

Supporting Information: Identifying Potential BO₂ Oxide Polymorphs for Epitaxial Growth Candidates

Prateek Mehta,[†] John R. Kitchin,^{*,†} and Paul A. Salvador[‡]

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

E-mail: jkitchin@andrew.cmu.edu

^{*}To whom correspondence should be addressed

[†]Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213

[‡]Department of Materials Science and Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213

1 Introduction

This document contains supporting data for the work, "Identifying Potential BO_2 Oxide Polymorphs for Epitaxial Growth Candidates." To make our work transparent and reproducible, we have stored all of the data used in this work in the JSON data format (<http://www.json.org>). The JSON file can be found here:

 (double-click to open).

JSON is a human-readable, language-independent data format. Code for parsing and generating JSON data is readily available in a large variety of programming languages like C, C++, Java, Python, Perl, etc. We have used Python to read and write JSON data for all the examples below. A number of online editors also exist, like <http://www.jsoneditoronline.org/>, see 1. Our JSON file contains information about how every calculation was setup, including all the computational parameters such as the planewave cutoff, exchange correlation functional, k -point grid, etc. It also contains relevant output from the calculation, like total energy, volume, unit cell parameters, atomic positions, forces, etc... We have stored equation of state calculation data and data from the final optimization calculation for each structure (see 2). Calculation data is stored in the standard dictionary syntax in Python, shown below.

```
1 d['<oxide>']['<polymorph>']['<functional>']['EOS']['calculations']
2 d['<oxide>']['<polymorph>']['<functional>']['final']['calculations']
```

The JSON file can be used to setup all our calculations, reproduce the figures in this work, gain additional information and to extend our results to other types of calculations. We now consider a few examples of how the JSON file can be used to do these things.

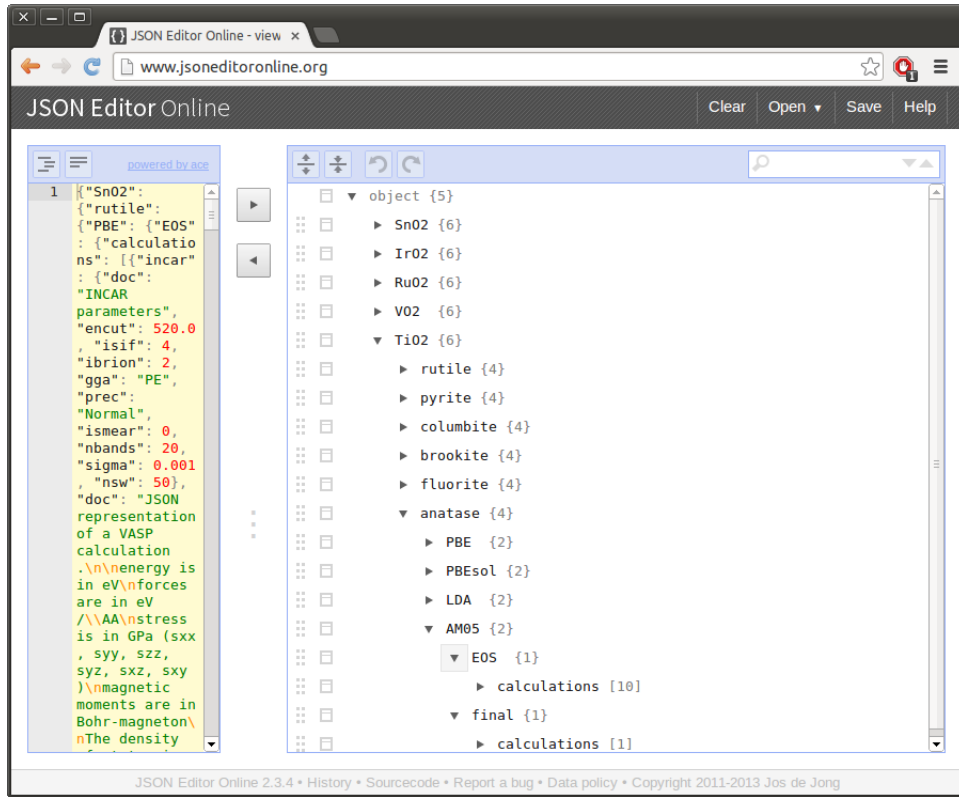


Figure 1: Snapshot of Online JSON Editor with our JSON tree

```

▼ calculations [1]
  ▼ 0 {6}
    ► incar {10}
      doc : JSON representation of a VASP calculation.\n\nenergy is
            in eV\nforces are in eV/\AA\nstress is in GPa (sxx, syx,
            syy, szz, syz, sxz, sxy)\nmagnetic moments are in Bohr-
            magneton\nThe density of states is reported with E_f at 0
            eV.\nVolume is reported in \AA^3\nCoordinates and cell
            parameters are reported in \AA\nIf atom-projected dos
            are included they are in the form:\n{ados:{energy:data,
            {atom index: {orbital : dos}}}\n
    ► atoms {5}
    ► input {7}
    ► potcar {2}
    ▼ data {6}
      ► stress [6]
        doc : Data from the output of the calculation
        volume : 70.25142348649388
        total_energy : -57.649399
      ► forces [6]
        fermi_level : 0.6502

```

Figure 2: A calculation represented in JSON

2 A Brief Note About The Supporting Information Document

The supporting information document was prepared in org-mode (<http://orgmode.org>) syntax, which was subsequently exported to L^AT_EX and converted to a PDF. Briefly, org-mode is a plain text format that enables intermingling of text and code, with markup for typical document elements such as headings, links, tables, etc..., and arbitrary inclusion of L^AT_EX for equations. With the Emacs editor, the code in an org-mode document can be executed in place, and the output captured in the document. For example, tables can be generated by code, or the code for generating a figure can be embedded in the document. The data in tables can be used as input to other code blocks in the document as well. Org-mode enables selective export of the content to various formats including html, L^AT_EX, and PDF. These features, and many others, make org-mode a convenient platform for reproducible research, where all the steps leading to conclusions can be documented in one place, but where it may be desirable not to show all the details in every view. For example, in an exported manuscript, or supporting information document such as this one. Nevertheless, it may still be valuable to go back to the source, for example, to figure out how some analysis was done, especially if all the code is not exported.

The org-mode source for this document can be found here: .

3 Computational Details of a Calculation

Here is an example of opening the JSON file, and printing the calculation details for one calculation, in this case the final geometry optimization for TiO₂ in the anatase polymorph, using the AM05 functional. We print the INCAR parameters used in the VASP calculations, the other input parameters such as the *k*-point grid, the POTCARs used, the final cell parameters, and other data saved from the calculation.

```

1  import json
2  import numpy as np
3
4  with open('supporting-information.json', 'rb') as f:
5      d = json.loads(f.read())
6
7  calc = d['TiO2']['anatase']['AM05']['final']['calculations'][0]
8
9  print 'INCAR:'
10 for key, val in calc['incar'].items():
11     print '{0} = {1}'.format(key, val)
12
13 print
14 print 'OTHER INPUT:'
15 for key, val in calc['input'].items():
16     print '{0} = {1}'.format(key, val)
17
18 print
19 print 'POTCARS:'
20 for sym,potcar, githash in calc['potcar']:
21     print sym, potcar, githash
22
23 print
24 # information about the geometry
25 print 'cell:\n', np.array(calc['atoms']['cell'])
26 print 'positions: \n',
27 for sym, pos in zip(calc['atoms']['symbols'], calc['atoms']['positions']):
28     print sym, pos
29
30 print
31 print 'output:'
32 for key, val in calc['data'].items():
33     if key == 'forces':
34         print 'forces = \n', np.array(val)
35     else:
36         print '{0} = {1}'.format(key, val)

```

INCAR:

doc = INCAR parameters

prec = Normal

```
isif = 3
ibrion = 2
gga = AM
encut = 520.0
ismear = 0
nbands = 20
sigma = 0.001
nsw = 50
```

OTHER INPUT:

```
kpts = [6, 6, 6]
reciprocal = False
xc = LDA
kpts_nintersections = None
setups = None
txt = -
gamma = False
```

POTCARS:

```
O potpaw/O/POTCAR 0cf2ce56049ca395c567026b700ed66c94a85161
Ti potpaw/Ti/POTCAR 51f7f05982d6b4052becc160375a8b8b670177a7
```

cell:

```
[[ 3.79437818  0.          0.          ]
 [ 0.          3.79437818  0.          ]
 [ 1.89718909  1.89718909  4.76872776]]
```

positions:

```
Ti [3.794378179384281, 2.845783634538211, 1.1921819398322235]
```

```
Ti [1.8971890896921404, 2.8457836345382104, 3.576545819496671]
O [3.794378179384281, 0.9485945448460702, 0.7914922560042584]
O [3.7943781793842803, 2.8457836345382104, 3.175856135668705]
O [1.8971890896921406, 4.742972724230351, 3.977235503324636]
O [1.8971890896921404, 2.845783634538211, 1.5928716236601885]
```

```
output:
```

```
stress = [0.09592973, 0.09592973, 1.57774819, 0.0, 0.0, 0.0]
```

```
doc = Data from the output of the calculation
```

```
volume = 68.6568316763
```

```
total_energy = -56.819327
```

```
forces =
```

```
[[ 0.      0.      0.      ]
 [ 0.      0.      0.      ]
 [ 0.      0.     -0.021397]
 [ 0.      0.     -0.021397]
 [ 0.      0.      0.021397]
 [ 0.      0.      0.021397]]
```

```
fermi_level = 0.7851
```

These results could have been read by hand from the JSON file, but it should be clear now that a machine can read them, and could even recreate the actual INCAR, KPOINTS, POSCAR and POTCAR files needed to rerun this calculation exactly the way it was done by us.

4 k -point Grids

Here we have a Python code block that reads the Monkhorst point k -point grids for each oxide polymorph from the JSON file and prints them in ??.

```
1 import json
2 with open('supporting-information.json', 'rb') as f:
3     d = json.loads(f.read())
4
5 print '#+caption:Monkhorst Point  $k$ -point meshes of all polymorphs of all oxides in this work\label{table:k-pts}'
6 print '|Oxide|Polymorph| $k$ -point grid|'
7
8     # oxide, latex key
9 oxides = [['TiO2', 'TiO $_2$ '],
10          ['VO2', 'VO $_2$ '],
11          ['RuO2', 'RuO $_2$ '],
12          ['IrO2', 'IrO $_2$ '],
13          ['SnO2', 'SnO $_2$ ']]
14 for B02,key in oxides:
15
16     print '|-'
17     for polymorph in ['rutile',
18                      'anatase',
19                      'brookite',
20                      'columbite',
21                      'pyrite',
22                      'fluorite']:
23         #  $k$ -points are consistent across functionals - we use LDA here
24         k1,k2,k3= d[B02][polymorph]['LDA']['final']['calculations'][0]['input']['kpts']
25         print '|{0}|{1}|{2}\\\times{3}\\\times{4}$|'.format(key,polymorph,k1,k2,k3)
```

Oxide	Polymorph	k -point grid
<hr/>		
TiO ₂	rutile	$6 \times 6 \times 6$
TiO ₂	anatase	$6 \times 6 \times 6$
TiO ₂	brookite	$6 \times 6 \times 6$
TiO ₂	columbite	$6 \times 6 \times 6$
TiO ₂	pyrite	$6 \times 6 \times 6$
TiO ₂	fluorite	$4 \times 4 \times 4$
<hr/>		
VO ₂	rutile	$6 \times 6 \times 6$
VO ₂	anatase	$16 \times 16 \times 16$
VO ₂	brookite	$6 \times 6 \times 6$
VO ₂	columbite	$6 \times 6 \times 6$
VO ₂	pyrite	$6 \times 6 \times 6$
VO ₂	fluorite	$6 \times 6 \times 6$
<hr/>		
RuO ₂	rutile	$6 \times 6 \times 6$
RuO ₂	anatase	$6 \times 6 \times 6$
RuO ₂	brookite	$8 \times 8 \times 8$
RuO ₂	columbite	$6 \times 6 \times 6$
RuO ₂	pyrite	$8 \times 8 \times 8$
RuO ₂	fluorite	$6 \times 6 \times 6$
<hr/>		
IrO ₂	rutile	$6 \times 6 \times 6$
IrO ₂	anatase	$7 \times 7 \times 7$
IrO ₂	brookite	$8 \times 8 \times 8$
IrO ₂	columbite	$6 \times 6 \times 6$
IrO ₂	pyrite	$6 \times 6 \times 6$
IrO ₂	fluorite	$6 \times 6 \times 6$
<hr/>		
SnO ₂	rutile	$8 \times 8 \times 8$
SnO ₂	anatase	$8 \times 8 \times 8$
SnO ₂	brookite	$8 \times 8 \times 8$
SnO ₂	columbite	$8 \times 8 \times 8$
SnO ₂	pyrite	$8 \times 8 \times 8$

5 Equations of State

5.1 Example Plots

Now we consider a more complicated example. Here we have a python code block to plot the equation of state for all polymorphs of all oxides using all functionals. We have used the *EquationOfState* module in *ASE*. Running this code will produce 120 equation of state plots, a representative example for rutile TiO_2 is shown in 3.

```
1 import json
2 import matplotlib.pyplot as plt
3 from ase.utils.eos import EquationOfState
4 with open('supporting-information.json', 'rb') as f:
5     d = json.loads(f.read())
6
7 for B02 in ['TiO2', 'VO2', 'RuO2', 'IrO2', 'SnO2']:
8     for polymorph in ['rutile', 'anatase', 'brookite', 'columbite', 'pyrite', 'fluorite']:
9         for xc in ['LDA', 'AM05', 'PBEsol', 'PBE']:
10             # number of atoms in the unit cell - used to normalize
11             natoms= len(d[B02][polymorph][xc]['EOS']['calculations']
12                        [0]['atoms']['symbols'])
13             volumes = [entry['data']['volume']*3./natoms for entry in
14                        d[B02][polymorph][xc]['EOS']['calculations']]
15             energies = [entry['data']['total_energy']*3./natoms for entry in
16                        d[B02][polymorph][xc]['EOS']['calculations']]
17
18             # Plotting EOS
19             eos = EquationOfState(volumes,energies,'birchmurnaghan')
20             v0, e0, B= eos.fit()
21             eos.plot('images/EOS/{0}/{0}-{1}-{2}-EOS.png'.format(B02,polymorph,xc))
```

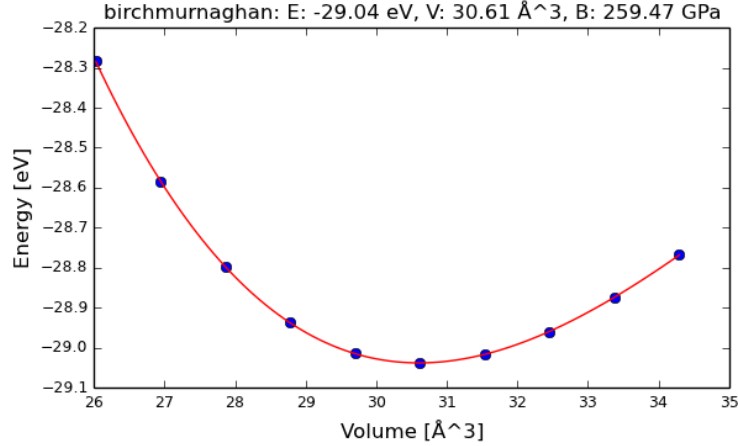


Figure 3: Equation of state for rutile TiO_2 using the LDA functional

5.2 Anatase VO_2

We have found the VO_2 anatase polymorph to be sensitive to the choice of functional. The LDA and PBEsol functionals produce good equation of state fits while AM05 and PBE do not (examples for LDA and PBE are shown in 4 and 5). Further, for AM05 and PBE we have had convergence issues in the final optimization, with the final volume found to be much higher than predicted in the equation of state. We have rechecked our calculations for a variety of different conditions, ranging from using very dense k -point grids, treating s and p semi-core states as valence states, and using both conjugate-gradient and quasi-Newton optimization algorithms. All these approaches have produced similar results. We have also found that anatase VO_2 has a very low bulk modulus compared to anatase polymorphs of other oxides. This leads one to believe that the anatase polymorph is very sensitive to small perturbations, which possibly explains why has been seen only as monolayers and not in the bulk form.

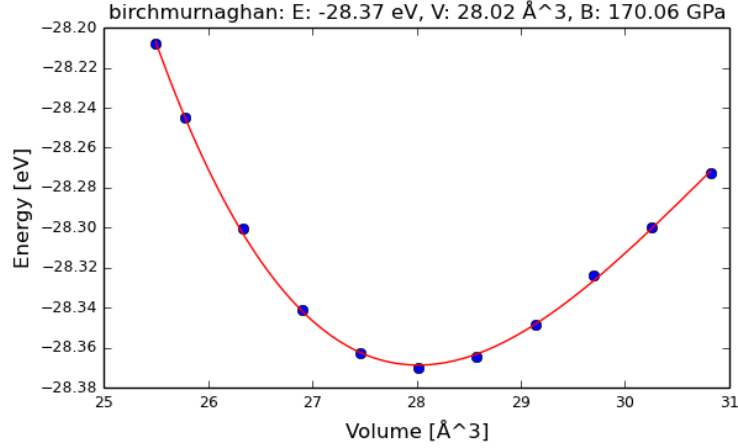


Figure 4: Equation of state for anatase VO₂ using the LDA functional

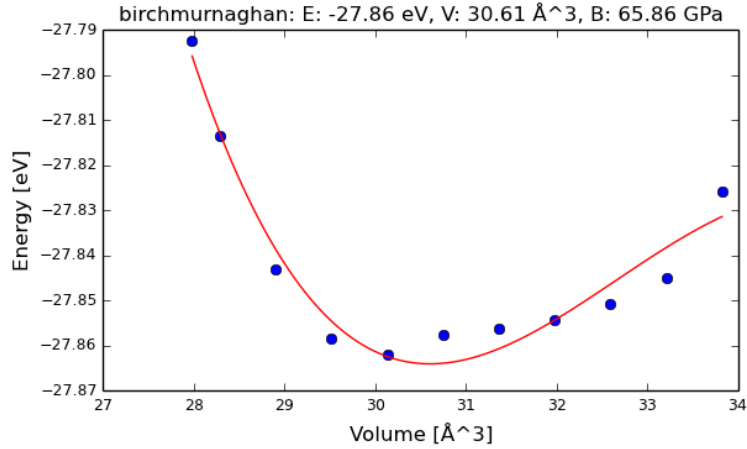


Figure 5: Equation of state for anatase VO₂ using the PBE functional

6 Relative Stability

Finally, we look at the relative stabilities of the different polymorphs. To do this we will use the JSON file to generate tables of data containing relevant information of the final optimized structures. We then use data from these tables to produce the plots of the relative stability. Note that to see how the data from the tables is used in the relative stability plots, one must view the org-mode source of this document. The scripts below actually print tables in org-mode syntax, which is exported to L^AT_EX. The details of this export are also contained

in the source of this document.

6.1 Relative Stability Tables

```
1 import json
2 import matplotlib.pyplot as plt
3 #from eos import EquationOfState
4 from ase.utils.eos import EquationOfState
5 with open('supporting-information.json', 'rb') as f:
6     d = json.loads(f.read())
7
8 for B02,key in [['TiO2','TiO$_2$'],
9                ['VO2','VO$_2$'],
10               ['RuO2','RuO$_2$'],
11               ['IrO2','IrO$_2$'],
12               ['SnO2','SnO$_2$']]:
13     print '#+caption: Computational energetics for {0} polymorphs'.format(key)
14     print '#+tblname:{0}-data'.format(B02)
15     print '|Oxide|Polymorph|Functional|E (kJ/mol)|V (\AA^3/f.u.)|Bulk Modulus (GPa)|'
16     print '|-'
17
18 for polymorph in ['rutile','anatase','brookite','columbite','pyrite','fluorite']:
19     for xc in ['LDA','AM05','PBEsol','PBE']:
20         # Equation of State Data
21         natoms= len(d[B02][polymorph][xc]['EOS']['calculations'][0]['atoms']['symbols'])
22         volumes = [entry['data']['volume']*3./natoms for entry in d[B02][polymorph][xc]['EOS']['calculations']]
23         energies = [entry['data']['total_energy']*3./natoms for entry in d[B02][polymorph][xc]['EOS']['calculations']]
24
25         # Fitting EOS to obtain bulk modulus
26         eos = EquationOfState(volumes,energies,'birchmurnaghan')
27         v0, e0, B= eos.fit()
28         try:
29             # Energy and Volume of final optimized structure
30             energy = d[B02][polymorph][xc]['final']['calculations'][0]['data']['total_energy']*3./natoms*96.4853075
31             volume = d[B02][polymorph][xc]['final']['calculations'][0]['data']['volume']*3./natoms
32
33             BM = B*160.2176487 # Converting to GPa
34             print '|{0}|{1}|{2}|{3:1.2f}|{4:1.2f}|{5:1.2f}|'.format(key, polymorph, xc, energy, volume, BM)
35
36             # Accounting for unavailable calculations
37         except (TypeError):
```

37 pass

38

39 print ''

Oxide	Polymorph	Functional	E (kJ/mol)	V ($\text{\AA}^3/\text{f.u.}$)	Bulk Modulus (GPa)
TiO ₂	rutile	LDA	-2801.64	30.58	259.47
TiO ₂	rutile	AM05	-2733.53	31.31	233.20
TiO ₂	rutile	PBEsol	-2759.29	31.22	239.76
TiO ₂	rutile	PBE	-2773.21	32.11	215.78
TiO ₂	anatase	LDA	-2802.73	33.62	187.40
TiO ₂	anatase	AM05	-2741.12	34.33	178.26
TiO ₂	anatase	PBEsol	-2763.61	34.25	178.71
TiO ₂	anatase	PBE	-2781.16	35.13	171.42
TiO ₂	brookite	LDA	-2803.26	31.56	238.06
TiO ₂	brookite	AM05	-2737.44	32.29	213.24
TiO ₂	brookite	PBEsol	-2762.06	32.18	220.77
TiO ₂	brookite	PBE	-2777.25	33.08	193.85
TiO ₂	columbite	LDA	-2803.53	30.00	246.07
TiO ₂	columbite	AM05	-2734.07	30.77	227.66
TiO ₂	columbite	PBEsol	-2760.35	30.64	233.23
TiO ₂	columbite	PBE	-2773.65	31.51	207.59
TiO ₂	pyrite	LDA	-2748.35	27.98	301.15
TiO ₂	pyrite	AM05	-2674.49	28.60	278.99
TiO ₂	pyrite	PBEsol	-2703.10	28.56	282.77
TiO ₂	pyrite	PBE	-2711.60	29.36	260.66
TiO ₂	fluorite	LDA	-2747.30	26.74	316.43
TiO ₂	fluorite	AM05	-2668.04	27.31	292.15
TiO ₂	fluorite	PBEsol	-2697.35	27.30	295.94
TiO ₂	fluorite	PBE	-2698.45	28.15	270.33

Oxide	Polymorph	Functional	E (kJ/mol)	V ($\text{\AA}^3/\text{f.u.}$)	Bulk Modulus (GPa)
VO ₂	rutile	LDA	-2752.92	28.02	293.60
VO ₂	rutile	AM05	-2678.15	28.63	266.03
VO ₂	rutile	PBEsol	-2703.13	28.58	271.91
VO ₂	rutile	PBE	-2703.29	29.52	243.10
VO ₂	anatase	LDA	-2737.35	28.09	170.06
VO ₂	anatase	PBEsol	-2688.97	28.65	146.11
VO ₂	brookite	LDA	-2746.79	27.70	205.39
VO ₂	brookite	AM05	-2672.36	28.48	178.82
VO ₂	brookite	PBEsol	-2697.20	28.46	185.02
VO ₂	brookite	PBE	-2697.13	29.65	148.08
VO ₂	columbite	LDA	-2745.93	27.78	260.85
VO ₂	columbite	AM05	-2670.49	28.33	238.78
VO ₂	columbite	PBEsol	-2695.87	28.38	243.00
VO ₂	columbite	PBE	-2696.23	29.25	215.62
VO ₂	pyrite	LDA	-2682.89	25.77	324.13
VO ₂	pyrite	AM05	-2602.87	26.28	299.24
VO ₂	pyrite	PBEsol	-2630.98	26.40	303.46
VO ₂	pyrite	PBE	-2627.41	27.21	276.50
VO ₂	fluorite	LDA	-2656.12	25.04	334.56
VO ₂	fluorite	AM05	-2571.99	25.53	308.16
VO ₂	fluorite	PBEsol	-2600.31	25.55	311.82
VO ₂	fluorite	PBE	-2591.40	26.41	281.88

Oxide	Polymorph	Functional	E (kJ/mol)	V ($\text{\AA}^3/\text{f.u.}$)	Bulk Modulus (GPa)
RuO ₂	rutile	LDA	-2439.03	30.74	309.10
RuO ₂	rutile	AM05	-2361.92	31.34	285.35
RuO ₂	rutile	PBEsol	-2382.47	31.37	288.43
RuO ₂	rutile	PBE	-2377.33	32.39	259.90
RuO ₂	anatase	LDA	-2372.64	35.33	234.61
RuO ₂	anatase	AM05	-2308.10	35.93	217.46
RuO ₂	anatase	PBEsol	-2323.91	36.00	218.84
RuO ₂	anatase	PBE	-2327.48	37.24	199.17
RuO ₂	brookite	LDA	-2398.53	32.57	238.10
RuO ₂	brookite	AM05	-2327.52	33.22	222.71
RuO ₂	brookite	PBEsol	-2345.92	33.23	223.84
RuO ₂	brookite	PBE	-2344.90	34.38	204.26
RuO ₂	columbite	LDA	-2426.20	30.16	260.71
RuO ₂	columbite	AM05	-2348.49	30.89	245.31
RuO ₂	columbite	PBEsol	-2368.64	30.88	246.17
RuO ₂	columbite	PBE	-2363.44	31.93	226.53
RuO ₂	pyrite	LDA	-2432.82	27.79	348.28
RuO ₂	pyrite	AM05	-2347.49	28.27	321.83
RuO ₂	pyrite	PBEsol	-2371.62	28.35	324.89
RuO ₂	pyrite	PBE	-2357.94	29.32	290.61
RuO ₂	fluorite	LDA	-2402.74	26.95	364.30
RuO ₂	fluorite	AM05	-2314.42	27.41	335.64
RuO ₂	fluorite	PBEsol	-2339.49	27.49	338.06
RuO ₂	fluorite	PBE	-2323.07	28.48	299.77

Oxide	Polymorph	Functional	E (kJ/mol)	V ($\text{\AA}^3/\text{f.u.}$)	Bulk Modulus (GPa)
IrO ₂	rutile	LDA	-2355.48	31.19	319.80
IrO ₂	rutile	AM05	-2284.25	31.66	297.77
IrO ₂	rutile	PBEsol	-2302.47	31.72	299.80
IrO ₂	rutile	PBE	-2295.17	32.77	270.36
IrO ₂	anatase	LDA	-2260.53	35.62	220.81
IrO ₂	anatase	AM05	-2202.43	36.38	204.41
IrO ₂	anatase	PBEsol	-2216.26	36.49	205.67
IrO ₂	anatase	PBE	-2218.82	37.59	186.42
IrO ₂	brookite	LDA	-2296.59	33.19	238.40
IrO ₂	brookite	AM05	-2232.30	33.76	229.08
IrO ₂	brookite	PBEsol	-2248.45	33.82	227.99
IrO ₂	brookite	PBE	-2246.65	34.97	211.78
IrO ₂	columbite	LDA	-2330.20	30.71	257.56
IrO ₂	columbite	AM05	-2258.69	31.31	244.62
IrO ₂	columbite	PBEsol	-2277.55	31.31	244.74
IrO ₂	columbite	PBE	-2270.61	32.46	227.07
IrO ₂	pyrite	LDA	-2339.81	28.42	359.14
IrO ₂	pyrite	AM05	-2260.58	28.81	335.26
IrO ₂	pyrite	PBEsol	-2282.49	28.90	336.06
IrO ₂	pyrite	PBE	-2267.50	29.90	301.47
IrO ₂	fluorite	LDA	-2245.17	28.91	349.08
IrO ₂	fluorite	AM05	-2168.09	29.31	324.58
IrO ₂	fluorite	PBEsol	-2192.18	29.40	326.79
IrO ₂	fluorite	PBE	-2179.70	30.44	291.28

Oxide	Polymorph	Functional	E (kJ/mol)	V ($\text{\AA}^3/\text{f.u.}$)	Bulk Modulus (GPa)
SnO ₂	rutile	LDA	-2052.71	35.76	212.39
SnO ₂	rutile	AM05	-2009.57	36.53	193.06
SnO ₂	rutile	PBEsol	-2025.23	36.48	197.04
SnO ₂	rutile	PBE	-2053.34	37.58	177.48
SnO ₂	anatase	LDA	-2026.74	40.36	164.17
SnO ₂	anatase	AM05	-1992.49	41.21	153.03
SnO ₂	anatase	PBEsol	-2004.00	41.17	154.01
SnO ₂	anatase	PBE	-2037.23	42.34	142.70
SnO ₂	brookite	LDA	-2034.33	37.38	182.29
SnO ₂	brookite	AM05	-1994.73	38.17	169.17
SnO ₂	brookite	PBEsol	-2008.86	38.12	170.85
SnO ₂	brookite	PBE	-2038.94	39.31	156.60
SnO ₂	columbite	LDA	-2049.45	35.03	192.05
SnO ₂	columbite	AM05	-2004.74	35.77	183.31
SnO ₂	columbite	PBEsol	-2021.17	35.72	183.81
SnO ₂	columbite	PBE	-2048.04	36.82	172.97
SnO ₂	pyrite	LDA	-2033.11	32.73	241.87
SnO ₂	pyrite	AM05	-1982.71	33.40	223.83
SnO ₂	pyrite	PBEsol	-2001.94	33.40	225.74
SnO ₂	pyrite	PBE	-2023.76	34.40	204.35
SnO ₂	fluorite	LDA	-1996.77	31.97	244.39
SnO ₂	fluorite	AM05	-1944.14	32.63	225.14
SnO ₂	fluorite	PBEsol	-1964.44	32.65	227.43
SnO ₂	fluorite	PBE	-1982.94	33.67	203.72

6.2 Relative Stability Plots

An example of how to produce relative stability plots and relative energetics data for TiO_2 is shown in 6 and 1. Plots for the other oxides can be produced in a similar fashion. Data has been read in from ??.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 B02 = 'TiO2'
5 key='TiO$_2$'
6 # polymorphs, color to make them, and position of label
7 polymorphs = [('Rutile', 'red',[29.65,4.0]),
8               ('Anatase', 'blue',[32.5,-15.115]),
9               ('Columbite', 'green',[28.36,-7.04]),
10              ('Brookite', 'goldenrod',[31,-10.04]),
11              ('Pyrite', 'gray',[27.433,47.6]),
12              ('Fluorite', 'brown',[26.32,68.7])]
13 # functional and marker for graph
14 xcs = [('LDA','o'),
15        ('PBEsol','d'),
16        ('AM05','v'),
17        ('PBE','s')]
18 # Printing Relative Energy Table
19 print '#+caption: Relative Energetics for TiO$_2$ Polymorphs \label{table:TiO2-relative}'
20 print '|Oxide|Polymorph|Functional|\Delta E(kJ/mol)|V (\AA^3/f.u.)|'
21 print '|-'
22 # Identifiers for data from TiO2 computational energetics table
23 OXIDE = 0
24 XC = 2
25 POLYMORPH = 1
26 E = 3
27 V = 4
28
29 plt.figure(figsize=(3,4))
30 for xc, marker in xcs:
31     # get rutile normalizing energy
32     rows = [row for row in data if (row[POLYMORPH].lower() == 'rutile'
33                                     and row[XC].lower() == xc.lower())]
34     for row in rows:
```

```

35     eref = row[E]
36     vref = row[V]
37     # Empty plots to generate legends
38     plt.plot([], [], marker, mfc='white', ms=9, mec='k', mew=1., label=xc)
39     for polymorph, color, xy in polymorphs:
40         x, y = xy
41         rows = [row for row in data if (row[POLYMORPH].lower() == polymorph.lower()
42                                         and row[XC].lower() == xc.lower())]
43         for row in rows:
44             e = row[E] - eref
45             v = row[V]
46
47             p, = plt.plot(v, e, ms=9.0, marker=marker, mec='k', mew=1, color=color)
48             print '|{0}|{1}|{2}|{3:1.3f}|{4:1.3f}|'.format(key, polymorph, xc, e, v)
49             # add a text label for clarity
50             if xc == 'PBEsol':
51                 plt.text(x, y, polymorph, color=color, size=9, weight=550, stretch=500)
52     print '|-'
53
54     plt.ylim([- 20, 80])
55     plt.xlabel('Volume ($\text{\AA}^3/\text{f.u.})', size=10)
56     plt.ylabel('Energy Relative to Rutile (kJ/mol)', size=11)
57     plt.tick_params(labelsize=9)
58     plt.legend(numpoints=1, markerscale = 0.6, prop={'size': 6})
59     plt.tight_layout()
60     plt.savefig('images/{0}-relative-xc-polymorphs'.format(B02), dpi = 300)

```

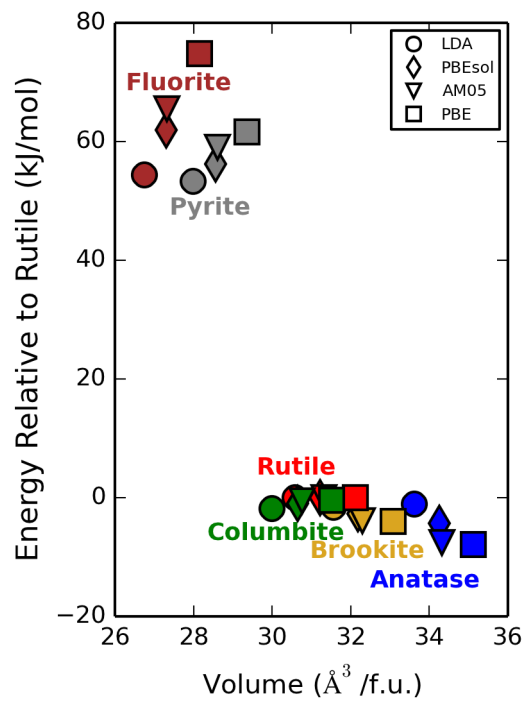


Figure 6: Relative Energetics for TiO_2 Polymorphs

Table 1: Relative Energetics for TiO₂ Polymorphs

Oxide	Polymorph	Functional	$\Delta E(\text{kJ/mol})$	$V (\text{\AA}^3/\text{f.u.})$
TiO ₂	Rutile	LDA	0.000	30.580
TiO ₂	Anatase	LDA	-1.090	33.620
TiO ₂	Columbite	LDA	-1.890	30.000
TiO ₂	Brookite	LDA	-1.620	31.560
TiO ₂	Pyrite	LDA	53.290	27.980
TiO ₂	Fluorite	LDA	54.340	26.740
TiO ₂	Rutile	PBEsol	0.000	31.220
TiO ₂	Anatase	PBEsol	-4.320	34.250
TiO ₂	Columbite	PBEsol	-1.060	30.640
TiO ₂	Brookite	PBEsol	-2.770	32.180
TiO ₂	Pyrite	PBEsol	56.190	28.560
TiO ₂	Fluorite	PBEsol	61.940	27.300
TiO ₂	Rutile	AM05	0.000	31.310
TiO ₂	Anatase	AM05	-7.590	34.330
TiO ₂	Columbite	AM05	-0.540	30.770
TiO ₂	Brookite	AM05	-3.910	32.290
TiO ₂	Pyrite	AM05	59.040	28.600
TiO ₂	Fluorite	AM05	65.490	27.310
TiO ₂	Rutile	PBE	0.000	32.110
TiO ₂	Anatase	PBE	-7.950	35.130
TiO ₂	Columbite	PBE	-0.440	31.510
TiO ₂	Brookite	PBE	-4.040	33.080
TiO ₂	Pyrite	PBE	61.610	29.360
TiO ₂	Fluorite	PBE	74.760	28.150