FRITZ-HABER-INSTITUT
MAX-PLANCK-GESELLSCHAFT

NOMAD
NOVEL MATERIALS DISCOVERY

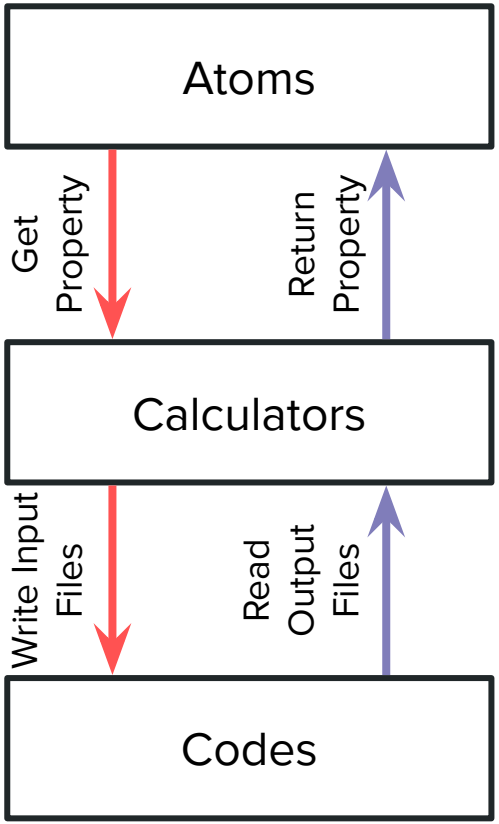
IRIS
ADLERSHOF

Alexander von Humboldt
Stiftung/Foundation

ASE: Current Status and Future Changes

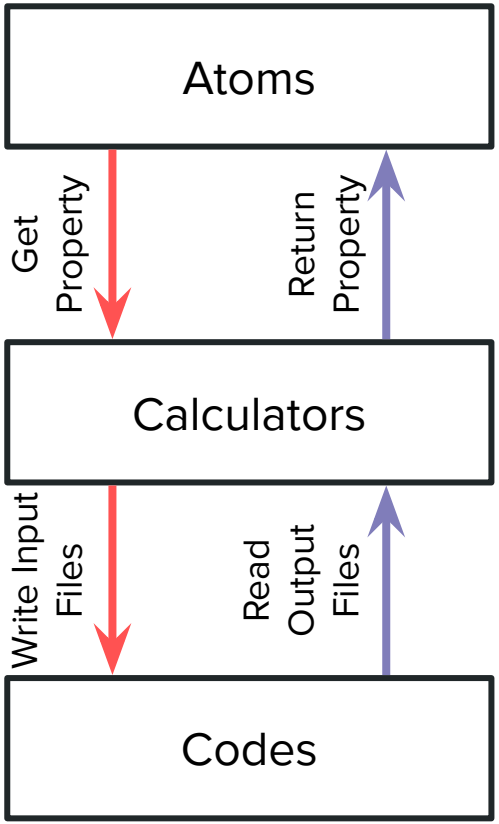
Thomas A. R. Purcell
August 3, 2023

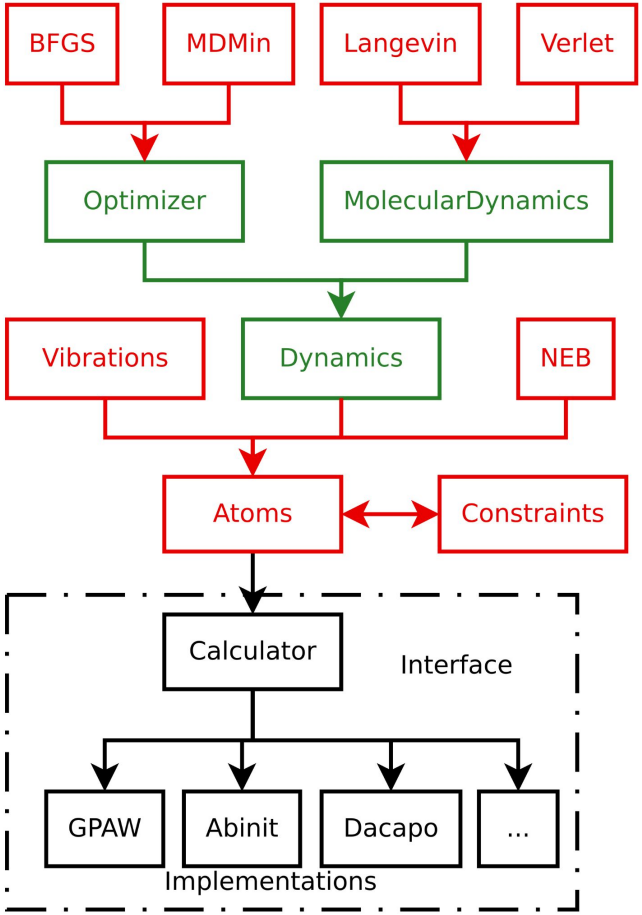
What is the Point of using ASE?



Ask Hjorth Larsen *et al* 2017 *J. Phys.: Condens. Matter* 29 273002
https://nano-bio.ehu.es/files/askhl-presentation-ase-abinitworkshop_0.pdf

What is the Point of using ASE?





Ask Hjorth Larsen *et al* 2017 *J. Phys.: Condens. Matter* 29 273002
https://nano-bio.ehu.es/files/askhl-presentation-ase-abinitworkshop_0.pdf

Finding the Lattice Constant of Silver with EMT Potential

```
import numpy as np

from ase import Atoms
from ase.io.trajectory import Trajectory
from ase.calculators.emt import EMT
```


https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html

Finding the Lattice Constant of Silver with EMT Potential

```python
import numpy as np

from ase import Atoms
from ase.io.trajectory import Trajectory
from ase.calculators.emt import EMT

a = 4.0  # approximate lattice constant
b = a / 2
ag = Atoms('Ag',
           cell=[(0, b, b), (b, 0, b), (b, b, 0)],
           pbc=1,
           calculator=EMT())  # use EMT potential
cell = ag.get_cell()
```


https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html

Finding the Lattice Constant of Silver with EMT Potential

```python
import numpy as np

from ase import Atoms
from ase.io.trajectory import Trajectory
from ase.calculators.emt import EMT

a = 4.0  # approximate lattice constant
b = a / 2
ag = Atoms('Ag',
           cell=[(0, b, b), (b, 0, b), (b, b, 0)],
           pbc=1,
           calculator=EMT())  # use EMT potential
cell = ag.get_cell()
traj = Trajectory('Ag.traj', 'w')
for x in np.linspace(0.95, 1.05, 5):
    ag.set_cell(cell * x, scale_atoms=True)
    ag.get_potential_energy()
    traj.write(ag)
```


https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html

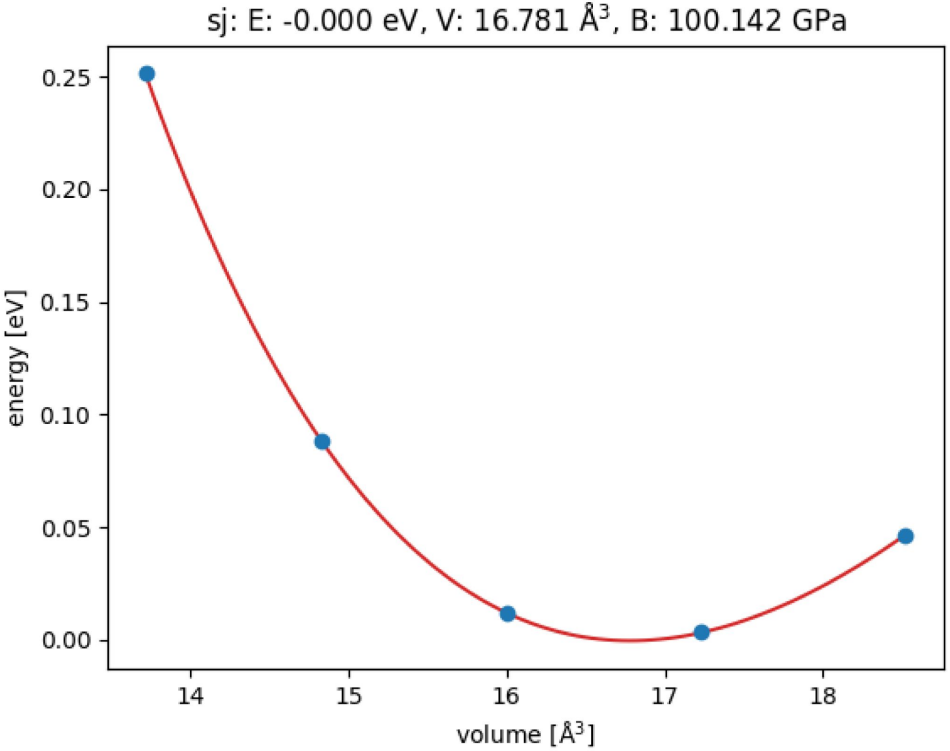
Finding the Lattice Constant of Silver with EMT Potential

```python
import numpy as np

from ase import Atoms
from ase.io.trajectory import Trajectory
from ase.calculators.emt import EMT

a = 4.0  # approximate lattice constant
b = a / 2
ag = Atoms('Ag',
           cell=[(0, b, b), (b, 0, b), (b, b, 0)],
           pbc=1,
           calculator=EMT())  # use EMT potential
cell = ag.get_cell()
traj = Trajectory('Ag.traj', 'w')
for x in np.linspace(0.95, 1.05, 5):
    ag.set_cell(cell * x, scale_atoms=True)
    ag.get_potential_energy()
    traj.write(ag)
```

```python
from ase.io import read
from ase.units import kJ
from ase.eos import EquationOfState
configs = read('Ag.traj@0:5')  # read 5 configurations
# Extract volumes and energies:
volumes = [ag.get_volume() for ag in configs]
energies = [ag.get_potential_energy() for ag in configs]
eos = EquationOfState(volumes, energies)
v0, e0, B = eos.fit()
print(B / kJ * 1.0e24, 'GPa')
eos.plot('Ag-eos.png')
```


https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html

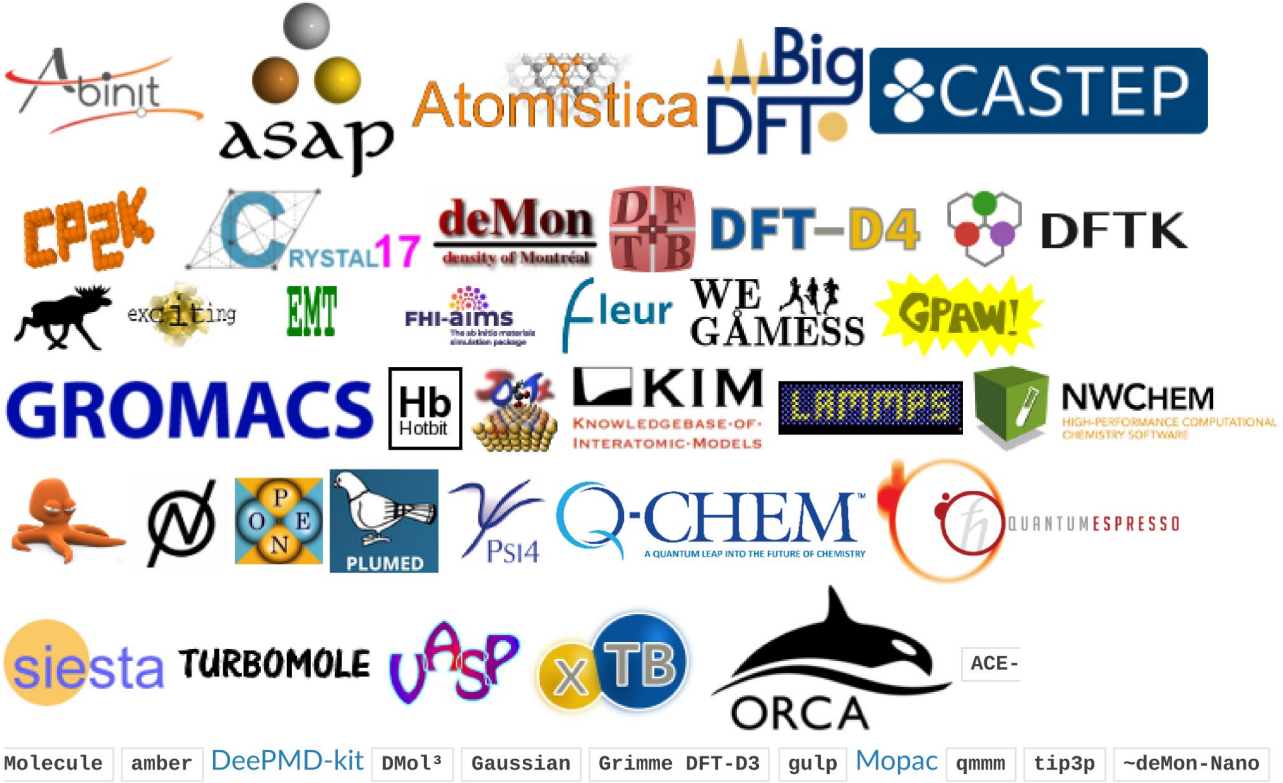
Finding the Lattice Constant of Silver with EMT Potential




```python
from ase.io import read
from ase.units import kJ
from ase.eos import EquationOfState
configs = read('Ag.traj@0:5')  # read 5 configurations
# Extract volumes and energies:
volumes = [ag.get_volume() for ag in configs]
energies = [ag.get_potential_energy() for ag in configs]
eos = EquationOfState(volumes, energies)
v0, e0, B = eos.fit()
print(B / kJ * 1.0e24, 'GPa')
eos.plot('Ag-eos.png')
```


https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html

Code Agnostic Framework



https://wiki.fysik.dtu.dk/ase/index.html

Code Agnostic Framework

https://wiki.fysik.dtu.dk/ase/index.html

Old Format had no Unified API for all Calculators

```python
class Aims(FileIOCalculator):
    # was "command" before the refactoring to dynamical commands
    __command_default = 'aims.version.serial.x > aims.out'
    __outfilename_default = 'aims.out'

    implemented_properties = ['energy', 'forces', 'stress', 'stresses',
                              'dipole', 'magmom']

    def __init__(self, restart=None,
                 ignore_bad_restart_file=FileIOCalculator._deprecated,
                 label=os.curdir, atoms=None, cubes=None, radmul=None,
                 tier=None, aims_command=None,
                 outfilename=None, **kwargs):
```

```python
class Vasp(GenerateVaspInput, Calculator):  # type: ignore
    """ASE interface for the Vienna Ab initio Simulation Package (VASP),
    with the Calculator interface.…
    """
    name = 'vasp'
    ase_objtype = 'vasp_calculator'  # For JSON storage

    # Environment commands
    env_commands = ('ASE_VASP_COMMAND', 'VASP_COMMAND', 'VASP_SCRIPT')

    implemented_properties = [
        'energy', 'free_energy', 'forces', 'dipole', 'fermi', 'stress',
        'magmom', 'magmoms'
    ]

    # Can be used later to set some ASE defaults
    default_parameters: Dict[str, Any] = {}

    def __init__(self,
                 atoms=None,
                 restart=None,
                 directory='.',
                 label='vasp',
                 ignore_bad_restart_file=Calculator._deprecated,
                 command=None,
                 txt='vasp.out',
                 **kwargs):
```

```python
class Espresso(FileIOCalculator):
    """
    """
    implemented_properties = ['energy', 'forces', 'stress', 'magmoms']
    command = 'pw.x -in PREFIX.pwi > PREFIX.pwo'
    discard_results_on_any_change = True

    def __init__(self, restart=None,
                 ignore_bad_restart_file=FileIOCalculator._deprecated,
                 label='espresso', atoms=None, **kwargs):
```

```python
class Abinit(FileIOCalculator):
    """Class for doing ABINIT calculations.

    The default parameters are very close to those that the ABINIT
    Fortran code would use.  These are the exceptions::…
    """

    implemented_properties = ['energy', 'forces', 'stress', 'magmom']
    ignored_changes = {'pbc'}  # In abinit, pbc is always effectively True.
    command = 'abinit < PREFIX.files > PREFIX.log'
    discard_results_on_any_change = True

    default_parameters = dict(
        xc='LDA',
        smearing=None,
        kpts=None,
        raw=None,
        pps='fhi')

    def __init__(self, restart=None,
                 ignore_bad_restart_file=FileIOCalculator._deprecated,
                 label='abinit', atoms=None, pp_paths=None,
                 v8_legacy_format=None,
                 **kwargs):
```

ASE Currently in a State of Flux

```python
class Espresso(GenericFileIOCalculator):
    def __init__(self, *, profile=None,
                 command=GenericFileIOCalculator._deprecated,
                 label=GenericFileIOCalculator._deprecated,
                 directory='.',
                 **kwargs):
```

```python
class Abinit(GenericFileIOCalculator):
    """Class for doing ABINIT calculations.

    The default parameters are very close to those that the ABINIT
    Fortran code would use.  These are the exceptions::...
    """

    def __init__(self, *, profile=None, directory='.', **kwargs):
```

```python
class Aims(GenericFileIOCalculator):
    def __init__(self, profile=None, directory='.', **kwargs):
```

```python
class Vasp(GenerateVaspInput, Calculator):  # type: ignore
    """ASE interface for the Vienna Ab initio Simulation Package (VASP),
    with the Calculator interface.
    """
    name = 'vasp'
    ase_objtype = 'vasp_calculator'  # For JSON storage

    # Environment commands
    env_commands = ('ASE_VASP_COMMAND', 'VASP_COMMAND', 'VASP_SCRIPT')

    implemented_properties = [
        'energy', 'free_energy', 'forces', 'dipole', 'fermi', 'stress',
        'magmom', 'magmoms'
    ]

    # Can be used later to set some ASE defaults
    default_parameters: Dict[str, Any] = {}

    def __init__(self,
                 atoms=None,
                 restart=None,
                 directory='.',
                 label='vasp',
                 ignore_bad_restart_file=Calculator._deprecated,
                 command=None,
                 txt='vasp.out',
                 **kwargs):
```

ASE new Release Soonish?

```python
class Espresso(GenericFileIOCalculator):
    def __init__(self, *, profile=None,
                 command=GenericFileIOCalculator._deprecated,
                 label=GenericFileIOCalculator._deprecated,
                 directory='.',
                 **kwargs):
```

```python
class Abinit(GenericFileIOCalculator):
    """Class for doing ABINIT calculations.

    The default parameters are very close to those that the ABINIT
    Fortran code would use.  These are the exceptions::...
    """

    def __init__(self, *, profile=None, directory='.', **kwargs):
```

```python
class Aims(GenericFileIOCalculator):
    def __init__(self, profile=None, directory='.', **kwargs):
```

News

- ASE version 3.22.1 released (1 December 2021).
- ASE version 3.22.0 released (24 June 2021).
- ASE version 3.21.1 released (24 January 2021).
- ASE version 3.21.0 released (18 January 2021).
- New bugfix releases 3.20.1 and 3.19.3 (11 August 2020).
- ASE version 3.20.0 released (8 August 2020).
- ASE version 3.19.2 released (22 July 2020).
- ASE version 3.19.1 released (4 April 2020).
- ASE version 3.19.0 released (16 December 2019).

What is new in ASE for FHI-aims

Major changes in ASE for FHI-aims

1. New GenericFileIOCalculator
2. IO Rework
   - Chunk Based Parser
   - Updated Results Dictionary
   - Functions to provide some customizable results reading
3. Improvements on the Horizon
   - Better system to define machine-specific default settings
   - Basis set manipulation

```python
@property
def results(self):
    """Convert an AimsOutChunk to a Results Dictionary"""
    results = {
        "energy": self.energy,
        "free_energy": self.free_energy,
        "forces": self.forces,
        "stress": self.stress,
        "stresses": self.stresses,
        "magmom": self.magmom,
        "dipole": self.dipole,
        "fermi_energy": self.E_f,
        "n_iter": self.n_iter,
        "hirshfeld_charges": self.hirshfeld_charges,
        "hirshfeld_dipole": self.hirshfeld_dipole,
        "hirshfeld_volumes": self.hirshfeld_volumes,
        "hirshfeld_atomic_dipoles": self.hirshfeld_atomic_dipoles,
        "eigenvalues": self.eigenvalues,
        "occupancies": self.occupancies,
        "dielectric_tensor": self.dielectric_tensor,
        "polarization": self.polarization,
    }
```

ASE Configure File

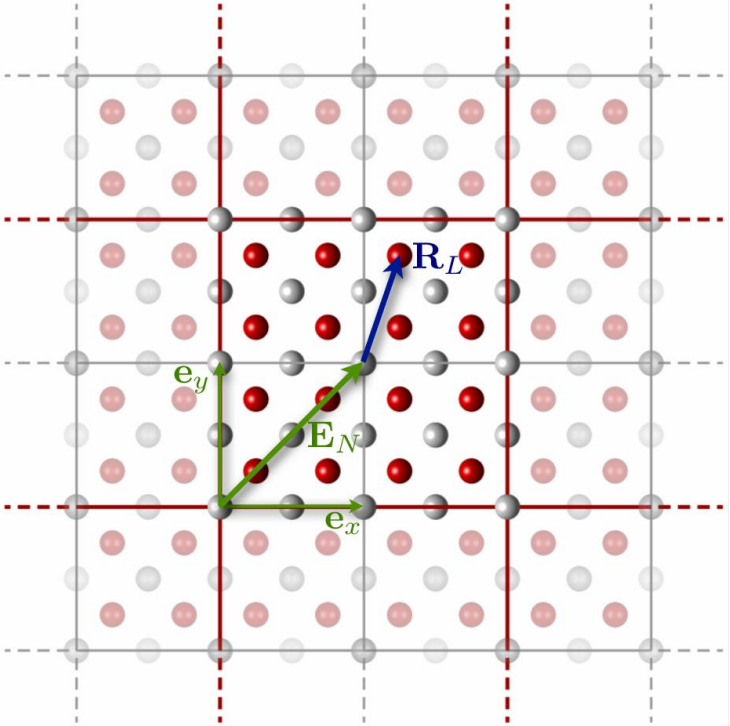
- Stored in: `~/.config/ase/config.ini`

```
[aims]
argv = /path/to/aims/aims.x
```

- Format will likely change upon release
  - I will make `species_dir` stored here
  - MPI calls still not fully defined
- FHI-aims version key?
  - Access the version of FHI-aims without having to run a full calculation
  - Needed for the configuration and profiles
- Will announce on Slack once this is merged in

How to Incorporate ASE into Packages

FHI-vibes: Modelling Vibrational Properties at all Levels of Anharmonicity



Complete Vibrational Properties with FHI-aims

1. Single point calculations
2. Phonopy
3. Phono3py calculations
4. Harmonic sampling
5. Molecular dynamics
6. Postprocessing tools for all steps

Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

FHI-vibes: Modelling Vibrational Properties at all Levels of Anharmonicity

```
[files]
geometry:                       geometry.in
[calculator]
name:                           aims
socketio:                       true
[calculator.parameters]
xc:                             pw-lda
compute_forces:                 true
[calculator.kpoints]
density:                        3
[calculator.basissets]
Si:                             light
```


Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

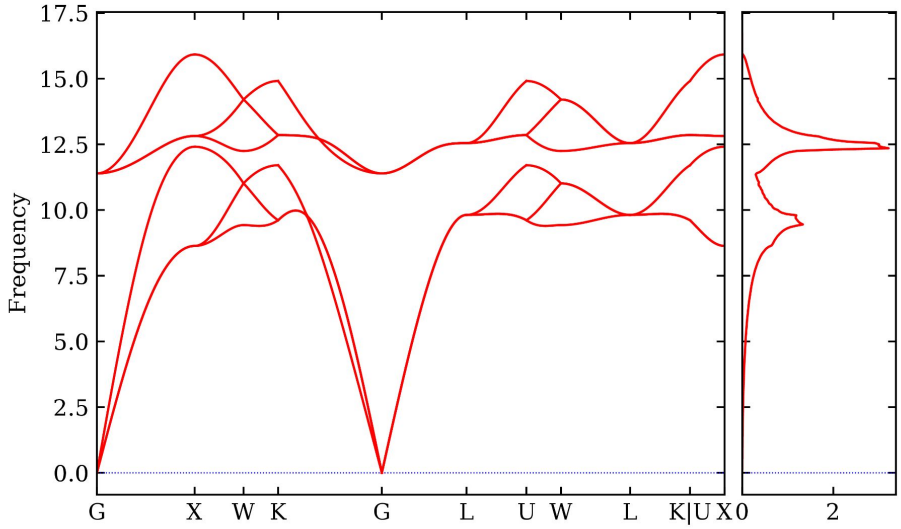
Calculating Phonon Properties with FHI-vibes

```
...
[phonopy]
supercell_matrix:               [1, 1, 1]
displacement:                   0.01
is_diagonal:                    False
is_plusminus:                   auto
symprec:                        1e-05
q_mesh:                         [45, 45, 45]
workdir:                        phonopy
```


Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

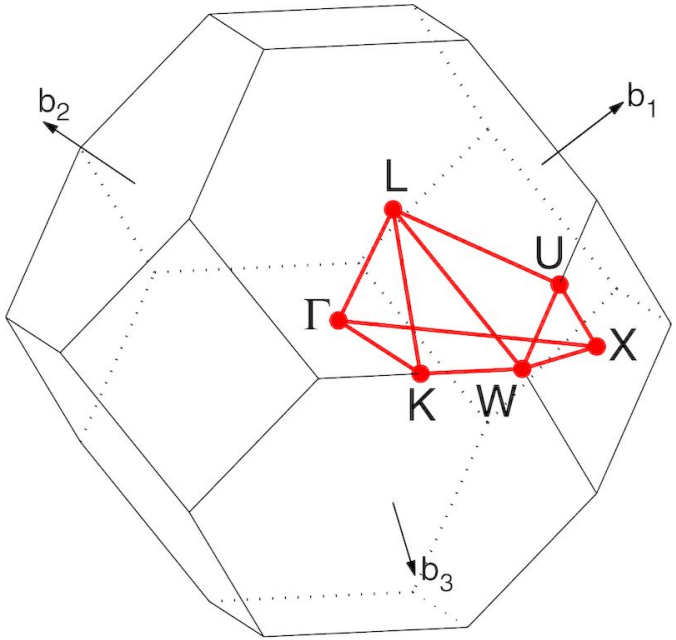
Calculating Phonon Properties with FHI-vibes

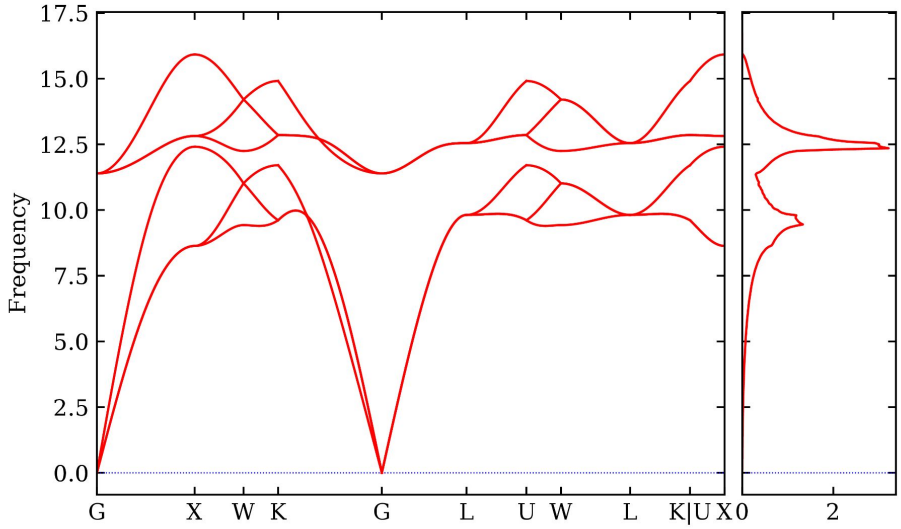
```
...
[phonopy]
supercell_matrix:                [1, 1, 1]
displacement:                    0.01
is_diagonal:                     False
is_plusminus:                    auto
symprec:                         1e-05
q_mesh:                          [45, 45, 45]
workdir:                         phonopy
```



Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

Calculating Phonon Properties with FHI-vibes



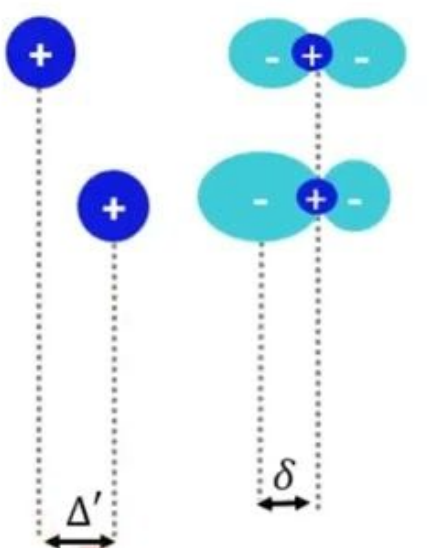


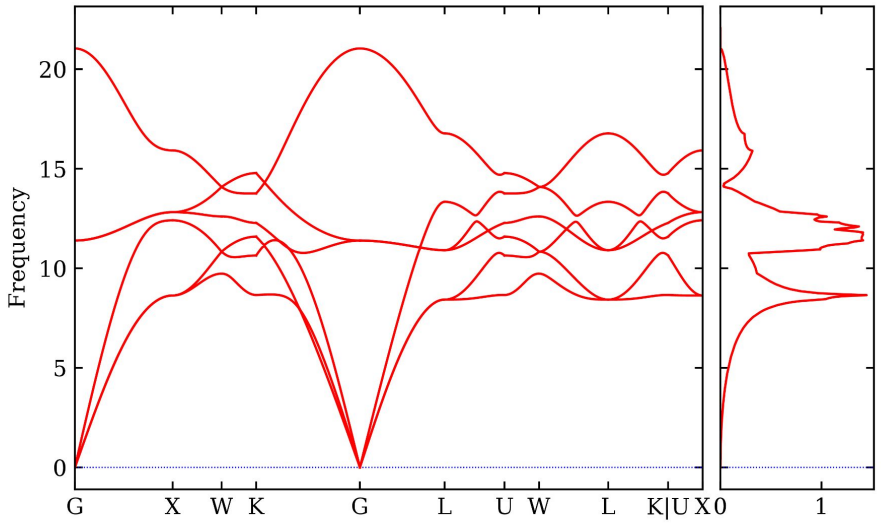
FCC path: Γ-X-W-K-Γ-L-U-W-L-K|U-X

[Setyawan & Curtarolo, DOI: 10.1016/j.commatsci.2010.05.010]

Setyawan, W.; Curtarolo, S. *Com. Mat. Sci.* **49**. 299 (2010)
Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

Born Effective Charges to correct the bandstructure

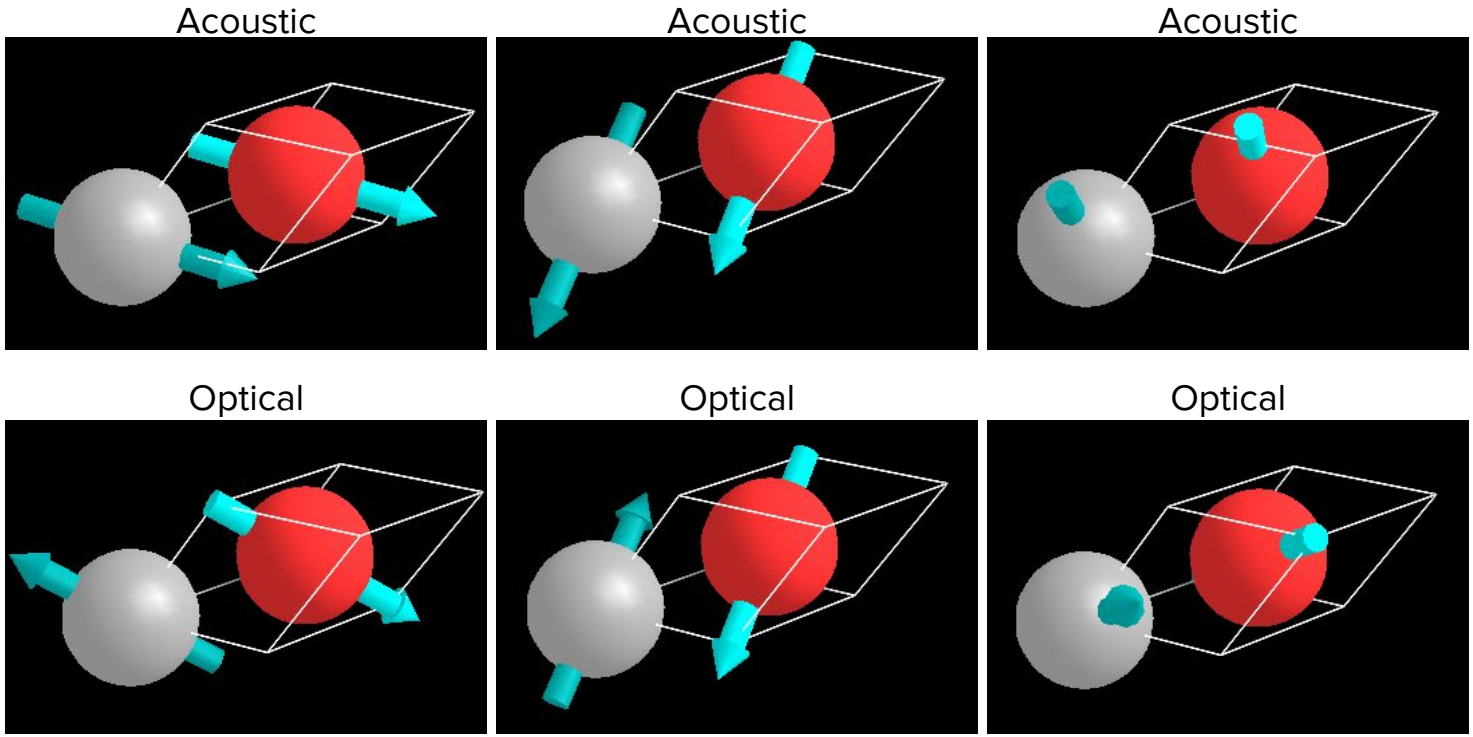




Script in FHI-aims `utilities/BEC.py`

Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

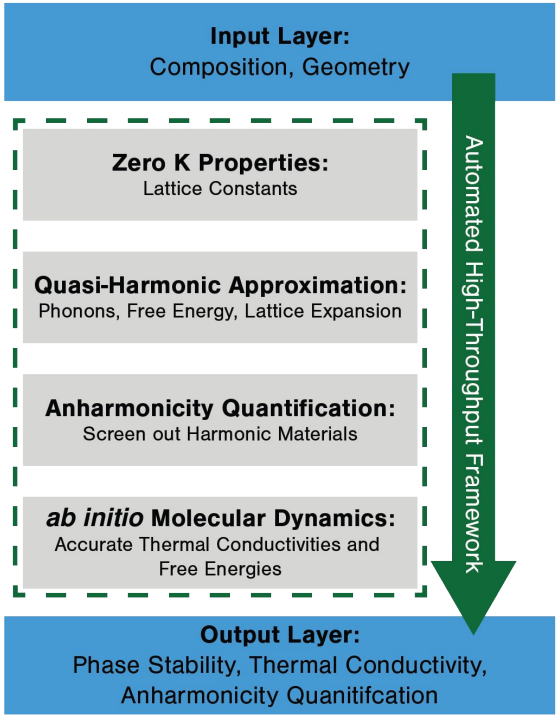
Visualizing the Vibrational Modes

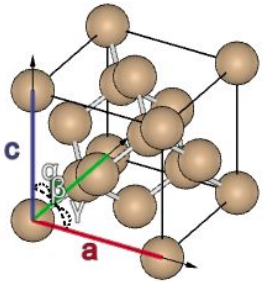


Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

FHI-aims Workflow Developments

Initiate the Workflow



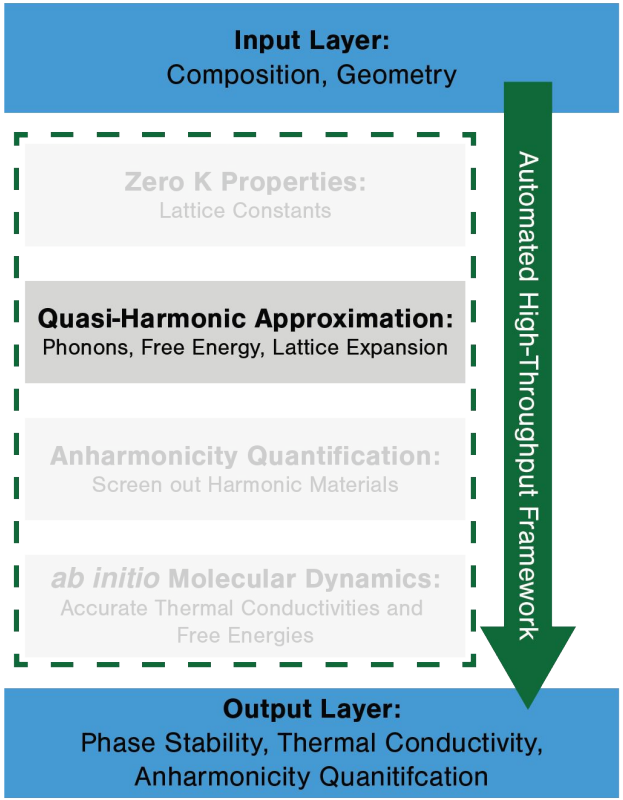


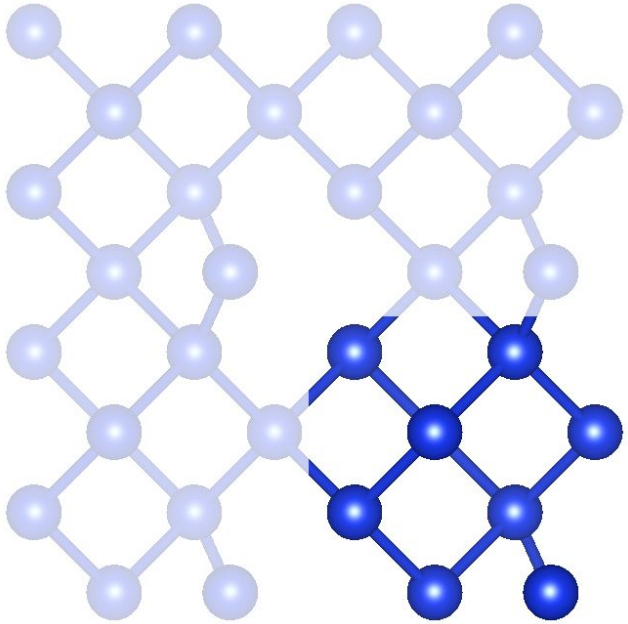

```
{
    "cell": [
        [0.000, 2.703, 2.703],
        [2.703, 0.000, 2.703],
        [2.703, 2.703, 0.000]
    ],
    "positions":[
        [0.67575, 0.67575, 0.67575],
        [4.73025, 4.73025, 4.73025]
     ],
    "symbols": ["Si", "Si"],
    "masses": [28.0855, 28.0855],
    "info": {},
    "constraints": [...]
}
```


```
[calculator]
...
[relaxation]
...
[phonopy]
...
[statistical_sampling]
...
```


Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

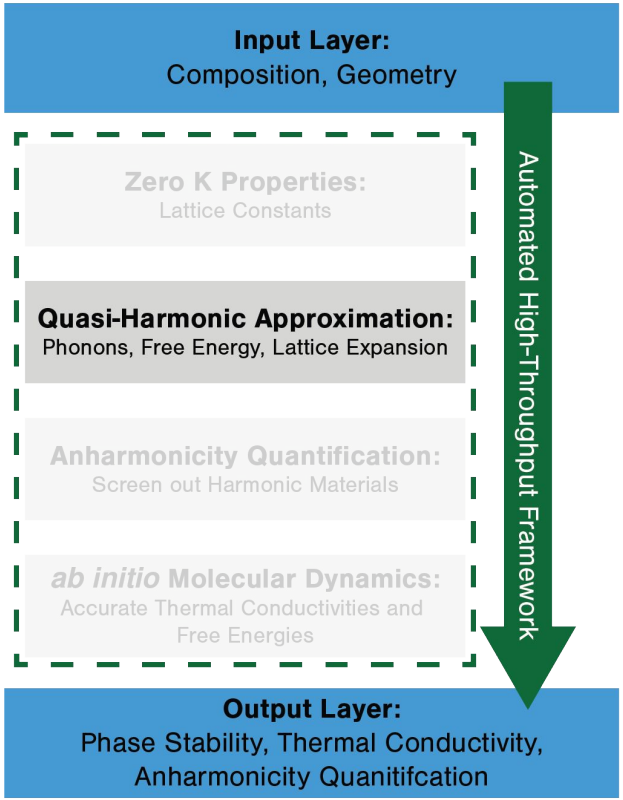
Building a Harmonic Model with Phonopy

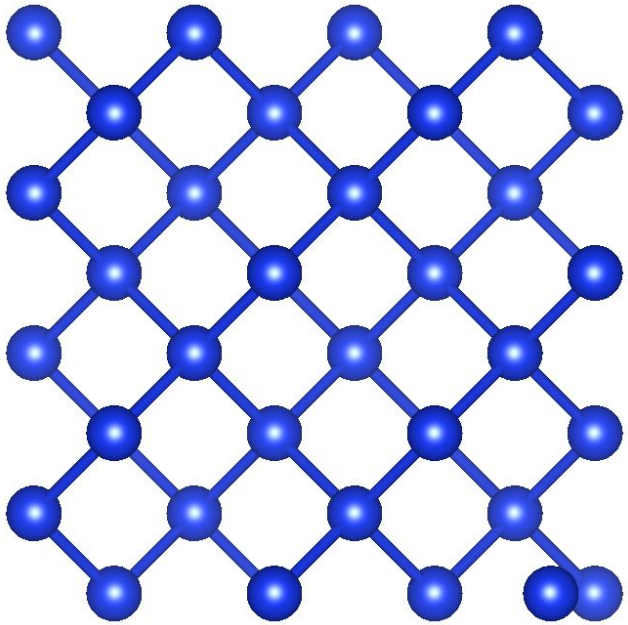




Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

Building a Harmonic Model with Phonopy

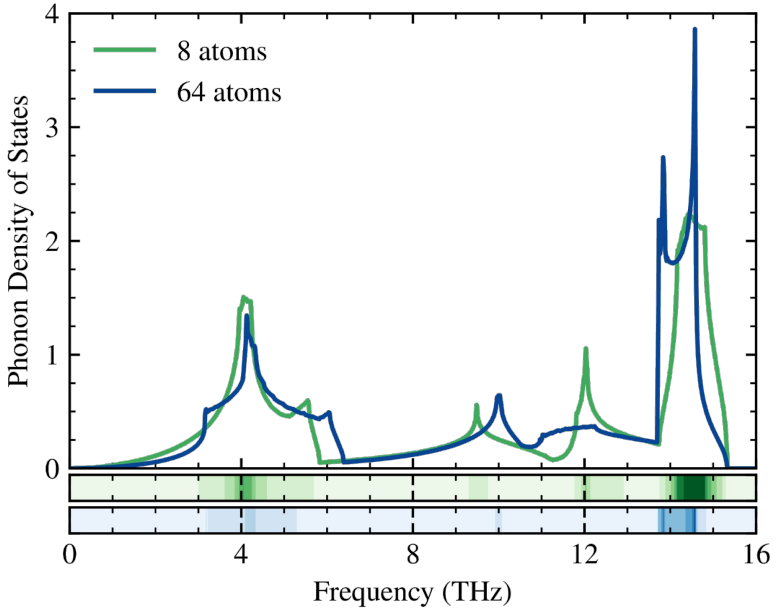


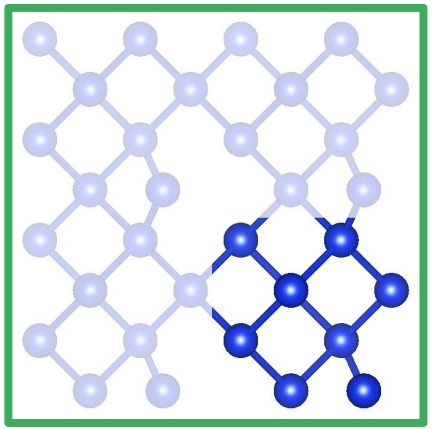


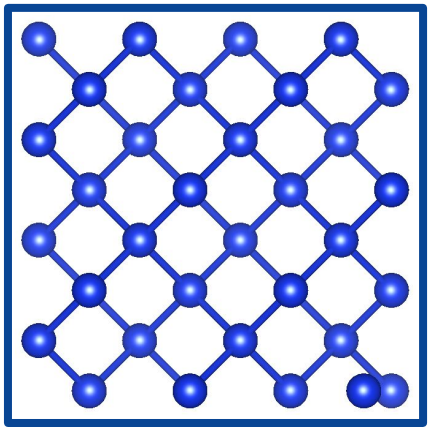
Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

How to Determine if the Model is Converged

$$f(A,B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$





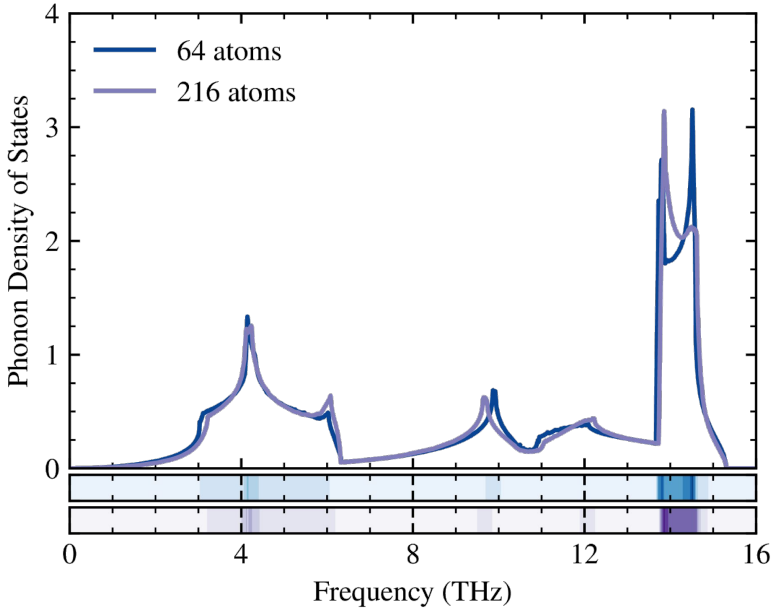


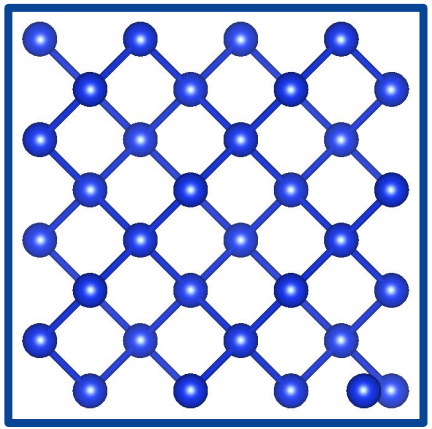
Isayev, O.; *et al. Chem. Mater.* 2015, 27, 3, 735–743

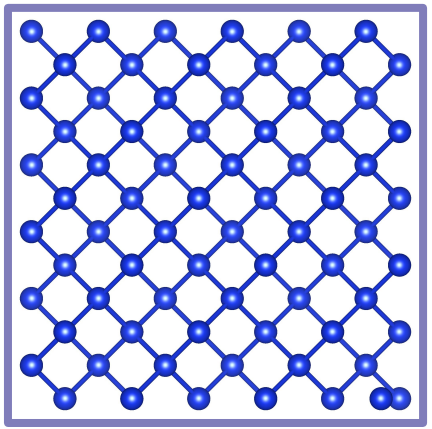
Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

How to Determine if the Model is Converged

$$f(A,B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$



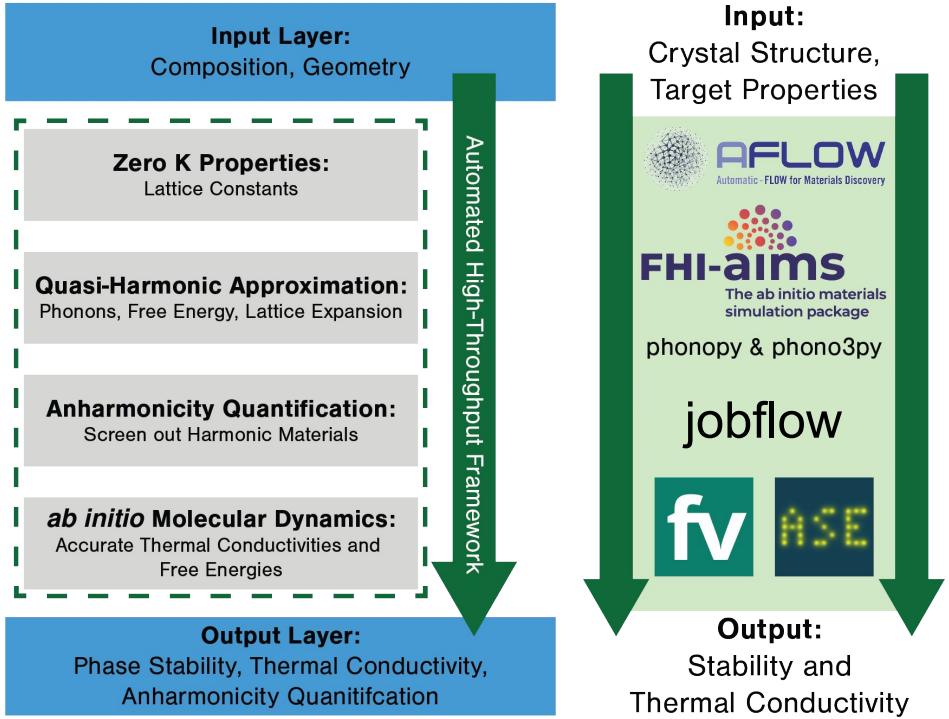




Isayev, O.; *et al. Chem. Mater.* 2015, 27, 3, 735–743

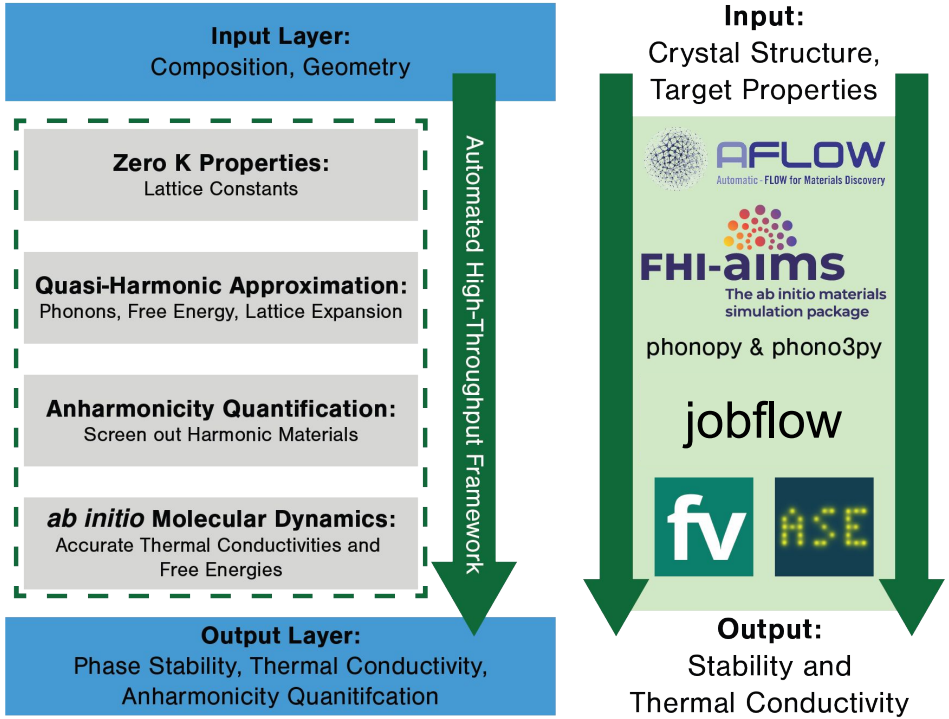
Knoop, F.; Purcell, T. A. R.; Scheffler, M. Carbogno, C. *J. Open Source Softw.* **5**, 2671. (2020)

Incorporating the Workflows into an Atomate2 Style Manager



https://github.com/materialsproject/jobflow

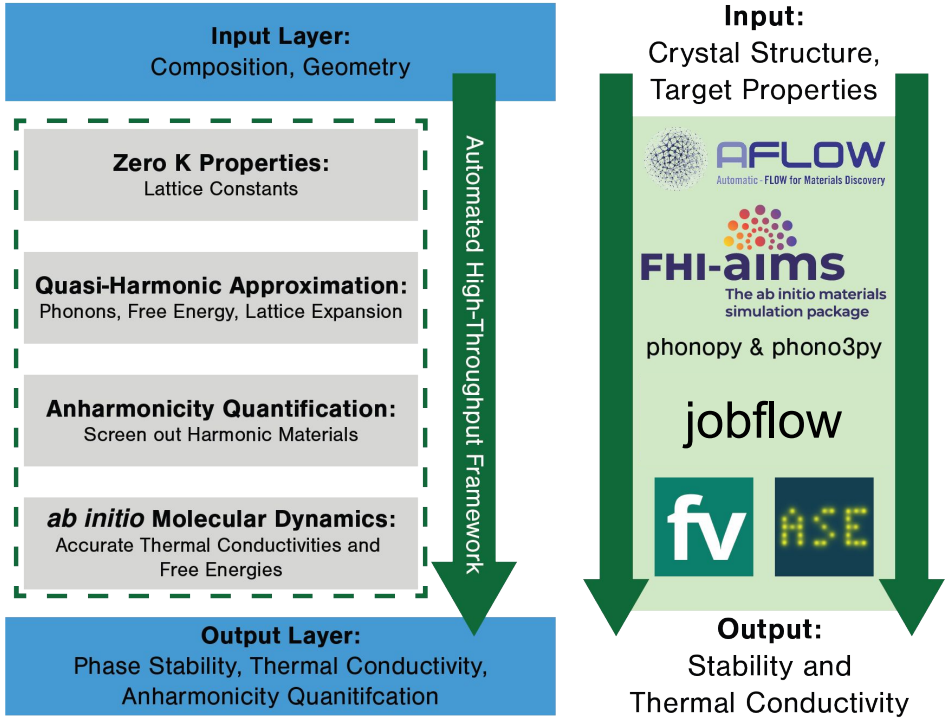
Incorporating the Workflows into an Atomate2 Style Manager



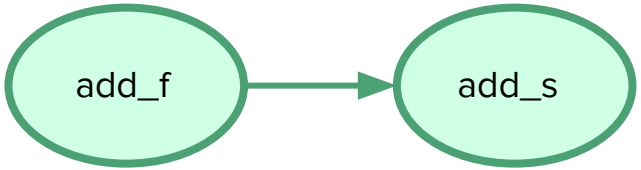

```python
from jobflow import job, Flow

@job
def add(a, b):
    return a + b

add_first = add(1, 5)
add_second = add(add_first.output, 5)

flow = Flow([add_first, add_second])
flow.draw_graph().show()
```


https://materialsproject.github.io/jobflow/tutorials/2-introduction.html

Incorporating the Workflows into an Atomate2 Style Manager




```python
from jobflow import job, Flow

@job
def add(a, b):
    return a + b

add_first = add(1, 5)
add_second = add(add_first.output, 5)

flow = Flow([add_first, add_second])
flow.draw_graph().show()
```



https://materialsproject.github.io/jobflow/tutorials/2-introduction.html

Accessing the Codes

FHI-vibes	ASE	jobflow
https://tinyurl.com/fhivibes	https://gitlab.com/ase/ase	https://github.com/materialsproject/jobflow

Questions? Suggestions? Requests?