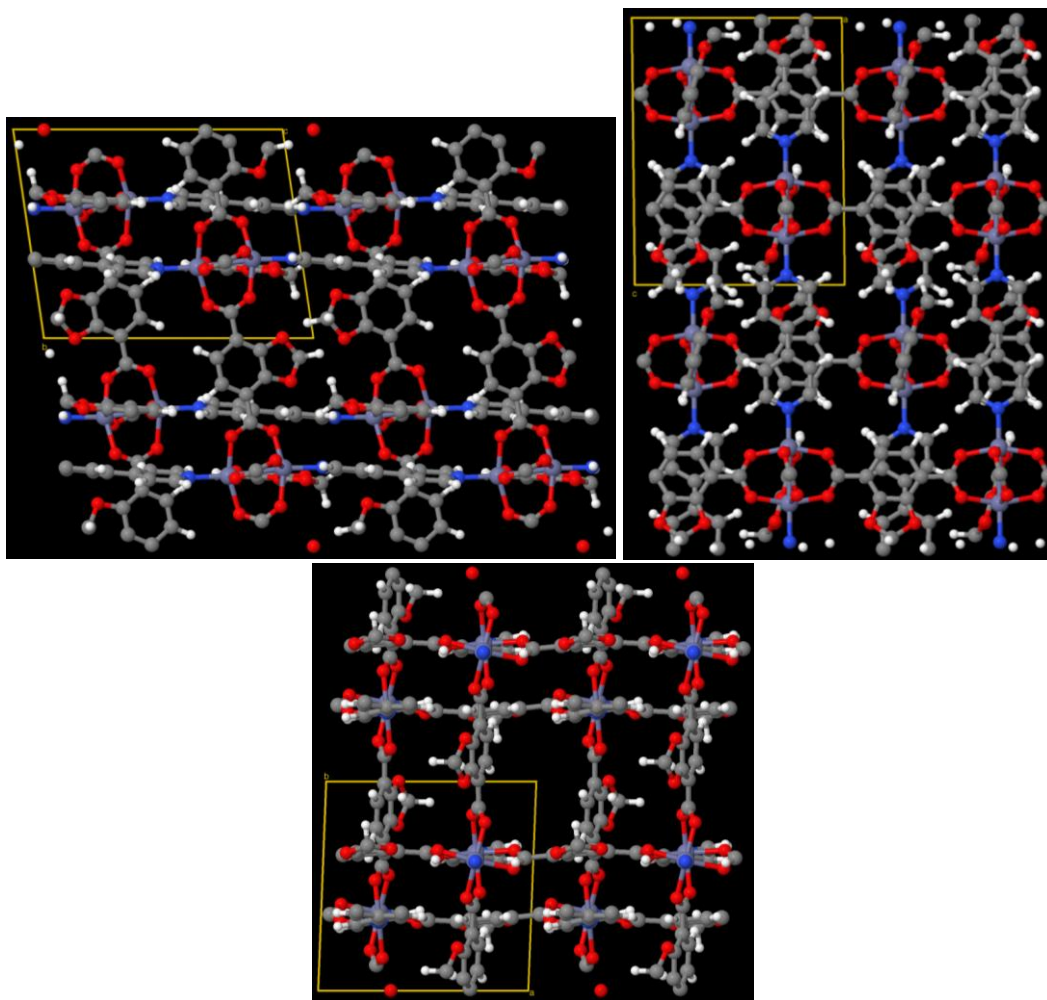


Figure S90: PEYPIU contains the chemical elements Zn, H, C, N, and O.

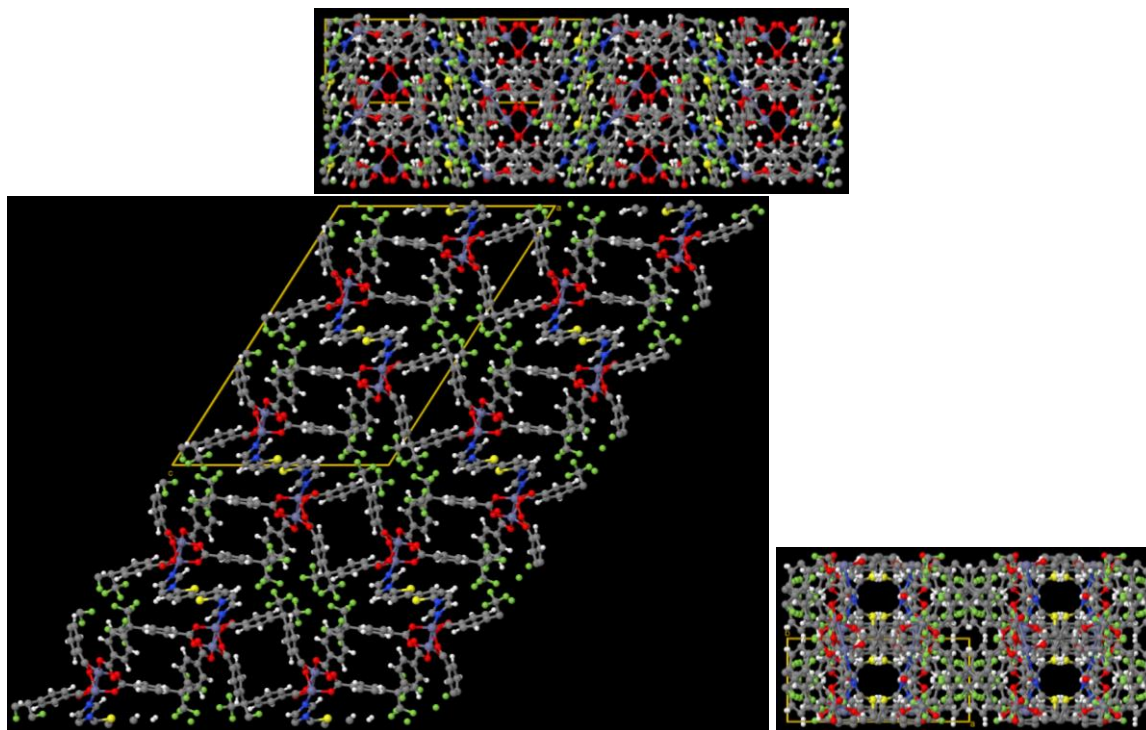


Figure S91: PITMOW contains the chemical elements F, C, H, O, Zn, N, and S.

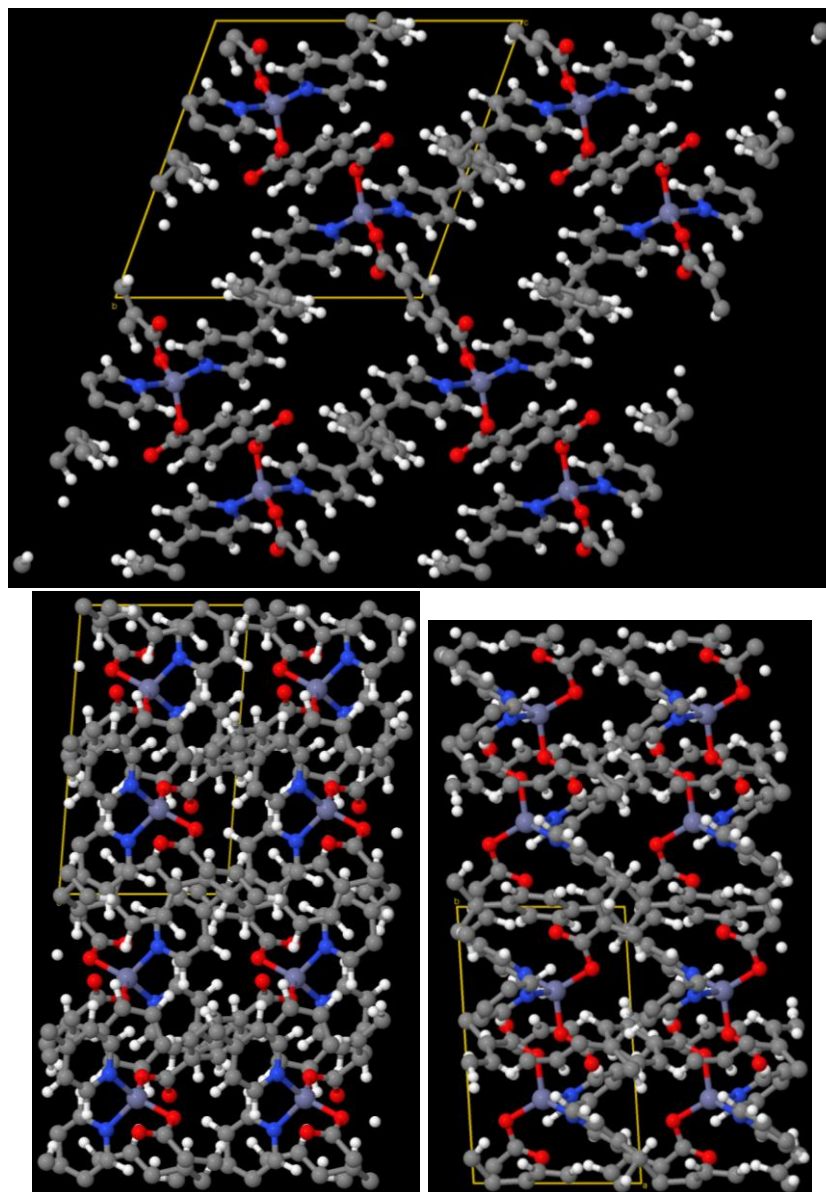


Figure S92: PIXHUB contains the chemical elements O, C, Zn, H, and N.

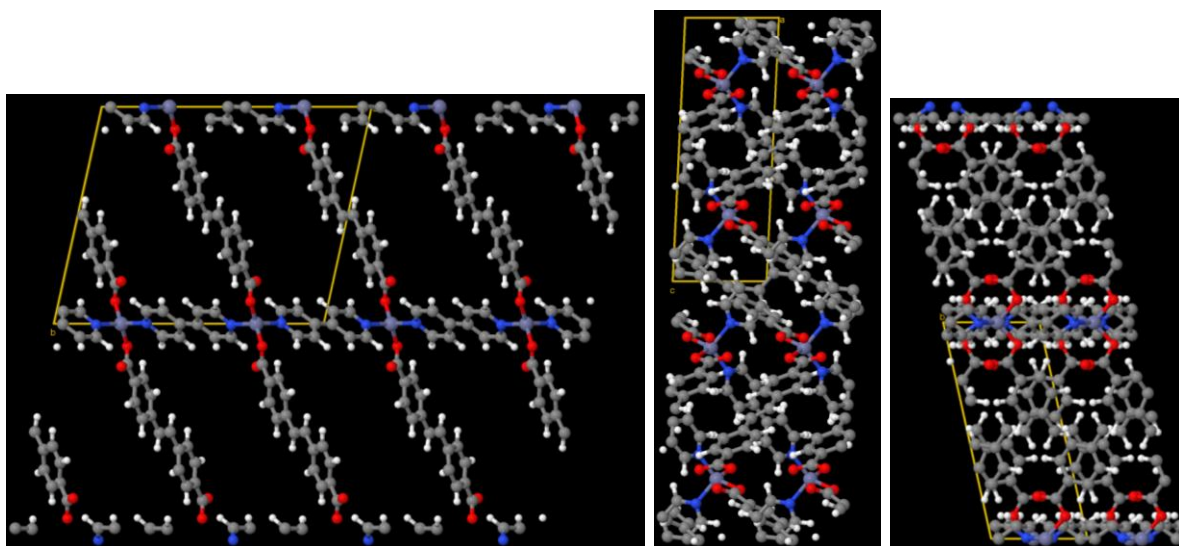


Figure S93: PORVUO contains the chemical elements Zn, H, C, N, and O.

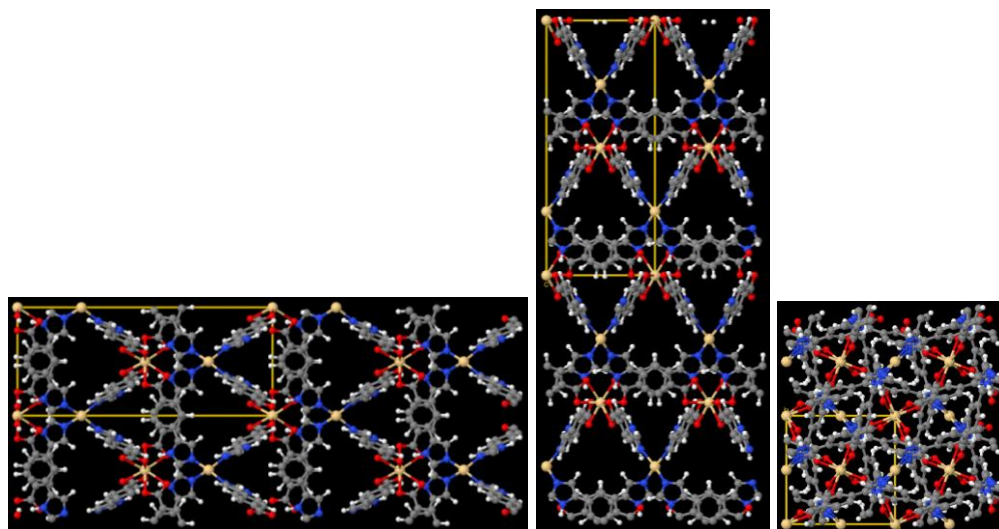


Figure S94: POSFOU contains the chemical elements Cd, H, C, N, and O.

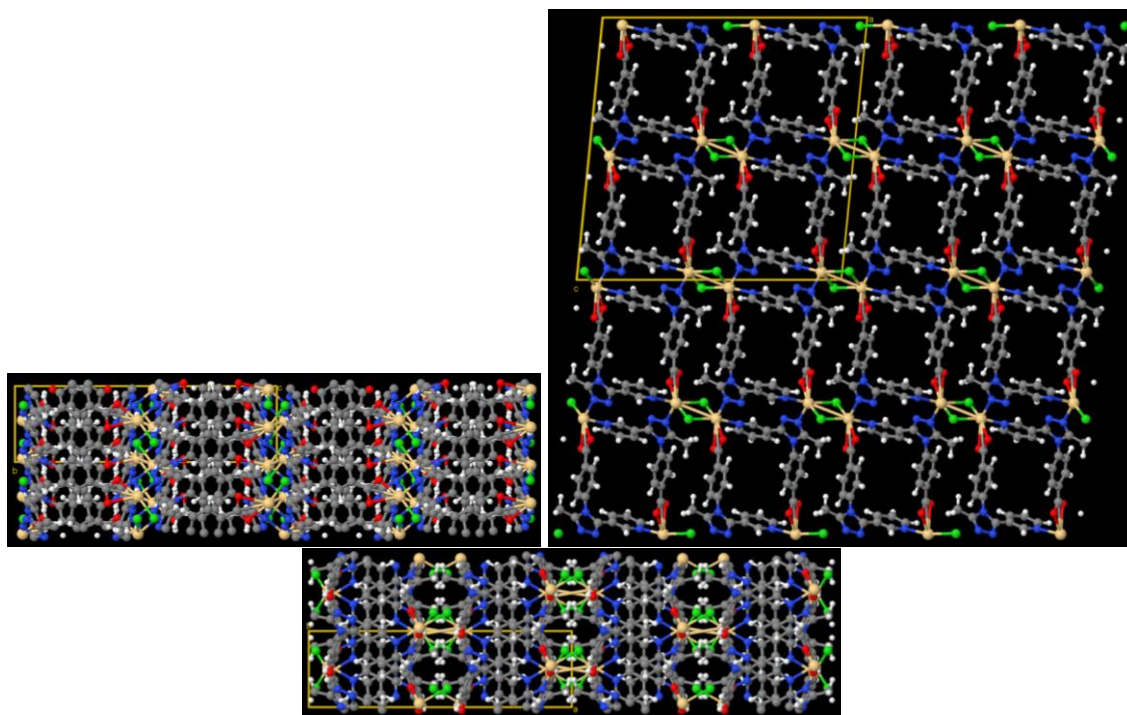


Figure S95: PUFVUJ contains the chemical elements C, H, Cd, Cl, N, and O.

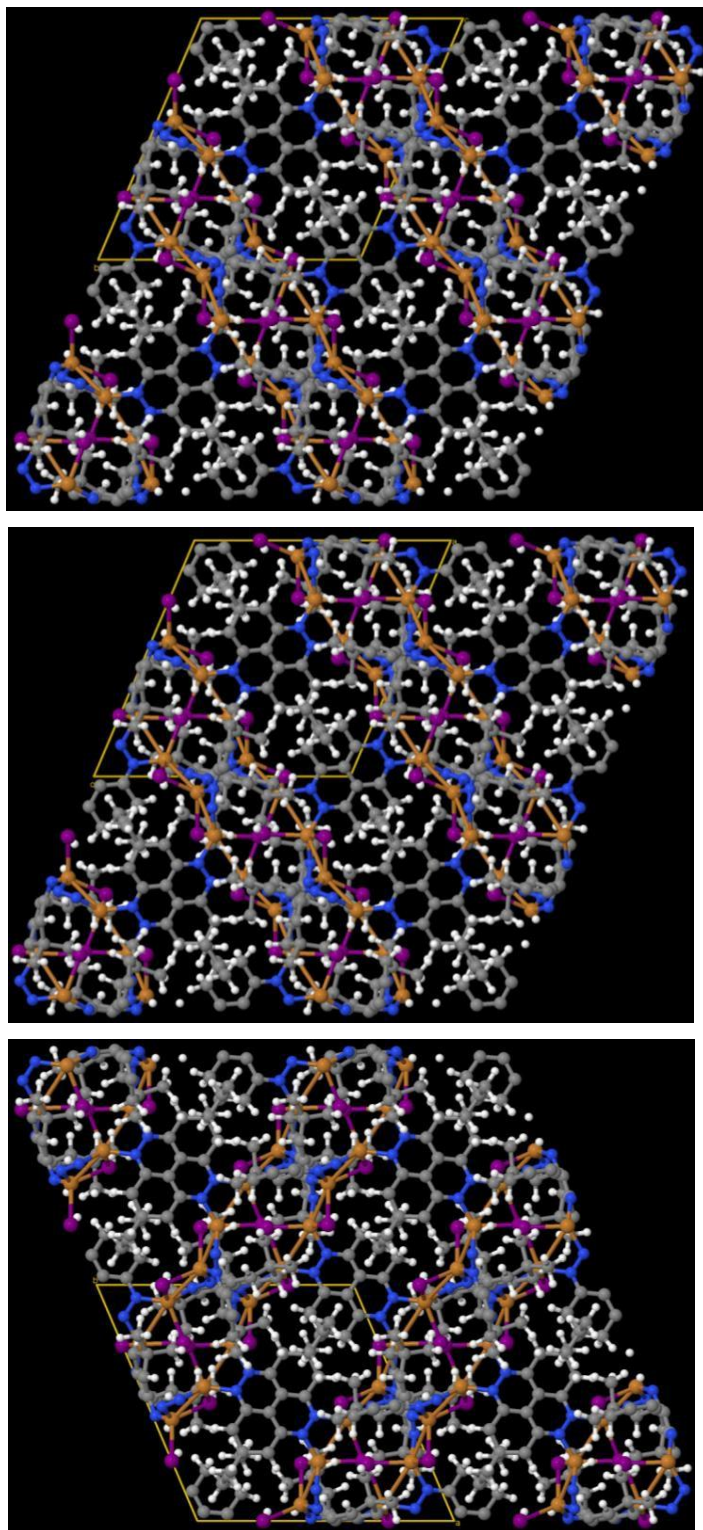


Figure S96: QESYOD contains the chemical elements Cu, H, C, I, and N.

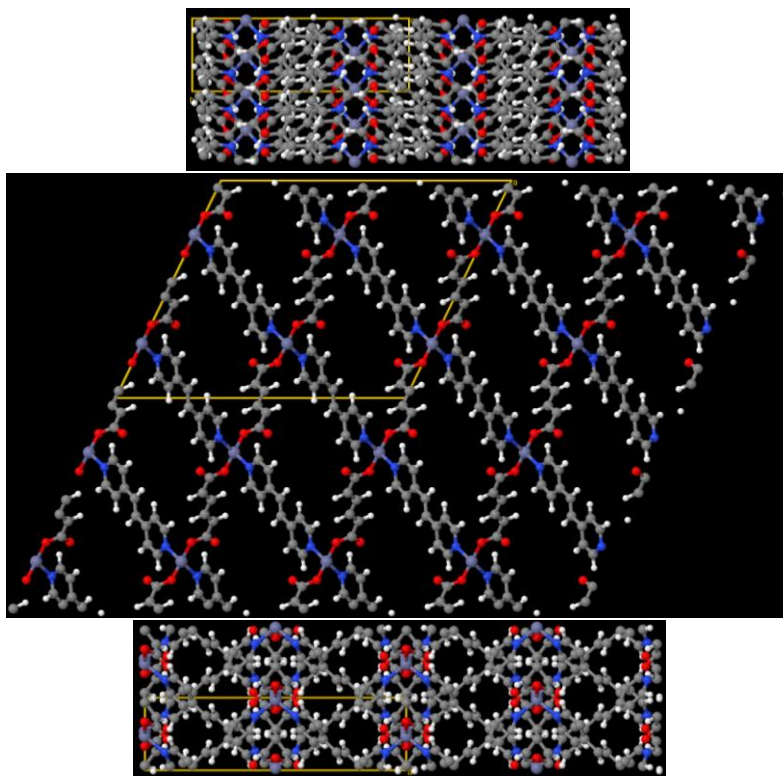


Figure S97: QIVYUR contains the chemical elements C, H, O, N, and Zn.

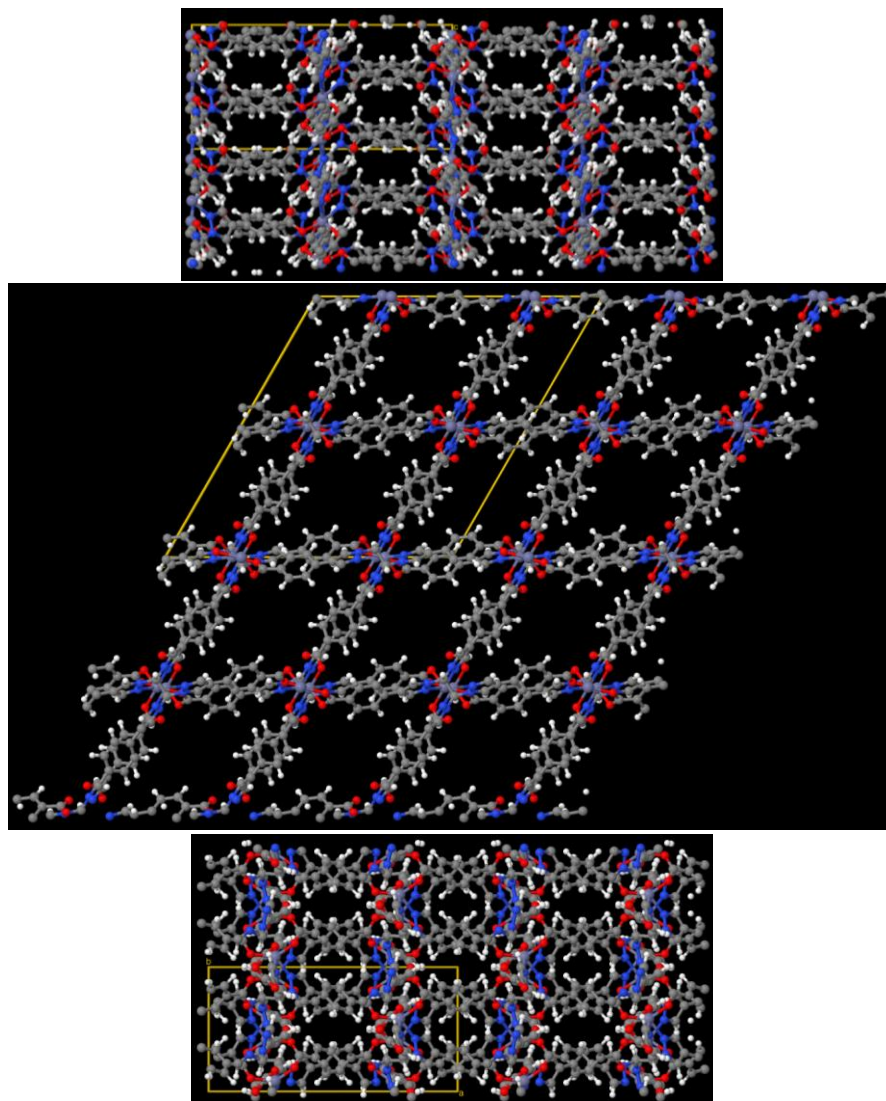


Figure S98: QIWCUW contains the chemical elements O, H, Zn, N, and C.

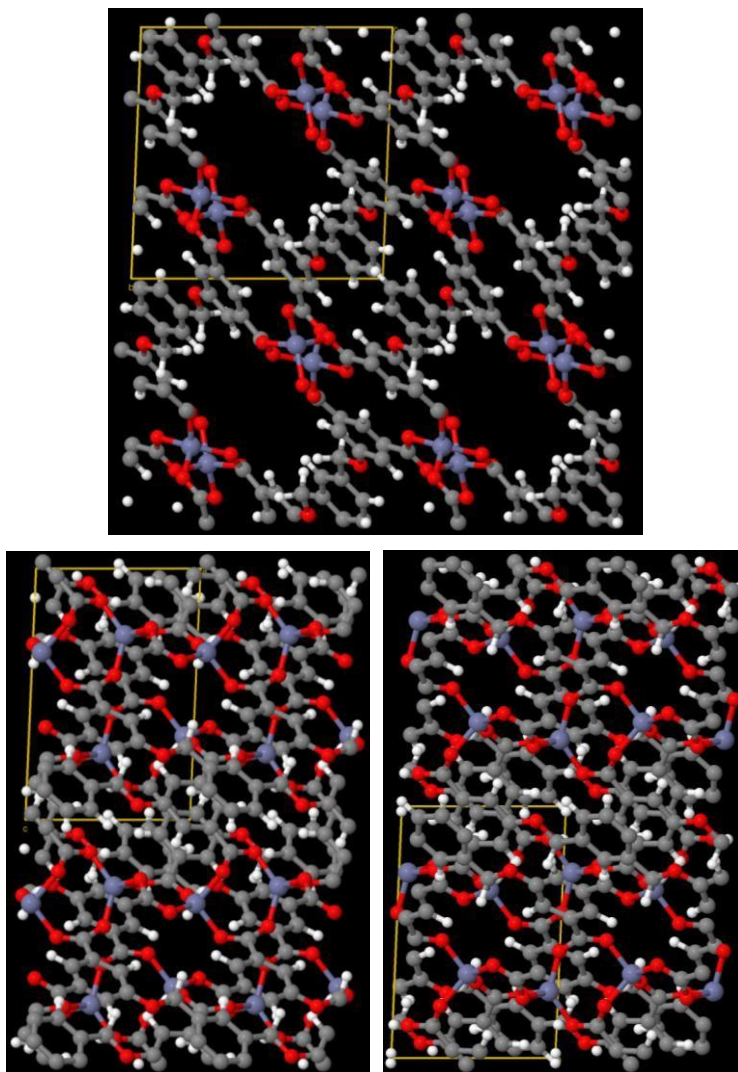


Figure S99: QOKCID contains the chemical elements Zn, H, C, and O.

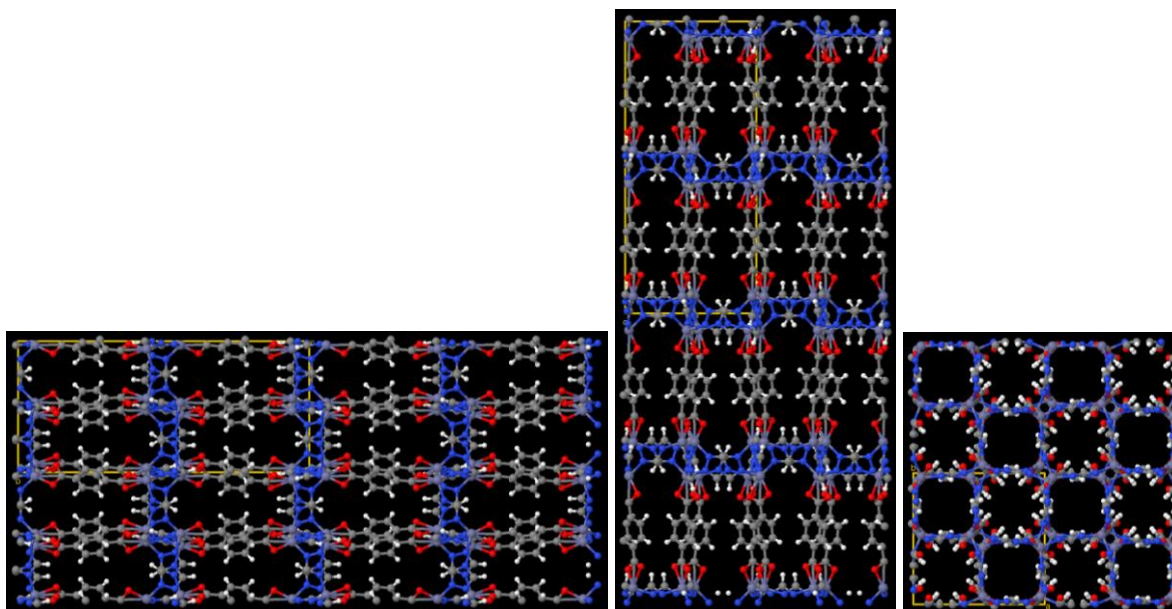


Figure S100: QOTBAE contains the chemical elements N, C, H, Zn, and O.

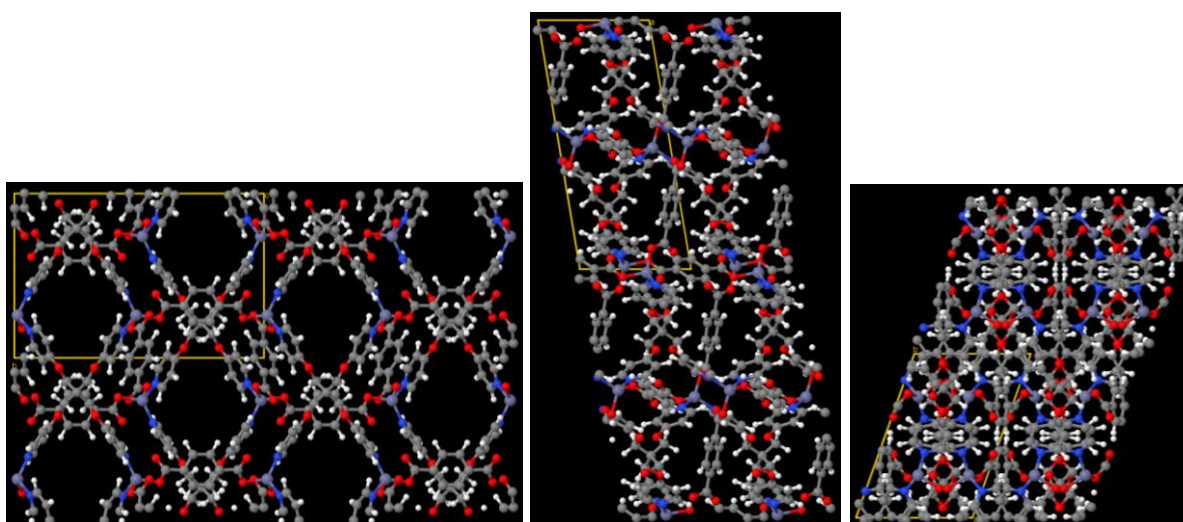


Figure S101: QUPHUF contains the chemical elements Zn, H, C, N, and O.

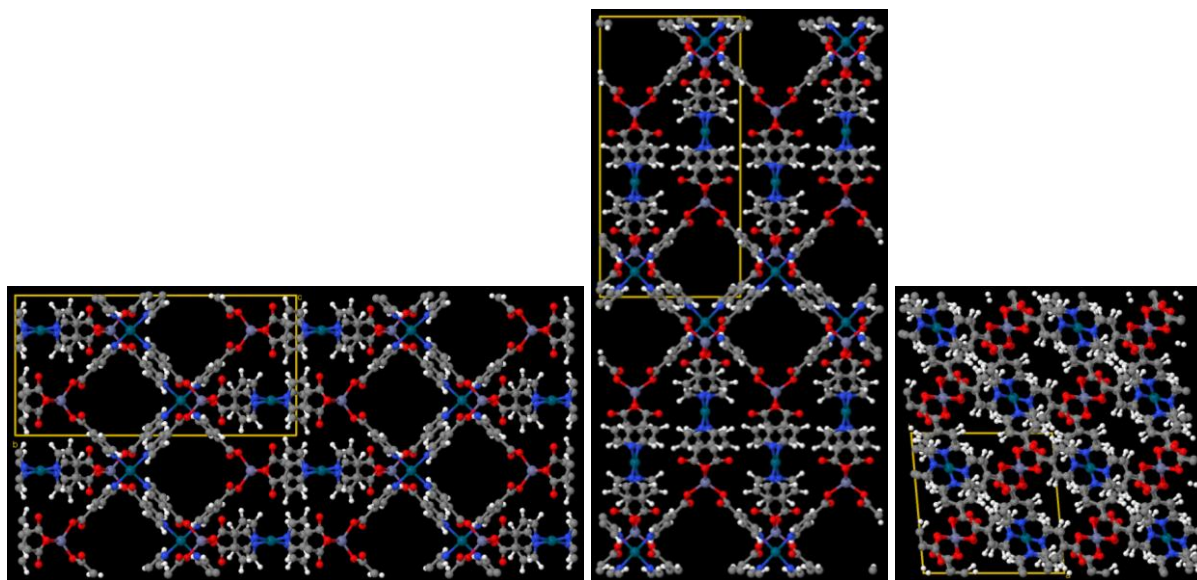


Figure S102: QURSEC contains the chemical elements Zn, H, Pd, C, N, and O.

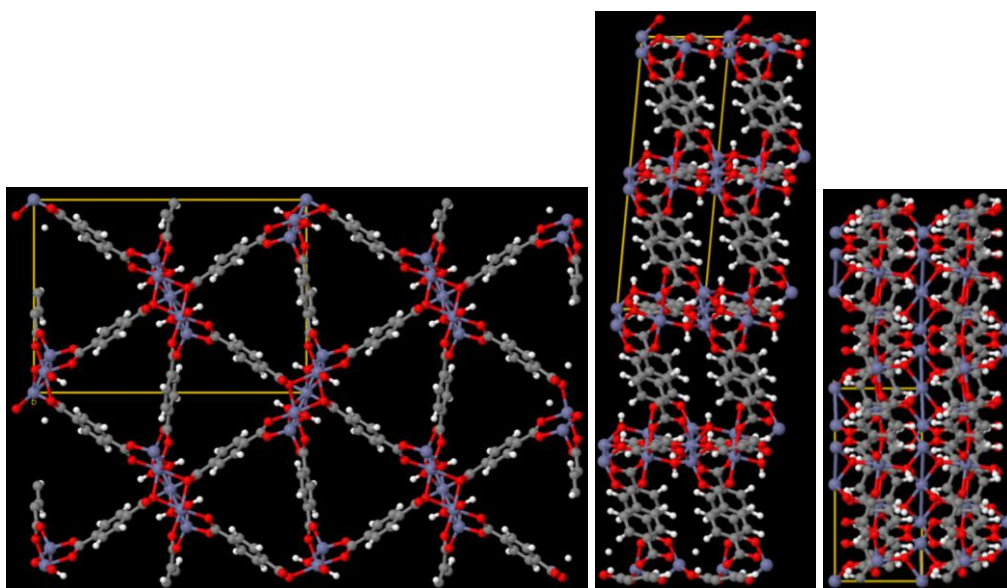


Figure S103: REDROI contains the chemical elements Zn, H, C, and O.

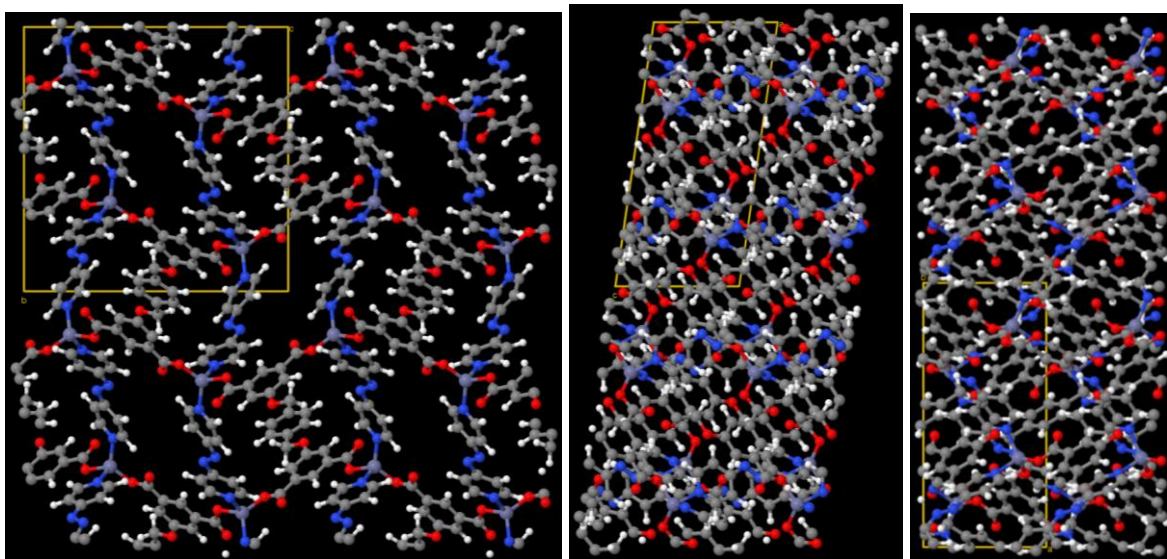


Figure S104: REYBAA contains the chemical elements Zn, H, C, N, and O.

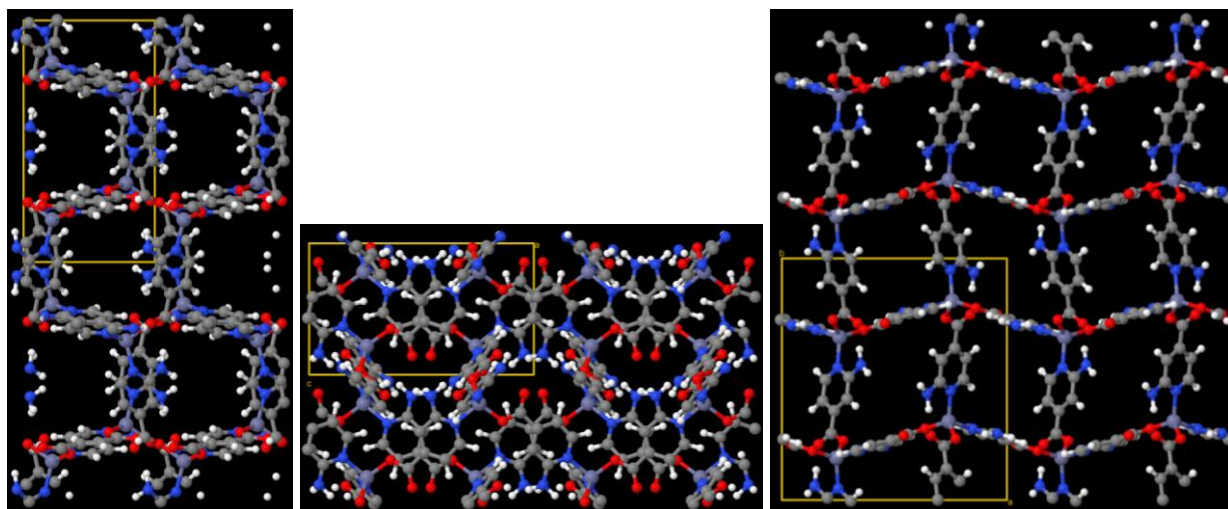


Figure S105: SARBOE contains the chemical elements Zn, O, C, H, and N.

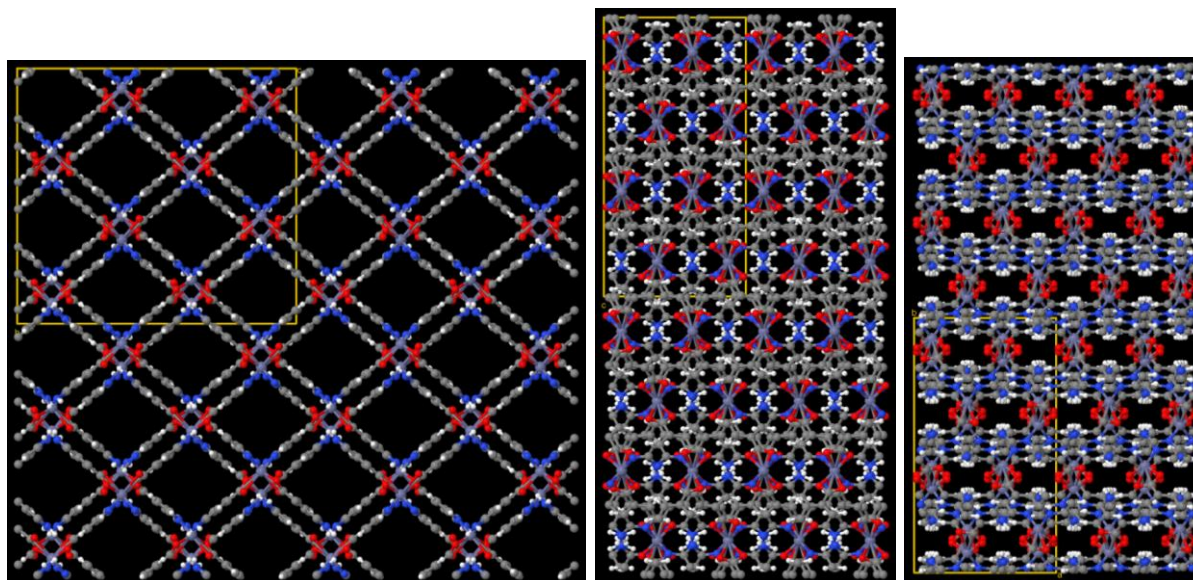


Figure S106: SARBUK contains the chemical elements Zn, C, H, N, and O.

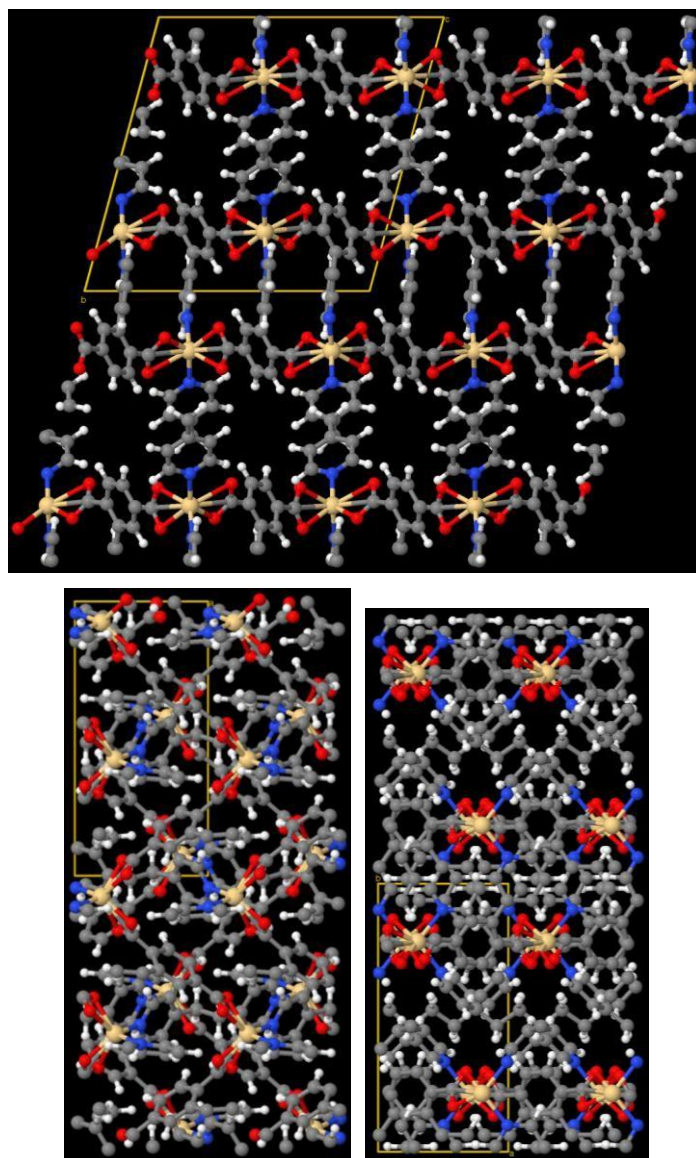


Figure S107: SOCLAZ contains the chemical elements Cd, H, C, N, and O.

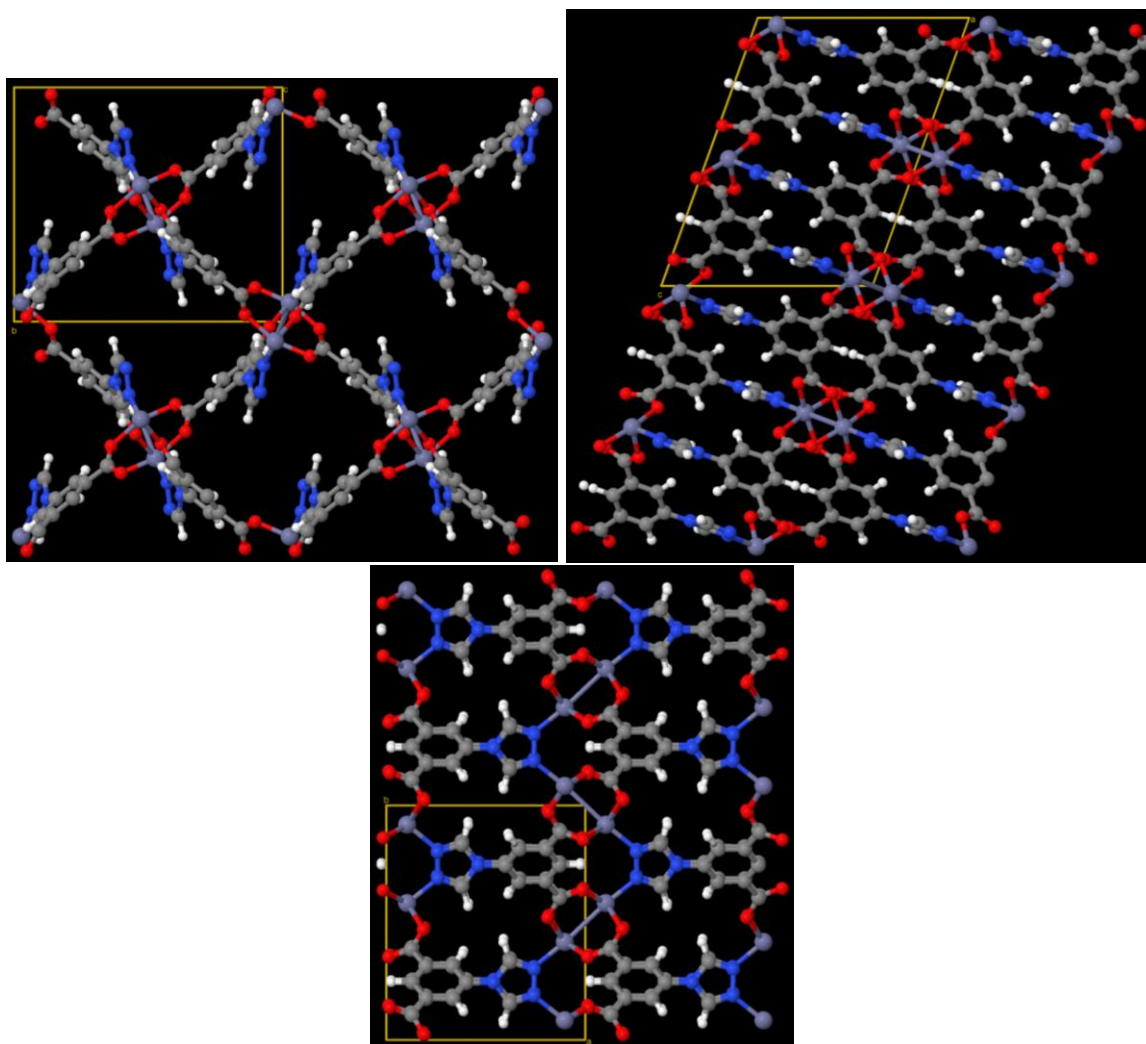


Figure S108: SOWCOY01 contains the chemical elements Zn, O, N, C, and H.

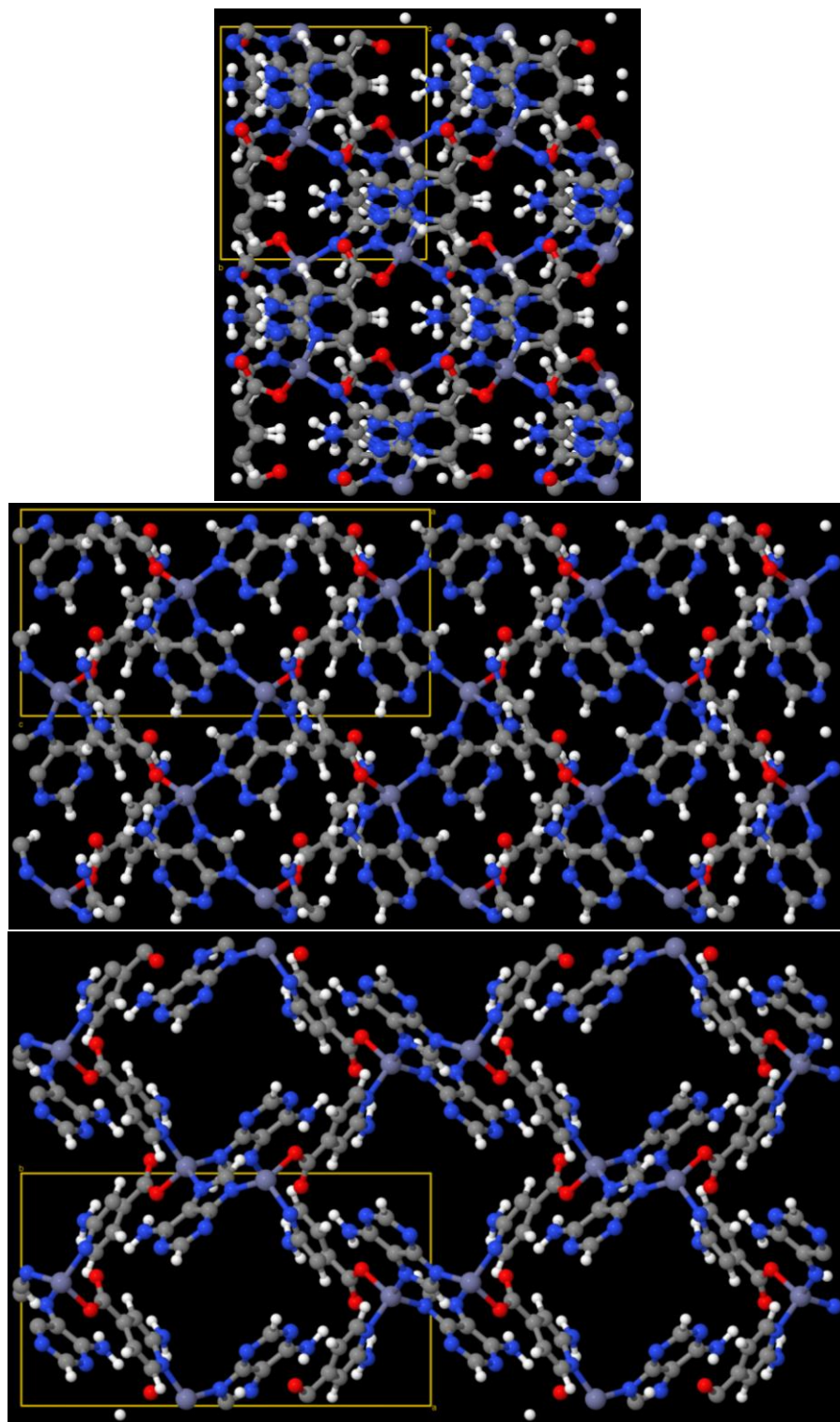


Figure S109: TEPGUS contains the chemical elements O, Zn, C, H, and N.

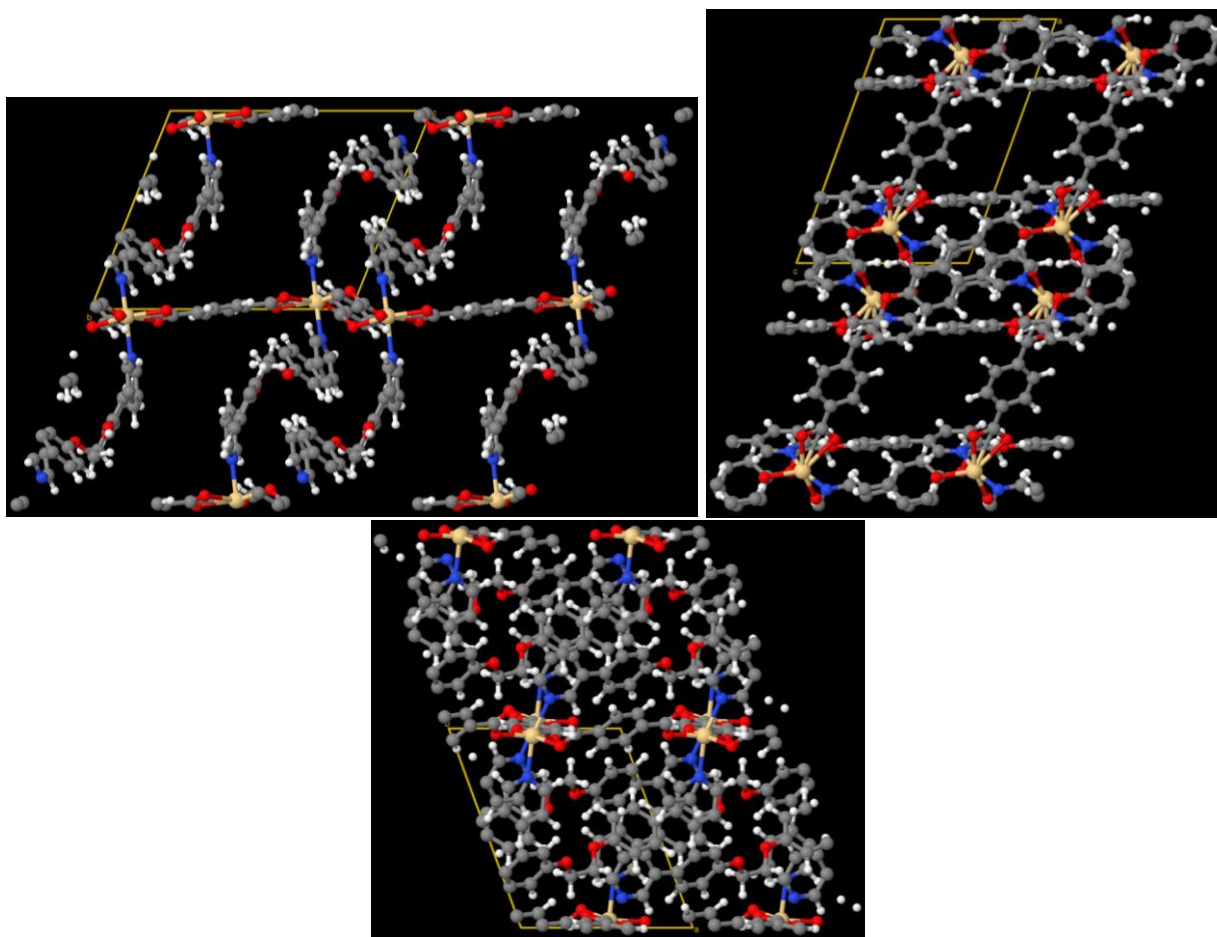


Figure S110: TUBLIN contains the chemical elements Cd, H, C, N, and O.

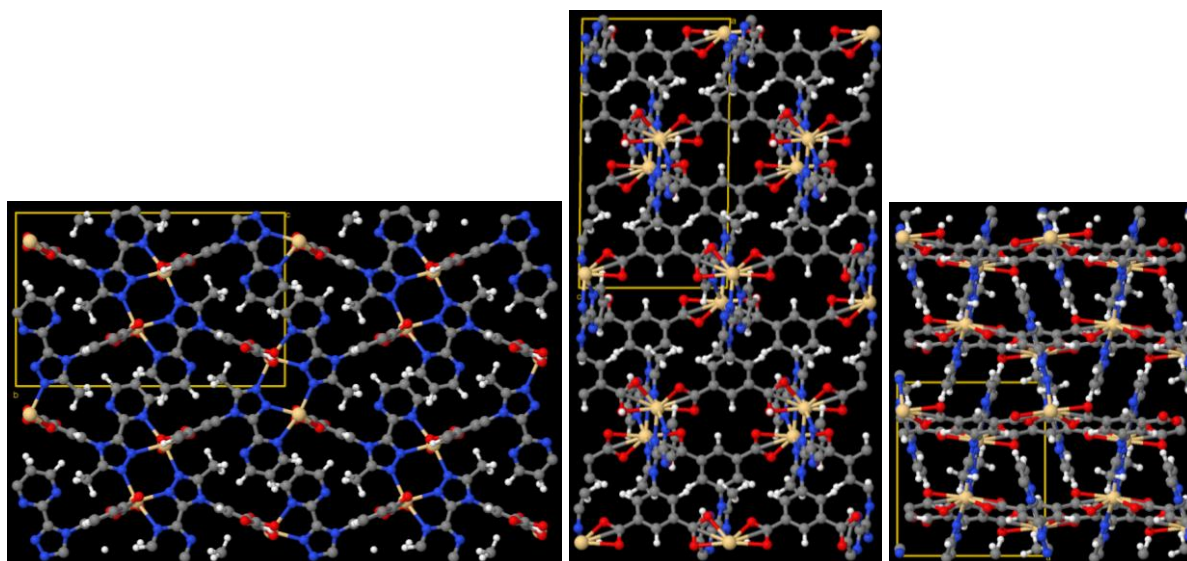


Figure S111: TUMDEM contains the chemical elements C, H, N, O, and Cd.

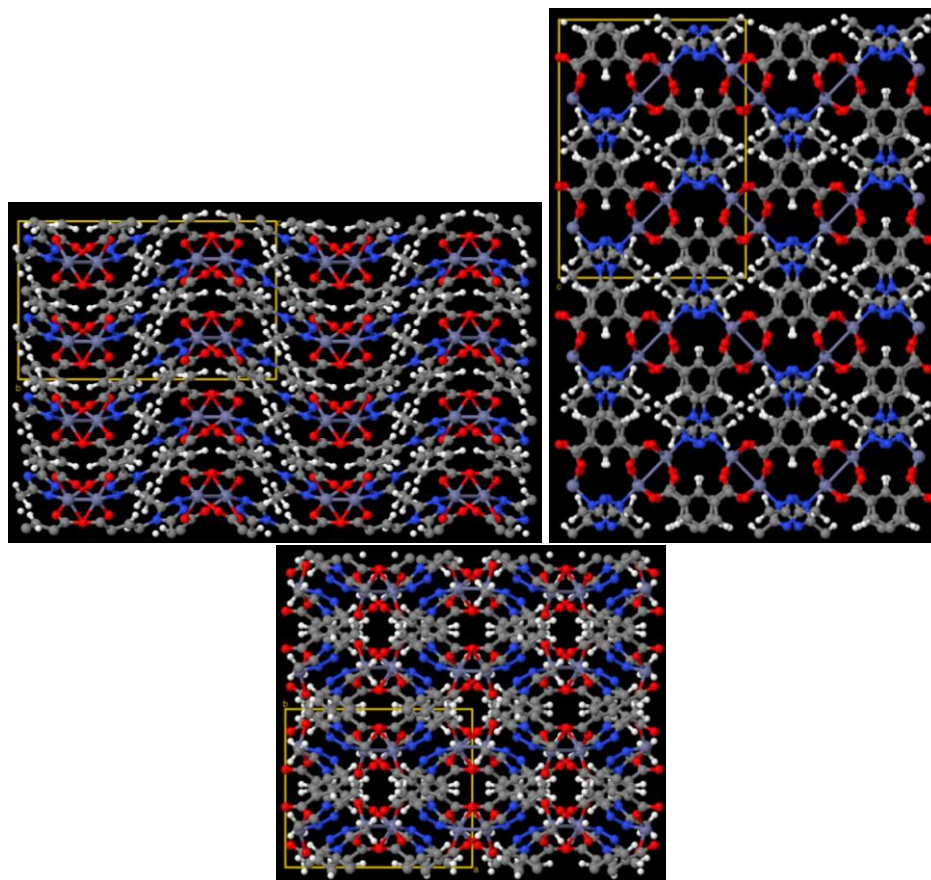


Figure S112: UKALEZ contains the chemical elements C, H, N, O, and Zn.

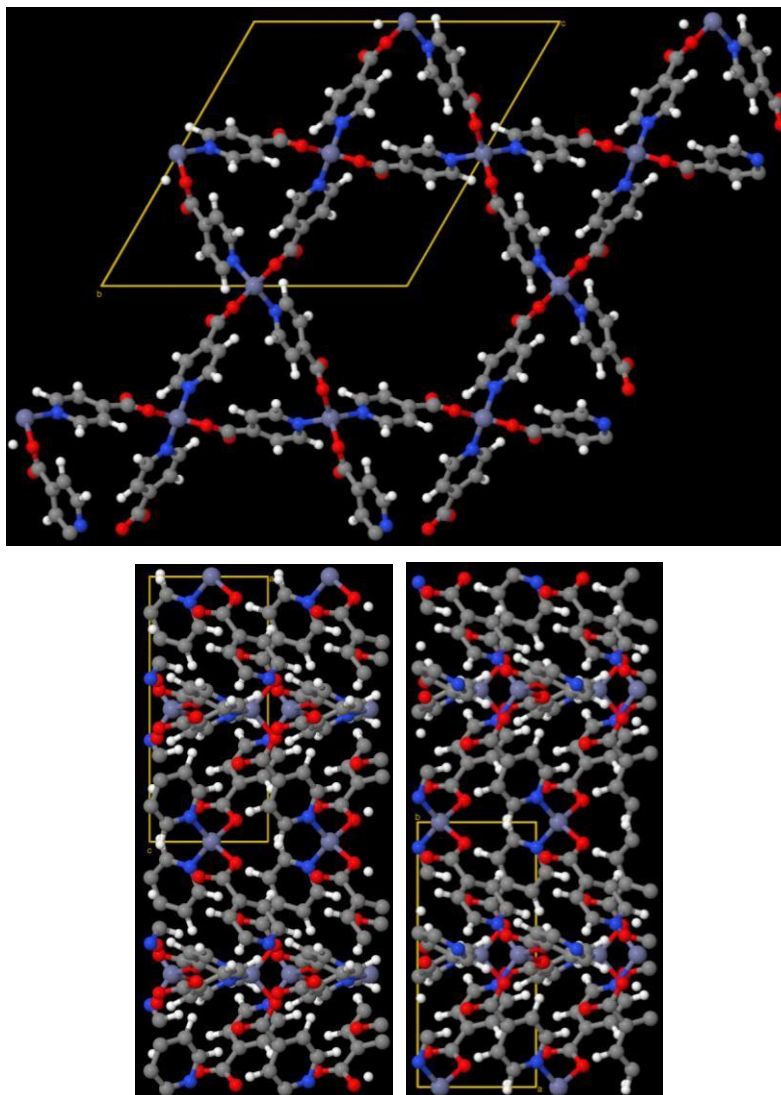


Figure S113: VACFUB01 contains the chemical elements Zn, H, C, N, and O.

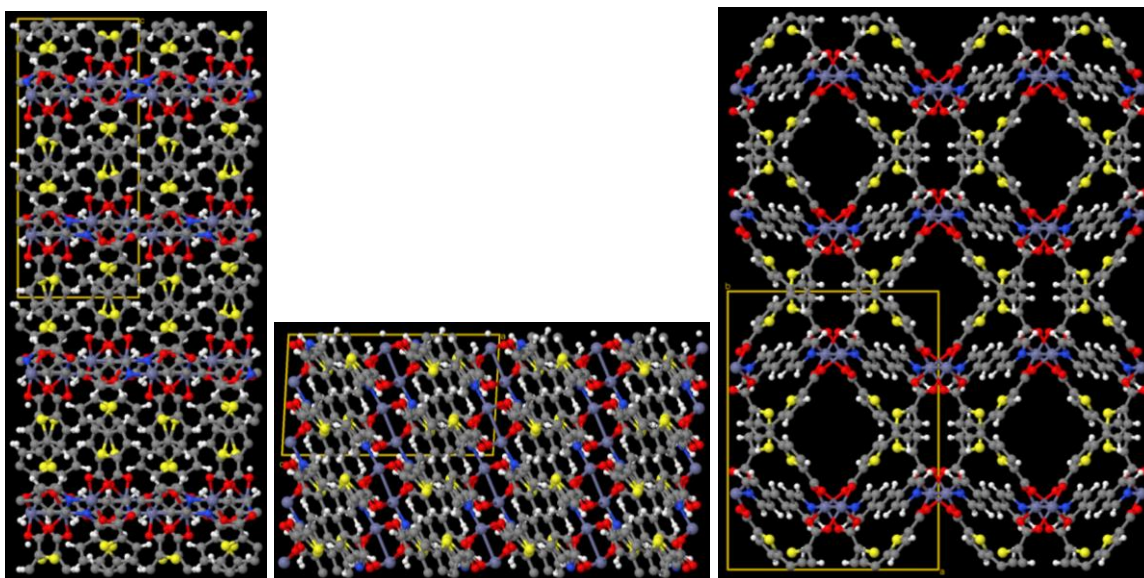


Figure S114: WECJAR contains the chemical elements Zn, C, H, N, O, and S.

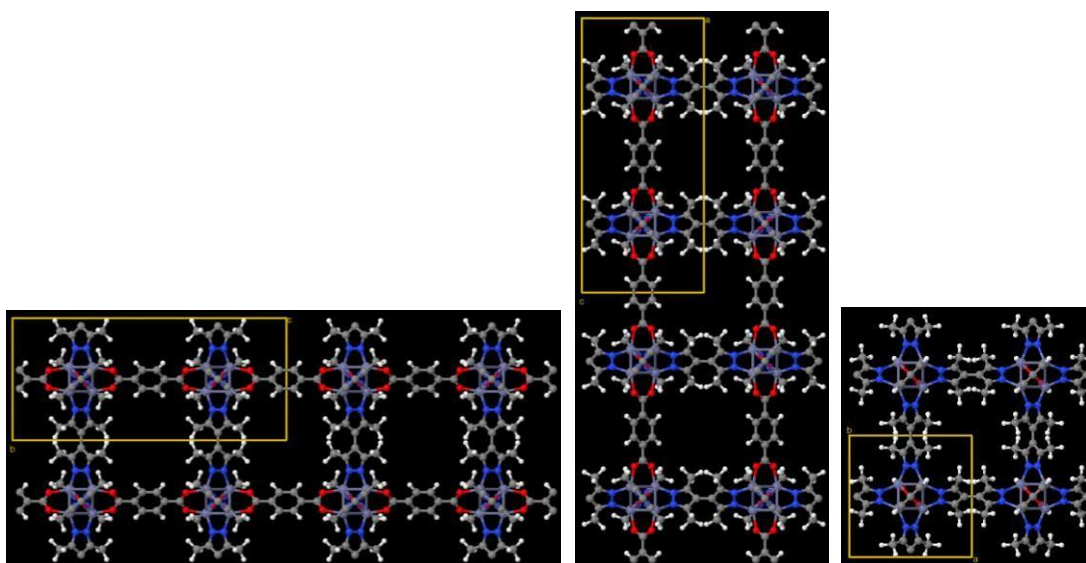


Figure S115: WIYFAM contains the chemical elements Zn, H, C, N, and O.

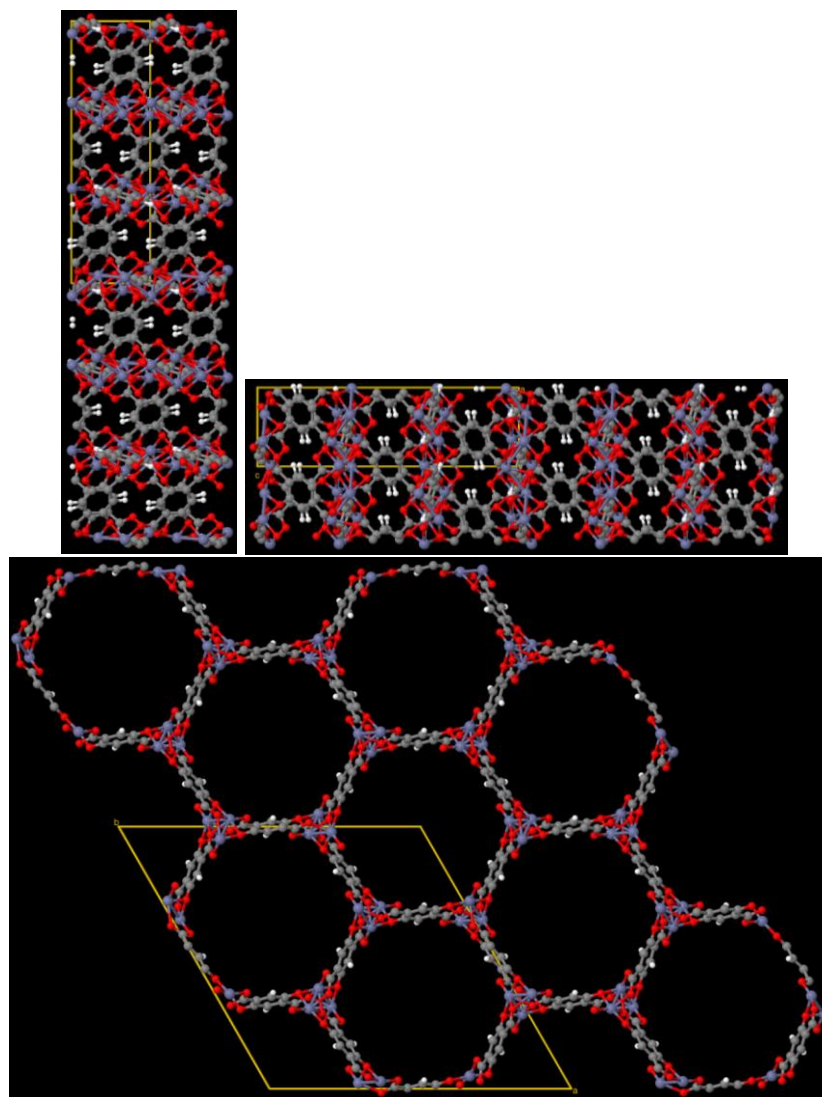


Figure S116: WOBHIF01 contains the chemical elements O, Zn, C, and H.

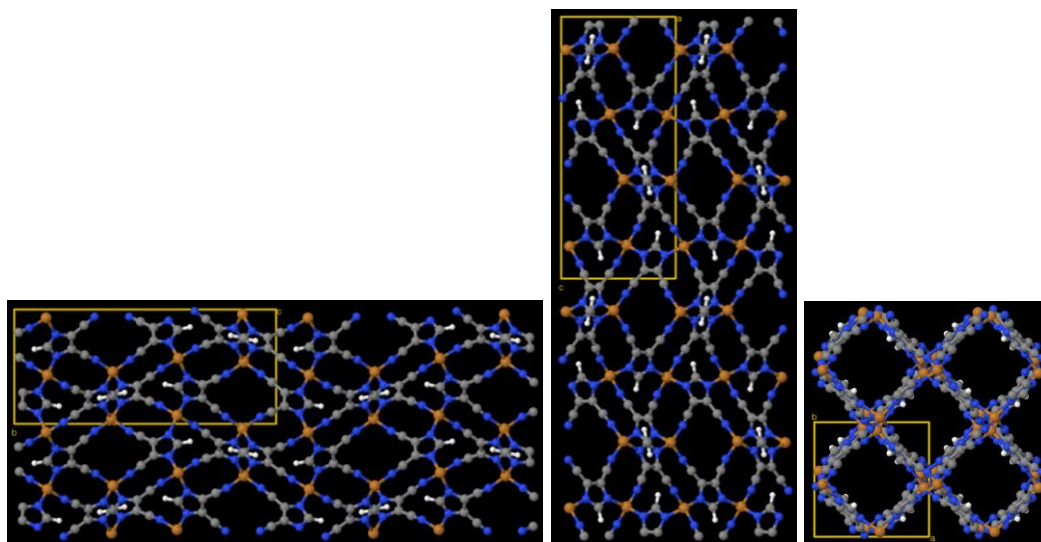


Figure S117: XAHROQ contains the chemical elements Cu, N, C, and H.

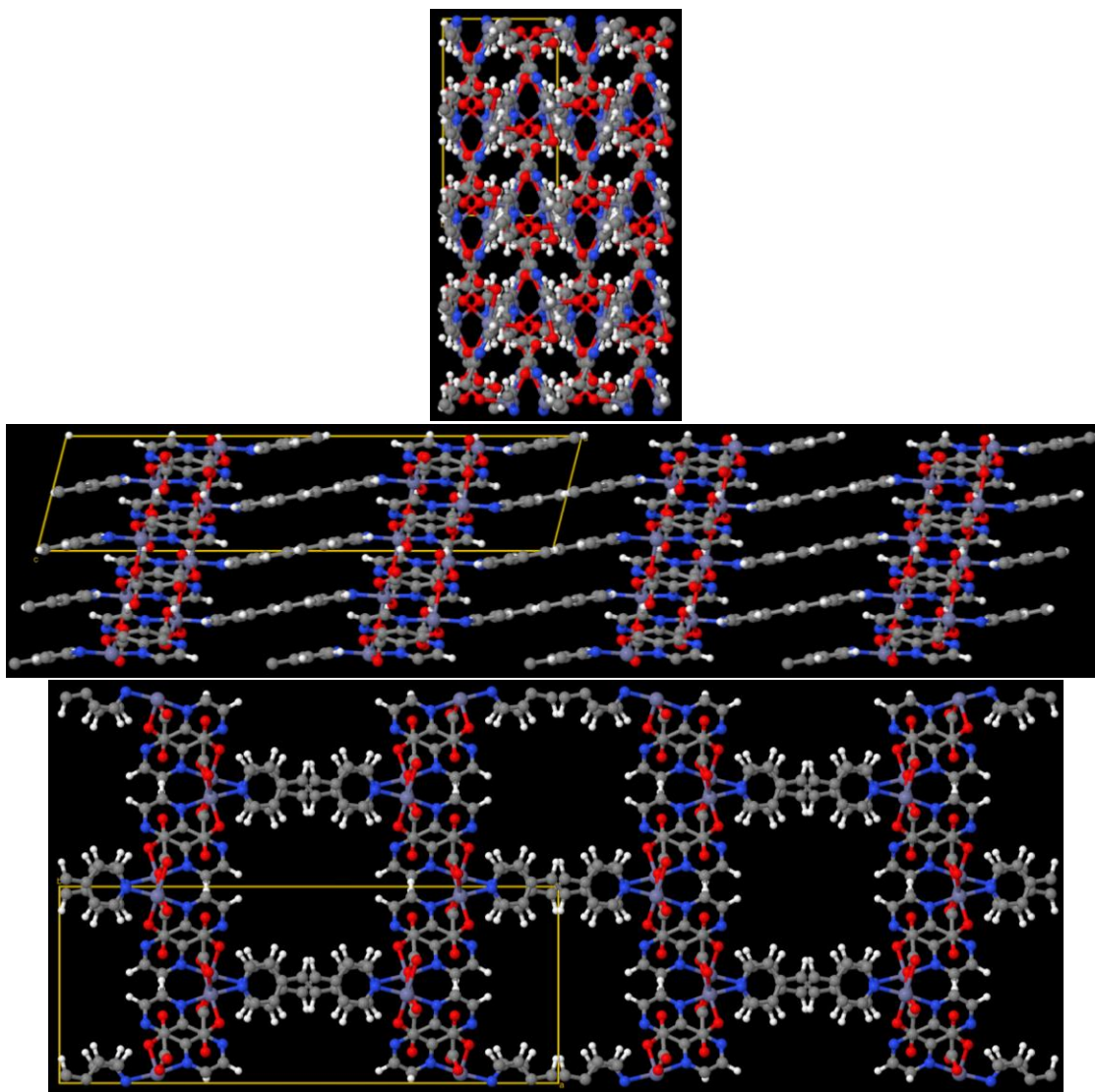


Figure S118: XARCEB contains the chemical elements H, Zn, C, N, and O.

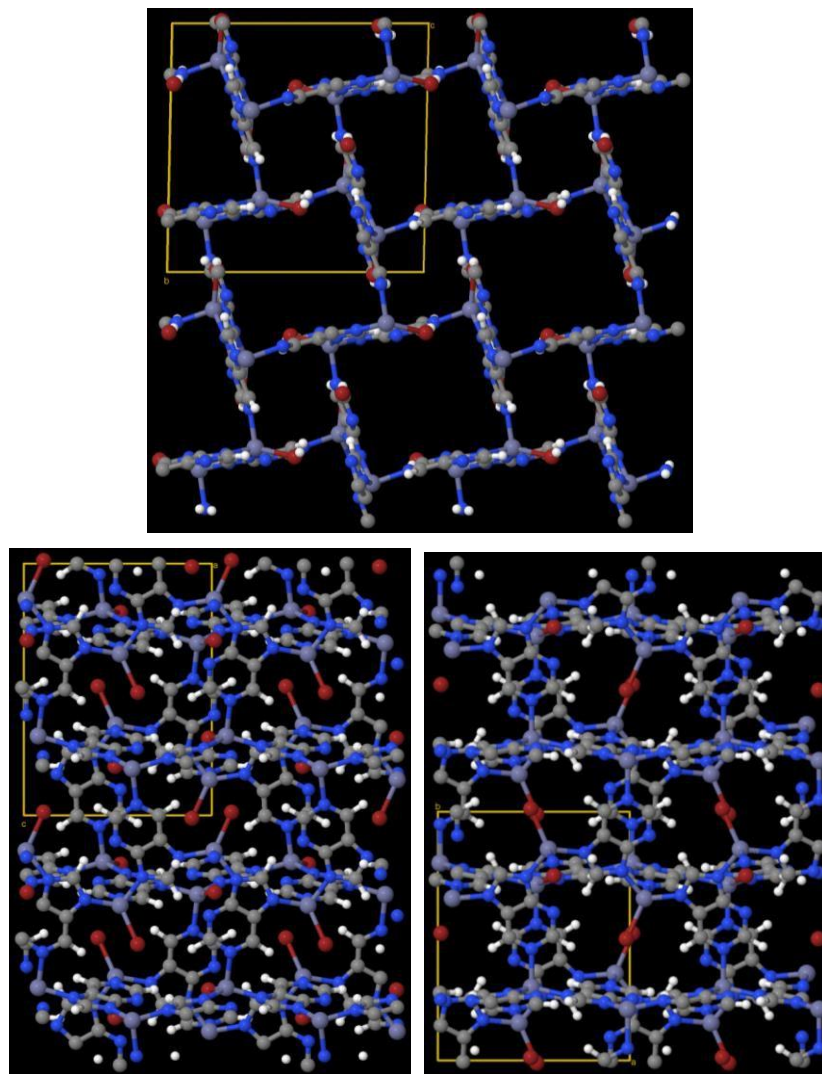


Figure S119: XEDPIH contains the chemical elements Zn, H, C, Br, and N.

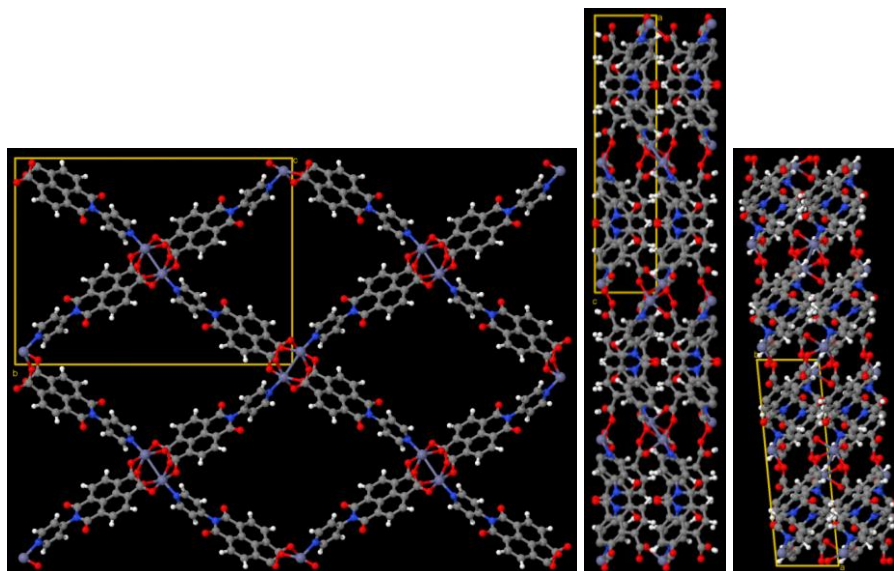


Figure S120: XEXMEU contains the chemical elements Zn, H, C, N, and O.

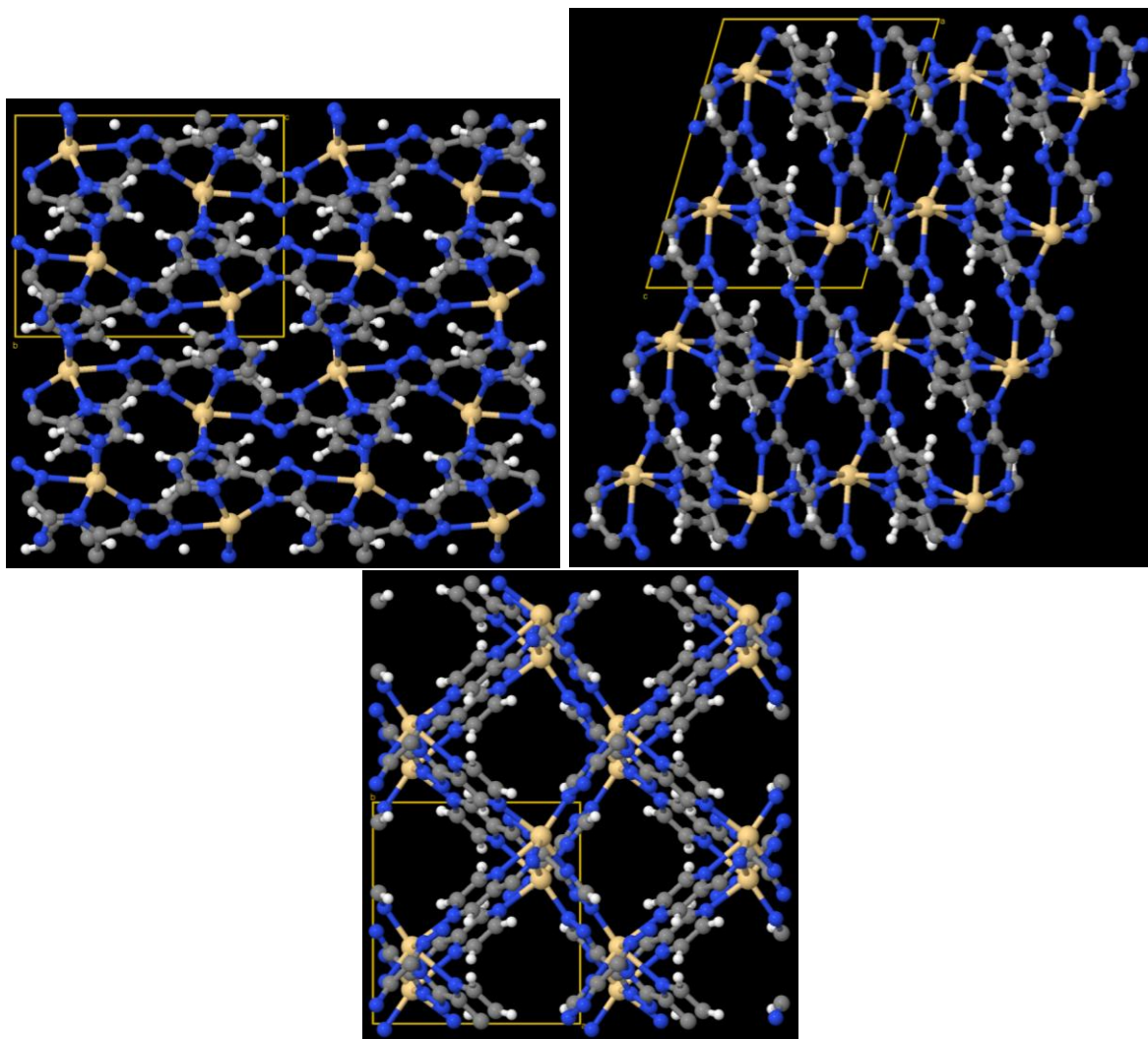


Figure S121: XORGUI contains the chemical elements Cd, N, C, and H.

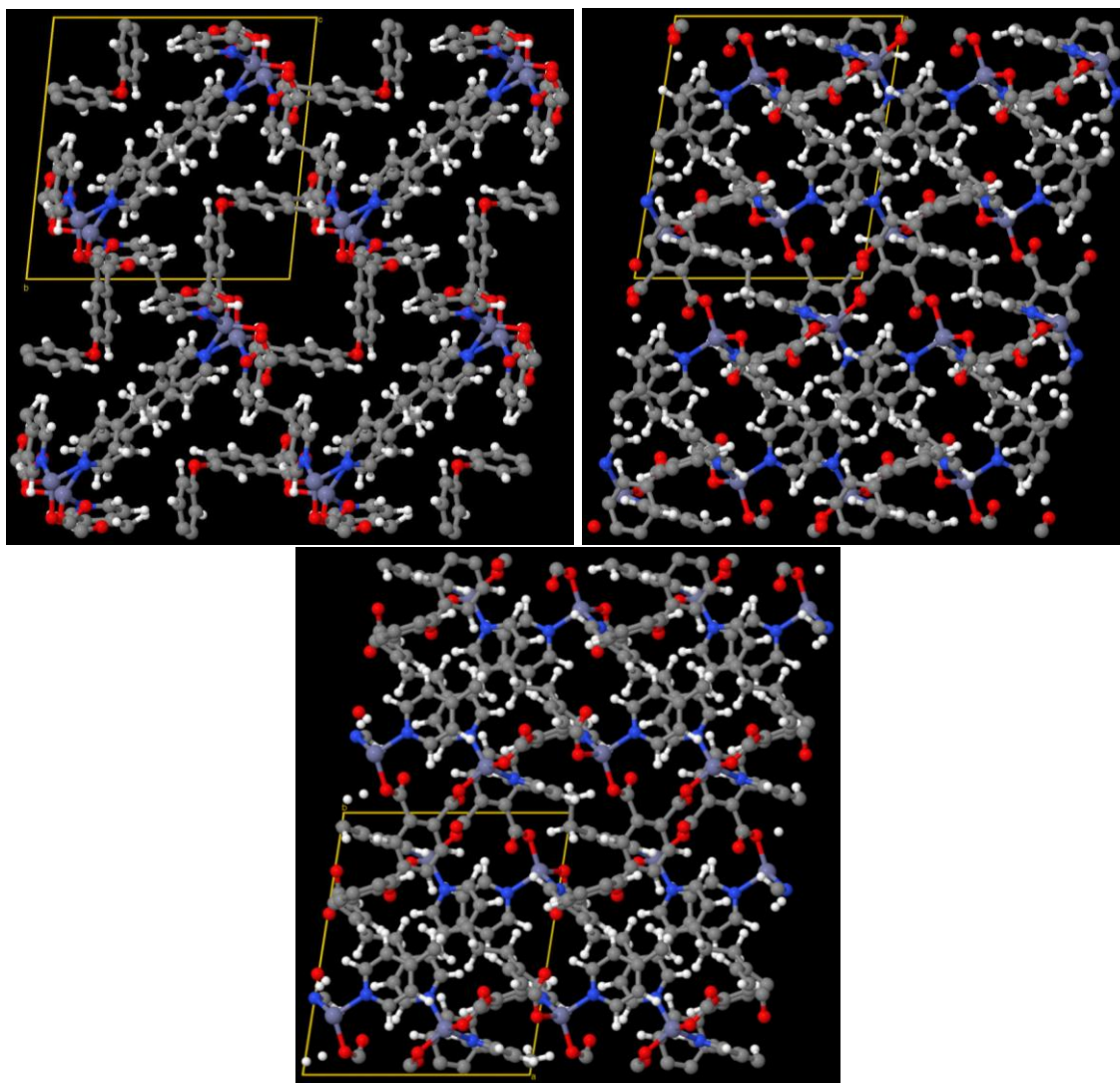


Figure S122: YARKEJ contains the chemical elements Zn, H, C, N, and O.

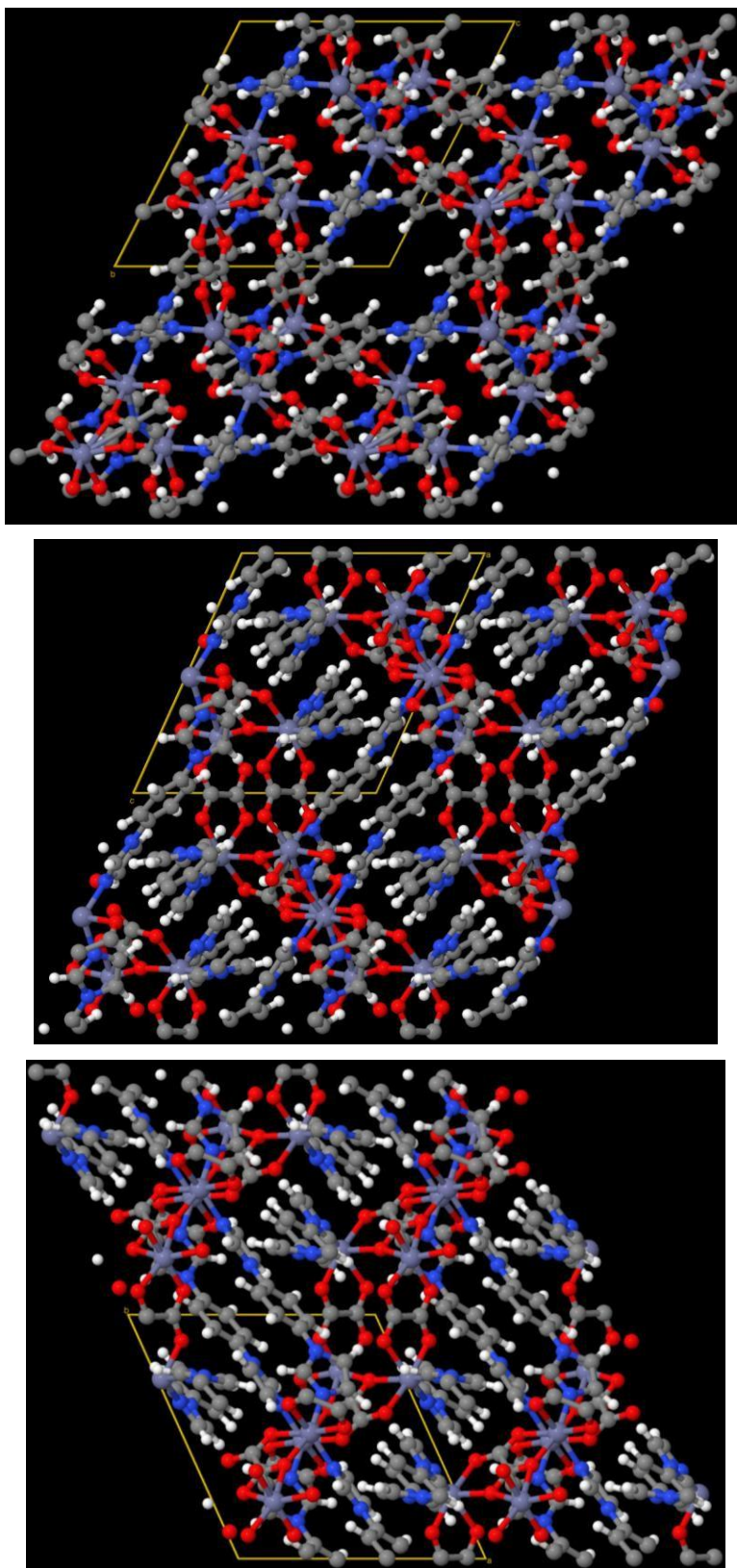


Figure S123: YARSIV contains the chemical elements Zn, H, C, N, and O.

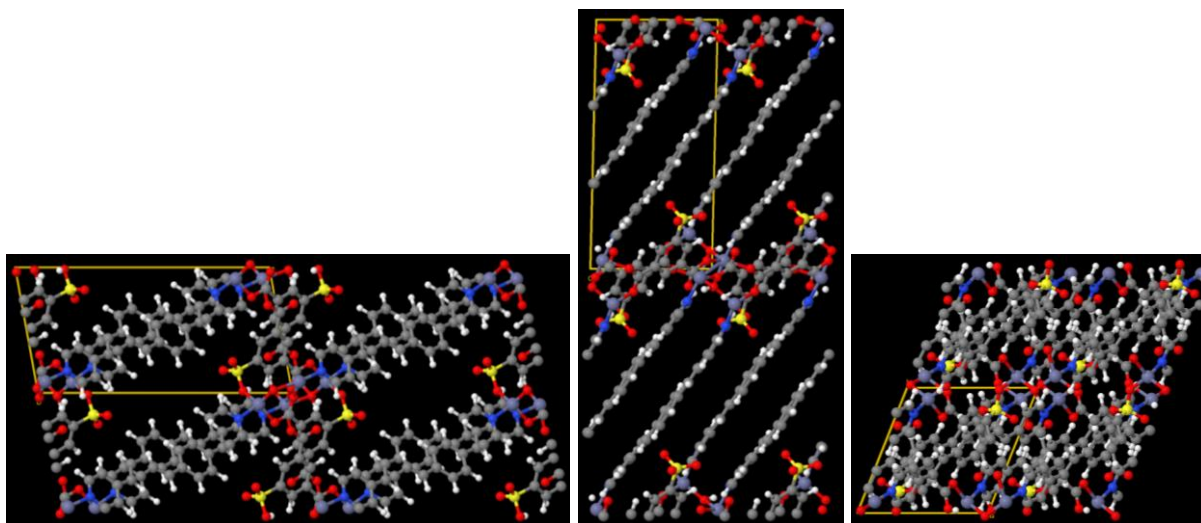


Figure S124: YUSGID contains the chemical elements Zn, H, C, S, N, and O.

Reference

- S1. T. A. Manz, "A Formally Exact Theory to Construct Nonreactive Forcefields using Linear Regression to Optimize Bonded Parameters," submitted to *RSC Advances*, 2024.