

**Relationships between the surface electronic and chemical
properties of doped 4d and 5d late transition metal dioxides:**

Supporting information

Zhongnan Xu and John R. Kitchin^{*}

Department of Chemical Engineering, Carnegie Mellon University,

5000 Forbes Ave, Pittsburgh, PA 15213

(Dated: February 4, 2016)

CONTENTS

I. Introduction	3
II. Bulk properties	3
III. Calculation of adsorption energies	4
A. Calculating relaxed bare surfaces	4
B. Adsorption energies on all structures	6
IV. Storage of structural data, adsorption energies, and DOS data	11
V. Analysis of adsorption energies and DOS	27
A. Parity plots between $\Delta\Delta E_{strain}^O$, $\Delta\Delta E_{ligand}^O$ and $\Delta\Delta E_{dopant}^O$	27
B. Example DOS	29
C. Relating the centers of the d , p , e_g , and t_{2g} -band to the adsorption energy	30
D. Ligand effect relationships	32
1. Relating changes in the center of the t_{2g} -band to $\Delta\Delta E_{ligand}^O$	33
2. Relating the t_{2g} -band center, width, and filling	34
3. Relating the d , p , and d^{host} -band fillings	35
4. Relating t_{2g} -band width and effective orbital radius r_d to $\Delta\Delta E_{ligand}^O$	37
E. Strain effect relationships	39
1. Relating strain to changes in adsorption energy	39
2. Relating strain, the t_{2g} -band width, and the t_{2g} -band center	40
3. Relating t_{2g} -band center and $\Delta\Delta E_{strain}^O$	42
VI. Additional of relationships	44
A. Relating the ligand effect to changes in the center of the d -band	44
B. Relating ligand induced changes to fillings of d and t_{2g} -band	46
C. Relating differences in the t_{2g} -band width and charge transfer	48
D. Relating charge transfer to changes in the adsorption energy	51
E. Relating strain to changes in band widths	53
F. Relating strain to changes in band centers	56
G. Relating strain to changes in fillings of t_{2g} -band	59

VII. Required modules	61
A. <code>numpy</code> , <code>scipy</code> , and <code>matplotlib</code>	61
B. <code>ase</code>	61
C. <code>jasp</code>	62
D. <code>pycse</code>	62
E. <code>ase_addons</code>	62
References	62

I. INTRODUCTION

The supporting information contains all the code necessary for reproducing the calculation and analysis of our data. Sections II and III contains the all of the DFT calculations performed in this study of both the bulk structures and adsorption energies. Section IV details the storage of all of the calculated data needed for analysis in a single JSON file, which is attached to this PDF and is titled **data.json**. The organization for this JSON file is also described in this section. All of the analysis required for the construction of all figures is done in Section V, while Section VI contains additional analysis done which we refer to in the manuscript. Finally, Section VII lists out the required software for reproducing all of our work.

II. BULK PROPERTIES

We first need to calculate the bulk properties in order to accurately construct surfaces used in this study. The initial guess of coordinates is taken from converged structures calculated in a previous study using QUANTUM-ESPRESSO and can be found at <http://dx.doi.org/10.5281/zenodo.12635>. From this initial guess, we perform a full shape and volume relaxation to the equilibrium a , c , and oxygen u parameters. In the previous study, we found that all oxides assumed a non-magnetic ground state. Hence, ground state magnetic structures were not compared. The calculations are saved in the folder named **{name}/ground**, where **{name}** is the name chemical formula of the material (IrO₂, RuO₂, etc...).

```

1  from jasp import *
2  JASPRC['queue.ppn'] = 4
3  JASPRC['queue.mem'] = '8GB'
4  from ase_addons.bulk import rutile
5
6  data = [['IrO2', 4.53, 3.18, 0.31],
7          ['RuO2', 4.53, 3.12, 0.31],
8          ['PtO2', 4.58, 3.22, 0.31],
9          ['RhO2', 4.52, 3.11, 0.31],
10         ['OsO2', 4.51, 3.21, 0.31],
11         ['PdO2', 4.58, 3.20, 0.31]]
12
13  print '#+CAPTION: Relaxed lattice parameters of rutile structures'
14  print '#+TBLNAME: rutile-struct'
15  print '| System |    a |    c |    u | a / c |'
16  print '|-----+-----+-----+-----|-----|'
17
18
19  for name, a, c, u in data:
20      atoms = rutile((name[:-2], 'O'), a=a, c=c, u=u, mags=(0, 0))
21      with jasp('{name}/ground'.format(**locals()), atoms=atoms,
22              xc='PBE', lreal=False,
23              encut=500, kpts=(7, 7, 7),
24              ispin=nspin, lorbit=11,
25              ibrion=2, isif=3, nsw=50, ediffg=-0.05,
26              npar=4, nsim=4, lplane=True, lscal=False,
27              lwave=False) as calc:
28          try:
29              calc.calculate()
30              pos = calc.atoms.get_positions()
31              cell = calc.atoms.get_cell()
32              a = cell[0][0]
33              c = cell[2][2]
34              aoverc = a / c
35              u = pos[2][0] / a
36              print '|{name}|{a:1.2f}|{c:1.2f}|{u:1.2f}|{aoverc:1.3f}|'.format(**locals())
37          except (VaspRunning, VaspSubmitted):
38              print calc.vasppdir, 'running'

```

TABLE I. Relaxed lattice parameters of rutile structures

System	a	c	u
IrO ₂	4.54	3.19	0.31
RuO ₂	4.55	3.14	0.31
PtO ₂	4.60	3.25	0.31
RhO ₂	4.55	3.13	0.31
OsO ₂	4.52	3.22	0.31
PdO ₂	4.58	3.20	0.31

III. CALCULATION OF ADSORPTION ENERGIES

A. Calculating relaxed bare surfaces

Before performing adsorption calculations, we first want to relax the surface without the adsorbate. Doing this gives us a good starting guess for the systems with an adsorbed species. This is performed with the script below. The calculations are stored in folders **{name}/bare**, where **{name}** is the chemical formula of the material (IrO₂, RuO₂, etc. . .).

```

1  from jasp import *
2  JASPRC['queue.ppn'] = 16
3  JASPRC['queue.mem'] = '32GB'
4  from ase_addons-surfaces import rutile110
5
6  data = [['IrO2', 4.54, 3.19, 0.31],
7          ['RuO2', 4.55, 3.14, 0.31],
8          ['PtO2', 4.60, 3.25, 0.31],
9          ['RhO2', 4.55, 3.13, 0.31],
10         ['OsO2', 4.52, 3.22, 0.31],
11         ['PdO2', 4.58, 3.20, 0.31]]
12
13  for name, a, c, u in data:
14      atoms = rutile110((name[:-2], 'O'), a, c, u, mag=0.0,
15                      base=3, layers=12, vacuum=12, fixlayers=6)
16
17      with jasp('{name}/bare'.format(**locals()), atoms=atoms,
18              xc='PBE', lreal=False,
19              encut=500, kpts=(7, 7, 1),
20              ispin=1, lorbit=11,
21              ibrion=2, isif=2, nsw=50, ediffg=-0.05,
```

```

22         npar=4, nsim=4, lplane=True, lscal=False,
23         lwave=False) as calc:
24     try:
25         calc.calculate()
26         print calc.vasmdir, 'Converged'
27     except (VaspSubmitted, VaspRunning, VaspQueued):
28         print calc.vasmdir, 'running'

```

B. Adsorption energies on all structures

The code below takes the relaxed bare surface and runs a series of calculations dopant-host calculations. Out of six transition metal oxide rutiles used in this study, each transition metal is doped into the structure of the other five rutiles. This totals to 30 dopant-host pairs. For each of these dopant-host pairs, we perform four adsorption energy calculations.

First, the adsorption energy on the dopant in its native structure is calculated. We call this adsorption energy the pure adsorption energy. Since the relaxed, the bare surface each structure was calculated in Section III A, we only need to perform a calculation of the bare surface with an adsorbate. This calculation is stored in the folder **{dopant}/O**, where **{dopant}** is the chemical name of the dopant system. Note this is only done once for all dopant-host pairs where the dopant is the same.

Second, the "strain effect" on the adsorption energy is evaluated by calculating the adsorption energy of the dopant in the host's lattice coordinates. We call the difference between this adsorption energy and the pure adsorption energy the "strain effect". To do this, we first read the lattice coordinates of the relaxed pure slab of the host system. We then construct a surface with these lattice coordinates with the chemical identities of the dopant system. Finally, we calculate the total energy of the bare surface slab and the surface slab with the adsorbate. These calculations are stored in the folders **{dopant}/strain/{host}/bare** and **{dopant}/strain/{host}/O**, respectively, where **{dopant}** and **{host}** are the chemical names of the dopant and host pair we are calculating.

Third, the "ligand effect" on the adsorption energy is evaluated by calculating the adsorption energy of the dopant in the dopant's lattice coordinates but with all cations except the cation at the adsorption site replaced with the chemical symbols of the host. We call the difference between this adsorption energy and the pure adsorption energy the "ligand effect". We calculate the total energy of the bare surface slab and the surface slab with the ad-

sorbate. These calculations are stored in the folders `{dopant}/ligand/{host}/bare` and `{dopant}/ligand/{host}/O`, respectively, where `{dopant}` and `{host}` are the chemical names of the dopant and host pair we are calculating.

Finally, the "dopant effect" on the adsorption energy is evaluated by calculating the adsorption energy of the dopant in the host's lattice coordinates with all of the cations except the cation at the adsorption site replaced with the chemical symbols of the host. Hence, the "dopant effect" can be seen as a combination of the "strain" and "ligand" effect. We calculate the total energy of the bare surface slab and the surface slab with the adsorbate. These calculations are stored in the folders `{dopant}/dopant/{host}/bare` and `{dopant}/dopant/{host}/O`, respectively, where `{dopant}` and `{host}` are the chemical names of the dopant and host pair we are calculating.

All of these calculations are performed with the single code below. In total, this code performs 186 DFT calculations. In addition to the six bare surface calculations done in Section III A this totals to 192 calculations, which is 96 adsorption energies.

```

1  from jasp import *
2  JASPRC['queue.ppn'] = 16
3  JASPRC['queue.mem'] = '32GB'
4  from ase_addons-surfaces import rutil110
5  from ase.lattice.surface import add_adsorbate
6  from ase.visualize import view
7
8  O = Atom('O', (0, 0, 0))
9
10 systems = ['RuO2', 'RhO2', 'PdO2', 'OsO2', 'IrO2', 'PtO2']
11 pairs = []
12 for p1 in systems:
13     for p2 in systems:
14         if p1 != p2:
15             pairs.append([p1, p2])
16
17 # Now calculate the contributions to the system of B doped with A
18 for pair in pairs:
19     dopant = pair[0]
20     host = pair[1]
21
22 #####
23 ### Adsorption energy on pure oxide A in struct of A ###
24 #####
25

```

```

26     with jasp('{dopant}/bare'.format(**locals())) as calc:
27         atoms = calc.get_atoms()
28
29     slab_x = atoms.get_cell()[0][0]
30     slab_y = atoms.get_cell()[1][1]
31
32     z_metal = atoms.get_positions()[44][2]
33     z_oxy = atoms.get_positions()[47][2]
34
35     if z_metal < z_oxy:
36         h = 2 - (z_oxy - z_metal)
37     else:
38         h = 2
39
40     add_adsorbate(atoms, 0, height=h,
41                  position=(slab_x * 0.5, slab_y * 0.5))
42
43     with jasp('{dopant}/O'.format(**locals()), atoms=atoms,
44              xc='PBE', lreal=False,
45              encut=500, kpts=(7, 7, 1),
46              ispin=1, lorbit=11,
47              ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
48              npar=4, nsim=4, lplane=True, lscal=False,
49              lwave=False, setups={'Nb': '_sv'}) as calc:
50         try:
51             calc.calculate()
52         except (VaspSubmitted, VaspRunning, VaspQueued):
53             print calc.vasppdir, 'running'
54
55     #####
56     ### Adsorption energy on pure oxide A in struct of B ###
57     #####
58
59     with jasp('{host}/bare'.format(**locals())) as calc:
60         atoms = calc.get_atoms()
61
62     symbols = atoms.get_chemical_symbols()
63     for i, sym in enumerate(symbols):
64         if sym == host[:-2]:
65             symbols[i] = dopant[:-2]
66
67     atoms.set_chemical_symbols(symbols)
68     slab = atoms.copy()
69     with jasp('{dopant}/strain/{host}/bare'.format(**locals()), atoms=slab,
70              xc='PBE', lreal=False,
71              encut=500, kpts=(7, 7, 1),
72              ispin=1, lorbit=11,

```



```

73         ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
74         npar=4, nsim=4, lplane=True, lscal=False,
75         lwave=False, setups={'Nb': '_sv'}) as calc:
76     try:
77         calc.calculate()
78     except (VaspSubmitted, VaspRunning, VaspQueued):
79         print calc.vasmdir, 'running'
80
81     slab_x = atoms.get_cell()[0][0]
82     slab_y = atoms.get_cell()[1][1]
83
84     z_metal = atoms.get_positions()[44][2]
85     z_oxy = atoms.get_positions()[47][2]
86
87     if z_metal < z_oxy:
88         h = 2 - (z_oxy - z_metal)
89     else:
90         h = 2
91
92     add_adsorbate(atoms, 0, height=h,
93                  position=(slab_x * 0.5, slab_y * 0.5))
94
95     with jasp('{dopant}/strain/{host}/0'.format(**locals()), atoms=atoms,
96              xc='PBE', lreal=False,
97              encut=500, kpts=(7, 7, 1),
98              ispin=1, lorbit=11,
99              ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
100             npar=4, nsim=4, lplane=True, lscal=False,
101             lwave=False, setups={'Nb': '_sv'}) as calc:
102     try:
103         calc.calculate()
104     except (VaspSubmitted, VaspRunning, VaspQueued):
105         print calc.vasmdir, 'running'
106
107     #####
108     ### Adsorption energy of A doped in B with struct of A ###
109     #####
110
111     with jasp('{dopant}/bare'.format(**locals())) as calc:
112         atoms = calc.get_atoms()
113         slab = atoms.copy()
114
115     symbols = atoms.get_chemical_symbols()
116     for i, sym in enumerate(symbols):
117         if (i is not 44 and sym == dopant[:-2]):
118             symbols[i] = host[:-2]
119

```

```

120 atoms.set_chemical_symbols(symbols)
121 slab = atoms.copy()
122 with jasp('{dopant}/ligand/{host}/bare'.format(**locals()), atoms=slab,
123          xc='PBE', lreal=False,
124          encut=500, kpts=(7, 7, 1),
125          ispin=1, lorbit=11,
126          ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
127          npar=4, nsim=4, lplane=True, lscal=False,
128          lwave=False, setups={'Nb': '_sv'}) as calc:
129     try:
130         calc.calculate()
131     except (VaspSubmitted, VaspRunning, VaspQueued):
132         print calc.vaspdir, 'running'
133
134 slab_x = atoms.get_cell()[0][0]
135 slab_y = atoms.get_cell()[1][1]
136
137 z_metal = atoms.get_positions()[44][2]
138 z_oxy = atoms.get_positions()[47][2]
139
140 if z_metal < z_oxy:
141     h = 2 - (z_oxy - z_metal)
142 else:
143     h = 2
144
145 add_adsorbate(atoms, 0, height=h,
146               position=(slab_x * 0.5, slab_y * 0.5))
147
148 with jasp('{dopant}/ligand/{host}/0'.format(**locals()), atoms=atoms,
149          xc='PBE', lreal=False,
150          encut=500, kpts=(7, 7, 1),
151          ispin=1, lorbit=11,
152          ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
153          npar=4, nsim=4, lplane=True, lscal=False,
154          lwave=False, setups={'Nb': '_sv'}) as calc:
155     try:
156         calc.calculate()
157     except (VaspSubmitted, VaspRunning, VaspQueued):
158         print calc.vaspdir, 'running'
159
160 #####
161 ### Adsorption energy of A doped in B with struct of B ###
162 #####
163
164 with jasp('{host}/bare'.format(**locals())) as calc:
165     atoms = calc.get_atoms()
166

```

```

167     symbols = atoms.get_chemical_symbols()
168     for i, sym in enumerate(symbols):
169         if (i is 44):
170             symbols[i] = dopant[:-2]
171
172     atoms.set_chemical_symbols(symbols)
173     slab = atoms.copy()
174     with jasp('{dopant}/dopant/{host}/bare'.format(**locals()), atoms=slab,
175             xc='PBE', lreal=False,
176             encut=500, kpts=(7, 7, 1),
177             ispin=1, lorbit=11,
178             ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
179             npar=4, nsim=4, lplane=True, lscalu=False,
180             lwave=False, setups={'Nb': '_sv'}) as calc:
181         try:
182             calc.calculate()
183         except (VaspSubmitted, VaspRunning, VaspQueued):
184             print calc.vasppdir, 'running'
185
186     slab_x = atoms.get_cell()[0][0]
187     slab_y = atoms.get_cell()[1][1]
188
189     z_metal = atoms.get_positions()[44][2]
190     z_oxy = atoms.get_positions()[47][2]
191
192     if z_metal < z_oxy:
193         h = 2 - (z_oxy - z_metal)
194     else:
195         h = 2
196
197     add_adsorbate(atoms, 0, height=h,
198                  position=(slab_x * 0.5, slab_y * 0.5))
199
200     with jasp('{dopant}/dopant/{host}/O'.format(**locals()), atoms=atoms,
201             xc='PBE', lreal=False,
202             encut=500, kpts=(7, 7, 1),
203             ispin=1, lorbit=11,
204             ibrion=2, isif=2, nsw=50, ediffg=-0.05, ediff=1e-6,
205             npar=4, nsim=4, lplane=True, lscalu=False,
206             lwave=False, setups={'Nb': '_sv'}) as calc:
207         try:
208             calc.calculate()
209         except (VaspSubmitted, VaspRunning, VaspQueued):
210             print calc.vasppdir, 'running'

```

IV. STORAGE OF STRUCTURAL DATA, ADSORPTION ENERGIES, AND DOS DATA

The code below stores all of the structural data, adsorption energies, and DOS data in a single json file titled **data.json** for further analysis. The organization of JSON file is as follows.

```
1 {
2   dopant:
3   {
4     "bare":
5     {
6       "symbols": symbols,
7       "pos"      : pos,
8       "cell"     : cell,
9       "E"        : E,
10    },
11    "O":
12    {
13      "symbols": symbols,
14      "pos"      : pos,
15      "cell"     : cell,
16      "E"        : E,
17    },
18    "DOS":
19    {
20      "E"          : energies,
21      "d_dos"      : d_dos,
22      "d_states"   : d_states,
23      "d_tot_states" : d_tot_states,
24      "d_filling"  : d_filling,
25      "d_center"   : d_center,
26      "d_width"    : d_width,
27      "t2g_..."  : t2g_...,
28      "eg_..."   : eg_...,
29      "p_..."    : p_...,
30    },
31    "Eads": Eads,
32    host:
33    {
34      "delta_E_strain": delta_E_strain,
35      "delta_E_ligand": delta_E_ligand,
36      "delta_E_dopant": delta_E_dopant,
37      "delta_E_predict": delta_E_predict,
```

```

38     "strain":
39     {
40         "bare":
41         {
42             "symbols": symbols,
43             "pos"      : pos,
44             "cell"     : cell,
45             "E"        : E,
46         },
47         "O":
48         {
49             "symbols": symbols,
50             "pos"      : pos,
51             "cell"     : cell,
52             "E"        : E,
53         },
54         "DOS":
55         {
56             "E"          : energies,
57             "d_dos"       : d_dos,
58             "d_states"    : d_states,
59             "d_d_states"  : d_d_states,
60             "d_tot_states": d_tot_states,
61             "d_d_tot_states": d_d_tot_states,
62             "d_filling"   : d_filling,
63             "d_d_filling" : d_d_filling,
64             "d_center"    : d_center,
65             "d_d_center"  : d_d_center,
66             "d_width"     : d_width,
67             "d_d_width"   : d_d_width,
68             "t2g_..."   : t2g_...,
69             "eg_..."    : eg_...,
70             "p_..."     : p_...,
71         },
72         "Eads": Eads,
73     }
74     "ligand":
75     {
76         ...
77     }
78     "dopant":
79     {
80         ...
81     }
82 }
83
84 }

```

The above JSON data structure displays the organization of data for a single dopant-host pair. This data structure is easily read into Python, where it is represented by a dictionary. For each dopant-host pair, the parent dictionary is given by the variable `dopant`, which is a string of the chemical symbol of the dopant in the dopant-host pair. Because we are looking at six systems, there will be six `dopant` dictionaries. The three immediate children dictionaries of `dopant` is the `"bare"`, `"O"`, `"DOS"`, `"Eads"`, and `host`. Each `dopant` parent dictionary will have five `host` child dictionaries – one for each of the metal cations that are not `dopant`.

`"bare"` and `"O"` are dictionaries that contain the structural and energetic information of the bare slab and slab with adsorbate, where both the structural and chemical identity of the slab is the pure dopant system's. Inside `"bare"` and `"O"`, the four sets of data given are the `symbols`, `pos`, `cell`, and `E`, the first three of which are lists and the last the total energy of the calculation. `"Eads"` is just a float of the adsorption energy of atomic oxygen on the pure slab.

`"DOS"` contains all of the density of states data needed for analysis. Inside the `"DOS"` dictionary, there are several variables. `"E"` contains the list of energies where each density is given. These are all given as a list of energies between -10 eV and 5 eV with respect to the fermi level. We then looked at properties of the d , t_{2g} , and e_g orbitals of the cation at the adsorption site and the p orbital of the surrounding five oxygen anions. All properties were recorded for the bare surface. Given orbital `a`, the variables given as `a_states`, `a_tot_states`, `a_filling`, `a_center`, `a_center`, and `a_width` are the number of occupied states in the band of orbital `a`, number of total states in the band of orbital `a`, the fractional filling of the band of orbital `a`, the first moment (center) of the band of orbital `a`, and the second moment around the center of the band (width) of orbital `a`, respectively.

The above description is a summary of the structural, energetic, and DOS data for all of pure component, given by string `dopant`. The same information for the dopant-host pairs is given in the `host`, dictionary, where `host` is a string variable that identifies the host in the dopant-host pair. Since we looked at six total systems, each `dopant` can be doped in five other `hosts`, so each unique `dopant` will have five `host` dictionaries.

The direct children of the `host` dictionary are the "`strain`", "`ligand`", and "`dopant`" dictionaries, which contain the same structural, energetic, and DOS information as described above but for the strain, ligand, and dopant calculations described in Section III B. The only addition to these calculation data is that the DOS data has added variables. Given orbital `a`, property `p` (occupied states, fractional filling, band center, etc...) and effect `f` (strain, ligand, dopant), variable `d_a.f` describes the difference in that property `p` of that effect `f` with respect to the same `a_p` of the `dopant` in its own structural. For example, given `dopant` is RuO_2 , `host` is PtO_2 , and the effect is "`strain`", `d_eg_p` describes the change in the width of the e_g band of the Ru atom when it is embedded in a RuO_2 lattice at the structure of the native PtO_2 lattice.

In addition to the "`strain`", "`ligand`", and "`dopant`" dictionary children of `host`, we also store three variables unique to each dopant-host pair, given as `delta_E_strain`, `delta_E_ligand`, `delta_E_dopant`, and `delta_E_predict`. Given dopant A and host B , `delta_E_strain`, `delta_E_ligand`, and `delta_E_dopant` are the change in the adsorption energy caused by the strain, ligand, and dopant effect. These are described as $\Delta\Delta E_{\text{strain}}^{A-\text{BO}_2}$, $\Delta\Delta E_{\text{ligand}}^{A-\text{BO}_2}$, and $\Delta\Delta E_{\text{dopant}}^{A-\text{BO}_2}$ in the manuscript. $\Delta\Delta E_{\text{predicted}}^{A-\text{BO}_2}$ is the predicted value of $\Delta\Delta E_{\text{dopant}}^{A-\text{BO}_2}$ given as the sum of $\Delta\Delta E_{\text{strain}}^{A-\text{BO}_2}$ and $\Delta\Delta E_{\text{ligand}}^{A-\text{BO}_2}$.

The code that generates the `data.json` file from the completed DFT calculations is shown below. The produced `data.json` file is attached here: .

```

1  from jasp import *
2  import json
3  JASPRC['queue.ppn'] = 16
4  JASPRC['queue.mem'] = '32GB'
5  from ase_addons-surfaces import rutil110
6  from ase.lattice.surface import add_adsorbate
7  from ase.visualize import view
8
9  # The dictionary we are storing all of our data in
10 data = {}
11
12 # Some important indexes for recording density of states
13 m_index = 44
14 o_indexes = [21, 22, 41, 45, 46]
15
16 d_index = range(4, 9)
17 t2g_index = (5, 7, 8)
18 eg_index = (4, 6)

```

```

19  p_index = range(1, 4)
20
21  # Save the O2 energy
22  O2 = -9.849707 + 1.198
23
24  # Create a list of dopant-host pairs we want to calculate
25  systems = ['RuO2', 'RhO2', 'PdO2', 'OsO2', 'IrO2', 'PtO2']
26  pairs = []
27  for p1 in systems:
28      for p2 in systems:
29          if p1 != p2:
30              pairs.append([p1, p2])
31
32  # Now calculate the contributions to the system of B doped with A
33  for pair in pairs:
34      dopant = pair[0]
35      host = pair[1]
36
37      if dopant not in data:
38          data[dopant] = {}
39      data[dopant][host] = {}
40
41      #####
42      ### Adsorption energy on pure oxide A in struct of A ###
43      #####
44
45      with jasp('{dopant}/bare'.format(**locals())) as calc:
46          atoms = calc.get_atoms()
47          symbols = atoms.get_chemical_symbols()
48          pos = atoms.get_positions().tolist()
49          cell = atoms.get_cell().tolist()
50          bare = atoms.get_potential_energy()
51          data[dopant]['bare'] = {}
52          data[dopant]['bare']['symbols'] = symbols
53          data[dopant]['bare']['pos'] = pos
54          data[dopant]['bare']['cell'] = cell
55          data[dopant]['bare']['E'] = bare
56
57      # Now read and store properties of the electronic structure
58      ados = VaspDos(efermi=calc.get_fermi_level())
59      energies = ados.energy
60      ind = (energies < 5) & (energies > -10)
61      energies = energies[ind]
62      occupied = energies <= 0.0
63
64      d_pure = np.zeros(len(energies))
65      for d_purei in d_index:

```



```

66         d_pure += np.array((ados.site_dos(calc.resort[m_index], d_purei)[ind]))
67     d_pure_occupied = np.trapz(d_pure[occupied], energies[occupied])
68     d_pure_states = np.trapz(d_pure, energies)
69     d_pure_filling = d_pure_occupied/d_pure_states
70     d_pure_center = np.trapz(energies * d_pure, energies) / d_pure_states
71     d_pure_width = np.sqrt(np.trapz((energies - d_pure_center) ** 2 * d_pure, energies) / d_pure_states)
72
73     # Read the t2g band center, width, filling, median, and center of bonding and anti-bonding states
74     t2g_pure = np.zeros(len(energies))
75     for t2g_purei in t2g_index:
76         t2g_pure += np.array((ados.site_dos(calc.resort[m_index], t2g_purei)[ind]))
77     t2g_pure_occupied = np.trapz(t2g_pure[occupied], energies[occupied])
78     t2g_pure_states = np.trapz(t2g_pure, energies)
79     t2g_pure_filling = t2g_pure_occupied/t2g_pure_states
80     t2g_pure_center = np.trapz(energies * t2g_pure, energies) / t2g_pure_states
81     t2g_pure_centers = t2g_pure_center * np.ones(len(energies))
82     t2g_pure_width = np.sqrt(np.trapz((energies - t2g_pure_center) ** 2 * t2g_pure, energies) / t2g_pure_states)
83
84     eg_pure = np.zeros(len(energies))
85     for eg_purei in eg_index:
86         eg_pure += np.array((ados.site_dos(calc.resort[m_index], eg_purei)[ind]))
87     eg_pure_occupied = np.trapz(eg_pure[occupied], energies[occupied])
88     eg_pure_states = np.trapz(eg_pure, energies)
89     eg_pure_filling = eg_pure_occupied/eg_pure_states
90     eg_pure_center = np.trapz(energies * eg_pure, energies) / eg_pure_states
91     eg_pure_centers = eg_pure_center * np.ones(len(energies))
92     eg_pure_width = np.sqrt(np.trapz((energies - eg_pure_center) ** 2 * eg_pure, energies) / eg_pure_states)
93
94     p_pure = np.zeros(len(energies))
95     for o_index in o_indexes:
96         for p_purei in p_index:
97             p_pure += np.array((ados.site_dos(calc.resort[o_index], p_purei)[ind]))
98     p_pure /= 5
99     p_pure_occupied = np.trapz(p_pure[occupied], energies[occupied])
100    p_pure_states = np.trapz(p_pure, energies)
101    p_pure_filling = p_pure_occupied/p_pure_states
102    p_pure_center = np.trapz(energies * p_pure, energies) / p_pure_states
103    p_pure_centers = p_pure_center * np.ones(len(energies))
104    p_pure_width = np.sqrt(np.trapz((energies - p_pure_center) ** 2 * p_pure, energies) / p_pure_states)
105
106    with jasp('{dopant}/0'.format(**locals())) as calc:
107        atoms = calc.get_atoms()
108        symbols = atoms.get_chemical_symbols()
109        pos = atoms.get_positions().tolist()
110        cell = atoms.get_cell().tolist()
111        ads = atoms.get_potential_energy()
112        data[dopant]['0'] = {}

```

```

113     data[dopant]['0']['symbols'] = symbols
114     data[dopant]['0']['pos'] = pos
115     data[dopant]['0']['cell'] = cell
116     data[dopant]['0']['E'] = bare
117
118     E_pure = ads - (bare + 0.5 * O2)
119     data[dopant]['E_ads'] = E_pure
120
121     # Now store all of the DOS data in a json file
122     data[dopant]['DOS'] = {}
123     data[dopant]['DOS']['E'] = energies.tolist()
124     data[dopant]['DOS']['d_dos'] = d_pure.tolist()
125     data[dopant]['DOS']['d_states'] = d_pure_occupied
126     data[dopant]['DOS']['d_tot_states'] = d_pure_states
127     data[dopant]['DOS']['d_filling'] = d_pure_filling
128     data[dopant]['DOS']['d_center'] = d_pure_center
129     data[dopant]['DOS']['d_width'] = d_pure_width
130     data[dopant]['DOS']['t2g_dos'] = t2g_pure.tolist()
131     data[dopant]['DOS']['t2g_states'] = t2g_pure_occupied
132     data[dopant]['DOS']['t2g_tot_states'] = t2g_pure_states
133     data[dopant]['DOS']['t2g_filling'] = t2g_pure_filling
134     data[dopant]['DOS']['t2g_center'] = t2g_pure_center
135     data[dopant]['DOS']['t2g_width'] = t2g_pure_width
136     data[dopant]['DOS']['eg_dos'] = eg_pure.tolist()
137     data[dopant]['DOS']['eg_states'] = eg_pure_occupied
138     data[dopant]['DOS']['eg_tot_states'] = eg_pure_states
139     data[dopant]['DOS']['eg_filling'] = eg_pure_filling
140     data[dopant]['DOS']['eg_center'] = eg_pure_center
141     data[dopant]['DOS']['eg_width'] = eg_pure_width
142     data[dopant]['DOS']['p_dos'] = p_pure.tolist()
143     data[dopant]['DOS']['p_states'] = p_pure_occupied
144     data[dopant]['DOS']['p_tot_states'] = p_pure_states
145     data[dopant]['DOS']['p_filling'] = p_pure_filling
146     data[dopant]['DOS']['p_center'] = p_pure_center
147     data[dopant]['DOS']['p_width'] = p_pure_width
148
149     #####
150     ### Adsorption energy on pure oxide A in struct of B ###
151     #####
152
153     data[dopant][host]['strain'] = {}
154
155     with jasp('{dopant}/strain/{host}/bare'.format(**locals())) as calc:
156         atoms = calc.get_atoms()
157         symbols = atoms.get_chemical_symbols()
158         pos = atoms.get_positions().tolist()
159         cell = atoms.get_cell().tolist()

```

```

160     bare = atoms.get_potential_energy()
161     data[dopant][host]['strain']['bare'] = {}
162     data[dopant][host]['strain']['bare']['symbols'] = symbols
163     data[dopant][host]['strain']['bare']['pos'] = pos
164     data[dopant][host]['strain']['bare']['cell'] = cell
165     data[dopant][host]['strain']['bare']['E'] = bare
166
167     # Now read and store properties of the electronic structure
168     ados = VaspDos(efermi=calc.get_fermi_level())
169     energies = ados.energy
170     ind = (energies < 5) & (energies > -10)
171     energies = energies[ind]
172     occupied = energies <= 0.0
173
174     d_strain = np.zeros(len(energies))
175     for d_straini in d_index:
176         d_strain += np.array((ados.site_dos(calc.resort[m_index], d_straini)[ind]))
177     d_strain_occupied = np.trapz(d_strain[occupied], energies[occupied])
178     d_strain_states = np.trapz(d_strain, energies)
179     d_strain_filling = d_strain_occupied/d_strain_states
180     d_strain_center = np.trapz(energies * d_strain, energies) / d_strain_states
181     d_strain_centers = d_strain_center * np.ones(len(energies))
182     d_strain_width = np.sqrt(np.trapz((energies - d_strain_center) ** 2 * d_strain, energies) / d_strain_states)
183
184     # Read the t2g band center, width, filling, median, and center of bonding and anti-bonding states
185     t2g_strain = np.zeros(len(energies))
186     for t2g_straini in t2g_index:
187         t2g_strain += np.array((ados.site_dos(calc.resort[m_index], t2g_straini)[ind]))
188     t2g_strain_occupied = np.trapz(t2g_strain[occupied], energies[occupied])
189     t2g_strain_states = np.trapz(t2g_strain, energies)
190     t2g_strain_filling = t2g_strain_occupied/t2g_strain_states
191     t2g_strain_center = np.trapz(energies * t2g_strain, energies) / t2g_strain_states
192     t2g_strain_centers = t2g_strain_center * np.ones(len(energies))
193     t2g_strain_width = np.sqrt(np.trapz((energies - t2g_strain_center) ** 2 * t2g_strain, energies) / t2g_strain_states)
194
195     eg_strain = np.zeros(len(energies))
196     for eg_straini in eg_index:
197         eg_strain += np.array((ados.site_dos(calc.resort[m_index], eg_straini)[ind]))
198     eg_strain_occupied = np.trapz(eg_strain[occupied], energies[occupied])
199     eg_strain_states = np.trapz(eg_strain, energies)
200     eg_strain_filling = eg_strain_occupied/eg_strain_states
201     eg_strain_center = np.trapz(energies * eg_strain, energies) / eg_strain_states
202     eg_strain_centers = eg_strain_center * np.ones(len(energies))
203     eg_strain_width = np.sqrt(np.trapz((energies - eg_strain_center) ** 2 * eg_strain, energies) / eg_strain_states)
204
205     p_strain = np.zeros(len(energies))
206     for o_index in o_indexes:

```

```

207         for p_straini in p_index:
208             p_strain += np.array((ados.site_dos(calc.resort[o_index], p_straini)[ind]))
209         p_strain /= 5
210         p_strain_occupied = np.trapz(p_strain[occupied], energies[occupied])
211         p_strain_states = np.trapz(p_strain, energies)
212         p_strain_filling = p_strain_occupied/p_strain_states
213         p_strain_center = np.trapz(energies * p_strain, energies) / p_strain_states
214         p_strain_centers = p_strain_center * np.ones(len(energies))
215         p_strain_width = np.sqrt(np.trapz((energies - p_strain_center) ** 2 * p_strain, energies) / p_strain_states)
216
217     with jasp('{dopant}/strain/{host}/0'.format(**locals())) as calc:
218         atoms = calc.get_atoms()
219         symbols = atoms.get_chemical_symbols()
220         pos = atoms.get_positions().tolist()
221         cell = atoms.get_cell().tolist()
222         ads = atoms.get_potential_energy()
223         data[dopant][host]['strain']['0'] = {}
224         data[dopant][host]['strain']['0']['symbols'] = symbols
225         data[dopant][host]['strain']['0']['pos'] = pos
226         data[dopant][host]['strain']['0']['cell'] = cell
227         data[dopant][host]['strain']['0']['E'] = bare
228
229     E_strain = ads - (bare + 0.5 * O2)
230     data[dopant][host]['strain']['E_ads'] = E_strain
231
232     # Now store all of the DOS data in a json file
233     data[dopant][host]['strain']['DOS'] = {}
234     data[dopant][host]['strain']['DOS']['E'] = energies.tolist()
235     data[dopant][host]['strain']['DOS']['d_dos'] = d_strain.tolist()
236     data[dopant][host]['strain']['DOS']['d_d_states'] = d_strain_occupied
237     data[dopant][host]['strain']['DOS']['d_tot_states'] = d_strain_states
238     data[dopant][host]['strain']['DOS']['d_d_filling'] = d_strain_filling
239     data[dopant][host]['strain']['DOS']['d_d_center'] = d_strain_center
240     data[dopant][host]['strain']['DOS']['d_d_width'] = d_strain_width
241
242     data[dopant][host]['strain']['DOS']['d_d_states'] = d_strain_occupied - d_pure_occupied
243     data[dopant][host]['strain']['DOS']['d_d_tot_states'] = d_strain_states - d_pure_states
244     data[dopant][host]['strain']['DOS']['d_d_filling'] = d_strain_filling - d_pure_filling
245     data[dopant][host]['strain']['DOS']['d_d_center'] = d_strain_center - d_pure_center
246     data[dopant][host]['strain']['DOS']['d_d_width'] = d_strain_width - d_pure_width
247
248     data[dopant][host]['strain']['DOS']['t2g_dos'] = t2g_strain.tolist()
249     data[dopant][host]['strain']['DOS']['t2g_states'] = t2g_strain_occupied
250     data[dopant][host]['strain']['DOS']['t2g_tot_states'] = t2g_strain_states
251     data[dopant][host]['strain']['DOS']['t2g_filling'] = t2g_strain_filling
252     data[dopant][host]['strain']['DOS']['t2g_center'] = t2g_strain_center
253     data[dopant][host]['strain']['DOS']['t2g_width'] = t2g_strain_width

```

```

254
255 data[dopant][host]['strain']['DOS']['d_t2g_states'] = t2g_strain_occupied - t2g_pure_occupied
256 data[dopant][host]['strain']['DOS']['d_t2g_tot_states'] = t2g_strain_states - t2g_pure_states
257 data[dopant][host]['strain']['DOS']['d_t2g_filling'] = t2g_strain_filling - t2g_pure_filling
258 data[dopant][host]['strain']['DOS']['d_t2g_center'] = t2g_strain_center - t2g_pure_center
259 data[dopant][host]['strain']['DOS']['d_t2g_width'] = t2g_strain_width - t2g_pure_width
260
261 data[dopant][host]['strain']['DOS']['eg_dos'] = eg_strain.tolist()
262 data[dopant][host]['strain']['DOS']['eg_states'] = eg_strain_occupied
263 data[dopant][host]['strain']['DOS']['eg_tot_states'] = eg_strain_states
264 data[dopant][host]['strain']['DOS']['eg_filling'] = eg_strain_filling
265 data[dopant][host]['strain']['DOS']['eg_center'] = eg_strain_center
266 data[dopant][host]['strain']['DOS']['eg_width'] = eg_strain_width
267
268 data[dopant][host]['strain']['DOS']['d_eg_states'] = eg_strain_occupied - eg_pure_occupied
269 data[dopant][host]['strain']['DOS']['d_eg_tot_states'] = eg_strain_states - eg_pure_states
270 data[dopant][host]['strain']['DOS']['d_eg_filling'] = eg_strain_filling - eg_pure_filling
271 data[dopant][host]['strain']['DOS']['d_eg_center'] = eg_strain_center - eg_pure_center
272 data[dopant][host]['strain']['DOS']['d_eg_width'] = eg_strain_width - eg_pure_width
273
274 data[dopant][host]['strain']['DOS']['p_dos'] = p_strain.tolist()
275 data[dopant][host]['strain']['DOS']['p_states'] = p_strain_occupied
276 data[dopant][host]['strain']['DOS']['p_tot_states'] = p_strain_states
277 data[dopant][host]['strain']['DOS']['p_filling'] = p_strain_filling
278 data[dopant][host]['strain']['DOS']['p_center'] = p_strain_center
279 data[dopant][host]['strain']['DOS']['p_width'] = p_strain_width
280
281 data[dopant][host]['strain']['DOS']['d_p_states'] = p_strain_occupied - p_pure_occupied
282 data[dopant][host]['strain']['DOS']['d_p_tot_states'] = p_strain_states - p_pure_states
283 data[dopant][host]['strain']['DOS']['d_p_filling'] = p_strain_filling - p_pure_filling
284 data[dopant][host]['strain']['DOS']['d_p_center'] = p_strain_center - p_pure_center
285 data[dopant][host]['strain']['DOS']['d_p_width'] = p_strain_width - p_pure_width
286
287
288 #####
289 ### Adsorption energy of A doped in B with struct of A ###
290 #####
291
292 data[dopant][host]['ligand'] = {}
293
294 with jasp('{dopant}/ligand/{host}/bare'.format(**locals())) as calc:
295     atoms = calc.get_atoms()
296     symbols = atoms.get_chemical_symbols()
297     pos = atoms.get_positions().tolist()
298     cell = atoms.get_cell().tolist()
299     bare = atoms.get_potential_energy()
300     data[dopant][host]['ligand']['bare'] = {}

```

```

301 data[dopant][host]['ligand']['bare']['symbols'] = symbols
302 data[dopant][host]['ligand']['bare']['pos'] = pos
303 data[dopant][host]['ligand']['bare']['cell'] = cell
304 data[dopant][host]['ligand']['bare']['E'] = bare
305
306 # Now read and store properties of the electronic structure
307 ados = VaspDos(efermi=calc.get_fermi_level())
308 energies = ados.energy
309 ind = (energies < 5) & (energies > -10)
310 energies = energies[ind]
311 occupied = energies <= 0.0
312
313 d_ligand = np.zeros(len(energies))
314 for d_ligandi in d_index:
315     d_ligand += np.array((ados.site_dos(calc.resort[m_index], d_ligandi)[ind]))
316 d_ligand_occupied = np.trapz(d_ligand[occupied], energies[occupied])
317 d_ligand_states = np.trapz(d_ligand, energies)
318 d_ligand_filling = d_ligand_occupied/d_ligand_states
319 d_ligand_center = np.trapz(energies * d_ligand, energies) / d_ligand_states
320 d_ligand_centers = d_ligand_center * np.ones(len(energies))
321 d_ligand_width = np.sqrt(np.trapz((energies - d_ligand_center) ** 2 * d_ligand, energies) / d_ligand_states)
322
323 # Read the t2g band center, width, filling, median, and center of bonding and anti-bonding states
324 t2g_ligand = np.zeros(len(energies))
325 for t2g_ligandi in t2g_index:
326     t2g_ligand += np.array((ados.site_dos(calc.resort[m_index], t2g_ligandi)[ind]))
327 t2g_ligand_occupied = np.trapz(t2g_ligand[occupied], energies[occupied])
328 t2g_ligand_states = np.trapz(t2g_ligand, energies)
329 t2g_ligand_filling = t2g_ligand_occupied/t2g_ligand_states
330 t2g_ligand_center = np.trapz(energies * t2g_ligand, energies) / t2g_ligand_states
331 t2g_ligand_centers = t2g_ligand_center * np.ones(len(energies))
332 t2g_ligand_width = np.sqrt(np.trapz((energies - t2g_ligand_center) ** 2 * t2g_ligand, energies) / t2g_ligand_states)
333
334 eg_ligand = np.zeros(len(energies))
335 for eg_ligandi in eg_index:
336     eg_ligand += np.array((ados.site_dos(calc.resort[m_index], eg_ligandi)[ind]))
337 eg_ligand_occupied = np.trapz(eg_ligand[occupied], energies[occupied])
338 eg_ligand_states = np.trapz(eg_ligand, energies)
339 eg_ligand_filling = eg_ligand_occupied/eg_ligand_states
340 eg_ligand_center = np.trapz(energies * eg_ligand, energies) / eg_ligand_states
341 eg_ligand_centers = eg_ligand_center * np.ones(len(energies))
342 eg_ligand_width = np.sqrt(np.trapz((energies - eg_ligand_center) ** 2 * eg_ligand, energies) / eg_ligand_states)
343
344 p_ligand = np.zeros(len(energies))
345 for o_index in o_indexes:
346     for p_ligandi in p_index:
347         p_ligand += np.array((ados.site_dos(calc.resort[o_index], p_ligandi)[ind]))

```

```

348     p_ligand /= 5
349     p_ligand_occupied = np.trapz(p_ligand[occupied], energies[occupied])
350     p_ligand_states = np.trapz(p_ligand, energies)
351     p_ligand_filling = p_ligand_occupied/p_ligand_states
352     p_ligand_center = np.trapz(energies * p_ligand, energies) / p_ligand_states
353     p_ligand_centers = p_ligand_center * np.ones(len(energies))
354     p_ligand_width = np.sqrt(np.trapz((energies - p_ligand_center) ** 2 * p_ligand, energies) / p_ligand_states)
355
356     with jasp('{dopant}/ligand/{host}/0'.format(**locals())) as calc:
357         atoms = calc.get_atoms()
358         symbols = atoms.get_chemical_symbols()
359         pos = atoms.get_positions().tolist()
360         cell = atoms.get_cell().tolist()
361         ads = atoms.get_potential_energy()
362         data[dopant][host]['ligand']['0'] = {}
363         data[dopant][host]['ligand']['0']['symbols'] = symbols
364         data[dopant][host]['ligand']['0']['pos'] = pos
365         data[dopant][host]['ligand']['0']['cell'] = cell
366         data[dopant][host]['ligand']['0']['E'] = bare
367
368     E_ligand = ads - (bare + 0.5 * 02)
369     data[dopant][host]['ligand']['E_ads'] = E_ligand
370
371     # Now store all of the DOS data in a json file
372     data[dopant][host]['ligand']['DOS'] = {}
373     data[dopant][host]['ligand']['DOS']['E'] = energies.tolist()
374     data[dopant][host]['ligand']['DOS']['d_dos'] = d_ligand.tolist()
375     data[dopant][host]['ligand']['DOS']['d_states'] = d_ligand_occupied
376     data[dopant][host]['ligand']['DOS']['d_tot_states'] = d_ligand_states
377     data[dopant][host]['ligand']['DOS']['d_filling'] = d_ligand_filling
378     data[dopant][host]['ligand']['DOS']['d_center'] = d_ligand_center
379     data[dopant][host]['ligand']['DOS']['d_width'] = d_ligand_width
380
381     data[dopant][host]['ligand']['DOS']['d_d_states'] = d_ligand_occupied - d_pure_occupied
382     data[dopant][host]['ligand']['DOS']['d_d_tot_states'] = d_ligand_states - d_pure_states
383     data[dopant][host]['ligand']['DOS']['d_d_filling'] = d_ligand_filling - d_pure_filling
384     data[dopant][host]['ligand']['DOS']['d_d_center'] = d_ligand_center - d_pure_center
385     data[dopant][host]['ligand']['DOS']['d_d_width'] = d_ligand_width - d_pure_width
386
387     data[dopant][host]['ligand']['DOS']['t2g_dos'] = t2g_ligand.tolist()
388     data[dopant][host]['ligand']['DOS']['t2g_states'] = t2g_ligand_occupied
389     data[dopant][host]['ligand']['DOS']['t2g_tot_states'] = t2g_ligand_states
390     data[dopant][host]['ligand']['DOS']['t2g_filling'] = t2g_ligand_filling
391     data[dopant][host]['ligand']['DOS']['t2g_center'] = t2g_ligand_center
392     data[dopant][host]['ligand']['DOS']['t2g_width'] = t2g_ligand_width
393
394     data[dopant][host]['ligand']['DOS']['d_t2g_states'] = t2g_ligand_occupied - t2g_pure_occupied

```

```

395 data[dopant][host]['ligand']['DOS']['d_t2g_tot_states'] = t2g_ligand_states - t2g_pure_states
396 data[dopant][host]['ligand']['DOS']['d_t2g_filling'] = t2g_ligand_filling - t2g_pure_filling
397 data[dopant][host]['ligand']['DOS']['d_t2g_center'] = t2g_ligand_center - t2g_pure_center
398 data[dopant][host]['ligand']['DOS']['d_t2g_width'] = t2g_ligand_width - t2g_pure_width
399
400 data[dopant][host]['ligand']['DOS']['eg_dos'] = eg_ligand.tolist()
401 data[dopant][host]['ligand']['DOS']['eg_states'] = eg_ligand_occupied
402 data[dopant][host]['ligand']['DOS']['eg_tot_states'] = eg_ligand_states
403 data[dopant][host]['ligand']['DOS']['eg_filling'] = eg_ligand_filling
404 data[dopant][host]['ligand']['DOS']['eg_center'] = eg_ligand_center
405 data[dopant][host]['ligand']['DOS']['eg_width'] = eg_ligand_width
406
407 data[dopant][host]['ligand']['DOS']['d_eg_states'] = eg_ligand_occupied - eg_pure_occupied
408 data[dopant][host]['ligand']['DOS']['d_eg_tot_states'] = eg_ligand_states - eg_pure_states
409 data[dopant][host]['ligand']['DOS']['d_eg_filling'] = eg_ligand_filling - eg_pure_filling
410 data[dopant][host]['ligand']['DOS']['d_eg_center'] = eg_ligand_center - eg_pure_center
411 data[dopant][host]['ligand']['DOS']['d_eg_width'] = eg_ligand_width - eg_pure_width
412
413 data[dopant][host]['ligand']['DOS']['p_dos'] = p_ligand.tolist()
414 data[dopant][host]['ligand']['DOS']['p_states'] = p_ligand_occupied
415 data[dopant][host]['ligand']['DOS']['p_tot_states'] = p_ligand_states
416 data[dopant][host]['ligand']['DOS']['p_filling'] = p_ligand_filling
417 data[dopant][host]['ligand']['DOS']['p_center'] = p_ligand_center
418 data[dopant][host]['ligand']['DOS']['p_width'] = p_ligand_width
419
420 data[dopant][host]['ligand']['DOS']['d_p_states'] = p_ligand_occupied - p_pure_occupied
421 data[dopant][host]['ligand']['DOS']['d_p_tot_states'] = p_ligand_states - p_pure_states
422 data[dopant][host]['ligand']['DOS']['d_p_filling'] = p_ligand_filling - p_pure_filling
423 data[dopant][host]['ligand']['DOS']['d_p_center'] = p_ligand_center - p_pure_center
424 data[dopant][host]['ligand']['DOS']['d_p_width'] = p_ligand_width - p_pure_width
425
426 #####
427 ### Adsorption energy of A doped in B with struct of B ###
428 #####
429
430 data[dopant][host]['dopant'] = {}
431
432 with jasp('{dopant}/dopant/{host}/bare'.format(**locals())) as calc:
433     atoms = calc.get_atoms()
434     symbols = atoms.get_chemical_symbols()
435     pos = atoms.get_positions().tolist()
436     cell = atoms.get_cell().tolist()
437     bare = atoms.get_potential_energy()
438     data[dopant][host]['dopant']['bare'] = {}
439     data[dopant][host]['dopant']['bare']['symbols'] = symbols
440     data[dopant][host]['dopant']['bare']['pos'] = pos
441     data[dopant][host]['dopant']['bare']['cell'] = cell

```



```

442 data[dopant][host]['dopant']['bare']['E'] = bare
443
444 # Now read and store properties of the electronic structure
445 ados = VaspDos(efermi=calc.get_fermi_level())
446 energies = ados.energy
447 ind = (energies < 5) & (energies > -10)
448 energies = energies[ind]
449 occupied = energies <= 0.0
450
451 d_dopant = np.zeros(len(energies))
452 for d_dopanti in d_index:
453     d_dopant += np.array((ados.site_dos(calc.resort[m_index], d_dopanti)[ind]))
454 d_dopant_occupied = np.trapz(d_dopant[occupied], energies[occupied])
455 d_dopant_states = np.trapz(d_dopant, energies)
456 d_dopant_filling = d_dopant_occupied/d_dopant_states
457 d_dopant_center = np.trapz(energies * d_dopant, energies) / d_dopant_states
458 d_dopant_centers = d_dopant_center * np.ones(len(energies))
459 d_dopant_width = np.sqrt(np.trapz((energies - d_dopant_center) ** 2 * d_dopant, energies) / d_dopant_states)
460
461 # Read the t2g band center, width, filling, median, and center of bonding and anti-bonding states
462 t2g_dopant = np.zeros(len(energies))
463 for t2g_dopanti in t2g_index:
464     t2g_dopant += np.array((ados.site_dos(calc.resort[m_index], t2g_dopanti)[ind]))
465 t2g_dopant_occupied = np.trapz(t2g_dopant[occupied], energies[occupied])
466 t2g_dopant_states = np.trapz(t2g_dopant, energies)
467 t2g_dopant_filling = t2g_dopant_occupied/t2g_dopant_states
468 t2g_dopant_center = np.trapz(energies * t2g_dopant, energies) / t2g_dopant_states
469 t2g_dopant_centers = t2g_dopant_center * np.ones(len(energies))
470 t2g_dopant_width = np.sqrt(np.trapz((energies - t2g_dopant_center) ** 2 * t2g_dopant, energies) / t2g_dopant_states)
471
472 eg_dopant = np.zeros(len(energies))
473 for eg_dopanti in eg_index:
474     eg_dopant += np.array((ados.site_dos(calc.resort[m_index], eg_dopanti)[ind]))
475 eg_dopant_occupied = np.trapz(eg_dopant[occupied], energies[occupied])
476 eg_dopant_states = np.trapz(eg_dopant, energies)
477 eg_dopant_filling = eg_dopant_occupied/eg_dopant_states
478 eg_dopant_center = np.trapz(energies * eg_dopant, energies) / eg_dopant_states
479 eg_dopant_centers = eg_dopant_center * np.ones(len(energies))
480 eg_dopant_width = np.sqrt(np.trapz((energies - eg_dopant_center) ** 2 * eg_dopant, energies) / eg_dopant_states)
481
482 p_dopant = np.zeros(len(energies))
483 for o_index in o_indexes:
484     for p_dopanti in p_index:
485         p_dopant += np.array((ados.site_dos(calc.resort[o_index], p_dopanti)[ind]))
486 p_dopant /= 5
487 p_dopant_occupied = np.trapz(p_dopant[occupied], energies[occupied])
488 p_dopant_states = np.trapz(p_dopant, energies)

```

```

489     p_dopant_filling = p_dopant_occupied/p_dopant_states
490     p_dopant_center = np.trapz(energies * p_dopant, energies) / p_dopant_states
491     p_dopant_centers = p_dopant_center * np.ones(len(energies))
492     p_dopant_width = np.sqrt(np.trapz((energies - p_dopant_center) ** 2 * p_dopant, energies) / p_dopant_states)
493
494
495     with jasp('{dopant}/dopant/{host}/0'.format(**locals())) as calc:
496         atoms = calc.get_atoms()
497         symbols = atoms.get_chemical_symbols()
498         pos = atoms.get_positions().tolist()
499         cell = atoms.get_cell().tolist()
500         ads = atoms.get_potential_energy()
501         data[dopant][host]['dopant']['0'] = {}
502         data[dopant][host]['dopant']['0']['symbols'] = symbols
503         data[dopant][host]['dopant']['0']['pos'] = pos
504         data[dopant][host]['dopant']['0']['cell'] = cell
505         data[dopant][host]['dopant']['0']['E'] = bare
506
507     E_dopant = ads - (bare + 0.5 * O2)
508     data[dopant][host]['ligand']['E_ads'] = E_ligand
509
510     # Now store all of the DOS data in a json file
511     data[dopant][host]['dopant']['DOS'] = {}
512     data[dopant][host]['dopant']['DOS']['E'] = energies.tolist()
513     data[dopant][host]['dopant']['DOS']['d_dos'] = d_dopant.tolist()
514     data[dopant][host]['dopant']['DOS']['d_states'] = d_dopant_occupied
515     data[dopant][host]['dopant']['DOS']['d_tot_states'] = d_dopant_states
516     data[dopant][host]['dopant']['DOS']['d_filling'] = d_dopant_filling
517     data[dopant][host]['dopant']['DOS']['d_center'] = d_dopant_center
518     data[dopant][host]['dopant']['DOS']['d_width'] = d_dopant_width
519
520     data[dopant][host]['dopant']['DOS']['d_d_states'] = d_dopant_occupied - d_pure_occupied
521     data[dopant][host]['dopant']['DOS']['d_d_tot_states'] = d_dopant_states - d_pure_states
522     data[dopant][host]['dopant']['DOS']['d_d_filling'] = d_dopant_filling - d_pure_filling
523     data[dopant][host]['dopant']['DOS']['d_d_center'] = d_dopant_center - d_pure_center
524     data[dopant][host]['dopant']['DOS']['d_d_width'] = d_dopant_width - d_pure_width
525
526     data[dopant][host]['dopant']['DOS']['t2g_dos'] = t2g_dopant.tolist()
527     data[dopant][host]['dopant']['DOS']['t2g_states'] = t2g_dopant_occupied
528     data[dopant][host]['dopant']['DOS']['t2g_tot_states'] = t2g_dopant_states
529     data[dopant][host]['dopant']['DOS']['t2g_filling'] = t2g_dopant_filling
530     data[dopant][host]['dopant']['DOS']['t2g_center'] = t2g_dopant_center
531     data[dopant][host]['dopant']['DOS']['t2g_width'] = t2g_dopant_width
532
533     data[dopant][host]['dopant']['DOS']['d_t2g_states'] = t2g_dopant_occupied - t2g_pure_occupied
534     data[dopant][host]['dopant']['DOS']['d_t2g_tot_states'] = t2g_dopant_states - t2g_pure_states
535     data[dopant][host]['dopant']['DOS']['d_t2g_filling'] = t2g_dopant_filling - t2g_pure_filling

```

```

536 data[dopant][host]['dopant']['DOS']['d_t2g_center'] = t2g_dopant_center - t2g_pure_center
537 data[dopant][host]['dopant']['DOS']['d_t2g_width'] = t2g_dopant_width - t2g_pure_width
538
539 data[dopant][host]['dopant']['DOS']['eg_dos'] = eg_dopant.tolist()
540 data[dopant][host]['dopant']['DOS']['eg_states'] = eg_dopant_occupied
541 data[dopant][host]['dopant']['DOS']['eg_tot_states'] = eg_dopant_states
542 data[dopant][host]['dopant']['DOS']['eg_filling'] = eg_dopant_filling
543 data[dopant][host]['dopant']['DOS']['eg_center'] = eg_dopant_center
544 data[dopant][host]['dopant']['DOS']['eg_width'] = eg_dopant_width
545
546 data[dopant][host]['dopant']['DOS']['d_eg_states'] = eg_dopant_occupied - eg_pure_occupied
547 data[dopant][host]['dopant']['DOS']['d_eg_tot_states'] = eg_dopant_states - eg_pure_states
548 data[dopant][host]['dopant']['DOS']['d_eg_filling'] = eg_dopant_filling - eg_pure_filling
549 data[dopant][host]['dopant']['DOS']['d_eg_center'] = eg_dopant_center - eg_pure_center
550 data[dopant][host]['dopant']['DOS']['d_eg_width'] = eg_dopant_width - eg_pure_width
551
552 data[dopant][host]['dopant']['DOS']['p_dos'] = p_dopant.tolist()
553 data[dopant][host]['dopant']['DOS']['p_states'] = p_dopant_occupied
554 data[dopant][host]['dopant']['DOS']['p_tot_states'] = p_dopant_states
555 data[dopant][host]['dopant']['DOS']['p_filling'] = p_dopant_filling
556 data[dopant][host]['dopant']['DOS']['p_center'] = p_dopant_center
557 data[dopant][host]['dopant']['DOS']['p_width'] = p_dopant_width
558
559 data[dopant][host]['dopant']['DOS']['d_p_states'] = p_dopant_occupied - p_pure_occupied
560 data[dopant][host]['dopant']['DOS']['d_p_tot_states'] = p_dopant_states - p_pure_states
561 data[dopant][host]['dopant']['DOS']['d_p_filling'] = p_dopant_filling - p_pure_filling
562 data[dopant][host]['dopant']['DOS']['d_p_center'] = p_dopant_center - p_pure_center
563 data[dopant][host]['dopant']['DOS']['d_p_width'] = p_dopant_width - p_pure_width
564
565 E_strain = E_strain - E_pure
566 E_ligand = E_ligand - E_pure
567 E_dopant = E_dopant - E_pure
568 E_pred = E_strain + E_ligand
569
570 data[dopant][host]['delta_E_predict'] = E_pred
571 data[dopant][host]['delta_E_strain'] = E_strain
572 data[dopant][host]['delta_E_ligand'] = E_ligand
573 data[dopant][host]['delta_E_dopant'] = E_dopant
574
575 myproj.cwd()
576 with open('data.json', 'w') as f:
577     json.dump(data, f)

```

V. ANALYSIS OF ADSORPTION ENERGIES AND DOS

The sections below contain all of the code necessary for construction of Figures 2-11 in the manuscript. Figure 1 is where we describe our scheme, the code for the construction of that Figure can be found in the org-file that constructed this PDF file.

A. Parity plots between $\Delta\Delta E_{strain}^O$, $\Delta\Delta E_{ligand}^O$ and $\Delta\Delta E_{dopant}^O$

The parity plots below compare the values of $\Delta\Delta E_{strain}^{A-BO_2} + \Delta\Delta E_{ligand}^{A-BO_2}$ and $\Delta\Delta E_{dopant}^{A-BO_2}$ as well as $\Delta\Delta E_{strain}^{A-BO_2}$ and $\Delta\Delta E_{ligand}^{A-BO_2}$. It reads the data from the **data.json** file and constructs Figure 2 of the manuscript.

```
1 import matplotlib
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import json
5
6 matplotlib.rc('xtick', labelsize=8)
7 matplotlib.rc('ytick', labelsize=8)
8
9 with open('data.json', 'r') as f:
10     data = json.load(f)
11
12 E_ligands, E_strains, E_dopants, E_predicts = [], [], [], []
13
14 for dopant in data:
15     for host in data[dopant]:
16         if host[-2:] != 'O2':
17             continue
18         E_dopants.append(data[dopant][host]['delta_E_dopant'])
19         E_predicts.append(data[dopant][host]['delta_E_strain']
20                           + data[dopant][host]['delta_E_ligand'])
21         E_ligands.append(abs(data[dopant][host]['delta_E_ligand']))
22         E_strains.append(abs(data[dopant][host]['delta_E_strain']))
23
24
25 fig = plt.figure(1, (3.3, 3))
26 ax = fig.add_axes([0.18, 0.2, 0.31, 0.7])
27 ax.plot((-2, 2), (-2, 2), ls='--', c='k')
28 ax.plot(E_predicts, E_dopants, marker='o', ls='none', c='grey')
29 ax.set_xlim(-1.75, 1.75)
30 ax.set_ylim(-1.75, 1.75)
```

```

31 ax.set_xticks([-1.0, 0, 1.0])
32 ax.set_yticks([-1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5])
33 ax.set_ylabel(r'$\Delta \Delta E_{\text{ligand}}^{\text{}} + \Delta \Delta E_{\text{strain}}^{\text{}}$ (eV)', size=8)
34 ax.set_xlabel(r'$\Delta \Delta E_{\text{dopant}}^{\text{}}$ (eV)', size=8)
35 ax.text(-1.4, 1.4, '(a)')
36
37 ax = fig.add_axes([0.51, 0.2, 0.31, 0.7])
38 ax.plot((-2, 2), (-2, 2), ls='--', c='k')
39 ax.plot(E_strains, E_ligands, marker='o', ls='none', c='grey')
40 ax.set_xlim(0, 0.2)
41 ax.set_ylim(0, 1.6)
42 ax.set_xticks([0, 0.1, 0.2])
43 ax.set_yticklabels([])
44 ax.set_xlabel(r'$|\Delta \Delta E_{\text{strain}}^{\text{}}|$ (eV)', size=9)
45 ax2 = ax.twinx()
46 ax2.set_xlim(0, 0.2)
47 ax2.set_ylim(0, 1.6)
48 ax2.set_xticks([0, 0.1, 0.2])
49 ax2.set_ylabel(r'$|\Delta \Delta E_{\text{ligand}}^{\text{}}|$ (eV)', size=9)
50 ax.text(0.145, 1.45, '(b)')
51
52 plt.savefig('figures/FIG2.png', dpi=300)
53 plt.savefig('figures/FIG2.eps', dpi=300)
54 plt.show()

```

B. Example DOS

The code below plots a sample DOS that we use as Figure 3 in the manuscript.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import json
5
6 matplotlib.rc('xtick', labelsize=8)
7 matplotlib.rc('ytick', labelsize=8)
8
9 with open('data.json', 'r') as f:
10     data = json.load(f)
11
12 energies = np.array(data['RuO2']['DOS']['E'])
13 d = np.array(data['RuO2']['DOS']['d_dos'])
14 eg = np.array(data['RuO2']['DOS']['eg_dos'])
15 t2g = np.array(data['RuO2']['DOS']['t2g_dos'])

```

```

16 p = np.array(data['RuO2']['DOS']['p_dos'])
17 occupied = energies <= 0.0
18
19 fig = plt.figure(1, (3.3, 2.5), dpi=300)
20 plt.plot(energies, p, c='r', label=r'$p$')
21 plt.fill_between(x=energies[occupied], y1=p[occupied],
22                 y2=np.zeros(p[occupied].shape), color='pink')
23 plt.plot(energies, d, c='b', label=r'$d$')
24 plt.fill_between(x=energies[occupied], y1=d[occupied],
25                 y2=np.zeros(d[occupied].shape), color='lightblue')
26 plt.plot(energies, t2g, c='m', label=r'$t_{2g}$')
27 plt.fill_between(x=energies[occupied], y1=t2g[occupied],
28                 y2=np.zeros(t2g[occupied].shape), color='plum')
29 plt.plot(energies, eg, c='g', label=r'$e_g$')
30 plt.fill_between(x=energies[occupied], y1=eg[occupied],
31                 y2=np.zeros(eg[occupied].shape), color='lightgreen')
32
33 plt.xlim(-9, 5)
34 plt.ylim(0, 4)
35 plt.xlabel('Energy (eV)', size=8)
36 plt.ylabel('DOS (states/M)', size=8)
37 plt.legend(loc=2, prop={'size': 'small'}, ncol=2)
38 plt.tight_layout()
39 plt.savefig('figures/FIG3.png', dpi=300)
40 plt.savefig('figures/FIG3.eps', dpi=300)
41 plt.show()

```

C. Relating the centers of the d , p , e_g , and t_{2g} -band to the adsorption energy

The figure below relates the center of the d , p , e_g , and t_{2g} -band to adsorption energies and constructs Figure 4 of the manuscript.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labels=8)
6 matplotlib.rc('ytick', labels=8)
7
8 with open('data.json', 'r') as f:
9     data = json.load(f)
10
11 d_Ru, eg_Ru, t2g_Ru, p_Ru, E_Ru = [], [], [], [], []
12 d_Rh, eg_Rh, t2g_Rh, p_Rh, E_Rh = [], [], [], [], []

```

```

13 d_Pd, eg_Pd, t2g_Pd, p_Pd, E_Pd = [], [], [], [], []
14 d_0s, eg_0s, t2g_0s, p_0s, E_0s = [], [], [], [], []
15 d_Ir, eg_Ir, t2g_Ir, p_Ir, E_Ir = [], [], [], [], []
16 d_Pt, eg_Pt, t2g_Pt, p_Pt, E_Pt = [], [], [], [], []
17
18 for dopant in data:
19     eval('E_' + dopant[:2]).append(data[dopant]['Eads'])
20     eval('d_' + dopant[:2]).append(data[dopant]['DOS']['d_center'])
21     eval('eg_' + dopant[:2]).append(data[dopant]['DOS']['eg_center'])
22     eval('t2g_' + dopant[:2]).append(data[dopant]['DOS']['t2g_center'])
23     eval('p_' + dopant[:2]).append(data[dopant]['DOS']['p_center'])
24     for host in data[dopant]:
25         if host[-2:] != 'O2':
26             continue
27         for effect in ['ligand', 'strain', 'dopant']:
28             eval('E_' + dopant[:2]).append(data[dopant]
29                                     [host][effect]['Eads'])
30             eval('d_' + dopant[:2]).append(data[dopant]
31                                     [host][effect]['DOS']['d_center'])
32             eval('eg_' + dopant[:2]).append(data[dopant]
33                                     [host][effect]['DOS']['eg_center'])
34             eval('t2g_' + dopant[:2]).append(data[dopant]
35                                     [host][effect]
36                                     ['DOS']['t2g_center'])
37             eval('p_' + dopant[:2]).append(data[dopant]
38                                     [host][effect]['DOS']['p_center'])
39
40 fig = plt.figure(1, (3.3, 4), dpi=300)
41 ax = fig.add_axes([0.14, 0.61, 0.35, 0.35])
42 ax.plot(d_Ru, E_Ru, ls='none', c='b', marker='o')
43 ax.plot(d_Rh, E_Rh, ls='none', c='b', marker='s')
44 ax.plot(d_Pd, E_Pd, ls='none', c='b', marker='^')
45 ax.plot(d_0s, E_0s, ls='none', c='c', marker='o')
46 ax.plot(d_Ir, E_Ir, ls='none', c='c', marker='s')
47 ax.plot(d_Pt, E_Pt, ls='none', c='c', marker='^')
48 ax.text(-2.95, 1.65, '(a)')
49 ax.set_xlabel(r'$E_{d}$ (eV)', size=8)
50 ax.set_xlim(-3.1, -0.9)
51 ax.set_xticks([-3, -2, -1])
52 ax.set_ylabel(r'$E_{ads}^{\mathrm{0}}$ (eV)', size=8)
53 ax.set_ylim(-3.5, 2.5)
54
55 ax = fig.add_axes([0.51, 0.61, 0.35, 0.35])
56 ax.plot(p_Ru, E_Ru, ls='none', c='b', marker='o')
57 ax.plot(p_Rh, E_Rh, ls='none', c='b', marker='s')
58 ax.plot(p_Pd, E_Pd, ls='none', c='b', marker='^')
59 ax.plot(p_0s, E_0s, ls='none', c='c', marker='o')

```

```

60 ax.plot(p_Ir, E_Ir, ls='none', c='c', marker='s')
61 ax.plot(p_Pt, E_Pt, ls='none', c='c', marker='^')
62 ax.text(-5.2, 1.65, '(b)')
63 ax.set_xlabel(r'$E_{\text{p}}$ (eV)', size=8)
64 ax.set_xlim(-5.5, -2.1)
65 ax.set_xticks([-5, -4, -3])
66 ax.set_ylabel('')
67 ax.set_ylim(-3.5, 2.5)
68 ax.set_yticklabels([])
69 ax = ax.twinx()
70 ax.set_ylabel(r'$E_{\text{ads}}^{\text{0}}$ (eV)', size=8)
71 ax.set_ylim(-3.5, 2.5)
72
73 ax = fig.add_axes([0.14, 0.13, 0.35, 0.35])
74 ax.plot(eg_Ru, E_Ru, ls='none', c='b', marker='o')
75 ax.plot(eg_Rh, E_Rh, ls='none', c='b', marker='s')
76 ax.plot(eg_Pd, E_Pd, ls='none', c='b', marker='^')
77 ax.plot(eg_Os, E_Os, ls='none', c='c', marker='o')
78 ax.plot(eg_Ir, E_Ir, ls='none', c='c', marker='s')
79 ax.plot(eg_Pt, E_Pt, ls='none', c='c', marker='^')
80 ax.text(-1.6, 1.65, '(c)')
81 ax.set_xlabel(r'$E_{\text{e}}$ (eV)', size=8)
82 ax.set_xlim(-4.3, -0.7)
83 ax.set_xticks([-4, -3, -2, -1])
84 ax.set_ylabel(r'$E_{\text{ads}}^{\text{0}}$ (eV)', size=8)
85 ax.set_ylim(-3.5, 2.5)
86
87 ax = fig.add_axes([0.51, 0.13, 0.35, 0.35])
88 ax.plot(t2g_Ru, E_Ru, ls='none', c='b', marker='o')
89 ax.plot(t2g_Rh, E_Rh, ls='none', c='b', marker='s')
90 ax.plot(t2g_Pd, E_Pd, ls='none', c='b', marker='^')
91 ax.plot(t2g_Os, E_Os, ls='none', c='c', marker='o')
92 ax.plot(t2g_Ir, E_Ir, ls='none', c='c', marker='s')
93 ax.plot(t2g_Pt, E_Pt, ls='none', c='c', marker='^')
94 ax.text(-1.5, 1.65, '(d)')
95 ax.set_xlabel(r'$E_{\text{t}_{2\text{g}}}$ (eV)', size=8)
96 ax.set_xlim(-3.1, -0.9)
97 ax.set_xticks([-3, -2, -1])
98 ax.set_ylabel('')
99 ax.set_ylim(-3.5, 2.5)
100 ax.set_yticklabels([])
101 ax = ax.twinx()
102 ax.set_ylabel(r'$E_{\text{ads}}^{\text{0}}$ (eV)', size=8)
103 ax.set_ylim(-3.5, 2.5)
104
105 plt.savefig('figures/FIG4.png', dpi=300)
106 plt.savefig('figures/FIG4.eps', dpi=300)

```


D. Ligand effect relationships

The sections below details the code for analysis of the ligand effect and construction of Figures 5-8 in the manuscript.

1. Relating changes in the center of the t_{2g} -band to $\Delta\Delta E_{ligand}^O$

The code below relates changes in the center of the t_{2g} -band to $\Delta\Delta E_{ligand}^O$ for all systems calculated in this study and constructs Figure 5 in the manuscript.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labels=8)
6  matplotlib.rc('ytick', labels=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  t2g_Ru, E_Ru = [], []
12  t2g_Rh, E_Rh = [], []
13  t2g_Pd, E_Pd = [], []
14  t2g_Os, E_Os = [], []
15  t2g_Ir, E_Ir = [], []
16  t2g_Pt, E_Pt = [], []
17
18  for dopant in data:
19      for host in data[dopant]:
20          if host[-2:] != 'O2':
21              continue
22          eval('E_' + dopant[:2]).append(data[dopant][host]['delta_E_ligand'])
23          eval('t2g_' + dopant[:2]).append(data[dopant]
24                                          [host]
25                                          ['ligand']['DOS']['d_t2g_center'])
26
27  fig = plt.figure(1, (2.3, 3), dpi=300)
28  ax = fig.add_subplot(111)
29  ax.axhline(0, ls='--', c='k')
30  ax.axvline(0, ls='--', c='k')

```

```

31 ax.plot(t2g_Ru, E_Ru, ls='none', c='b', marker='o')
32 ax.plot(t2g_Rh, E_Rh, ls='none', c='b', marker='s')
33 ax.plot(t2g_Pd, E_Pd, ls='none', c='b', marker='^')
34 ax.plot(t2g_Os, E_Os, ls='none', c='c', marker='o')
35 ax.plot(t2g_Ir, E_Ir, ls='none', c='c', marker='s')
36 ax.plot(t2g_Pt, E_Pt, ls='none', c='c', marker='^')
37 ax.set_xlim(-0.8, 0.85)
38 ax.set_xticks([-0.6, -0.3, 0, 0.3, 0.6])
39 ax.set_ylim(-1.6, 1.6)
40 ax.set_xlabel(r'$\Delta E_{t_{2g}}$ (eV)', size=8)
41 ax.set_ylabel(r'$\Delta E_{\text{ligand}}^{\text{0}}$ (eV)', size=8)
42 plt.tight_layout()
43 plt.savefig('figures/FIG5.png', dpi=300)
44 plt.savefig('figures/FIG5.eps', dpi=300)
45 plt.show()

```

2. Relating the t_{2g} -band center, width, and filling

The code below analyzes relationships between the t_{2g} -band center, width, and filling and constructs Figure 6 in the manuscript.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labels=8)
6  matplotlib.rc('ytick', labels=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  center_Ru, width_Ru, filling_Ru = [], [], []
12  center_Rh, width_Rh, filling_Rh = [], [], []
13  center_Pd, width_Pd, filling_Pd = [], [], []
14  center_Os, width_Os, filling_Os = [], [], []
15  center_Ir, width_Ir, filling_Ir = [], [], []
16  center_Pt, width_Pt, filling_Pt = [], [], []
17
18  for dopant in data:
19      eval('center_' + dopant[:2]).append(data[dopant]['DOS']['t2g_center'])
20      eval('width_' + dopant[:2]).append(data[dopant]['DOS']['t2g_width'])
21      eval('filling_' + dopant[:2]).append(data[dopant]['DOS']['t2g_filling'])
22      for host in data[dopant]:
23          if host[-2:] != '02':

```

```

24         continue
25         eval('center_' + dopant[:2]).append(data[dopant][host]
26                                             ['ligand']['DOS']['t2g_center'])
27         eval('width_' + dopant[:2]).append(data[dopant][host]
28                                             ['ligand']['DOS']['t2g_width'])
29         eval('filling_' + dopant[:2]).append(data[dopant][host]
30                                             ['ligand']['DOS']['t2g_filling'])
31
32 fig = plt.figure(1, (2.5, 4), dpi=300)
33
34 ax = fig.add_axes([0.3, 0.63, 0.6, 0.34])
35 ax.plot(center_Ru, width_Ru, ls='none', c='b', marker='o')
36 ax.plot(center_Rh, width_Rh, ls='none', c='b', marker='s')
37 ax.plot(center_Pd, width_Pd, ls='none', c='b', marker='^')
38 ax.plot(center_Os, width_Os, ls='none', c='c', marker='o')
39 ax.plot(center_Ir, width_Ir, ls='none', c='c', marker='s')
40 ax.plot(center_Pt, width_Pt, ls='none', c='c', marker='^')
41 ax.text(-2.9, 3.1, '(a)')
42 ax.set_xlabel(r'$E_{t_{2g}}$ (eV)', size=8)
43 ax.set_xlim(-3, -1)
44 ax.set_ylabel(r'$W_{t_{2g}}$ (eV)', size=8)
45
46 ax = fig.add_axes([0.3, 0.15, 0.6, 0.34])
47 ax.plot(filling_Ru, center_Ru, ls='none', c='b', marker='o')
48 ax.plot(filling_Rh, center_Rh, ls='none', c='b', marker='s')
49 ax.plot(filling_Pd, center_Pd, ls='none', c='b', marker='^')
50 ax.plot(filling_Os, center_Os, ls='none', c='c', marker='o')
51 ax.plot(filling_Ir, center_Ir, ls='none', c='c', marker='s')
52 ax.plot(filling_Pt, center_Pt, ls='none', c='c', marker='^')
53 ax.text(0.62, -1.35, '(b)')
54 ax.set_xlabel(r'$f_{t_{2g}}$', size=8)
55 ax.set_xticks([0.6, 0.7, 0.8, 0.9, 1.0])
56 ax.set_ylabel(r'$E_{t_{2g}}$ (eV)', size=8)
57 ax.set_ylim(-3, -1)
58
59
60 plt.savefig('figures/FIG6.png', dpi=300)
61 plt.savefig('figures/FIG6.eps', dpi=300)
62 plt.show()

```

3. Relating the d , p , and d^{host} -band fillings

The code below analyzes relationships between changes in the d^{dopant} , p , and d^{host} -band fillings caused by the ligand effect and constructs Figure 7 in the manuscript.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labelsize=8)
6  matplotlib.rc('ytick', labelsize=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  d_Ru, p_Ru, d_Ru_host = [], [], []
12  d_Rh, p_Rh, d_Rh_host = [], [], []
13  d_Pd, p_Pd, d_Pd_host = [], [], []
14  d_Os, p_Os, d_Os_host = [], [], []
15  d_Ir, p_Ir, d_Ir_host = [], [], []
16  d_Pt, p_Pt, d_Pt_host = [], [], []
17
18  for dopant in data:
19      for host in data[dopant]:
20          if host[-2:] != '02':
21              continue
22          eval('d_' + dopant[:2]).append(data[dopant][host]
23                                          ['ligand']['DOS']['d_d_filling'])
24          eval('p_' + dopant[:2]).append(data[dopant][host]
25                                          ['ligand']['DOS']['d_p_filling'])
26          eval('d_' + dopant[:2] + '_host').append(data[host]
27                                                    [dopant]
28                                                    ['dopant']
29                                                    ['DOS']['d_p_filling'])
30
31
32  fig = plt.figure(1, (2.5, 4), dpi=300)
33  ax = fig.add_axes([0.3, 0.63, 0.6, 0.34])
34  ax.axhline(0, ls='--', c='k')
35  ax.axvline(0, ls='--', c='k')
36  ax.plot(d_Ru, p_Ru, ls='none', c='b', marker='o')
37  ax.plot(d_Rh, p_Rh, ls='none', c='b', marker='s')
38  ax.plot(d_Pd, p_Pd, ls='none', c='b', marker='^')
39  ax.plot(d_Os, p_Os, ls='none', c='c', marker='o')
40  ax.plot(d_Ir, p_Ir, ls='none', c='c', marker='s')
41  ax.plot(d_Pt, p_Pt, ls='none', c='c', marker='^')
42  ax.text(-0.11, 0.07, '(a)')
43  ax.set_xlim(-0.12, 0.06)
44  ax.set_xticks([-0.12, -0.06, 0.0, 0.06])
45  ax.set_xlabel(r'$\Delta f_{\text{d}}$', size=8)
46  ax.set_ylim(-0.1, 0.1)

```

```

47 ax.set_ylabel(r'$\Delta f_{\text{p}}$', size=8)
48
49 ax = fig.add_axes([0.3, 0.15, 0.6, 0.34])
50 ax.plot([-0.1, 0.1], [0.1, -0.1], ls='--', c='k')
51 ax.plot(p_Ru, d_Ru_host, ls='none', c='b', marker='o')
52 ax.plot(p_Rh, d_Rh_host, ls='none', c='b', marker='s')
53 ax.plot(p_Pd, d_Pd_host, ls='none', c='b', marker='^')
54 ax.plot(p_Os, d_Os_host, ls='none', c='c', marker='o')
55 ax.plot(p_Ir, d_Ir_host, ls='none', c='c', marker='s')
56 ax.plot(p_Pt, d_Pt_host, ls='none', c='c', marker='^')
57 ax.text(0.065, 0.07, '(a)')
58 ax.set_xlim(-0.1, 0.1)
59 ax.set_xlabel(r'$\Delta f_{\text{p}}$', size=8)
60 ax.set_ylim(-0.1, 0.1)
61 ax.set_ylabel(r'$\Delta f_{\text{d}}^{\text{host}}$', size=8)
62
63
64 plt.savefig('figures/FIG7.png', dpi=300)
65 plt.savefig('figures/FIG7.eps', dpi=300)
66 plt.show()

```

4. Relating t_{2g} -band width and effective orbital radius r_d to $\Delta\Delta E_{\text{ligand}}^{\text{O}}$

The code below relates the ligand effect to differences between the dopant and hosts metal cations t_{2g} -band width and effective orbital radius and constructs Figure 8 of the manuscript.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labelsize=8)
6 matplotlib.rc('ytick', labelsize=8)
7
8 with open('data.json', 'r') as f:
9     data = json.load(f)
10
11 rds = {'Ru': 1.05**(3.0/2.0),
12        'Rh': 0.99**(3.0/2.0),
13        'Pd': 0.94**(3.0/2.0),
14        'Os': 1.13**(3.0/2.0),
15        'Ir': 1.08**(3.0/2.0),
16        'Pt': 1.04**(3.0/2.0),

```

```

17     }
18
19     rd_Ru, t2g_Ru, E_Ru = [], [], []
20     rd_Rh, t2g_Rh, E_Rh = [], [], []
21     rd_Pd, t2g_Pd, E_Pd = [], [], []
22     rd_Os, t2g_Os, E_Os = [], [], []
23     rd_Ir, t2g_Ir, E_Ir = [], [], []
24     rd_Pt, t2g_Pt, E_Pt = [], [], []
25
26     for dopant in data:
27         for host in data[dopant]:
28             if host[-2:] != 'O2':
29                 continue
30             eval('E_' + dopant[:2]).append(data[dopant][host]
31                                             ['delta_E_ligand'])
32             eval('t2g_' + dopant[:2]).append(data[dopant]
33                                             ['DOS']['t2g_width']
34                                             - data[host]
35                                             ['DOS']['t2g_width'])
36             eval('rd_' + dopant[:2]).append(rds[dopant[:2]]
37                                             - rds[host[:2]])
38
39     fig = plt.figure(1, (3.3, 2), dpi=300)
40
41     ax = fig.add_axes([0.18, 0.25, 0.395, 0.7])
42     ax.axhline(0, ls='--', c='k')
43     ax.axvline(0, ls='--', c='k')
44     ax.plot(t2g_Ru, E_Ru, ls='none', c='b', marker='o')
45     ax.plot(t2g_Rh, E_Rh, ls='none', c='b', marker='s')
46     ax.plot(t2g_Pd, E_Pd, ls='none', c='b', marker='^')
47     ax.plot(t2g_Os, E_Os, ls='none', c='c', marker='o')
48     ax.plot(t2g_Ir, E_Ir, ls='none', c='c', marker='s')
49     ax.plot(t2g_Pt, E_Pt, ls='none', c='c', marker='^')
50     ax.text(-0.7, 1.2, '(a)')
51     ax.set_xlim(-0.85, 0.85)
52     ax.set_xticks([-0.80, -0.40, 0.0, 0.40, 0.80])
53     ax.set_xlabel('$W_{t_{2g}}^{\mathrm{dopant}}$'
54                 ' - $W_{t_{2g}}^{\mathrm{host}}$', size=8)
55     ax.set_ylim(-1.75, 1.75)
56     ax.set_ylabel('$\Delta \Delta E_{\mathrm{ligand}}^{\mathrm{0}}$', size=8)
57
58     ax = fig.add_axes([0.59, 0.25, 0.395, 0.7])
59     ax.axhline(0, ls='--', c='k')
60     ax.axvline(0, ls='--', c='k')
61     ax.plot(rd_Ru, E_Ru, ls='none', c='b', marker='o')
62     ax.plot(rd_Rh, E_Rh, ls='none', c='b', marker='s')
63     ax.plot(rd_Pd, E_Pd, ls='none', c='b', marker='^')

```

```

64 ax.plot(rd_Os, E_Os, ls='none', c='c', marker='o')
65 ax.plot(rd_Ir, E_Ir, ls='none', c='c', marker='s')
66 ax.plot(rd_Pt, E_Pt, ls='none', c='c', marker='^')
67 ax.text(-0.3, 1.2, '(b)')
68 ax.set_xlim(-0.35, 0.35)
69 ax.set_xticks([-0.2, 0.0, 0.2])
70 ax.set_xlabel(r'\left(r_{d}^{3/2}\right)^{\mathdefault{dopant}}',
71              r' - \left(r_{d}^{3/2}\right)^{\mathdefault{host}}$',
72              size=8)
73 ax.set_ylim(-1.75, 1.75)
74 ax.set_yticklabels([])
75
76 plt.savefig('figures/FIG8.png', dpi=300)
77 plt.savefig('figures/FIG8.eps', dpi=300)
78 plt.show()

```

E. Strain effect relationships

The sections below details the code for analysis of the strain effect and construction of Figures 9-11 in the manuscript.

1. Relating strain to changes in adsorption energy

The code below relates changes to the adsorption energy caused by strain and constructs Figure 9 in the manuscript.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labelsize=8)
6  matplotlib.rc('ytick', labelsize=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  E_Ru, strain_Ru = [], []
12  E_Rh, strain_Rh = [], []
13  E_Pd, strain_Pd = [], []
14  E_Os, strain_Os = [], []
15  E_Ir, strain_Ir = [], []
16  E_Pt, strain_Pt = [], []

```

```

17
18 for dopant in data:
19     d0 = data[dopant]['bare']['cell'][0][0]
20     for host in data[dopant]:
21         if host[-2:] != 'O2':
22             continue
23         eval('E_' + dopant[:2]).append(data[dopant][host]['delta_E_strain'])
24         d = data[dopant][host]['strain']['bare']['cell'][0][0]
25         eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
26
27 fig = plt.figure(1, (2.3, 3), dpi=300)
28
29 ax = fig.add_subplot(111)
30 ax.plot(strain_Ru, E_Ru, ls='none', c='b', marker='o')
31 ax.plot(strain_Rh, E_Rh, ls='none', c='b', marker='s')
32 ax.plot(strain_Pd, E_Pd, ls='none', c='b', marker='^')
33 ax.plot(strain_Os, E_Os, ls='none', c='c', marker='o')
34 ax.plot(strain_Ir, E_Ir, ls='none', c='c', marker='s')
35 ax.plot(strain_Pt, E_Pt, ls='none', c='c', marker='^')
36 ax.set_xlim(-5, 5)
37 ax.set_xticks([-4, -2, 0, 2, 4])
38 ax.set_xlabel('Strain %', size=8)
39 ax.set_ylabel(r'$\Delta E_{\text{strain}}$ (eV)', size=8)
40 plt.tight_layout()
41 plt.savefig('figures/FIG9.png', dpi=300)
42 plt.savefig('figures/FIG9.eps', dpi=300)
43 plt.show()

```

2. Relating strain, the t_{2g} -band width, and the t_{2g} -band center

The code below relates strain induced changes to the t_{2g} -band width and center and constructs Figure 10 in the manuscript.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labelsizes=8)
6 matplotlib.rc('ytick', labelsizes=8)
7
8 with open('data.json', 'r') as f:
9     data = json.load(f)
10
11 center_Ru, width_Ru, strain_Ru = [], [], []

```



```

12 center_Rh, width_Rh, strain_Rh = [], [], []
13 center_Pd, width_Pd, strain_Pd = [], [], []
14 center_Os, width_Os, strain_Os = [], [], []
15 center_Ir, width_Ir, strain_Ir = [], [], []
16 center_Pt, width_Pt, strain_Pt = [], [], []
17
18 for dopant in data:
19     d0 = data[dopant]['bare']['cell'][0][0]
20     for host in data[dopant]:
21         if host[-2:] != 'O2':
22             continue
23         eval('center_' + dopant[:2]).append(data[dopant]
24                                             [host]
25                                             ['strain']
26                                             ['DOS']['d_t2g_center'])
27         eval('width_' + dopant[:2]).append(data[dopant]
28                                             [host]['strain']
29                                             ['DOS']['d_t2g_width'])
30         d = data[dopant][host]['strain']['bare']['cell'][0][0]
31         eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
32
33 fig = plt.figure(1, (2.5, 3.5), dpi=300)
34
35 ax = fig.add_axes([0.3, 0.56, 0.6, 0.4])
36 ax.plot(strain_Ru, width_Ru, ls='none', c='b', marker='o')
37 ax.plot(strain_Rh, width_Rh, ls='none', c='b', marker='s')
38 ax.plot(strain_Pd, width_Pd, ls='none', c='b', marker='^')
39 ax.plot(strain_Os, width_Os, ls='none', c='c', marker='o')
40 ax.plot(strain_Ir, width_Ir, ls='none', c='c', marker='s')
41 ax.plot(strain_Pt, width_Pt, ls='none', c='c', marker='^')
42 ax.text(-4.2, 0.17, '(a)')
43 ax.set_xlim(-5, 5)
44 ax.set_xticklabels([])
45 ax.set_ylim(-0.25, 0.25)
46 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
47 ax.set_ylabel(r'$\Delta W_{t_{2g}}$ (eV)', size=8)
48
49 ax = fig.add_axes([0.3, 0.14, 0.6, 0.4])
50 ax.plot(strain_Ru, center_Ru, ls='none', c='b', marker='o')
51 ax.plot(strain_Rh, center_Rh, ls='none', c='b', marker='s')
52 ax.plot(strain_Pd, center_Pd, ls='none', c='b', marker='^')
53 ax.plot(strain_Os, center_Os, ls='none', c='c', marker='o')
54 ax.plot(strain_Ir, center_Ir, ls='none', c='c', marker='s')
55 ax.plot(strain_Pt, center_Pt, ls='none', c='c', marker='^')
56 ax.text(-4.2, 0.12, '(b)')
57 ax.set_xlim(-5, 5)
58 ax.set_xlabel(r'Strain %', size=8)

```

```

59 ax.set_ylim(-0.12, 0.17)
60 ax.set_ylabel(r'$\Delta E_{t_{2g}}$ (eV)', size=8)
61
62
63 plt.savefig('figures/FIG10.png', dpi=300)
64 plt.savefig('figures/FIG10.eps', dpi=300)
65 plt.show()

```

3. Relating t_{2g} -band center and $\Delta\Delta E_{strain}^O$

The code below relates changes in the t_{2g} -band center to strain effect as well the slope of this relationship to the idealized occupancy of electrons in the metal cation. The code constructs Figure 11 in the manuscript.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import json
5  from pycse import regress
6
7  matplotlib.rc('xtick', labelsize=8)
8  matplotlib.rc('ytick', labelsize=8)
9
10 with open('data.json', 'r') as f:
11     data = json.load(f)
12
13 slope_Ru, dn_Ru, filling_Ru, center_Ru, strain_Ru, E_Ru = [], [], [], [], [], []
14 slope_Rh, dn_Rh, filling_Rh, center_Rh, strain_Rh, E_Rh = [], [], [], [], [], []
15 slope_Pd, dn_Pd, filling_Pd, center_Pd, strain_Pd, E_Pd = [], [], [], [], [], []
16 slope_Os, dn_Os, filling_Os, center_Os, strain_Os, E_Os = [], [], [], [], [], []
17 slope_Ir, dn_Ir, filling_Ir, center_Ir, strain_Ir, E_Ir = [], [], [], [], [], []
18 slope_Pt, dn_Pt, filling_Pt, center_Pt, strain_Pt, E_Pt = [], [], [], [], [], []
19
20 for dopant in data:
21     if dopant in ['RuO2', 'OsO2']:
22         eval('dn_' + dopant[:2]).append(4)
23     elif dopant in ['RhO2', 'IrO2']:
24         eval('dn_' + dopant[:2]).append(5)
25     else:
26         eval('dn_' + dopant[:2]).append(6)
27     d0 = data[dopant]['bare']['cell'][0][0]
28     t2g, E = [], []
29     for host in data[dopant]:

```

```

30         if host[-2:] != '02':
31             continue
32         E.append(data[dopant][host]['delta_E_strain'])
33         t2g.append(data[dopant][host]['strain']['DOS']['d_t2g_center'])
34         eval('E_' + dopant[:2]).append(data[dopant][host]['delta_E_strain'])
35         eval('center_' + dopant[:2]).append(data[dopant]
36                                             [host]['strain']
37                                             ['DOS']['d_t2g_center'])
38         eval('filling_' + dopant[:2]).append(data[dopant][host]
39                                             ['strain']['DOS']
40                                             ['d_t2g_filling'])
41         d = data[dopant][host]['strain']['bare']['cell'][0][0]
42         eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
43     E.append(0)
44     t2g.append(0)
45     E = np.array(E)
46     t2g = np.array(t2g)
47     A = np.column_stack([t2g**0, t2g])
48     alpha = 0.05
49     p, pint, se = regress(A, E, alpha)
50     m = p[1]
51     mint = pint[1]
52     eval('slope_' + dopant[:2]).append(m)
53     eval('slope_' + dopant[:2]).append(abs(m - mint[1]))
54     eval('slope_' + dopant[:2]).append(abs(m - mint[0]))
55
56
57 fig = plt.figure(1, (2.5, 4), dpi=300)
58
59 ax = fig.add_axes([0.3, 0.63, 0.6, 0.34])
60 ax.plot(center_Ru, E_Ru, ls='none', c='b', marker='o')
61 ax.plot(center_Rh, E_Rh, ls='none', c='b', marker='s')
62 ax.plot(center_Pd, E_Pd, ls='none', c='b', marker='^')
63 ax.plot(center_Os, E_Os, ls='none', c='c', marker='o')
64 ax.plot(center_Ir, E_Ir, ls='none', c='c', marker='s')
65 ax.plot(center_Pt, E_Pt, ls='none', c='c', marker='^')
66 ax.text(-0.13, 0.12, '(a)')
67 ax.set_xlim(-0.15, 0.15)
68 ax.set_xticks([-0.1, 0, 0.1])
69 ax.set_xlabel(r'$\Delta E_{t_{2g}}$', size=8)
70 ax.set_ylim(-0.175, 0.175)
71 ax.set_yticks([-0.1, 0, 0.1])
72 ax.set_ylabel(r'$\Delta \Delta E_{\text{strain}}^{\text{0}}$ (eV)', size=8)
73
74 ax = fig.add_axes([0.3, 0.15, 0.6, 0.34])
75 ax.errorbar(dn_Ru, slope_Ru[0], ls='none', yerr=slope_Ru[1-2],
76            c='b', marker='o')

```

```

77 ax.errorbar(dn_Rh, slope_Rh[0], ls='none', yerr=slope_Rh[1-2],
78             c='b', marker='s')
79 ax.errorbar(dn_Pd, slope_Pd[0], ls='none', yerr=slope_Pd[1-2],
80             c='b', marker='^')
81 ax.errorbar(dn_Os, slope_Os[0], ls='none', yerr=slope_Os[1-2],
82             c='c', marker='o')
83 ax.errorbar(dn_Ir, slope_Ir[0], ls='none', yerr=slope_Ir[1-2],
84             c='c', marker='s')
85 ax.errorbar(dn_Pt, slope_Pt[0], ls='none', yerr=slope_Pt[1-2],
86             c='c', marker='^')
87 ax.text(5.9, 1.7, '(b)')
88
89 # Now want to fit a line to this
90 xs = dn_Ru + dn_Rh + dn_Pd + dn_Os + dn_Ir + dn_Pt
91 ys = [slope_Ru[0], slope_Rh[0], slope_Pd[0],
92       slope_Os[0], slope_Ir[0], slope_Pt[0]]
93 yfit = np.poly1d(np.polyfit(xs, ys, 1))
94 ax.plot([3, 7], yfit([3, 7]), zorder=0)
95
96 ax.set_xlim(3.5, 6.5)
97 ax.set_xticks([4, 5, 6])
98 ax.set_xlabel(r'$d^{\{n\}}$', size=8)
99 ax.set_ylim(-2.5, 2.5)
100 ax.set_yticks([-2, -1, 0, 1, 2])
101 ax.set_ylabel(r'$\frac{d\Delta}{\Delta E_{\text{strain}}}$',
102             r'$\frac{d\Delta E_{\{2g\}}}{\Delta E_{\{2g\}}}$', size=8)
103
104 plt.savefig('figures/FIG11.png', dpi=300)
105 plt.savefig('figures/FIG11.eps', dpi=300)
106 plt.show()

```

VI. ADDITIONAL OF RELATIONSHIPS

The sections below some additional relations between the strain, the electronic structure, and adsorption energies we performed but not include as figures in the manuscript. Some of these figures are referred to in the manuscript.

A. Relating the ligand effect to changes in the center of the d -band

The code relates the ligand effect to changes in the center of the d -band. The correlation is weak, and the correlation with changes in the t_{2g} -band was stronger. The figure is stored

in the supporting-figures, named as ligand-vs-delta-d-center.png.

```
1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labels=8)
6 matplotlib.rc('ytick', labels=8)
7
8 with open('data.json', 'r') as f:
9     data = json.load(f)
10
11 d_Ru, E_Ru = [], []
12 d_Rh, E_Rh = [], []
13 d_Pd, E_Pd = [], []
14 d_Os, E_Os = [], []
15 d_Ir, E_Ir = [], []
16 d_Pt, E_Pt = [], []
17
18 for dopant in data:
19     for host in data[dopant]:
20         if host[-2:] != 'O2':
21             continue
22         eval('E_' + dopant[:2]).append(data[dopant]
23                                         [host]['delta_E_ligand'])
24         eval('d_' + dopant[:2]).append(data[dopant]
25                                         [host]['ligand'])
26         ['DOS']['d_d_center'])
27
28 fig = plt.figure(1, (2.3, 3), dpi=300)
29 ax = fig.add_subplot(111)
30 ax.axhline(0, ls='--', c='k')
31 ax.axvline(0, ls='--', c='k')
32 ax.plot(d_Ru, E_Ru, ls='none', c='b', marker='o')
33 ax.plot(d_Rh, E_Rh, ls='none', c='b', marker='s')
34 ax.plot(d_Pd, E_Pd, ls='none', c='b', marker='^')
35 ax.plot(d_Os, E_Os, ls='none', c='c', marker='o')
36 ax.plot(d_Ir, E_Ir, ls='none', c='c', marker='s')
37 ax.plot(d_Pt, E_Pt, ls='none', c='c', marker='^')
38 ax.set_xlim(-0.8, 0.85)
39 ax.set_xticks([-0.6, -0.3, 0, 0.3, 0.6])
40 ax.set_ylim(-1.6, 1.6)
41 ax.set_xlabel(r'$\Delta E_{t_{2g}}$ (eV)', size=8)
42 ax.set_ylabel(r'$\Delta \Delta E_{ligand}^{\mathrm{0}}$ (eV)', size=8)
43 plt.tight_layout()
44 plt.savefig('supporting-figures/ligand-vs-delta-d-center.png', dpi=300)
```

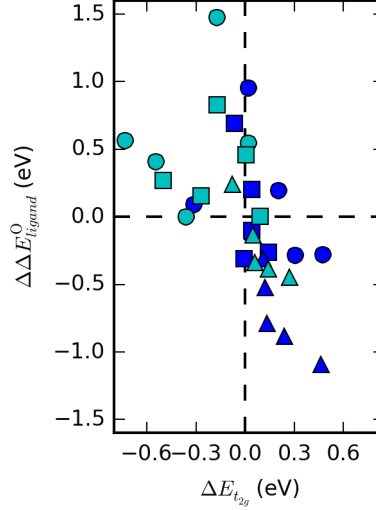


FIG. 1. Relationship between the ligand effect ($\Delta\Delta E_{ligand}^O$) in system $M^D-M^H O_2$ and the corresponding difference in the center of the d -band (E_d) between the system $M^D O_2$ and $M^D-M^H O_2$ with the lattice constant of $M^D O_2$. Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

B. Relating ligand induced changes to fillings of d and t_{2g} -band

The figure below relates changes in the filling of the t_{2g} -band to changes in the filling of the d -band caused by the ligand effect. A constant d -band filling would imply no charge transfer to the metal cation caused by the ligand effect. The figure is stored in the `supporting-figures`, named as `ligand-d-vs-t2g-filling.png`.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labels=8)
6 matplotlib.rc('ytick', labels=8)
7
8 with open('data.json', 'r') as f:

```

```

9     data = json.load(f)
10
11     t2g_Ru, d_Ru = [], []
12     t2g_Rh, d_Rh = [], []
13     t2g_Pd, d_Pd = [], []
14     t2g_Os, d_Os = [], []
15     t2g_Ir, d_Ir = [], []
16     t2g_Pt, d_Pt = [], []
17
18     for dopant in data:
19         for host in data[dopant]:
20             if host[-2:] != 'O2':
21                 continue
22             eval('d_' + dopant[:2]).append(data[dopant]
23                                             [host]['ligand']
24                                             ['DOS']['d_d_filling'])
25             eval('t2g_' + dopant[:2]).append(data[dopant]
26                                             [host]['ligand']
27                                             ['DOS']
28                                             ['d_t2g_filling'])
29
30     fig = plt.figure(1, (2.3, 3))
31     ax = fig.add_subplot(111)
32     ax.axhline(0, ls='--', c='k')
33     ax.axvline(0, ls='--', c='k')
34     ax.plot(t2g_Ru, d_Ru, ls='none', c='b', marker='o')
35     ax.plot(t2g_Rh, d_Rh, ls='none', c='b', marker='s')
36     ax.plot(t2g_Pd, d_Pd, ls='none', c='b', marker='^')
37     ax.plot(t2g_Os, d_Os, ls='none', c='c', marker='o')
38     ax.plot(t2g_Ir, d_Ir, ls='none', c='c', marker='s')
39     ax.plot(t2g_Pt, d_Pt, ls='none', c='c', marker='^')
40     ax.set_xlim(-0.2, 0.2)
41     ax.set_xticks([-0.2, -0.1, 0, 0.1, 0.2])
42     ax.set_xlabel(r'$\Delta f_{t_{2g}}$ (eV)', size=8)
43     ax.set_ylabel(r'$\Delta f_{d}$ (eV)', size=8)
44     plt.tight_layout()
45     plt.savefig('supporting-figures/ligand-d-vs-t2g-filling.png', dpi=300)
46     plt.show()

```

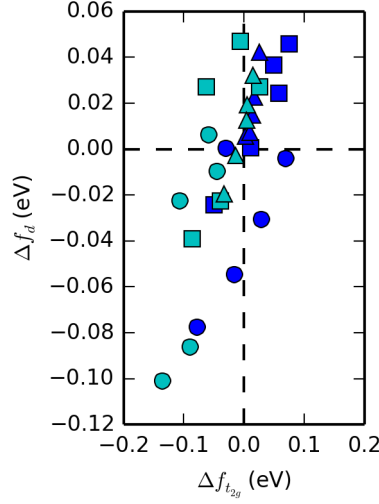


FIG. 2. Relationship between the change on the M^D t_{2g} -band filling ($\Delta f_{t_{2g}}$) and d -band filling (Δf_d) caused by the ligand effect. Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

C. Relating differences in the t_{2g} -band width and charge transfer

One of the conclusions made in the manuscript was that differences in the overlap of the t_{2g} -orbitals caused by the ligand effect results in charge transfer. This relationship was implied by relationships shown in Figure 6, 7, and 8, but this is verified below. The figure below shows that differences in the overlap of atomic orbitals between the host and dopant, which shown by differences in the calculated t_{2g} -band width ($W_{t_{2g}}$) and tabulated effective orbital radius to the 3/2 power ($r_d^{3/2}$), results in charge transfer, which is shown by changes in the filling of the t_{2g} -orbital.

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 import json
4
5 matplotlib.rc('xtick', labels=8)
6 matplotlib.rc('ytick', labels=8)
7
8 with open('data.json', 'r') as f:

```



```

9     data = json.load(f)
10
11     rds = {'Ru': 1.05**(3.0/2.0),
12           'Rh': 0.99**(3.0/2.0),
13           'Pd': 0.94**(3.0/2.0),
14           'Os': 1.13**(3.0/2.0),
15           'Ir': 1.08**(3.0/2.0),
16           'Pt': 1.04**(3.0/2.0),
17           }
18
19     rd_Ru, t2g_Ru, filling_Ru = [], [], []
20     rd_Rh, t2g_Rh, filling_Rh = [], [], []
21     rd_Pd, t2g_Pd, filling_Pd = [], [], []
22     rd_Os, t2g_Os, filling_Os = [], [], []
23     rd_Ir, t2g_Ir, filling_Ir = [], [], []
24     rd_Pt, t2g_Pt, filling_Pt = [], [], []
25
26     for dopant in data:
27         for host in data[dopant]:
28             if host[-2:] != 'O2':
29                 continue
30             eval('filling_' + dopant[:2]).append(data[dopant][host]['ligand']
31                                                  ['DOS']['d_t2g_filling'])
32             eval('t2g_' + dopant[:2]).append(data[dopant]
33                                              ['DOS']['t2g_width']
34                                              - data[host]
35                                              ['DOS']['t2g_width'])
36             eval('rd_' + dopant[:2]).append(rds[dopant[:2]]
37                                             - rds[host[:2]])
38
39     fig = plt.figure(1, (3.3, 2), dpi=300)
40
41     ax = fig.add_axes([0.18, 0.25, 0.395, 0.7])
42     ax.axhline(0, ls='--', c='k')
43     ax.axvline(0, ls='--', c='k')
44     ax.plot(t2g_Ru, filling_Ru, ls='none', c='b', marker='o')
45     ax.plot(t2g_Rh, filling_Rh, ls='none', c='b', marker='s')
46     ax.plot(t2g_Pd, filling_Pd, ls='none', c='b', marker='^')
47     ax.plot(t2g_Os, filling_Os, ls='none', c='c', marker='o')
48     ax.plot(t2g_Ir, filling_Ir, ls='none', c='c', marker='s')
49     ax.plot(t2g_Pt, filling_Pt, ls='none', c='c', marker='^')
50     ax.text(-0.7, 1.2, '(a)')
51     ax.set_xlim(-0.85, 0.85)
52     ax.set_xticks([-0.80, -0.40, 0.0, 0.40, 0.80])
53     ax.set_xlabel('$W_{t_{2g}}^{\mathsf{\mathsf{dopant}}}$'
54                  ' - $W_{t_{2g}}^{\mathsf{\mathsf{host}}}$', size=8)
55     ax.set_ylim(-0.15, 0.15)

```

```

56 ax.set_yticks([-0.1, 0.0, 0.1])
57 ax.set_ylabel('$\Delta f_{t_{2g}}$', size=8)
58
59 ax = fig.add_axes([0.59, 0.25, 0.395, 0.7])
60 ax.axhline(0, ls='--', c='k')
61 ax.axvline(0, ls='--', c='k')
62 ax.plot(rd_Ru, filling_Ru, ls='none', c='b', marker='o')
63 ax.plot(rd_Rh, filling_Rh, ls='none', c='b', marker='s')
64 ax.plot(rd_Pd, filling_Pd, ls='none', c='b', marker='^')
65 ax.plot(rd_Os, filling_Os, ls='none', c='c', marker='o')
66 ax.plot(rd_Ir, filling_Ir, ls='none', c='c', marker='s')
67 ax.plot(rd_Pt, filling_Pt, ls='none', c='c', marker='^')
68 ax.text(-0.3, 1.2, '(b)')
69 ax.set_ylim(-0.15, 0.15)
70 ax.set_yticks([-0.1, 0.0, 0.1])
71 ax.set_xlim(-0.35, 0.35)
72 ax.set_xticks([-0.2, 0.0, 0.2])
73 ax.set_xlabel(r'$\left(r_{d}^{3/2}\right)^{\mathrm{dopant}}$'
74             r' - $\left(r_{d}^{3/2}\right)^{\mathrm{host}}$',
75             size=8)
76 #ax.set_ylim(-1.75, 1.75)
77 ax.set_yticklabels([])
78
79 plt.savefig('supporting-figures/width-vs-charge.png', dpi=300)
80 plt.savefig('supporting-figures/width-vs-charge.eps', dpi=300)
81 plt.show()

```

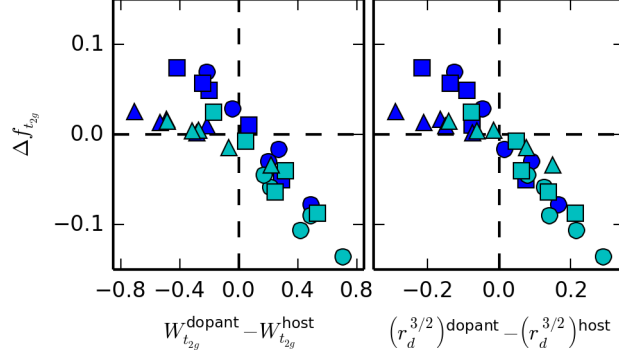


FIG. 3. Relationship between ligand effect induced charge transfer ($\Delta f_{t_{2g}}$) in the dopant-host system $M^D-M^H\text{O}_2$ and the difference in (a) the t_{2g} -band width ($W_{t_{2g}}$) and (b) the tabulated effective orbital radius to the 3/2 power ($r_d^{3/2}$) of dopant M^D and host M^H . Black dashed line in both (a) and (b) show the $x=0$ and $y=0$ axis. Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

D. Relating charge transfer to changes in the adsorption energy

One of the conclusions made in the manuscript was that ligand induced charge transfer results in changes in the adsorption energy. This relationship was implied by relationships shown in Figure 5 and 6 but this is verified below.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labels=8)
6  matplotlib.rc('ytick', labels=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  t2g_Ru, E_Ru = [], []
12  t2g_Rh, E_Rh = [], []
13  t2g_Pd, E_Pd = [], []
14  t2g_Os, E_Os = [], []
15  t2g_Ir, E_Ir = [], []
16  t2g_Pt, E_Pt = [], []

```

```

17
18 for dopant in data:
19     for host in data[dopant]:
20         if host[-2:] != 'O2':
21             continue
22         eval('E_' + dopant[:2]).append(data[dopant][host]['delta_E_ligand'])
23         eval('t2g_' + dopant[:2]).append(data[dopant]
24                                         [host]
25                                         ['ligand']['DOS']['d_t2g_filling'])
26
27 fig = plt.figure(1, (2.3, 3))
28 ax = fig.add_subplot(111)
29 ax.axhline(0, ls='--', c='k')
30 ax.axvline(0, ls='--', c='k')
31 ax.plot(t2g_Ru, E_Ru, ls='none', c='b', marker='o')
32 ax.plot(t2g_Rh, E_Rh, ls='none', c='b', marker='s')
33 ax.plot(t2g_Pd, E_Pd, ls='none', c='b', marker='^')
34 ax.plot(t2g_Os, E_Os, ls='none', c='c', marker='o')
35 ax.plot(t2g_Ir, E_Ir, ls='none', c='c', marker='s')
36 ax.plot(t2g_Pt, E_Pt, ls='none', c='c', marker='^')
37 ax.set_xlim(-0.16, 0.16)
38 ax.set_xticks([-0.16, -0.08, 0, 0.08, 0.16])
39 ax.set_ylim(-1.6, 1.6)
40 ax.set_xlabel(r'$\Delta f_{t_{2g}}$ (eV)', size=8)
41 ax.set_ylabel(r'$\Delta \Delta E_{\text{ligand}}^{\mathrm{0}}$ (eV)', size=8)
42 plt.tight_layout()
43 plt.savefig('supporting-figures/charge-vs-energy.png', dpi=300)
44 plt.savefig('supporting-figures/charge-vs-energy.eps', dpi=300)
45 plt.show()

```

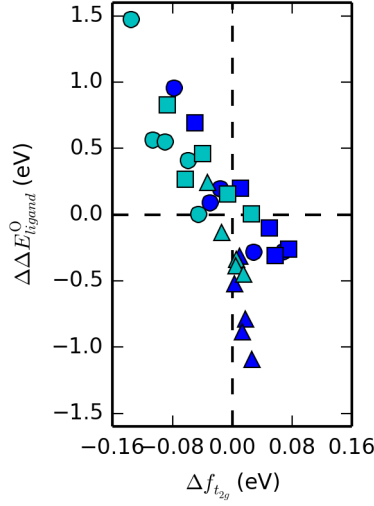


FIG. 4. Relationship between the ligand effect ($\Delta\Delta E_{ligand}^O$) in system $M^D-M^H\text{O}_2$ and the corresponding difference in the filling of the t_{2g} -band ($\Delta f_{t_{2g}}$) between the system $M^D\text{O}_2$ and $M^D-M^H\text{O}_2$ with the lattice constant of $M^D\text{O}_2$. Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

E. Relating strain to changes in band widths

The code below reads data from the **data.json** file and constructs figures that show the dependence of the change in the d , p , t_{2g} , and e_g -band widths when applying a certain amount of strain. The figure is stored in the **supporting-figures**, named as **strain-vs-widths.png**

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labels=8)
6  matplotlib.rc('ytick', labels=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  d_Ru, t2g_Ru, eg_Ru, p_Ru, strain_Ru = [], [], [], [], []

```

```

12 d_Rh, t2g_Rh, eg_Rh, p_Rh, strain_Rh = [], [], [], [], []
13 d_Pd, t2g_Pd, eg_Pd, p_Pd, strain_Pd = [], [], [], [], []
14 d_Os, t2g_Os, eg_Os, p_Os, strain_Os = [], [], [], [], []
15 d_Ir, t2g_Ir, eg_Ir, p_Ir, strain_Ir = [], [], [], [], []
16 d_Pt, t2g_Pt, eg_Pt, p_Pt, strain_Pt = [], [], [], [], []
17
18 for dopant in data:
19     d0 = data[dopant]['bare']['cell'][0][0]
20     for host in data[dopant]:
21         if host[-2:] != 'O2':
22             continue
23         eval('d_' + dopant[:2]).append(data[dopant]
24                                         [host]['strain']
25                                         ['DOS']['d_d_width'])
26         eval('t2g_' + dopant[:2]).append(data[dopant][host]
27                                         ['strain']['DOS']
28                                         ['d_t2g_width'])
29         eval('eg_' + dopant[:2]).append(data[dopant][host]
30                                         ['strain']['DOS']
31                                         ['d_eg_width'])
32         eval('p_' + dopant[:2]).append(data[dopant][host]
33                                         ['strain']['DOS']
34                                         ['d_p_width'])
35         d = data[dopant][host]['strain']['bare']['cell'][0][0]
36         eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
37
38 fig = plt.figure(1, (3.3, 3.3), dpi=300)
39
40 ax = fig.add_axes([0.17, 0.51, 0.32, 0.36])
41 ax.plot(strain_Ru, d_Ru, ls='none', c='b', marker='o')
42 ax.plot(strain_Rh, d_Rh, ls='none', c='b', marker='s')
43 ax.plot(strain_Pd, d_Pd, ls='none', c='b', marker='^')
44 ax.plot(strain_Os, d_Os, ls='none', c='c', marker='o')
45 ax.plot(strain_Ir, d_Ir, ls='none', c='c', marker='s')
46 ax.plot(strain_Pt, d_Pt, ls='none', c='c', marker='^')
47 ax.set_xlim(-5, 5)
48 ax.set_ylim(-0.25, 0.25)
49 ax.set_xticklabels([])
50 ax.set_ylabel(r'$\Delta W_{d}$ (eV)', size=8)
51 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
52
53 ax = fig.add_axes([0.51, 0.51, 0.32, 0.36])
54 ax.plot(strain_Ru, p_Ru, ls='none', c='b', marker='o')
55 ax.plot(strain_Rh, p_Rh, ls='none', c='b', marker='s')
56 ax.plot(strain_Pd, p_Pd, ls='none', c='b', marker='^')
57 ax.plot(strain_Os, p_Os, ls='none', c='c', marker='o')
58 ax.plot(strain_Ir, p_Ir, ls='none', c='c', marker='s')

```

```

59 ax.plot(strain_Pt, p_Pt, ls='none', c='c', marker='^')
60 ax.set_xlim(-5, 5)
61 ax.set_ylim(-0.25, 0.25)
62 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
63 ax.set_xticklabels([])
64 ax.set_yticklabels([])
65 ax = ax.twinx()
66 ax.set_ylim(-0.25, 0.25)
67 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
68 ax.set_ylabel(r'$\Delta W_{\text{p}}$ (eV)', size=8)
69
70 ax = fig.add_axes([0.17, 0.13, 0.32, 0.36])
71 ax.plot(strain_Ru, eg_Ru, ls='none', c='b', marker='o')
72 ax.plot(strain_Rh, eg_Rh, ls='none', c='b', marker='s')
73 ax.plot(strain_Pd, eg_Pd, ls='none', c='b', marker='^')
74 ax.plot(strain_0s, eg_0s, ls='none', c='c', marker='o')
75 ax.plot(strain_Ir, eg_Ir, ls='none', c='c', marker='s')
76 ax.plot(strain_Pt, eg_Pt, ls='none', c='c', marker='^')
77 ax.set_xlim(-5, 5)
78 ax.set_xlabel(r'Strain %', size=8)
79 ax.set_ylim(-0.35, 0.35)
80 ax.set_ylabel(r'$\Delta W_{\text{e}}$ (eV)', size=8)
81
82 ax = fig.add_axes([0.51, 0.13, 0.32, 0.36])
83 ax.plot(strain_Ru, t2g_Ru, ls='none', c='b', marker='o')
84 ax.plot(strain_Rh, t2g_Rh, ls='none', c='b', marker='s')
85 ax.plot(strain_Pd, t2g_Pd, ls='none', c='b', marker='^')
86 ax.plot(strain_0s, t2g_0s, ls='none', c='c', marker='o')
87 ax.plot(strain_Ir, t2g_Ir, ls='none', c='c', marker='s')
88 ax.plot(strain_Pt, t2g_Pt, ls='none', c='c', marker='^')
89 ax.set_xlim(-5, 5)
90 ax.set_ylim(-0.25, 0.25)
91 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
92 ax.set_yticklabels([])
93 ax.set_xlabel(r'Strain %', size=8)
94 ax = ax.twinx()
95 ax.set_ylim(-0.25, 0.25)
96 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
97 ax.set_ylabel(r'$\Delta W_{\text{t}_{2\text{g}}}$ (eV)', size=8)
98
99 plt.savefig('supporting-figures/strain-vs-widths.png', dpi=300)
100 plt.show()

```

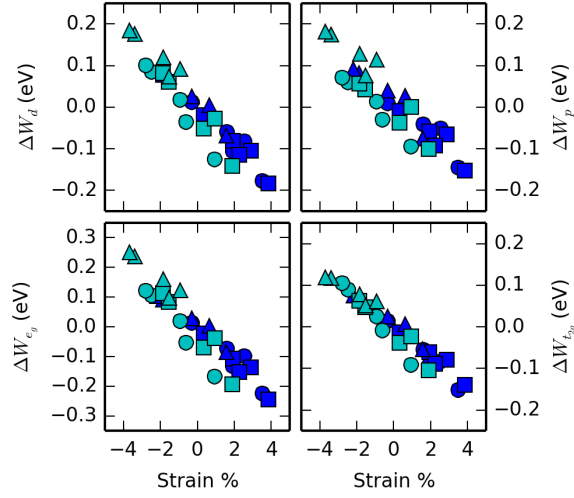


FIG. 5. Relationship between the relative strain in the direction parallel to the surface and the change in the (a) d -band width (ΔW_d), (b) p -band width (ΔW_p), (c) e_g -band width (ΔW_{e_g}), and (d) t_{2g} -band width ($\Delta W_{t_{2g}}$). Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

F. Relating strain to changes in band centers

The code below reads data from the **data.json** file and constructs figures that show the dependence of the change in the d , p , t_{2g} , and e_g -band centers when applying a certain amount of strain. The figure is stored in the **supporting-figures**, named as **strain-vs-widths.png**.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labelsiz=8)
6  matplotlib.rc('ytick', labelsiz=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  d_Ru, t2g_Ru, eg_Ru, p_Ru, strain_Ru = [], [], [], [], []

```



```

12 d_Rh, t2g_Rh, eg_Rh, p_Rh, strain_Rh = [], [], [], [], []
13 d_Pd, t2g_Pd, eg_Pd, p_Pd, strain_Pd = [], [], [], [], []
14 d_Os, t2g_Os, eg_Os, p_Os, strain_Os = [], [], [], [], []
15 d_Ir, t2g_Ir, eg_Ir, p_Ir, strain_Ir = [], [], [], [], []
16 d_Pt, t2g_Pt, eg_Pt, p_Pt, strain_Pt = [], [], [], [], []
17
18 for dopant in data:
19     d0 = data[dopant]['bare']['cell'][0][0]
20     for host in data[dopant]:
21         if host[-2:] != 'O2':
22             continue
23         eval('d_' + dopant[:2]).append(data[dopant]
24                                         [host]['strain']
25                                         ['DOS']['d_d_center'])
26         eval('t2g_' + dopant[:2]).append(data[dopant][host]
27                                         ['strain']['DOS']
28                                         ['d_t2g_center'])
29         eval('eg_' + dopant[:2]).append(data[dopant][host]
30                                         ['strain']['DOS']
31                                         ['d_eg_center'])
32         eval('p_' + dopant[:2]).append(data[dopant][host]
33                                         ['strain']['DOS']
34                                         ['d_p_center'])
35         d = data[dopant][host]['strain']['bare']['cell'][0][0]
36         eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
37
38 fig = plt.figure(1, (3.3, 3.3), dpi=300)
39
40 ax = fig.add_axes([0.17, 0.51, 0.32, 0.36])
41 ax.plot(strain_Ru, d_Ru, ls='none', c='b', marker='o')
42 ax.plot(strain_Rh, d_Rh, ls='none', c='b', marker='s')
43 ax.plot(strain_Pd, d_Pd, ls='none', c='b', marker='^')
44 ax.plot(strain_Os, d_Os, ls='none', c='c', marker='o')
45 ax.plot(strain_Ir, d_Ir, ls='none', c='c', marker='s')
46 ax.plot(strain_Pt, d_Pt, ls='none', c='c', marker='^')
47 ax.set_xlim(-5, 5)
48 ax.set_ylim(-0.25, 0.25)
49 ax.set_xticklabels([])
50 ax.set_ylabel(r'$\Delta E_{\text{d}}$ (eV)', size=8)
51 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
52
53 ax = fig.add_axes([0.51, 0.51, 0.32, 0.36])
54 ax.plot(strain_Ru, p_Ru, ls='none', c='b', marker='o')
55 ax.plot(strain_Rh, p_Rh, ls='none', c='b', marker='s')
56 ax.plot(strain_Pd, p_Pd, ls='none', c='b', marker='^')
57 ax.plot(strain_Os, p_Os, ls='none', c='c', marker='o')
58 ax.plot(strain_Ir, p_Ir, ls='none', c='c', marker='s')

```

```

59 ax.plot(strain_Pt, p_Pt, ls='none', c='c', marker='^')
60 ax.set_xlim(-5, 5)
61 ax.set_ylim(-0.25, 0.25)
62 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
63 ax.set_xticklabels([])
64 ax.set_yticklabels([])
65 ax = ax.twinx()
66 ax.set_ylim(-0.25, 0.25)
67 ax.set_yticks([-0.2, -0.1, 0, 0.1, 0.2])
68 ax.set_ylabel(r'$\Delta E_{\text{p}}$ (eV)', size=8)
69
70 ax = fig.add_axes([0.17, 0.13, 0.32, 0.36])
71 ax.plot(strain_Ru, eg_Ru, ls='none', c='b', marker='o')
72 ax.plot(strain_Rh, eg_Rh, ls='none', c='b', marker='s')
73 ax.plot(strain_Pd, eg_Pd, ls='none', c='b', marker='^')
74 ax.plot(strain_Os, eg_Os, ls='none', c='c', marker='o')
75 ax.plot(strain_Ir, eg_Ir, ls='none', c='c', marker='s')
76 ax.plot(strain_Pt, eg_Pt, ls='none', c='c', marker='^')
77 ax.set_xlim(-5, 5)
78 ax.set_xlabel(r'Strain %', size=8)
79 ax.set_ylim(-0.35, 0.35)
80 ax.set_ylabel(r'$\Delta E_{\text{e}}$ (eV)', size=8)
81
82 ax = fig.add_axes([0.51, 0.13, 0.32, 0.36])
83 ax.plot(strain_Ru, t2g_Ru, ls='none', c='b', marker='o')
84 ax.plot(strain_Rh, t2g_Rh, ls='none', c='b', marker='s')
85 ax.plot(strain_Pd, t2g_Pd, ls='none', c='b', marker='^')
86 ax.plot(strain_Os, t2g_Os, ls='none', c='c', marker='o')
87 ax.plot(strain_Ir, t2g_Ir, ls='none', c='c', marker='s')
88 ax.plot(strain_Pt, t2g_Pt, ls='none', c='c', marker='^')
89 ax.set_xlim(-5, 5)
90 ax.set_ylim(-0.15, 0.15)
91 ax.set_yticks([-0.1, 0, 0.1])
92 ax.set_yticklabels([])
93 ax.set_xlabel(r'Strain %', size=8)
94 ax = ax.twinx()
95 ax.set_ylim(-0.15, 0.15)
96 ax.set_yticks([-0.1, 0, 0.1])
97 ax.set_ylabel(r'$\Delta E_{\text{t}_{2\text{g}}}$ (eV)', size=8)
98
99 plt.savefig('supporting-figures/strain-vs-centers.png', dpi=300)
100 plt.show()

```

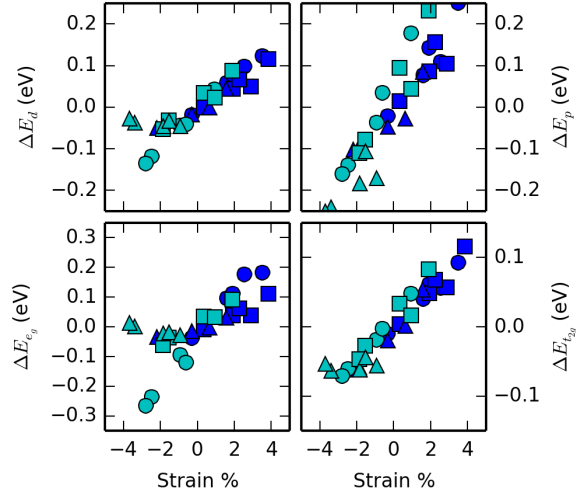


FIG. 6. Relationship between the relative strain in the direction parallel to the surface and the change in the (a) d -band center (ΔE_d), (b) p -band center (ΔE_p), (c) e_g -band center (ΔE_{e_g}), and (d) t_{2g} -band center ($\Delta E_{t_{2g}}$). Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

G. Relating strain to changes in fillings of t_{2g} -band

The code below relates strain to changes in the filling of the t_{2g} -band to assure strain did not cause significant charge transfer.

```

1  import matplotlib
2  import matplotlib.pyplot as plt
3  import json
4
5  matplotlib.rc('xtick', labels=8)
6  matplotlib.rc('ytick', labels=8)
7
8  with open('data.json', 'r') as f:
9      data = json.load(f)
10
11  t2g_Ru, strain_Ru = [], []
12  t2g_Rh, strain_Rh = [], []
13  t2g_Pd, strain_Pd = [], []
14  t2g_Os, strain_Os = [], []

```

```

15  t2g_Ir, strain_Ir = [], []
16  t2g_Pt, strain_Pt = [], []
17
18  for dopant in data:
19      d0 = data[dopant]['bare']['cell'][0][0]
20      for host in data[dopant]:
21          if host[-2:] != 'O2':
22              continue
23          eval('t2g_' + dopant[:2]).append(data[dopant]
24                                             [host]['strain']
25                                             ['DOS']['d_t2g_filling'])
26          d = data[dopant][host]['strain']['bare']['cell'][0][0]
27          eval('strain_' + dopant[:2]).append((d - d0)/d0 * 100)
28
29
30  fig = plt.figure(1, (2.3, 3), dpi=300)
31  ax = fig.add_subplot(111)
32  ax.axhline(0, ls='--', c='k')
33  ax.axvline(0, ls='--', c='k')
34  ax.plot(strain_Ru, t2g_Ru, ls='none', c='b', marker='o')
35  ax.plot(strain_Rh, t2g_Rh, ls='none', c='b', marker='s')
36  ax.plot(strain_Pd, t2g_Pd, ls='none', c='b', marker='^')
37  ax.plot(strain_Os, t2g_Os, ls='none', c='c', marker='o')
38  ax.plot(strain_Ir, t2g_Ir, ls='none', c='c', marker='s')
39  ax.plot(strain_Pt, t2g_Pt, ls='none', c='c', marker='^')
40  ax.set_xlim(-5, 5)
41  ax.set_xlabel(r'Strain %', size=8)
42  ax.set_ylabel(r'$\Delta E_{t_{2g}}$ (eV)', size=8)
43  plt.tight_layout()
44  plt.savefig('supporting-figures/strain-d-vs-t2g-filling.png', dpi=300)
45  plt.show()

```

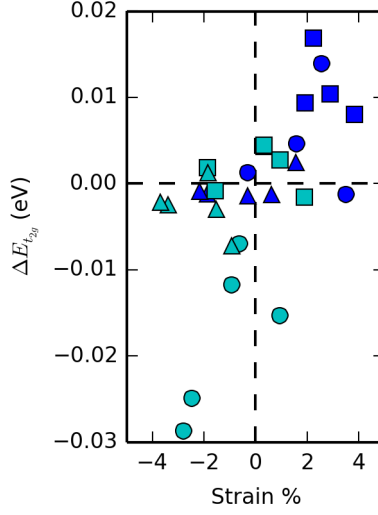


FIG. 7. Change in the t_{2g} -band filling ($\Delta\Delta f_{t_{2g}}$) caused by strain. Blue and cyan markers show systems where M^D is a row 5 and 6 metal, respectively. Circle, square, and triangle markers show systems where M^D is a column 8, 9, and 10 metal, respectively.

VII. REQUIRED MODULES

In addition to having a working license to the Vienna Ab-initio Simulation Package (VASP), one also requires at least Python 2.7 and several Python modules to reproduce all of our data. These modules are summarized below.

A. `numpy`, `scipy`, and `matplotlib`

`numpy`, `scipy`, and `matplotlib` are standard suites for performing scientific analysis using Python. Most linux distributions provide these packages along with Python, but they can also be easily downloaded via the Enthought package (www.enthought.com).

B. `ase`

`ase` is the Atomic Simulation Environment, which is wrapper for performing and analyzing quantum-chemical calculations using Python. It can be downloaded at <https://wiki.fysik.dtu.dk/ase/>.

C. `jasp`

`jasp` is a Python wrapper of the `ase.calculators.vasp` module written by Professor John Kitchin. This module facilitates performing calculations on a super computer with the submission of jobs, organization, and I/O control schemes. It allows for easy job organization and I/O of calculations. This can be found at <https://github.com/jkitchin/jasp>.

D. `pycse`

`pycse` is short of python computations in science and engineering, which is a module for performing a suite of mathematics and statistics functions for research and engineering. The primary function we have used in this work is the `regress` function, which is useful for performing linear regression on correlations. This code can be found at <http://github.com/jkitchin/pycse>

E. `ase_addons`

The python module `ase_addons` is a convenience module written by Zhongnan Xu and can be found at http://github.com/zhongnanxu/ase_addons. It contains bulk and surface structures along with a number of convenience functions. In this work, we primarily use it to load the rutile bulk unit cell and surface slab.

* jkitchin@andrew.cmu.edu