

Supporting information for:

Probing the Coverage Dependence of Site and

Adsorbate Configurational Correlations on (111)

Surfaces of Late Transition Metals

Zhongnan Xu and John R. Kitchin*

*Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave,
Pittsburgh, PA 15213*

E-mail: jkitchin@andrew.cmu.edu

*To whom correspondence should be addressed

Contents


1	Introduction	S3
2	Calculation of adsorption energies	S4
2.1	Gas-phase adsorbate calculations	S4
2.2	Slab calculations	S5
3	Storage of adsorption energies	S8
4	Storage of structures	S11
5	Analysis of adsorption energies	S13
5.1	Correlations between adsorption energies on different sites	S13
5.2	Correlations between adsorption energies on different sites for presentation .	S16
5.2.1	Iteration 1	S16
5.2.2	Iteration 2	S19
5.2.3	Iteration 3	S22
5.2.4	Iteration 4	S25
5.2.5	Iteration 5	S28
5.2.6	Iteration 6	S31
5.3	Correlations between adsorption energies of different adsorbates	S34
5.3.1	Correlations between F, Cl, and Br	S34
5.3.2	Correlations between O and S	S38
5.3.3	Correlations between C, N and O	S40
6	Discussion of preferential binding site	S44
7	Sample calculations including dispersion	S45
7.1	Dispersion calculations from relaxed structures	S46
7.2	Storage of dispersion calculations	S50

7.3	Site site correlations with dispersion	S53
7.4	Scaling relationships with dispersion	S56
8	Required modules	S60
8.1	numpy, scipy, and matplotlib	S60
8.2	ase	S60
8.3	jasp	S60
8.4	pycse	S61
9	Creating the TOC figure	S61

1 Introduction

This document details the complete calculation and analysis of our data, which resides in the following sections. Section Calculation of adsorption energies details the calculation of the adsorption energies. Section Storage of adsorption energies details the storage of all adsorption energies in a single energies.json file, which is attached to this manuscript. Section Analysis of adsorption energies contains code for reading the energies.json file and making Figures 1-4 of the manuscript.

In Section Discussion of preferential binding site, we also discuss trends in the preferential binding site we can obtain from Figure 1 in the manuscript. We believe this discussion is useful for determining what combination of surface, adsorbate, and coverage prefers which binding site. Section Required modules is a list of the required python modules needed to reproduce our results.

The source for this document can be found here: . The source document is an 'org' file, which is a plain text file constructed in the Emacs editing environment.

2 Calculation of adsorption energies

The code below details the calculation of the adsorption energies. The first section contains the calculation of the adsorbate in the gas phase as a single molecule. The second section is the calculation of the bare surface slabs along with slabs with adsorbates. The table below details the combinations of surface, adsorbate, site, and coverage we tested.

Table S1: The combinations of adsorbate, surface, coverage, and sites we tested in this work. This totaled to 896 adsorption energies.

Adsorbates	Surfaces	Coverages	Sites
H, C, N, O	Cu, Rh, Pd, Ag	0.25ML, 0.5ML	fcc, hcp
F, S, Cl, Br	Ir, Pt, Au	0.75ML, 1.0ML	bridge, ontop

We note that in many cases, a poor guess of the initial starting height can lead a lack of bonding between the surface and the adsorbate. Therefore, it is not likely that 100% of these calculations will converge to meaningful structure in one run.

2.1 Gas-phase adsorbate calculations

The code below simply calculates the adsorbates in a $8 \times 9 \times 10$ Å box. We use an asymmetric box to break symmetry and ensure a ground state electronic structure.

```

1  # orthrhombic box center
2  from ase import Atom, Atoms
3  from jasp import *
4
5  for adsorbate in ['H', 'C', 'N', 'O', 'S',
6                  'F', 'Cl', 'Br']:
7      atoms = Atoms([Atom(adsorbate, [4, 4.5, 5], magmom=2)],
8                    cell=(8,9,10))
9
10     wd = 'adsorbates/{adsorbate}-1'.format(**locals())
11     with jasp(wd,
12              xc='PBE',
13              prec='normal',
14              encut=400,
```

```

15         ismear=0,
16         sigma=0.01,
17         ispin=2,
18         atoms=atoms) as calc:
19     try:
20         calc.calculate()
21         print '{0}: {1}'.format(wd, atoms.get_potential_energy())
22     except (VaspSubmitted, VaspQueued):
23         print '{0}: queued'.format(wd)
24         pass

```

2.2 Slab calculations

In addition running the bare surface, the code belows performs every possible combination of adsorbate, surface, coverage, and site shown in Table S1. In total, this code runs a total of 903 DFT calculations.

```

1  from ase import *
2  from ase.constraints import FixAtoms,FixScaled
3  from ase.lattice.surface import *
4  from jasp import *
5  import sys
6
7  JASPRC['queue.mem'] = '4GB'
8
9  LC = {'Rh':3.84,
10        'Ir':3.88,
11        'Pd':3.95,
12        'Pt':3.98,
13        'Cu':3.64,
14        'Ag':4.16,
15        'Au':4.17}
16
17  CWD = os.getcwd()
18
19  for metal in ['Rh', 'Ir', 'Pd', 'Pt', 'Cu', 'Ag', 'Au']:
20      for adsorbate in ['H', 'C', 'N', 'O', 'S', 'F', 'Cl', 'Br']:
21          for site in ['fcc', 'hcp', 'bridge', 'ontop']:
22              for coverage in [0.0, 0.25, 0.5, 0.75, 1.0]:

```

```

23
24     atoms = fcc111(metal,
25                     size=(2,2,4),
26                     vacuum=12,
27                     a=LC[metal])
28
29     if site == 'ontop':
30         height = 2.0 #this is from the center of the atom
31     else:
32         height = 1.2
33
34     # add the adsorbates
35     if coverage == 0.25:
36         add_adsorbate(atoms,
37                       adsorbate,
38                       height=height,
39                       position=site)
40     elif coverage == 0.5:
41         add_adsorbate(atoms,
42                       adsorbate,
43                       height=height,
44                       position=site)
45         add_adsorbate(atoms,
46                       adsorbate,
47                       height=height,
48                       position=site,
49                       offset=(1,0))
50     elif coverage == 0.75:
51         add_adsorbate(atoms,
52                       adsorbate,
53                       height=height,
54                       position=site)
55         add_adsorbate(atoms,
56                       adsorbate,
57                       height=height,
58                       position=site,
59                       offset=(1,0))
60         add_adsorbate(atoms,
61                       adsorbate,
62                       height=height,
63                       position=site,

```

```

64             offset=(0,1))
65 elif coverage == 1.0:
66     add_adsorbate(atoms,
67                   adsorbate,
68                   height=height,
69                   position=site)
70     add_adsorbate(atoms,
71                   adsorbate,
72                   height=height,
73                   position=site,
74                   offset=(1,0))
75     add_adsorbate(atoms,
76                   adsorbate,
77                   height=height,
78                   position=site,
79                   offset=(0,1))
80     add_adsorbate(atoms,adsorbate,
81                   height=height,
82                   position=site,
83                   offset=(1,1))
84
85     # We apply constraints so that the top two layers along with
86     # the adsorbate are allowed to relax. Note that in systems
87     # where the site is the bridge, the x and y directions of
88     # the surface atoms must also be fixed to ensure no major
89     # surface reconstructions.
90
91     constraints = []
92
93     if site == 'bridge' and coverage != 0.0:
94         surf_constr = [True, True, False]
95     else:
96         surf_constr = [False, False, False]
97
98     for i,atom in enumerate(atoms):
99         if atom.symbol == adsorbate:
100             constraints.append(FixScaled(atoms.get_cell(), i,
101                                         [True, True, False]))
102         elif atom.tag < 3:
103             constraints.append(FixScaled(atoms.get_cell(), i,
104                                         surf_constr))

```

```

105         else:
106             constraints.append(FixScaled(atoms.get_cell(), i,
107                                         [True, True, True]))
108         atoms.set_constraint(constraints)
109
110     if coverage == 0.0:
111         # no need to run many of these. We only need one clean surface
112         wd = 'phase1/{metal}/{coverage}'.format(**locals())
113     else:
114         wd = 'phase1/{metal}/{adsorbate}/{site}/{coverage}'.format(**locals())
115     with jasp(wd, atoms=atoms,
116             encut=400,
117             prec='normal', ediff=1e-5,
118             xc='PBE',
119             kpts=(6,6,1),
120             ibrion=1, ediffg=-0.05,
121             nsw=100,
122             ismear=0, sigma=0.1,
123             lwave=False) as calc:
124         calc.create_metadata()
125         calc.set_nbands(f=2) # more bands than default for TMs
126         try:
127             calc.calculate()
128             print '{0}|{1}|'.format(wd, atoms.get_potential_energy())
129         except (VaspSubmitted, VaspQueued):
130             print '{0}|queued|'.format(wd)
131         except:
132             print '{0}|unexpected error|'.format(wd)
133         raise

```

3 Storage of adsorption energies

The code below reads the calculations and stores the data into a json file. Note that we had a few calculations that failed to converge to a stable structure. These are shown in the table below.

```

1 import os
2 from jasp import *

```

Table S2: These are the systems that could not reach a stable state after many attempts at relaxation with a variety of parameters

Surface	Adsorbate	Site	Coverage
Pd	N	bridge	0.75
Cu	C	ontop	0.75
Cu	N	bridge	0.75
Cu	F	ontop	0.75
Cu	O	ontop	0.75
Ag	N	bridge	0.75
Ag	N	ontop	0.75
Ag	C	bridge	0.5
Ag	N	brdige	0.5
Ag	O	ontop	0.75
Ag	F	ontop	0.75
Au	C	hcp	0.5
Au	C	bridge	0.75
Au	N	bridge	0.75
Au	O	hcp	0.5

```

3
4 CWD = os.getcwd()
5
6 ignore_data = (('Pd', 'N', 'bridge', 0.75 ),
7                ('Cu', 'C', 'ontop', 0.75 ),
8                ('Cu', 'N', 'bridge', 0.75 ),
9                ('Cu', 'F', 'ontop', 0.75 ),
10               ('Cu', 'O', 'ontop', 0.75 ),
11               ('Ag', 'N', 'bridge', 0.75 ),
12               ('Ag', 'N', 'ontop', 0.75 ),
13               ('Ag', 'C', 'bridge', 0.5 ),
14               ('Ag', 'N', 'brdige', 0.5 ),
15               ('Ag', 'O', 'ontop', 0.75 ),
16               ('Ag', 'F', 'ontop', 0.75 ),
17               ('Au', 'C', 'hcp', 0.5 ),
18               ('Au', 'C', 'bridge', 0.75 ),
19               ('Au', 'N', 'bridge', 0.75 ),
20               ('Au', 'O', 'hcp', 0.5 ))
21
22 data = {}
23
24 for metal in ['Rh', 'Ir', 'Pd', 'Pt', 'Cu', 'Ag', 'Au']:
25     # clean slab

```

```

26     wd = 'phase3/{metal}/0.0'.format(**locals())
27     with jasp(wd) as calc:
28         atoms = calc.get_atoms()
29         clean_slab_energy = atoms.get_potential_energy()
30
31     if metal not in data:
32         data[metal] = {}
33
34     for adsorbate in ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']:
35         # get adsorbate energy
36
37         if adsorbate not in data[metal]:
38             data[metal][adsorbate] = {}
39
40         wd = 'adsorbates/{adsorbate}-1'.format(**locals())
41         with jasp(wd) as calc:
42             atoms = calc.get_atoms()
43             ads_energy = atoms.get_potential_energy()
44
45         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
46
47             if site not in data[metal][adsorbate]:
48                 data[metal][adsorbate][site] = {}
49
50             for coverage in [0.25, 0.5, 0.75, 1.0]:
51
52                 if coverage not in data[metal][adsorbate][site]:
53                     data[metal][adsorbate][site][coverage] = {}
54
55                 if [metal, adsorbate, site, coverage] in ignore_data:
56                     data[metal][adsorbate][site][coverage] = 'Q'
57                     continue
58
59                 wd = 'phase3/{metal}/{adsorbate}/{site}/{coverage}'.format(**locals())
60                 with jasp(wd) as calc:
61                     energy = calc.read_energy()[1]
62                     Hads = energy
63                     Hads -= clean_slab_energy
64                     for atom in atoms:
65                         if atom.symbol == adsorbate:
66                             Hads -= ads_energy

```

```

67
68         Hads /= coverage*4 # normalize by number of adsorbates
69
70         data[metal][adsorbate][site][coverage] = Hads
71
72     f = open('energies.json', 'w')
73     import json
74     json.dump(data, f)
75     f.close()

```

The json file can be found here: .

4 Storage of structures

The code below reads the atomic structures of all finished calculations and stores them inside a single json file. These structures can be used as the starting point for future studies by us or any other researcher. The json file is organized that given a specific metal, adsorbate, site, and coverage, one can obtain the positions, cell, and symbols as lists to be easily used in constructing an atoms object. We will show how this can be easily used in a proceeding section in the supporting information looking at dispersion calculations.

structures.json: .

```

1  import os
2  import myproj
3  from jasp import *
4
5  CWD = os.getcwd()
6
7  ignore_data = (('Pd', 'N', 'bridge', 0.75 ),
8                ('Cu', 'C', 'ontop', 0.75 ),
9                ('Cu', 'N', 'bridge', 0.75 ),
10               ('Cu', 'F', 'ontop', 0.75 ),
11               ('Cu', 'O', 'ontop', 0.75 ),
12               ('Ag', 'N', 'bridge', 0.75 ),
13               ('Ag', 'N', 'ontop', 0.75 ),

```

```

14         ('Ag', 'C', 'bridge', 0.5 ),
15         ('Ag', 'N', 'bridge', 0.5 ),
16         ('Ag', 'O', 'ontop', 0.75 ),
17         ('Ag', 'F', 'ontop', 0.75 ),
18         ('Au', 'C', 'hcp', 0.5 ),
19         ('Au', 'C', 'bridge', 0.75 ),
20         ('Au', 'N', 'bridge', 0.75 ),
21         ('Au', 'O', 'hcp', 0.5 ))
22
23 data = {}
24
25 for metal in ['Rh', 'Ir', 'Pd', 'Pt', 'Cu', 'Ag', 'Au']:
26     # clean slab
27     wd = 'phase3/{metal}/0.0'.format(**locals())
28     with jasp(wd) as calc:
29         atoms = calc.get_atoms()
30         clean_symbols = atoms.get_chemical_symbols()
31         clean_pos = atoms.get_positions().tolist()
32         clean_cell = atoms.get_cell().tolist()
33
34     if metal not in data:
35         data[metal] = {}
36
37     for adsorbate in ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']:
38         # get adsorbate energy
39
40         if adsorbate not in data[metal]:
41             data[metal][adsorbate] = {}
42
43         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
44
45             if site not in data[metal][adsorbate]:
46                 data[metal][adsorbate][site] = {}
47
48             for coverage in [0.25, 0.5, 0.75, 1.0]:
49
50                 if coverage not in data[metal][adsorbate][site]:
51                     data[metal][adsorbate][site][coverage] = {}
52
53                 if [metal, adsorbate, site, coverage] in ignore_data:
54                     data[metal][adsorbate][site][coverage] = 'Q'

```

```

55         continue
56
57     wd = 'phase3/{metal}/{adsorbate}/{site}/{coverage}'.format(**locals())
58     with jasp(wd) as calc:
59         atoms = calc.get_atoms()
60         symbols = atoms.get_chemical_symbols()
61         pos = atoms.get_positions().tolist()
62         cell = atoms.get_cell().tolist()
63
64         data[metal][adsorbate][site][coverage]['symbols'] = symbols
65         data[metal][adsorbate][site][coverage]['pos'] = pos
66         data[metal][adsorbate][site][coverage]['cell'] = cell
67
68         data[metal][adsorbate][site][0.0] = {}
69         data[metal][adsorbate][site][0.0]['symbols'] = clean_symbols
70         data[metal][adsorbate][site][0.0]['pos'] = clean_pos
71         data[metal][adsorbate][site][0.0]['cell'] = clean_cell
72
73 myproj.cwd()
74 f = open('structures.json', 'w')
75 import json
76 json.dump(data, f)
77 f.close()

```

5 Analysis of adsorption energies

Below is code for the analysis of all adsorption energies to create all figures in the manuscript. The first section details the construction of Figure 1, while the second section details the construction of Figures 2-4. All code constructs the figures directly from the energies.json file.

5.1 Correlations between adsorption energies on different sites

This code compares the adsorption energy of a system with a specific metal, adsorbate, adsorption site, and coverage with another system with the exact same properties *except* the adsorption site. We make comparisons between the fcc site and the hcp, bridge, and ontop

sites. This code produces Figure 1 found in the manuscript.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import json
4 from matplotlib import rcParams, rc
5 # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6 # rc('text', usetex=True)
7 rcParams['mathtext.default'] = 'regular'
8
9 fig = plt.figure(1, (6.5, 4.5))
10
11 f = open('energies.json', 'r')
12 data = json.load(f)
13
14 '''
15 color for metal
16 letter symbol for adsorbate
17 '''
18 colors = {'Cu':'Orange',
19           'Ag':'Silver',
20           'Au':'Yellow',
21           'Pd':'Green',
22           'Pt':'Red',
23           'Rh':'Blue',
24           'Ir':'Purple'}
25
26 simple_ads = ['H', 'C', 'N', 'O']
27 all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
28
29 axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
30
31 for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
32     for adsorbate in all_ads:
33         E1, E2 = [], []
34         for coverage in '0.25', '0.5', '0.75', '1.0':
35             if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
36                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
37                 E1.append(data[metal][adsorbate]['hcp'][coverage])
38                 E2.append(data[metal][adsorbate]['fcc'][coverage])
39
```

```

40         axes.plot(E1, E2,
41                     marker='$_s$' % adsorbate,
42                     color=colors[metal])
43
44     axes.text(-7.5, -0.8, '(a)', size='large')
45     axes.plot([-8, 0], [-8, 0], 'k--')
46     axes.set_xlabel('$_\Delta H_{ads,hcp}$ (eV)')
47     axes.set_ylabel('$_\Delta H_{ads,fcc}$ (eV)')
48     axes.set_xlim(-8, 0)
49     axes.set_ylim(-8, 0)
50     axes.set_xticks([-7, -5, -3, -1])
51
52     axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
53
54     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
55         for adsorbate in all_ads:
56             E1, E2 = [], []
57             for coverage in '0.25', '0.5', '0.75', '1.0':
58                 if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
59                     isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
60                     E1.append(data[metal][adsorbate]['bridge'][coverage])
61                     E2.append(data[metal][adsorbate]['fcc'][coverage])
62
63             axes.plot(E1, E2,
64                       marker='$_s$' % adsorbate,
65                       color=colors[metal])
66
67     axes.text(-7.5, -0.8, '(b)', size='large')
68     axes.plot([-8, 0], [-8, 0], 'k--')
69     axes.set_xlabel('$_\Delta H_{ads,bridge}$ (eV)')
70     axes.set_xlim(-8, 0)
71     axes.set_ylim(-8, 0)
72     axes.set_yticklabels([])
73     axes.set_xticks([-7, -5, -3, -1])
74
75     axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
76
77     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
78         for adsorbate in all_ads:
79             E1, E2 = [], []
80             for coverage in '0.25', '0.5', '0.75', '1.0':

```

```

81         if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
82             isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
83             E1.append(data[metal][adsorbate]['ontop'][coverage])
84             E2.append(data[metal][adsorbate]['fcc'][coverage])
85
86     axes.plot(E1, E2,
87              marker='%s$' % adsorbate,
88              color=colors[metal])
89
90     axes.text(-7.5, -0.8, '(c)', size='large')
91     axes.plot([-8, 0], [-8, 0], 'k--')
92     axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
93     axes.set_xlim(-8, 0)
94     axes.set_ylim(-8, 0)
95     axes.set_yticklabels([])
96     axes.set_xticks([-7, -5, -3, -1])
97
98     axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
99                  arrowprops=dict(arrowstyle='simple', color='r',
100                                connectionstyle='arc3,rad=-0.5'))
101     axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
102
103     from matplotlib.patches import FancyArrow
104     l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
105                    transform=fig.transFigure, figure=fig)
106     fig.patches.extend([l1])
107     fig.text(0.555, 0.94, r'Increasing Geometric Similarity', size='large', ha='center',)
108     plt.savefig('figures/FIG1.png')
109     plt.savefig('figures/FIG1.pdf')
110
111     plt.show()

```

5.2 Correlations between adsorption energies on different sites for presentation

5.2.1 Iteration 1

```

1 import matplotlib.pyplot as plt
2 import numpy as np

```

```

3  import json
4  from matplotlib import rcParams, rc
5  # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6  # rc('text', usetex=True)
7  rcParams['mathtext.default'] = 'regular'
8
9  fig = plt.figure(1, (6.5, 4.5))
10
11  f = open('energies.json', 'r')
12  data = json.load(f)
13
14  '''
15  color for metal
16  letter symbol for adsorbate
17  '''
18  colors = {'Cu':'Orange',
19            'Ag':'Silver',
20            'Au':'Yellow',
21            'Pd':'Green',
22            'Pt':'Red',
23            'Rh':'Blue',
24            'Ir':'Purple'}
25
26  # all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
27  all_ads = ['S', 'C', 'Cl']
28
29  all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
30  all_metals = ['Pd']
31
32  axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
33
34  for metal in all_metals:
35      for adsorbate in all_ads:
36          E1, E2 = [], []
37          for coverage in '0.25', '0.5', '0.75', '1.0':
38              if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
39                  isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
40                  E1.append(data[metal][adsorbate]['hcp'][coverage])
41                  E2.append(data[metal][adsorbate]['fcc'][coverage])
42
43      axes.plot(E1, E2,

```

```

44         marker='%s$' % adsorbate,
45         color=colors[metal])
46
47 axes.text(-7.5, -0.8, '(a)', size='large')
48 axes.plot([-8, 0], [-8, 0], 'k--')
49 axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')
50 axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
51 axes.set_xlim(-8, 0)
52 axes.set_ylim(-8, 0)
53 axes.set_xticks([-7, -5, -3, -1])
54
55 axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
56
57 for metal in all_metals:
58     for adsorbate in all_ads:
59         E1, E2 = [], []
60         for coverage in '0.25', '0.5', '0.75', '1.0':
61             if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
62                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
63                 E1.append(data[metal][adsorbate]['bridge'][coverage])
64                 E2.append(data[metal][adsorbate]['fcc'][coverage])
65
66         axes.plot(E1, E2,
67                 marker='%s$' % adsorbate,
68                 color=colors[metal])
69
70 axes.text(-7.5, -0.8, '(b)', size='large')
71 axes.plot([-8, 0], [-8, 0], 'k--')
72 axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
73 axes.set_xlim(-8, 0)
74 axes.set_ylim(-8, 0)
75 axes.set_yticklabels([])
76 axes.set_xticks([-7, -5, -3, -1])
77
78 axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
79
80 for metal in all_metals:
81     for adsorbate in all_ads:
82         E1, E2 = [], []
83         for coverage in '0.25', '0.5', '0.75', '1.0':
84             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and

```

```

85         isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
86         E1.append(data[metal][adsorbate]['ontop'][coverage])
87         E2.append(data[metal][adsorbate]['fcc'][coverage])
88
89     axes.plot(E1, E2,
90               marker='%s$' % adsorbate,
91               color=colors[metal])
92
93     axes.text(-7.5, -0.8, '(c)', size='large')
94     axes.plot([-8, 0], [-8, 0], 'k--')
95     axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
96     axes.set_xlim(-8, 0)
97     axes.set_ylim(-8, 0)
98     axes.set_yticklabels([])
99     axes.set_xticks([-7, -5, -3, -1])
100
101     axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
102                  arrowprops=dict(arrowstyle='simple', color='r',
103                                  connectionstyle='arc3,rad=-0.5'))
104     axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
105
106     from matplotlib.patches import FancyArrow
107     l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
108                    transform=fig.transFigure, figure=fig)
109     fig.patches.extend([l1])
110     fig.text(0.555, 0.94, 'Increasing Geometric Similarity', size='large', ha='center',)
111     plt.savefig('talk-figures/site-site-fig1.png')
112
113     plt.show()

```

5.2.2 Iteration 2

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import json
4  from matplotlib import rcParams, rc
5  # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6  # rc('text', usetex=True)
7  rcParams['mathtext.default'] = 'regular'
8

```

```

9  fig = plt.figure(1, (6.5, 4.5))
10
11  f = open('energies.json', 'r')
12  data = json.load(f)
13
14  '''
15  color for metal
16  letter symbol for adsorbate
17  '''
18  colors = {'Cu':'Orange',
19            'Ag':'Silver',
20            'Au':'Yellow',
21            'Pd':'Green',
22            'Pt':'Red',
23            'Rh':'Blue',
24            'Ir':'Purple'}
25
26  # all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
27  all_ads = ['S', 'C', 'Cl']
28
29  all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
30  all_metals = ['Pd', 'Rh', 'Ir']
31
32  axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
33
34  for metal in all_metals:
35      for adsorbate in all_ads:
36          E1, E2 = [], []
37          for coverage in '0.25', '0.5', '0.75', '1.0':
38              if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
39                  isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
40                  E1.append(data[metal][adsorbate]['hcp'][coverage])
41                  E2.append(data[metal][adsorbate]['fcc'][coverage])
42
43          axes.plot(E1, E2,
44                   marker='%s$' % adsorbate,
45                   color=colors[metal])
46
47  axes.text(-7.5, -0.8, '(a)', size='large')
48  axes.plot([-8, 0], [-8, 0], 'k--')
49  axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')

```

```

50 axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
51 axes.set_xlim(-8, 0)
52 axes.set_ylim(-8, 0)
53 axes.set_xticks([-7, -5, -3, -1])
54
55 axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
56
57 for metal in all_metals:
58     for adsorbate in all_ads:
59         E1, E2 = [], []
60         for coverage in '0.25', '0.5', '0.75', '1.0':
61             if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
62                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
63                 E1.append(data[metal][adsorbate]['bridge'][coverage])
64                 E2.append(data[metal][adsorbate]['fcc'][coverage])
65
66         axes.plot(E1, E2,
67                   marker='$s$' % adsorbate,
68                   color=colors[metal])
69
70 axes.text(-7.5, -0.8, '(b)', size='large')
71 axes.plot([-8, 0], [-8, 0], 'k--')
72 axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
73 axes.set_xlim(-8, 0)
74 axes.set_ylim(-8, 0)
75 axes.set_yticklabels([])
76 axes.set_xticks([-7, -5, -3, -1])
77
78 axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
79
80 for metal in all_metals:
81     for adsorbate in all_ads:
82         E1, E2 = [], []
83         for coverage in '0.25', '0.5', '0.75', '1.0':
84             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
85                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
86                 E1.append(data[metal][adsorbate]['ontop'][coverage])
87                 E2.append(data[metal][adsorbate]['fcc'][coverage])
88
89         axes.plot(E1, E2,
90                   marker='$s$' % adsorbate,

```

```

91         color=colors[metal])
92
93 axes.text(-7.5, -0.8, '(c)', size='large')
94 axes.plot([-8, 0], [-8, 0], 'k--')
95 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
96 axes.set_xlim(-8, 0)
97 axes.set_ylim(-8, 0)
98 axes.set_yticklabels([])
99 axes.set_xticks([-7, -5, -3, -1])
100
101 axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
102               arrowprops=dict(arrowstyle='simple', color='r',
103                               connectionstyle='arc3,rad=-0.5'))
104 axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
105
106 from matplotlib.patches import FancyArrow
107 l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
108                transform=fig.transFigure, figure=fig)
109 fig.patches.extend([l1])
110 fig.text(0.555, 0.94, 'Increasing Geometric Similarity', size='large', ha='center',)
111 plt.savefig('talk-figures/site-site-fig2.png')
112
113 plt.show()

```

5.2.3 Iteration 3

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import json
4  from matplotlib import rcParams, rc
5  # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6  # rc('text', usetex=True)
7  rcParams['mathtext.default'] = 'regular'
8
9  fig = plt.figure(1, (6.5, 4.5))
10
11 f = open('energies.json', 'r')
12 data = json.load(f)
13
14 '''

```

```

15  color for metal
16  letter symbol for adsorbate
17  '''
18  colors = {'Cu':'Orange',
19            'Ag':'Silver',
20            'Au':'Yellow',
21            'Pd':'Green',
22            'Pt':'Red',
23            'Rh':'Blue',
24            'Ir':'Purple'}
25
26  # all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
27  all_ads = ['S', 'C', 'Cl', 'F', 'O', 'H']
28
29  all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
30  all_metals = ['Pd', 'Rh', 'Ir']
31
32  axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
33
34  for metal in all_metals:
35      for adsorbate in all_ads:
36          E1, E2 = [], []
37          for coverage in '0.25', '0.5', '0.75', '1.0':
38              if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
39                  isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
40                  E1.append(data[metal][adsorbate]['hcp'][coverage])
41                  E2.append(data[metal][adsorbate]['fcc'][coverage])
42
43          axes.plot(E1, E2,
44                   marker='s%s' % adsorbate,
45                   color=colors[metal])
46
47  axes.text(-7.5, -0.8, '(a)', size='large')
48  axes.plot([-8, 0], [-8, 0], 'k--')
49  axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')
50  axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
51  axes.set_xlim(-8, 0)
52  axes.set_ylim(-8, 0)
53  axes.set_xticks([-7, -5, -3, -1])
54
55  axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])

```

```

56
57 for metal in all_metals:
58     for adsorbate in all_ads:
59         E1, E2 = [], []
60         for coverage in '0.25', '0.5', '0.75', '1.0':
61             if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
62                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
63                 E1.append(data[metal][adsorbate]['bridge'][coverage])
64                 E2.append(data[metal][adsorbate]['fcc'][coverage])
65
66         axes.plot(E1, E2,
67                   marker='s' % adsorbate,
68                   color=colors[metal])
69
70 axes.text(-7.5, -0.8, '(b)', size='large')
71 axes.plot([-8, 0], [-8, 0], 'k--')
72 axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
73 axes.set_xlim(-8, 0)
74 axes.set_ylim(-8, 0)
75 axes.set_yticklabels([])
76 axes.set_xticks([-7, -5, -3, -1])
77
78 axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
79
80 for metal in all_metals:
81     for adsorbate in all_ads:
82         E1, E2 = [], []
83         for coverage in '0.25', '0.5', '0.75', '1.0':
84             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
85                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
86                 E1.append(data[metal][adsorbate]['ontop'][coverage])
87                 E2.append(data[metal][adsorbate]['fcc'][coverage])
88
89         axes.plot(E1, E2,
90                   marker='s' % adsorbate,
91                   color=colors[metal])
92
93 axes.text(-7.5, -0.8, '(c)', size='large')
94 axes.plot([-8, 0], [-8, 0], 'k--')
95 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
96 axes.set_xlim(-8, 0)

```

```

97 axes.set_ylim(-8, 0)
98 axes.set_yticklabels([])
99 axes.set_xticks([-7, -5, -3, -1])
100
101 axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
102               arrowprops=dict(arrowstyle='simple', color='r',
103                               connectionstyle='arc3,rad=-0.5'))
104 axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
105
106 from matplotlib.patches import FancyArrow
107 l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
108                transform=fig.transFigure, figure=fig)
109 fig.patches.extend([l1])
110 fig.text(0.555, 0.94, r'Increasing Geometric Similarity', size='large', ha='center',)
111 plt.savefig('talk-figures/site-site-fig3.png')
112
113 plt.show()

```

5.2.4 Iteration 4

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import json
4  from matplotlib import rcParams, rc
5  # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6  # rc('text', usetex=True)
7  rcParams['mathtext.default'] = 'regular'
8
9  fig = plt.figure(1, (6.5, 4.5))
10
11  f = open('energies.json', 'r')
12  data = json.load(f)
13
14  '''
15  color for metal
16  letter symbol for adsorbate
17  '''
18  colors = {'Cu':'Orange',
19           'Ag':'Silver',
20           'Au':'Yellow',

```

```

21         'Pd': 'Green',
22         'Pt': 'Red',
23         'Rh': 'Blue',
24         'Ir': 'Purple'}
25
26 # all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
27 all_ads = ['S', 'C', 'Cl', 'F', 'O', 'H']
28
29 all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
30 all_metals = ['Pd', 'Rh', 'Ir', 'Au', 'Ag']
31
32 axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
33
34 for metal in all_metals:
35     for adsorbate in all_ads:
36         E1, E2 = [], []
37         for coverage in '0.25', '0.5', '0.75', '1.0':
38             if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
39                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
40                 E1.append(data[metal][adsorbate]['hcp'][coverage])
41                 E2.append(data[metal][adsorbate]['fcc'][coverage])
42
43         axes.plot(E1, E2,
44                   marker='%s$' % adsorbate,
45                   color=colors[metal])
46
47 axes.text(-7.5, -0.8, '(a)', size='large')
48 axes.plot([-8, 0], [-8, 0], 'k--')
49 axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')
50 axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
51 axes.set_xlim(-8, 0)
52 axes.set_ylim(-8, 0)
53 axes.set_xticks([-7, -5, -3, -1])
54
55 axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
56
57 for metal in all_metals:
58     for adsorbate in all_ads:
59         E1, E2 = [], []
60         for coverage in '0.25', '0.5', '0.75', '1.0':
61             if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and

```

```

62         isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
63             E1.append(data[metal][adsorbate]['bridge'][coverage])
64             E2.append(data[metal][adsorbate]['fcc'][coverage])
65
66     axes.plot(E1, E2,
67               marker='s' % adsorbate,
68               color=colors[metal])
69
70     axes.text(-7.5, -0.8, '(b)', size='large')
71     axes.plot([-8, 0], [-8, 0], 'k--')
72     axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
73     axes.set_xlim(-8, 0)
74     axes.set_ylim(-8, 0)
75     axes.set_yticklabels([])
76     axes.set_xticks([-7, -5, -3, -1])
77
78     axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
79
80     for metal in all_metals:
81         for adsorbate in all_ads:
82             E1, E2 = [], []
83             for coverage in '0.25', '0.5', '0.75', '1.0':
84                 if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
85                     isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
86                     E1.append(data[metal][adsorbate]['ontop'][coverage])
87                     E2.append(data[metal][adsorbate]['fcc'][coverage])
88
89             axes.plot(E1, E2,
90                       marker='s' % adsorbate,
91                       color=colors[metal])
92
93     axes.text(-7.5, -0.8, '(c)', size='large')
94     axes.plot([-8, 0], [-8, 0], 'k--')
95     axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
96     axes.set_xlim(-8, 0)
97     axes.set_ylim(-8, 0)
98     axes.set_yticklabels([])
99     axes.set_xticks([-7, -5, -3, -1])
100
101     axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
102                  arrowprops=dict(arrowstyle='simple', color='r',

```

```

103             connectionstyle='arc3,rad=-0.5'))
104 axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
105
106 from matplotlib.patches import FancyArrow
107 l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
108               transform=fig.transFigure, figure=fig)
109 fig.patches.extend([l1])
110 fig.text(0.555, 0.94, r'Increasing Geometric Similarity', size='large', ha='center',)
111 plt.savefig('talk-figures/site-site-fig4.png')
112
113 plt.show()

```

5.2.5 Iteration 5

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import json
4 from matplotlib import rcParams, rc
5 # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6 # rc('text', usetex=True)
7 rcParams['mathtext.default'] = 'regular'
8
9 fig = plt.figure(1, (6.5, 4.5))
10
11 f = open('energies.json', 'r')
12 data = json.load(f)
13
14 '''
15 color for metal
16 letter symbol for adsorbate
17 '''
18 colors = {'Cu':'Orange',
19          'Ag':'Silver',
20          'Au':'Yellow',
21          'Pd':'Green',
22          'Pt':'Red',
23          'Rh':'Blue',
24          'Ir':'Purple'}
25
26 all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']

```

```

27
28 all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
29 all_metals = ['Pd', 'Rh', 'Ir', 'Au', 'Ag']
30
31 axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
32
33 for metal in all_metals:
34     for adsorbate in all_ads:
35         E1, E2 = [], []
36         for coverage in '0.25', '0.5', '0.75', '1.0':
37             if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
38                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
39                 E1.append(data[metal][adsorbate]['hcp'][coverage])
40                 E2.append(data[metal][adsorbate]['fcc'][coverage])
41
42         axes.plot(E1, E2,
43                   marker='s' % adsorbate,
44                   color=colors[metal])
45
46 axes.text(-7.5, -0.8, '(a)', size='large')
47 axes.plot([-8, 0], [-8, 0], 'k--')
48 axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')
49 axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
50 axes.set_xlim(-8, 0)
51 axes.set_ylim(-8, 0)
52 axes.set_xticks([-7, -5, -3, -1])
53
54 axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
55
56 for metal in all_metals:
57     for adsorbate in all_ads:
58         E1, E2 = [], []
59         for coverage in '0.25', '0.5', '0.75', '1.0':
60             if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
61                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
62                 E1.append(data[metal][adsorbate]['bridge'][coverage])
63                 E2.append(data[metal][adsorbate]['fcc'][coverage])
64
65         axes.plot(E1, E2,
66                   marker='s' % adsorbate,
67                   color=colors[metal])

```

```

68
69 axes.text(-7.5, -0.8, '(b)', size='large')
70 axes.plot([-8, 0], [-8, 0], 'k--')
71 axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
72 axes.set_xlim(-8, 0)
73 axes.set_ylim(-8, 0)
74 axes.set_yticklabels([])
75 axes.set_xticks([-7, -5, -3, -1])
76
77 axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
78
79 for metal in all_metals:
80     for adsorbate in all_ads:
81         E1, E2 = [], []
82         for coverage in '0.25', '0.5', '0.75', '1.0':
83             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
84                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
85                 E1.append(data[metal][adsorbate]['ontop'][coverage])
86                 E2.append(data[metal][adsorbate]['fcc'][coverage])
87
88         axes.plot(E1, E2,
89                   marker='$s$' % adsorbate,
90                   color=colors[metal])
91
92 axes.text(-7.5, -0.8, '(c)', size='large')
93 axes.plot([-8, 0], [-8, 0], 'k--')
94 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
95 axes.set_xlim(-8, 0)
96 axes.set_ylim(-8, 0)
97 axes.set_yticklabels([])
98 axes.set_xticks([-7, -5, -3, -1])
99
100 axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
101               arrowprops=dict(arrowstyle='simple', color='r',
102                               connectionstyle='arc3,rad=-0.5'))
103 axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
104
105 from matplotlib.patches import FancyArrow
106 l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
107                 transform=fig.transFigure, figure=fig)
108 fig.patches.extend([l1])

```

```

109 fig.text(0.555, 0.94, r'Increasing Geometric Similarity', size='large', ha='center',)
110 plt.savefig('talk-figures/site-site-fig5.png')
111
112 plt.show()

```

5.2.6 Iteration 6

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import json
4  from matplotlib import rcParams, rc
5  # rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
6  # rc('text', usetex=True)
7  rcParams['mathtext.default'] = 'regular'
8
9  fig = plt.figure(1, (6.5, 4.5))
10
11  f = open('energies.json', 'r')
12  data = json.load(f)
13
14  '''
15  color for metal
16  letter symbol for adsorbate
17  '''
18  colors = {'Cu':'Orange',
19           'Ag':'Silver',
20           'Au':'Yellow',
21           'Pd':'Green',
22           'Pt':'Red',
23           'Rh':'Blue',
24           'Ir':'Purple'}
25
26  all_ads = ['H', 'C', 'N', 'O', 'F', 'S', 'Cl', 'Br']
27
28  all_metals = ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']
29
30  axes = fig.add_axes([0.1, 0.15, 0.28, 0.73])
31
32  for metal in all_metals:
33      for adsorbate in all_ads:

```

```

34     E1, E2 = [], []
35     for coverage in '0.25', '0.5', '0.75', '1.0':
36         if (isinstance(data[metal][adsorbate]['hcp'][coverage], float) and
37             isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
38             E1.append(data[metal][adsorbate]['hcp'][coverage])
39             E2.append(data[metal][adsorbate]['fcc'][coverage])
40
41     axes.plot(E1, E2,
42              marker='s%s' % adsorbate,
43              color=colors[metal])
44
45     axes.text(-7.5, -0.8, '(a)', size='large')
46     axes.plot([-8, 0], [-8, 0], 'k--')
47     axes.set_xlabel('$\Delta H_{ads,hcp}$ (eV)')
48     axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
49     axes.set_xlim(-8, 0)
50     axes.set_ylim(-8, 0)
51     axes.set_xticks([-7, -5, -3, -1])
52
53     axes = fig.add_axes([0.4, 0.15, 0.28, 0.73])
54
55     for metal in all_metals:
56         for adsorbate in all_ads:
57             E1, E2 = [], []
58             for coverage in '0.25', '0.5', '0.75', '1.0':
59                 if (isinstance(data[metal][adsorbate]['bridge'][coverage], float) and
60                     isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
61                     E1.append(data[metal][adsorbate]['bridge'][coverage])
62                     E2.append(data[metal][adsorbate]['fcc'][coverage])
63
64             axes.plot(E1, E2,
65                      marker='s%s' % adsorbate,
66                      color=colors[metal])
67
68     axes.text(-7.5, -0.8, '(b)', size='large')
69     axes.plot([-8, 0], [-8, 0], 'k--')
70     axes.set_xlabel('$\Delta H_{ads,bridge}$ (eV)')
71     axes.set_xlim(-8, 0)
72     axes.set_ylim(-8, 0)
73     axes.set_yticklabels([])
74     axes.set_xticks([-7, -5, -3, -1])

```

```

75
76 axes = fig.add_axes([0.70, 0.15, 0.28, 0.73])
77
78 for metal in all_metals:
79     for adsorbate in all_ads:
80         E1, E2 = [], []
81         for coverage in '0.25', '0.5', '0.75', '1.0':
82             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
83                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
84                 E1.append(data[metal][adsorbate]['ontop'][coverage])
85                 E2.append(data[metal][adsorbate]['fcc'][coverage])
86
87         axes.plot(E1, E2,
88                 marker='%s$' % adsorbate,
89                 color=colors[metal])
90
91 axes.text(-7.5, -0.8, '(c)', size='large')
92 axes.plot([-8, 0], [-8, 0], 'k--')
93 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
94 axes.set_xlim(-8, 0)
95 axes.set_ylim(-8, 0)
96 axes.set_yticklabels([])
97 axes.set_xticks([-7, -5, -3, -1])
98
99 axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
100               arrowprops=dict(arrowstyle='simple', color='r',
101                               connectionstyle='arc3,rad=-0.5'))
102 axes.text(-7.4, -7.8, 'Increasing coverage', size=12, color='r')
103
104 from matplotlib.patches import FancyArrow
105 l1 = FancyArrow(0.92, 0.92, -0.73, 0, fc='k', width=0.0015, head_width=0.03,
106               transform=fig.transFigure, figure=fig)
107 fig.patches.extend([l1])
108 fig.text(0.555, 0.94, r'Increasing Geometric Similarity', size='large', ha='center',)
109 plt.savefig('talk-figures/site-site-fig6.png')
110
111 plt.show()

```

5.3 Correlations between adsorption energies of different adsorbates

This code compares the adsorption energy of a system with a specific metal, adsorbate, adsorption site, and coverage with another system with the exact same properties *except* the adsorbate. We make comparisons between same Group elements and same Period elements. The first and second section compares adsorption energies between Group 17 (halogens) and Group 16 (chalcogens), while the last section compares adsorption energies within Period 3. Since we found same period correlations with the halogens were weak, these were not included in our analysis.

The code below also uses the `pycse` module to use linear regression to compute the 95% confidence intervals of the correlations. This allows us to draw conclusions on whether correlations are coverage dependent. This module was written by Professor John Kitchen and can be downloaded at <https://github.com/jkitchen/pycse>.

5.3.1 Correlations between F, Cl, and Br

```
1 from pylab import *
2 import numpy as np
3 import json
4
5 from pycse import regress
6
7 f = open('energies.json', 'r')
8 data = json.load(f)
9
10 colors = {'0.25': 'b',
11          '0.5': 'g',
12          '0.75': 'm',
13          '1.0': 'c'}
14
15 markers = {'0.25': 'o',
16           '0.5': 's',
17           '0.75': '^',
```

```

18         '1.0': 'd'})
19
20 fig = plt.figure(1, (4, 6))
21
22 inset = fig.add_axes([0.51, 0.1, 0.33, 0.39])
23 inset.text(0.25, 1.222, '(d)')
24 inset.set_xlim(0.2, 1.05)
25 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
26 inset.set_xticklabels([])
27 inset.set_yticklabels([])
28 inset.axhline(1.0, ls='--', c='k')
29
30 pair = ('F', 'Cl')
31 axes = fig.add_axes([0.16, 0.1, 0.33, 0.39])
32 ms, mhighs, m lows = [], [], []
33 for coverage in ('0.25', '0.5', '0.75', '1.0'):
34     E1, E2 = [], []
35     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
36         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
37             if (isinstance(data[metal][pair[0]][site][coverage], float) and
38                 isinstance(data[metal][pair[1]][site][coverage], float)):
39                 E1.append(data[metal][pair[0]][site][coverage])
40                 E2.append(data[metal][pair[1]][site][coverage])
41     Efits = np.linspace(min(E1), max(E1))
42     func = np.poly1d(np.polyfit(E1, E2, 1))
43     E1 = np.array(E1)
44     E2 = np.array(E2)
45     A = np.column_stack([E1**0, E1])
46     alpha = 0.05
47     p, pint, se = regress(A, E2, alpha)
48     m = p[1]
49     mint = pint[1]
50     ms.append(m)
51     mhighs.append(abs(m - mint[1]))
52     m lows.append(abs(m - mint[0]))
53     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='o')
54     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
55 inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,
56               marker='o', c='r')
57 axes.text(-4.35, -0.85, '(c)')
58 axes.set_xlim(-4.5, -2)

```

```

59 axes.set_ylim(-3.5, -0.5)
60 axes.set_xticks([-4, -3])
61 axes.set_yticks([-1, -2, -3])
62 axes.set_ylabel(r'$\Delta \mathit{E}_{\text{ads}}^{\text{Cl}}$ (eV)')
63
64 pair = ('F', 'Br')
65 axes = fig.add_axes([0.16, 0.51, 0.33, 0.39])
66 ms, mhighs, m lows = [], [], []
67 for coverage in ('0.25', '0.5', '0.75', '1.0'):
68     E1, E2 = [], []
69     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
70         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
71             if (isinstance(data[metal][pair[0]][site][coverage], float) and
72                 isinstance(data[metal][pair[1]][site][coverage], float)):
73                 E1.append(data[metal][pair[0]][site][coverage])
74                 E2.append(data[metal][pair[1]][site][coverage])
75     Efits = np.linspace(min(E1), max(E1))
76     func = np.poly1d(np.polyfit(E1, E2, 1))
77     E1 = np.array(E1)
78     E2 = np.array(E2)
79     A = np.column_stack([E1**0, E1])
80     alpha = 0.05
81     p, pint, se = regress(A, E2, alpha)
82     m = p[1]
83     mint = pint[1]
84     ms.append(m)
85     mhighs.append(abs(m - mint[1]))
86     m lows.append(abs(m - mint[0]))
87     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='s')
88     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
89 inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,
90               marker='s', c='y')
91 axes.text(-4.35, -0.4, '(a)')
92 axes.set_xlim(-4.5, -2)
93 axes.set_xticks([-4, -3])
94 axes.set_xticklabels([])
95 ylim = axes.get_ylim()
96 axes.set_yticks([-1, -2, -3])
97 axes.set_ylabel(r'$\Delta \mathit{E}_{\text{ads}}^{\text{Br}}$ (eV)')
98 axes = axes.twin()
99 axes.set_xlim(-4.5, -2)

```

```

100 axes.set_ylim(ylim)
101 axes.set_yticks([-1, -2, -3])
102 axes.set_xticks([-4, -3])
103 axes.set_xlabel(r'$\Delta \mathit{E}_{\text{ads}}^{\text{F}}$ (eV)')
104
105 pair = ('Cl', 'Br')
106 axes = fig.add_axes([0.51, 0.51, 0.33, 0.39])
107 ms, mhighs, m lows = [], [], []
108 for coverage in ('0.25', '0.5', '0.75', '1.0'):
109     E1, E2 = [], []
110     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
111         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
112             if (isinstance(data[metal][pair[0]][site][coverage], float) and
113                 isinstance(data[metal][pair[1]][site][coverage], float)):
114                 E1.append(data[metal][pair[0]][site][coverage])
115                 E2.append(data[metal][pair[1]][site][coverage])
116     Efits = np.linspace(min(E1), max(E1))
117     func = np.poly1d(np.polyfit(E1, E2, 1))
118     E1 = np.array(E1)
119     E2 = np.array(E2)
120     A = np.column_stack([E1**0, E1])
121     alpha = 0.05
122     p, pint, se = regress(A, E2, alpha)
123     m = p[1]
124     mint = pint[1]
125     ms.append(m)
126     mhighs.append(abs(m - mint[1]))
127     m lows.append(abs(m - mint[0]))
128     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='^')
129     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
130 inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,
131                marker='^', c='orange')
132 axes.text(-3.3, -0.4, '(b)')
133 axes.set_xlim(-3.5, -0.5)
134 axes.set_xticks([-3, -2, -1])
135 axes.set_yticks([-1, -2, -3])
136 axes.set_xticklabels([])
137 axes.set_yticklabels([])
138
139 axesy = axes.twinx()
140 axesy.set_xlim(-3.5, -0.5)

```

```

141 axesy.set_xticks([-3, -2, -1])
142 axesy.set_yticks([-1, -2, -3])
143 axes.set_yticklabels([])
144 axesy.set_xlabel(r'$\Delta \mathit{E}_{\mathrm{ads}}^{\mathrm{C1}}$ (eV)')
145
146 inset.set_xlim(0.2, 1.05)
147 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
148 inset.set_xlabel('Coverage')
149 insetx = inset.twinx()
150 insetx.set_xlim(0.2, 1.05)
151 insetx.set_xticks([0.25, 0.5, 0.75, 1.0])
152 insetx.set_xticklabels([r'$\mathit{frac{1}{4}}$',
153                        r'$\mathit{frac{1}{2}}$',
154                        r'$\mathit{frac{3}{4}}$',
155                        '1'])
156 insetx.set_ylabel('Slope')
157 insetx.set_yticks([1, 0.5, 0, -0.5, -1])
158 insetx.set_ylim(inset.get_ylim())
159
160 plt.savefig('figures/FIG2.png')
161 plt.savefig('figures/FIG2.pdf')
162 plt.show()

```

5.3.2 Correlations between O and S

```

1  from pylab import *
2  import numpy as np
3  import json
4
5  from pycse import regress
6
7  f = open('energies.json', 'r')
8  data = json.load(f)
9
10 colors = {'0.25': 'b',
11          '0.5': 'g',
12          '0.75': 'm',
13          '1.0': 'c'}
14
15 markers = {'0.25': 'o',

```

```

16         '0.5': 's',
17         '0.75': '^',
18         '1.0': 'd'}
19
20 pairs = (('C', 'N'),
21         ('N', 'O'),
22         ('C', 'O'),
23         ('C', 'F'))
24
25 fig = plt.figure(1, (4, 3))
26
27 inset = fig.add_axes([0.51, 0.2, 0.32, 0.78])
28 inset.set_xlim(0.2, 1.05)
29 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
30 inset.set_xticklabels([])
31 inset.set_yticklabels([])
32 inset.axhline(1.0, ls='--', c='k')
33
34 pair = ('O', 'S')
35 axes = fig.add_axes([0.17, 0.2, 0.32, 0.78])
36 ms, mhighs, m lows = [], [], []
37 for coverage in ('0.25', '0.5', '0.75', '1.0'):
38     E1, E2 = [], []
39     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
40         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
41             if (isinstance(data[metal][pair[0]][site][coverage], float) and
42                 isinstance(data[metal][pair[1]][site][coverage], float)):
43                 E1.append(data[metal][pair[0]][site][coverage])
44                 E2.append(data[metal][pair[1]][site][coverage])
45     Efits = np.linspace(min(E1), max(E1))
46     func = np.poly1d(np.polyfit(E1, E2, 1))
47     E1 = np.array(E1)
48     E2 = np.array(E2)
49     A = np.column_stack([E1**0, E1])
50     alpha = 0.05
51     p, pint, se = regress(A, E2, alpha)
52     m = p[1]
53     mint = pint[1]
54     ms.append(m)
55     mhighs.append(abs(m - mint[1]))
56     m lows.append(abs(m - mint[0]))

```

```

57     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='o')
58     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
59     inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,
60                  marker='o', c='r')
61     axes.text(-6.4, -2.05, '(a)')
62     axes.set_xlim(-7, -1)
63     axes.set_xticks([-6, -4, -2])
64     axes.set_ylim(-6, -1.5)
65     axes.set_yticks([-2, -3, -4, -5])
66
67     axes.set_xlabel(r'$\Delta \mathdefault{E_{\text{ads}}^0}$ (eV)')
68     axes.set_ylabel(r'$\Delta \mathdefault{E_{\text{ads}}^S}$ (eV)')
69
70     inset.set_xlim(0.2, 1.05)
71     inset.set_xticks([0.25, 0.5, 0.75, 1.0])
72     inset.text(0.85, 1.07, '(b)')
73     inset.set_xlabel('Coverage')
74     insetx = inset.twinx()
75     insetx.set_xticklabels([r'$\mathdefault{\frac{1}{4}}$',
76                            r'$\mathdefault{\frac{1}{2}}$',
77                            r'$\mathdefault{\frac{3}{4}}$',
78                            '1'])
79     insetx.set_xlim(0.2, 1.05)
80     insetx.set_xticks([0.25, 0.5, 0.75, 1.0])
81     insetx.set_ylabel('Slope')
82     insetx.set_yticks([0.4, 0.6, 0.8, 1.0])
83     insetx.set_ylim(inset.get_ylim())
84
85     plt.savefig('figures/FIG3.png')
86     plt.savefig('figures/FIG3.pdf')
87     plt.show()

```

5.3.3 Correlations between C, N and O

```

1  from pylab import *
2  import numpy as np
3  import json
4
5  from pycse import regress
6

```

```

7  f = open('energies.json', 'r')
8  data = json.load(f)
9
10 colors = {'0.25': 'b',
11           '0.5': 'g',
12           '0.75': 'm',
13           '1.0': 'c'}
14
15 markers = {'0.25': 'o',
16           '0.5': 's',
17           '0.75': '^',
18           '1.0': 'd'}
19
20 fig = plt.figure(1, (4, 6))
21
22 inset = fig.add_axes([0.51, 0.1, 0.33, 0.39])
23 inset.text(0.26, 1.11, '(d)')
24 inset.set_xlim(0.2, 1.05)
25 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
26 inset.set_xticklabels([])
27 inset.set_yticklabels([])
28
29 pair = ('C', 'N')
30 axes = fig.add_axes([0.16, 0.1, 0.33, 0.39])
31 ms, mhighs, m lows = [], [], []
32 for coverage in ('0.25', '0.5', '0.75', '1.0'):
33     E1, E2 = [], []
34     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
35         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
36             if (isinstance(data[metal][pair[0]][site][coverage], float) and
37                 isinstance(data[metal][pair[1]][site][coverage], float)):
38                 E1.append(data[metal][pair[0]][site][coverage])
39                 E2.append(data[metal][pair[1]][site][coverage])
40     Efits = np.linspace(min(E1), max(E1))
41     func = np.poly1d(np.polyfit(E1, E2, 1))
42     E1 = np.array(E1)
43     E2 = np.array(E2)
44     A = np.column_stack([E1**0, E1])
45     alpha = 0.05
46     p, pint, se = regress(A, E2, alpha)
47     m = p[1]

```

```

48     mint = pint[1]
49     ms.append(m)
50     mhighs.append(abs(m - mint[1]))
51     mlows.append(abs(m - mint[0]))
52     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='o')
53     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
54     inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, mlows), elinewidth=1.5,
55                   marker='o', c='r')
56     inset.axhline(0.75, c='r', ls='--')
57     axes.text(-7.5, -0.7, '(c)')
58     axes.set_xlim(-8, -0)
59     axes.set_xticks([-2, -4, -6])
60     axes.set_yticks([-1, -2, -3, -4, -5, -6])
61     axes.set_xticklabels([])
62     axes.set_ylabel(r'$\Delta \mathit{E}_{\text{ads}}^{\text{N}}$ (eV)')
63
64     pair = ('C', 'O')
65     axes = fig.add_axes([0.16, 0.51, 0.33, 0.39])
66     ms, mhighs, mlows = [], [], []
67     for coverage in ('0.25', '0.5', '0.75', '1.0'):
68         E1, E2 = [], []
69         for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
70             for site in ['fcc', 'hcp', 'bridge', 'ontop']:
71                 if (isinstance(data[metal][pair[0]][site][coverage], float) and
72                     isinstance(data[metal][pair[1]][site][coverage], float)):
73                     E1.append(data[metal][pair[0]][site][coverage])
74                     E2.append(data[metal][pair[1]][site][coverage])
75     Efits = np.linspace(min(E1), max(E1))
76     func = np.poly1d(np.polyfit(E1, E2, 1))
77     E1 = np.array(E1)
78     E2 = np.array(E2)
79     A = np.column_stack([E1**0, E1])
80     alpha = 0.05
81     p, pint, se = regress(A, E2, alpha)
82     m = p[1]
83     mint = pint[1]
84     ms.append(m)
85     mhighs.append(abs(m - mint[1]))
86     mlows.append(abs(m - mint[0]))
87     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='s')
88     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')

```

```

89 inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,
90               marker='s', c='y')
91 inset.axhline(0.5, c='y', ls='--')
92 axes.text(-7.5, -1.55, '(a)')
93 axes.set_xlim(-8, -0)
94 axes.set_ylim(-6, -1)
95 axes.set_xticks([-2, -4, -6])
96 axes.set_yticks([-2, -3, -4, -5])
97 axes.set_xticklabels([])
98 axes.set_ylabel(r'$\Delta \mathit{\text{default}}\{E_{\text{ads}}^{\text{0}}\}$ (eV)')
99 axes = axes.twinx()
100 axes.set_xlim(-8, -0)
101 axes.set_xticks([-2, -4, -6])
102 axes.set_xlabel(r'$\Delta \mathit{\text{default}}\{E_{\text{ads}}^{\text{C}}\}$ (eV)')
103
104 pair = ('N', 'O')
105 axes = fig.add_axes([0.51, 0.51, 0.33, 0.39])
106 ms, mhighs, m lows = [], [], []
107 for coverage in ('0.25', '0.5', '0.75', '1.0'):
108     E1, E2 = [], []
109     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
110         for site in ['fcc', 'hcp', 'bridge', 'ontop']:
111             if (isinstance(data[metal][pair[0]][site][coverage], float) and
112                 isinstance(data[metal][pair[1]][site][coverage], float)):
113                 E1.append(data[metal][pair[0]][site][coverage])
114                 E2.append(data[metal][pair[1]][site][coverage])
115     Efits = np.linspace(min(E1), max(E1))
116     func = np.poly1d(np.polyfit(E1, E2, 1))
117     E1 = np.array(E1)
118     E2 = np.array(E2)
119     A = np.column_stack([E1**0, E1])
120     alpha = 0.05
121     p, pint, se = regress(A, E2, alpha)
122     m = p[1]
123     mint = pint[1]
124     ms.append(m)
125     mhighs.append(abs(m - mint[1]))
126     m lows.append(abs(m - mint[0]))
127     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='^')
128     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
129 inset.errorbar([0.25, 0.5, 0.75, 1.0], ms, yerr=(mhighs, m lows), elinewidth=1.5,

```

```

130         marker='^', c='orange')
131 inset.axhline(0.66, c='orange', ls='--')
132 axes.text(-7.5, -1.55, '(b)')
133 axes.set_xlim(-8, -0)
134 axes.set_ylim(-6, -1)
135 axes.set_xticks([-2, -4, -6])
136 axes.set_yticks([-2, -3, -4, -5])
137 axes.set_yticklabels([])
138 axes.set_xticklabels([])
139
140 axes = axes.twinx()
141 axes.set_xlim(-8, -0)
142 axes.set_xticks([-2, -4, -6])
143 axes.set_xlabel(r'$\Delta \mathit{\text{default}}{E_{\text{ads}}^{\text{N}}}$ (eV)')
144 axes.set_yticklabels([])
145
146 inset.set_xlim(0.2, 1.05)
147 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
148 inset.set_xlabel('Coverage')
149 insetx = inset.twinx()
150 insetx.set_xlim(0.2, 1.05)
151 insetx.set_xticks([0.25, 0.5, 0.75, 1.0])
152 insetx.set_ylabel('Slope')
153 insetx.set_xticklabels([r'$\mathit{\text{default}}{\frac{1}{4}}$',
154                        r'$\mathit{\text{default}}{\frac{1}{2}}$',
155                        r'$\mathit{\text{default}}{\frac{3}{4}}$',
156                        '1'])
157 insetx.set_ylim(inset.get_ylim())
158
159 plt.savefig('figures/FIG4.png')
160 plt.savefig('figures/FIG4.pdf')
161 plt.show()

```

6 Discussion of preferential binding site

We can also extract information about the general trends of preferential binding sites from our data. Figure 1 (b) and (c) from the manuscript clearly show that while a majority of adsorbates prefer the fcc site over either the ontop or bridge sites, there are some clear excep-

tions. Group 17 elements (halogens), in addition to sulfur adsorption on Pt and Ir at high coverages, widely prefer the top site over the fcc site over a variety of coverages and materials. This is consistent with previous studies looking at adsorption of various electronegative species missing a single electron^{S1,S2} on late transition metals. A wide majority of elements prefer adsorption on the fcc site over the bridge site, except for some weak binding elements which have similar adsorption on both.

From Figure 1 (a) from the manuscript, there is no clear indication of which combinations of surface, adsorbate, and coverage produce a more favorable coverage on the hcp site over the fcc site. In Figure S1, we create a histogram counting the frequency of systems that prefer the hcp site. We see clear trends that Group 10 surfaces (Pd and Pt) prefer the fcc site, while Group 9 and 11 surfaces tend to prefer the hcp site. Furthermore, we see that adsorbates that are more electron deficient (C and N) and larger adsorbates (S, Cl, and Br) prefer the hcp site. Finally, we see that preference over the hcp site is largely coverage independent.

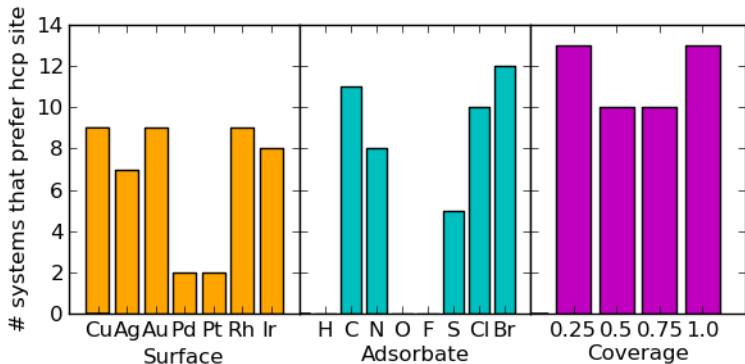


Figure S1: Histogram detailing system preference of hcp versus the fcc site.

7 Sample calculations including dispersion

One strength of the approach used to store the results in this work is that it is not only reproducible but easy to use the data (adsorption energies and structures) we have already calculated for new calculations. In this section, we perform sample calculations that include

dispersion, starting from the geometries calculated by the PBE functional. We use the DFT+D2 method of Grimme, implemented in VASP.^{S3} We start these calculations from the relaxed structures of our finished calculations. We choose to study dispersion because of its importance in accurately describing direct interactions between adsorbates, which is more important at high coverages. We to choose to study the O and S adsorbates adsorbed at the fcc and ontop site.

We choose to study the fcc and ontop site because we expect the fcc site to have minimal dispersion interactions due to screening from the electron density in the metal that spills out into the vacuum, while the ontop site should have maximum dispersion interactions. These differences between the sites should highlight the effect of dispersion on the correlations in adsorption energies between these two sites.

We choose to study the O and S adsorption also because of differences in their expected dispersion interactions. S is notably larger than O and could give rise to more dispersion interactions. There are large amount of metal sulfide materials that are layered and held together by dispersion forces, further emphasizing the importance of dispersion in S. O and S showed strong correlations at all coverages and a clear coverage dependence on their scaling relationship without dispersion. Hence, differences in expected dispersion interactions between O and S should highlight the effect of dispersion on scaling relationships.

7.1 Dispersion calculations from relaxed structures

The code below takes small set of metals, adsorbates, sites, and coverages and calculates their adsorption energies with the inclusion of adsorption. The starting structure of the calculation is taken from the relaxed structure after a finished calculation without dispersion. For our trial, we performed calculations of the S and O adsorbates on the Rh, Pd, Cu and Ag surface on the fcc and ontop sites at all coverages.

```
1 import os
2 import sys
```

```

3  import json
4  with open('structures.json', 'r') as f:
5      structure_data = json.load(f)
6
7  from ase import *
8  from ase.constraints import FixAtoms, FixScaled
9  from ase.lattice.surface import *
10 from ase.visualize import view
11 from jasp import *
12
13 JASPRC['queue.ppn'] = 4
14 JASPRC['queue.mem'] = '8GB'
15
16
17 CWD = os.getcwd()
18
19 LC = {'Rh':3.84,
20       'Ir':3.88,
21       'Pd':3.95,
22       'Pt':3.98,
23       'Cu':3.64,
24       'Ag':4.16,
25       'Au':4.17}
26
27 ads_indexes = {0.25: [16],
28                0.5: [16, 17],
29                0.75: [16, 17, 18],
30                1.0: [16, 17, 18, 19]}
31
32
33
34 for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
35     for adsorbate in ['O', 'S']:
36         for site in ['fcc', 'ontop']:
37             for coverage in [0.0, 0.25, 0.5, 0.75, 1.0]:
38
39                 atoms = fcc111(metal,
40                                size=(2,2,4),
41                                vacuum=12,
42                                a=LC[metal])
43

```

```

44     if site == 'ontop':
45         height = 2.0 #this is from the center of the atom
46     else:
47         height = 1.2
48
49     # add the adsorbates
50     if coverage == 0.25:
51         add_adsorbate(atoms,
52                       adsorbate,
53                       height=height,
54                       position=site)
55     elif coverage == 0.5:
56         add_adsorbate(atoms,
57                       adsorbate,
58                       height=height,
59                       position=site)
60         add_adsorbate(atoms,
61                       adsorbate,
62                       height=height,
63                       position=site,
64                       offset=(1,0))
65     elif coverage == 0.75:
66         add_adsorbate(atoms,
67                       adsorbate,
68                       height=height,
69                       position=site)
70         add_adsorbate(atoms,
71                       adsorbate,
72                       height=height,
73                       position=site,
74                       offset=(1,0))
75         add_adsorbate(atoms,
76                       adsorbate,
77                       height=height,
78                       position=site,
79                       offset=(0,1))
80     elif coverage == 1.0:
81         add_adsorbate(atoms,
82                       adsorbate,
83                       height=height,
84                       position=site)

```

```

85         add_adsorbate(atoms,
86                         adsorbate,
87                         height=height,
88                         position=site,
89                         offset=(1,0))
90         add_adsorbate(atoms,
91                         adsorbate,
92                         height=height,
93                         position=site,
94                         offset=(0,1))
95         add_adsorbate(atoms,adsorbate,
96                         height=height,
97                         position=site,
98                         offset=(1,1))
99
100     try:
101         pos = structure_data[metal][adsorbate][site][str(coverage)]['pos']
102     except (TypeError):
103         print metal, adsorbate, site, coverage, 'not converged'
104         continue
105
106     atoms.set_positions(pos)
107
108     # freeze only the top two layers of metal atoms
109     constraints = []
110
111     if site == 'bridge' and coverage != 0.0:
112         surf_constr = [True, True, False]
113     else:
114         surf_constr = [False, False, False]
115
116     for i,atom in enumerate(atoms):
117         if atom.symbol == adsorbate:
118             constraints.append(FixScaled(atoms.get_cell(), i,
119                                         [True, True, False]))
120         elif atom.tag < 3:
121             constraints.append(FixScaled(atoms.get_cell(), i,
122                                         surf_constr))
123         else:
124             constraints.append(FixScaled(atoms.get_cell(), i,
125                                         [True, True, True]))

```

```

126         atoms.set_constraint(constraints)
127
128         if coverage == 0.0:
129             # no need to run many of these. We only need one clean surface
130             wd = 'dispersion/{metal}/{coverage}'.format(**locals())
131         else:
132             wd = 'dispersion/{metal}/{adsorbate}/{site}/{coverage}'.format(**locals())
133
134         cwd = os.getcwd()
135         try:
136             with jasp(wd, atoms=atoms,
137                     encut=400,
138                     prec='normal', ediff=1e-5,
139                     potim=0.15,
140                     xc='PBE',
141                     kpts=(6,6,1),
142                     ibrion=1, ediffg=-0.05,
143                     nsw=100,
144                     ismear=0, sigma=0.1,
145                     lwave=False,
146                     lvdw=True) as calc:
147                 try:
148                     calc.calculate()
149                     print '|{0}|{1}|'.format(wd, atoms.get_potential_energy())
150                 except (VaspSubmitted, VaspQueued):
151                     print '|{0}|queued|'.format(wd)
152                 except (VaspNotConverged):
153                     print wd, 'not converged'
154             except VaspNotFinished:
155                 print wd, 'not finished'
156         os.chdir(cwd)

```

7.2 Storage of dispersion calculations

The code below reads the calculations we performed with dispersion and stores the data into a json file. Note that we had a few calculations that failed to converge to a stable structure. These are shown in the table below.

Table S3: These are the systems that could not reach a stable state after turning on dispersion.

Surface	Adsorbate	Site	Coverage
Cu	O	ontop	0.75
Ag	O	ontop	0.5
Ag	O	ontop	0.75
Ag	S	ontop	1.0

```

1  import os
2  from jasp import *
3
4  ignore_calcs = [['Cu', 'O', 'ontop', 0.75],
5                  ['Ag', 'O', 'ontop', 0.5],
6                  ['Ag', 'O', 'ontop', 0.75],
7                  ['Ag', 'S', 'ontop', 1.0]]
8
9  CWD = os.getcwd()
10
11  data = {}
12
13  for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
14      # clean slab
15      wd = 'dispersion/{metal}/0.0'.format(**locals())
16      with jasp(wd) as calc:
17          atoms = calc.get_atoms()
18          clean_slab_energy = atoms.get_potential_energy()
19
20      if metal not in data:
21          data[metal] = {}
22
23      for adsorbate in ['O', 'S']:
24          # get adsorbate energy
25
26          if adsorbate not in data[metal]:
27              data[metal][adsorbate] = {}
28
29          wd = 'adsorbates/{adsorbate}-d'.format(**locals())
30          with jasp(wd) as calc:
31              atoms = calc.get_atoms()
32              ads_energy = atoms.get_potential_energy()
33

```

```

34     for site in ['fcc', 'ontop']:
35
36         if site not in data[metal][adsorbate]:
37             data[metal][adsorbate][site] = {}
38
39         for coverage in [0.25, 0.5, 0.75, 1.0]:
40             if coverage not in data[metal][adsorbate][site]:
41                 data[metal][adsorbate][site][coverage] = {}
42
43             if [metal, adsorbate, site, coverage] in ignore_calcs:
44                 data[metal][adsorbate][site][coverage] = 'Q'
45
46             wd = 'dispersion/{metal}/{adsorbate}/{site}/{coverage}'.format(**locals())
47             cwd = os.getcwd()
48             try:
49                 with jasp(wd) as calc:
50                     try:
51                         atoms = calc.get_atoms()
52                         energy = atoms.get_potential_energy()
53
54                         Hads = energy
55                         Hads -= clean_slab_energy
56                         for atom in atoms:
57                             if atom.symbol == adsorbate:
58                                 Hads -= ads_energy
59
60                         Hads /= coverage*4 # normalize by number of adsorbates
61
62                         data[metal][adsorbate][site][coverage] = Hads
63                         s = '|{metal}|{adsorbate}|{site}|{coverage}|{Hads}|'.format(**locals())
64                         print s
65                     except (VaspSubmitted, VaspQueued):
66                         data[metal][adsorbate][site][coverage] = 'Q'
67                     except (VaspNotConverged):
68                         data[metal][adsorbate][site][coverage] = 'Q'
69             except (VaspNotFinished, AttributeError):
70                 data[metal][adsorbate][site][coverage] = 'Q'
71             os.chdir(cwd)
72
73
74

```

```

75 myproj.cwd()
76 f = open('dispersion-energies.json', 'w')
77 import json
78 json.dump(data, f)
79 f.close()

```

dispersion-energies.json: 

7.3 Site site correlations with dispersion

The code below reads data from the dispersion-energies.json file we created from the calculation date and looks at correlations between sites. The figure and discussion is shown following the code.

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import json
4  from matplotlib import rcParams, rc
5  rcParams['mathtext.default'] = 'regular'
6
7  fig = plt.figure(1, (4.5, 4.5))
8
9  with open('energies.json', 'r') as f:
10     data = json.load(f)
11
12  with open('dispersion-energies.json', 'r') as f:
13     dispersion_data = json.load(f)
14
15
16  '''
17  color for metal
18  letter symbol for adsorbate
19  '''
20  colors = {'Cu':'Orange',
21           'Ag':'Silver',
22           'Au':'Yellow',
23           'Pd':'Green',
24           'Pt':'Red',

```

```

25         'Rh': 'Blue',
26         'Ir': 'Purple'}
27
28 all_ads = ['O', 'S']
29
30 axes = fig.add_axes([0.15, 0.15, 0.35, 0.73])
31
32 for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
33     for adsorbate in all_ads:
34         E1, E2 = [], []
35         for coverage in '0.25', '0.5', '0.75', '1.0':
36             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
37                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
38                 E1.append(data[metal][adsorbate]['ontop'][coverage])
39                 E2.append(data[metal][adsorbate]['fcc'][coverage])
40         axes.plot(E1, E2,
41                 marker='$_{s}$' % adsorbate,
42                 color=colors[metal])
43
44 axes.text(-5.5, -1.5, '(a)', size='large')
45 axes.plot([-6, 0], [-6, 0], 'k--')
46 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
47 axes.set_ylabel('$\Delta H_{ads,fcc}$ (eV)')
48 axes.set_xlim(-6, -1)
49 axes.set_ylim(-6, -1)
50 axes.set_xticks([-5, -3, -1])
51
52 axes = fig.add_axes([0.55, 0.15, 0.35, 0.73])
53
54 for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
55     for adsorbate in all_ads:
56         E1, E2 = [], []
57         for coverage in '0.25', '0.5', '0.75', '1.0':
58             if (isinstance(dispersion_data[metal][adsorbate]['ontop'][coverage], float) and
59                 isinstance(dispersion_data[metal][adsorbate]['fcc'][coverage], float)):
60                 E1.append(dispersion_data[metal][adsorbate]['ontop'][coverage])
61                 E2.append(dispersion_data[metal][adsorbate]['fcc'][coverage])
62         axes.plot(E1, E2,
63                 marker='$_{s}$' % adsorbate,
64                 color=colors[metal])
65

```

```

66 axes.text(-5.5, -1.5, '(b)', size='large')
67 axes.plot([-6, 0], [-6, 0], 'k--')
68 axes.set_xlabel('$\Delta H_{ads,ontop}$ (eV)')
69 axes.set_xlim(-6, -1)
70 axes.set_ylim(-6, -1)
71 axes.set_yticklabels([])
72 axes.set_xticks([-5, -3, -1])
73
74 axes.annotate('', xytext=(-5, -7.5), xy=(-6.2, -6.5), size=10,
75               arrowprops=dict(arrowstyle='simple', color='r',
76                               connectionstyle='arc3,rad=-0.5'))
77
78 plt.savefig('supporting-figures/dispersion-site-site.png')
79 plt.savefig('supporting-figures/dispersion-site-site.pdf')
80
81 plt.show()

```

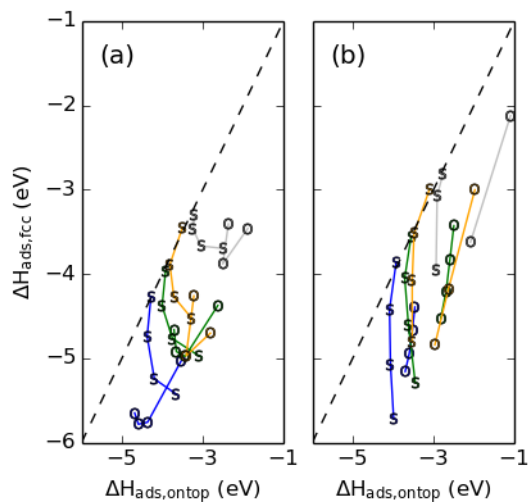


Figure S2: Comparison between correlations between the ontop and fcc site adsorption energies of O and S without dispersion (a) and with dispersion (b).

One of the key observations from calculations without dispersion was that coverage has an important role in breaking down correlations adsorption sites. This is clear from Figure S2 (a), where an increase in coverage causes the adsorption energy on the fcc site to increase but the ontop site to decrease. Interestingly, this effect is lessened with the inclusion of

dispersion, but not eliminated. With dispersion on, the slope becomes closer to unity, but the effect of coverage is still results in varying behavior on different sites. For example, changing coverage produces little change in the adsorption energy on the ontop site but large changes on the fcc site.

7.4 Scaling relationships with dispersion

The code below reads data from the dispersion-energies.json file we created from the calculation date and looks at correlations between the O and S adsorbate. The figure and discussion is shown following the code.

```
1  from pylab import *
2  import numpy as np
3  import json
4
5  from pycse import regress
6
7  with open('energies.json', 'r') as f:
8      data = json.load(f)
9
10 with open('dispersion-energies.json', 'r') as f:
11     dispersion_data = json.load(f)
12
13 colors = {'0.25': 'b',
14          '0.5': 'g',
15          '0.75': 'm',
16          '1.0': 'c'}
17
18 markers = {'0.25': 'o',
19           '0.5': 's',
20           '0.75': '^',
21           '1.0': 'd'}
22
23 pairs = (('C', 'N'),
24         ('N', 'O'),
25         ('C', 'O'),
26         ('C', 'F'))
```

```

27
28 fig = plt.figure(1, (6, 3))
29
30 inset = fig.add_axes([0.66, 0.2, 0.25, 0.78])
31 inset.set_xlim(0.2, 1.05)
32 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
33 inset.set_xticklabels([])
34 inset.set_yticklabels([])
35 inset.axhline(1.0, ls='--', c='k')
36
37 pair = ('O', 'S')
38 axes = fig.add_axes([0.14, 0.2, 0.25, 0.78])
39 ms, mhighs, m lows = [], [], []
40 for coverage in ('0.25', '0.5', '0.75', '1.0'):
41     E1, E2 = [], []
42     for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
43         for site in ['fcc', 'ontop']:
44             if (isinstance(data[metal][pair[0]][site][coverage], float) and
45                 isinstance(data[metal][pair[1]][site][coverage], float)):
46                 E1.append(data[metal][pair[0]][site][coverage])
47                 E2.append(data[metal][pair[1]][site][coverage])
48     Efits = np.linspace(min(E1), max(E1))
49     func = np.poly1d(np.polyfit(E1, E2, 1))
50     E1 = np.array(E1)
51     E2 = np.array(E2)
52     A = np.column_stack([E1**0, E1])
53     alpha = 0.05
54     p, pint, se = regress(A, E2, alpha)
55     m = p[1]
56     mint = pint[1]
57     ms.append(m)
58     mhighs.append(abs(m - mint[1]))
59     m lows.append(abs(m - mint[0]))
60     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='o')
61     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
62 inset.errorbar([0.25 - 0.025, 0.5 - 0.025, 0.75 - 0.025, 1.0 - 0.025], ms, yerr=(mhighs, m lows),
63               elinewidth=1.5, marker='o', c='r')
64 axes.text(-6.4, -2.05, '(a)')
65 axes.set_xlim(-7, -1)
66 axes.set_xticks([-6, -4, -2])
67 axes.set_ylim(-6, -1.5)

```

```

68 axes.set_yticks([-2, -3, -4, -5])
69
70 axes.set_xlabel(r'$\Delta \mathdefault{E_{ads}}^{\{0\}}$ (eV)')
71 axes.set_ylabel(r'$\Delta \mathdefault{E_{ads}}^{\{S\}}$ (eV)')
72
73 pair = ('O', 'S')
74 axes = fig.add_axes([0.4, 0.2, 0.25, 0.78])
75 ms, mhighs, m lows = [], [], []
76 for coverage in ('0.25', '0.5', '0.75', '1.0'):
77     E1, E2 = [], []
78     for metal in ['Rh', 'Pd', 'Cu', 'Ag']:
79         for site in ['fcc', 'ontop']:
80             if (isinstance(dispersion_data[metal][pair[0]][site][coverage], float) and
81                 isinstance(dispersion_data[metal][pair[1]][site][coverage], float)):
82                 E1.append(dispersion_data[metal][pair[0]][site][coverage])
83                 E2.append(dispersion_data[metal][pair[1]][site][coverage])
84     Efits = np.linspace(min(E1), max(E1))
85     func = np.poly1d(np.polyfit(E1, E2, 1))
86     E1 = np.array(E1)
87     E2 = np.array(E2)
88     A = np.column_stack([E1**0, E1])
89     alpha = 0.05
90     p, pint, se = regress(A, E2, alpha)
91     m = p[1]
92     mint = pint[1]
93     ms.append(m)
94     mhighs.append(abs(m - mint[1]))
95     m lows.append(abs(m - mint[0]))
96     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='o')
97     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
98 inset.errorbar([0.25 + 0.025, 0.5 + 0.025, 0.75 + 0.025, 1.0 + 0.025], ms, yerr=(mhighs, m lows),
99               elinewidth=1.5, marker='o', c='y')
100 axes.text(-6.4, -2.05, '(b)')
101 axes.set_xlim(-7, -1)
102 axes.set_xticks([-6, -4, -2])
103 axes.set_ylim(-6, -1.5)
104 axes.set_yticks([-2, -3, -4, -5])
105 axes.set_yticklabels([])
106
107 axes.set_xlabel(r'$\Delta \mathdefault{E_{ads}}^{\{0\}}$ (eV)')
108

```

```

109 inset.set_xlim(0.15, 1.1)
110 inset.set_xticks([0.25, 0.5, 0.75, 1.0])
111 inset.text(0.85, 1.07, '(c)')
112 inset.set_xlabel('Coverage')
113 insetx = inset.twinx()
114 insetx.set_xticklabels([r'\mathdefault{\frac{1}{4}}$',
115                        r'\mathdefault{\frac{1}{2}}$',
116                        r'\mathdefault{\frac{3}{4}}$',
117                        '1'])
118 inset.set_xlim(0.15, 1.1)
119 insetx.set_xticks([0.25, 0.5, 0.75, 1.0])
120 insetx.set_ylabel('Slope')
121 insetx.set_yticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
122 insetx.set_ylim(inset.get_ylim())
123
124 plt.savefig('supporting-figures/dispersion-scaling.png')
125 plt.savefig('supporting-figures/dispersion-scaling.pdf')
126 plt.show()

```

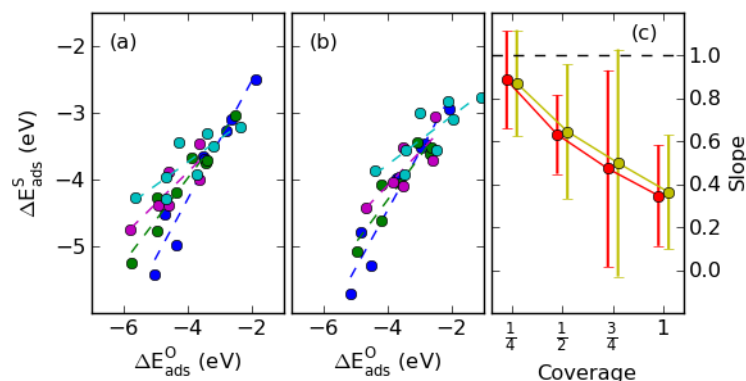


Figure S3: Correlations between adsorption of O and S using only fcc and ontop sites without dispersion (a) and with dispersion (b). (c) The slope of the correlation with respect to increasing coverage without dispersion (red) and with dispersion (yellow).

Figure S3 shows the effect of dispersion on the scaling relationship between S and O. The calculations used for both with and without dispersion only include adsorption energies on Rh, Pd, Cu, and Ag on the fcc and ontop site. Hence, the confidence intervals in comparison to the ones that include all of the sites and metals will be larger. Comparison between Figures S3 (a) and (b) show that dispersion tends to make the adsorption energies more

endothermic. The slopes of the scaling relationships at varying coverage shows little change when dispersion is added. At 0.25 and 1.0 ML coverage, the confidence intervals between with and without dispersion are not too different, but the confidence intervals at 0.5 and 0.75 ML with dispersion are larger than those without dispersion. This comparison shows that though dispersion can have a significant effect on the absolute values of the adsorption energies, the scaling relationships are preserved and similar to those without dispersion.

8 Required modules

In addition to having a working license to the Vienna Ab-initio Simulation Package (VASP), one also requires at least Python 2.5 and several Python modules to reproduce all of our data. These modules are summarized below.

8.1 `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).

8.2 `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/>.

8.3 `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>.

8.4 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>

9 Creating the TOC figure

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import json
4 from matplotlib.patches import FancyArrow, FancyArrowPatch
5 from matplotlib.lines import Line2D
6 from matplotlib import rc, rcParams
7 rcParams['mathtext.fontset'] = 'cm'
8 rcParams['mathtext.default'] = 'regular'
9 rc('font', **{'size': 8})
10
11 f = open('energies.json', 'r')
12 data = json.load(f)
13
14 fig = plt.figure(1, (2, 2), facecolor='white')
15
16 colors = {'Cu': 'Orange',
17           'Ag': 'Silver',
18           'Au': 'Yellow',
19           'Pd': 'Green',
20           'Pt': 'Red',
21           'Rh': 'Blue',
22           'Ir': 'Purple'}
```

```

24 simple_ads = ['H', 'C', 'N', 'O']
25
26 axes = fig.add_axes([0.14, 0.2, 0.28, 0.7], frameon=False)
27
28 for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
29     for adsorbate in simple_ads:
30         E1, E2 = [], []
31         for coverage in '0.25', '0.5', '0.75', '1.0':
32             if (isinstance(data[metal][adsorbate]['ontop'][coverage], float) and
33                 isinstance(data[metal][adsorbate]['fcc'][coverage], float)):
34                 E1.append(data[metal][adsorbate]['ontop'][coverage])
35                 E2.append(data[metal][adsorbate]['fcc'][coverage])
36
37         axes.plot(E1, E2,
38                   color=colors[metal], ms=3)
39
40 axes.plot([-9, 0], [-9, 0], 'k--')
41 axes.set_xlabel('$\Delta E_{ads,top}^{\{A\}}$')
42 ylab = axes.set_ylabel('$\Delta E_{ads,fcc}^{\{A\}}$', labelpad=0)
43
44 axes.set_xlim(-7, 0)
45 axes.set_ylim(-9, 0)
46 axes.set_xticklabels([])
47 axes.set_xticks([])
48 axes.set_yticklabels([])
49 axes.set_yticks([])
50
51 xmin, xmax = axes.get_xaxis().get_view_interval()
52 ymin, ymax = axes.get_yaxis().get_view_interval()
53 axes.add_artist(Line2D((xmin, xmax), (ymin, ymin), color='black', linewidth=2))
54 axes.add_artist(Line2D((xmin, xmin), (ymin, ymax), color='black', linewidth=2))
55
56 axes.annotate('', xy=(-3, -6), xytext=(-6.5, -7),
57               arrowprops=dict(arrowstyle='<-', color='r',
58                               connectionstyle='arc3,rad=0.5'))
59 axes.text(-6.5, -8.8, 'Increasing\ncoverage', size=8, color='r')
60
61
62 colors = {'0.25': 'b',
63           '0.5': 'g',
64           '0.75': 'm',

```

```

65         '1.0': 'c'}
66
67 markers = {'0.25': 'o',
68            '0.5': 's',
69            '0.75': '^',
70            '1.0': 'd'}
71
72 pair = ('F', 'Br')
73 axes = fig.add_axes([0.55, 0.2, 0.3, 0.7], frameon=False)
74 ms, mhighs, m lows = [], [], []
75 for coverage in ('0.25', '0.5', '0.75', '1.0'):
76     E1, E2 = [], []
77     for metal in ['Cu', 'Ag', 'Au', 'Pd', 'Pt', 'Rh', 'Ir']:
78         for site in ['fcc', 'hcp', ]:
79             if (isinstance(data[metal][pair[0]][site][coverage], float) and
80                 isinstance(data[metal][pair[1]][site][coverage], float)):
81                 E1.append(data[metal][pair[0]][site][coverage])
82                 E2.append(data[metal][pair[1]][site][coverage])
83     Efits = np.linspace(min(E1), max(E1))
84     func = np.poly1d(np.polyfit(E1, E2, 1))
85     E1 = np.array(E1)
86     E2 = np.array(E2)
87     axes.plot(E1, E2, ls='none', c=colors[coverage], marker='s', ms=3)
88     axes.plot(Efits, func(Efits), c=colors[coverage], ls='--')
89
90 axes.set_xlim(-4.5, -2)
91 axes.set_xticks([])
92 axes.set_xticklabels([])
93 axes.set_yticks([])
94 axes.set_xticklabels([])
95
96 axes.set_xlabel(r'$\Delta E_{\text{ads}}^{\text{B}}$', size=8)
97 axes.set_ylabel(r'$\Delta E_{\text{ads}}^{\text{A}}$', size=8, labelpad=0)
98
99 xmin, xmax = axes.get_xaxis().get_view_interval()
100 ymin, ymax = axes.get_yaxis().get_view_interval()
101 axes.add_artist(Line2D((xmin, xmax), (ymin, ymin), color='black', linewidth=2))
102 axes.add_artist(Line2D((xmin, xmin), (ymin, ymax), color='black', linewidth=2))
103
104 l1 = FancyArrow(0.88, 0.3, 0, 0.45, fc='r', width=0.0015, head_width=0.03,
105                 transform=fig.transFigure, figure=fig, color='r')

```

```

106 fig.patches.extend([l1])
107 fig.text(0.95, 0.55, 'Increasing\n Coverage', ha='center', va='center',
108          size=8, rotation='vertical', color='r')
109
110
111
112 fig.savefig('figures/TOC.png', dpi=300)
113 fig.savefig('figures/TOC.eps', dpi=300)
114 fig.savefig('figures/TOC.pdf', dpi=300)
115 plt.show()

```

References

- (S1) Papoian, G.; Nørskov, J. K.; Hoffmann, R. A Comparative Theoretical Study of the Hydrogen, Methyl, and Ethyl Chemisorption on the Pt(111) Surface. **2000**, *122*, 4129–4144.
- (S2) Xin, H.; Linic, S. Communications: Exceptions To the d-band Model of Chemisorption on Metal Surfaces: The Dominant Role of Repulsion Between Adsorbate States and Metal d-states. **2010**, *132*, 221101.
- (S3) Grimme, S. Semiempirical GGA-Type Density Functional Constructed With a Long-Range Dispersion Correction. **2006**, *27*, 1787–1799.