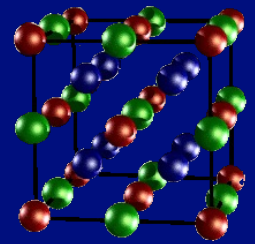
Los Alamos NATIONAL LABORATORY

EXASCALE COMPUTING PROJECT

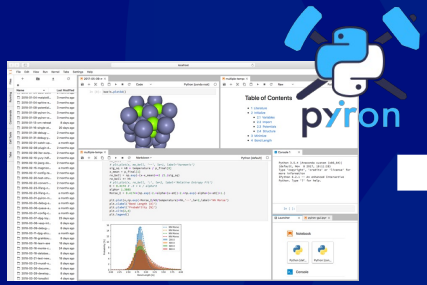
pyiron – Rapid-prototyping and Up-scaling Workflows for the Exascale

Jan Janssen and Danny Perez

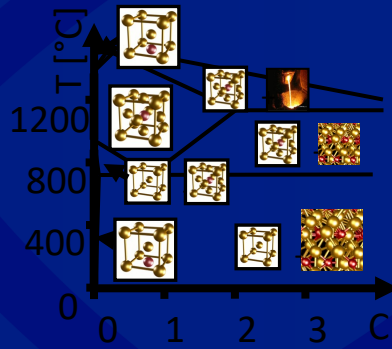
05/01/2023



Atomic Structure

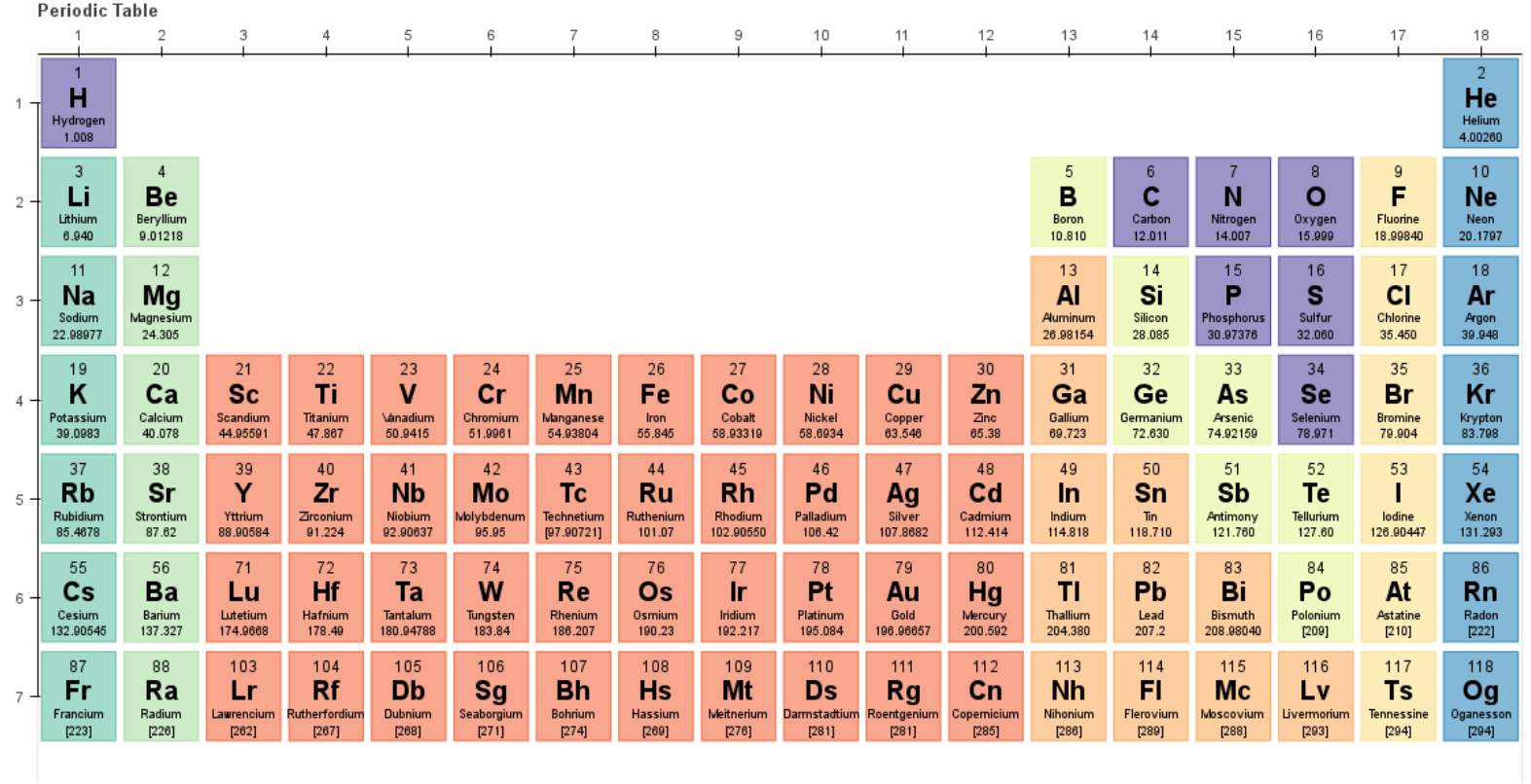


Autonomous Workflows



Material Properties

Materials Science

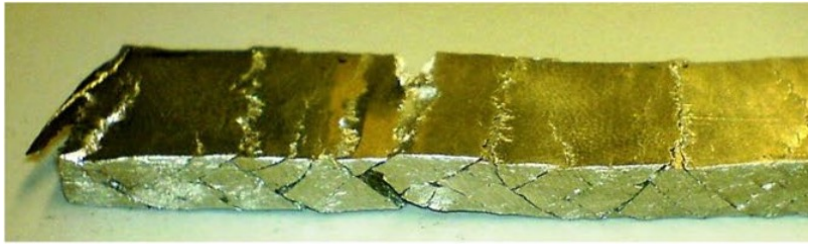


Materials Science

Periodic Table

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1 H Hydrogen 1.008																	2 He Helium 4.00260
3 Li Lithium 6.940	4 Be Beryllium 9.01218											5 B Boron 10.810	6 C Carbon 12.011	7 N Nitrogen 14.007	8 O Oxygen 15.999	9 F Fluorine 18.99840	10 Ne Neon 20.1797
11 Na Sodium 22.98977	12 Mg Magnesium 24.305											13 Al Aluminum 26.98154	14 Si Silicon 28.085	15 P Phosphorus 30.97376	16 S Sulfur 32.060	17 Cl Chlorine 35.450	18 Ar Argon 39.948
19 K Potassium 39.0983	20 Ca Calcium 40.078											31 Ga Gallium 69.723	32 Ge Germanium 72.630	33 As Arsenic 74.92159	34 Se Selenium 78.971	35 Br Bromine 79.904	36 Kr Krypton 83.798
37 Rb Rubidium 85.4678	38 Sr Strontium 87.62	39 Y Yttrium 88.90584	40 Zr Zirconium 91.224	41 Nb Niobium 92.90637	42 Mo Molybdenum 95.95	43 Tc Technetium [97.90721]	44 Ru Ruthenium 101.07	45 Rh Rhodium 102.90550	46 Pd Palladium 106.42	47 Ag Silver 107.8682	48 Cd Cadmium 112.414	49 In Indium 114.818	50 Sn Tin 118.710	51 Sb Antimony 121.760	52 Te Tellurium 127.60	53 I Iodine 126.90447	54 Xe Xenon 131.293
55 Cs Cesium 132.90545	56 Ba Barium 137.327	71 Lu Lutetium 174.9668	72 Hf Hafnium 178.49	73 Ta Tantalum 180.94788	74 W Tungsten 183.84	75 Re Rhenium 186.207	76 Os Osmium 190.23	77 Ir Iridium 192.217	78 Pt Platinum 195.084	79 Au Gold 196.96657	80 Hg Mercury 200.592	81 Tl Thallium 204.380	82 Pb Lead 207.2	83 Bi Bismuth 208.98040	84 Po Polonium [209]	85 At Astatine [210]	86 Rn Radon [222]
87 Fr Francium [223]	88 Ra Radium [226]	103 Lr Lawrencium [262]	104 Rf Rutherfordium [267]	105 Db Dubnium [268]	106 Sg Seaborgium [271]	107 Bh Bohrium [274]	108 Hs Hassium [269]	109 Mt Meitnerium [276]	110 Ds Darmstadtium [281]	111 Rg Roentgenium [281]	112 Cn Copernicium [285]	113 Nh Nihonium [286]	114 Fl Flerovium [289]	115 Mc Moscovium [288]	116 Lv Livermorium [293]	117 Ts Tennessine [294]	118 Og Oganesson [294]

Pure Magnesium is brittle

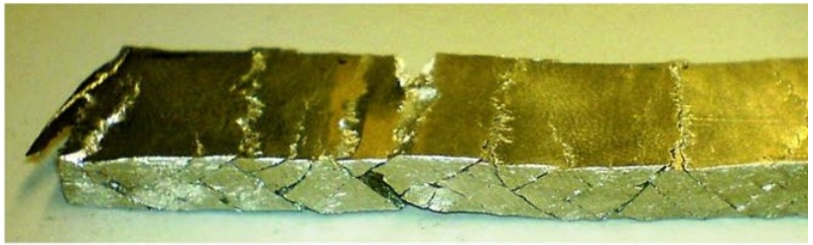


10% Compression (CR)

Materials Science

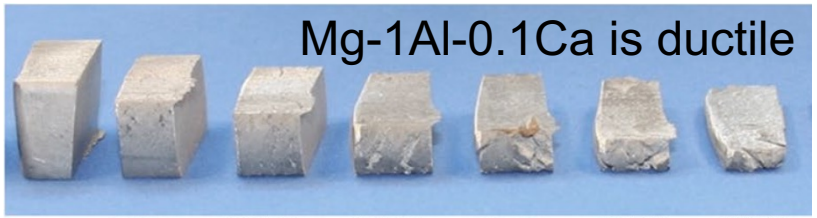
Periodic Table

Pure Magnesium is brittle



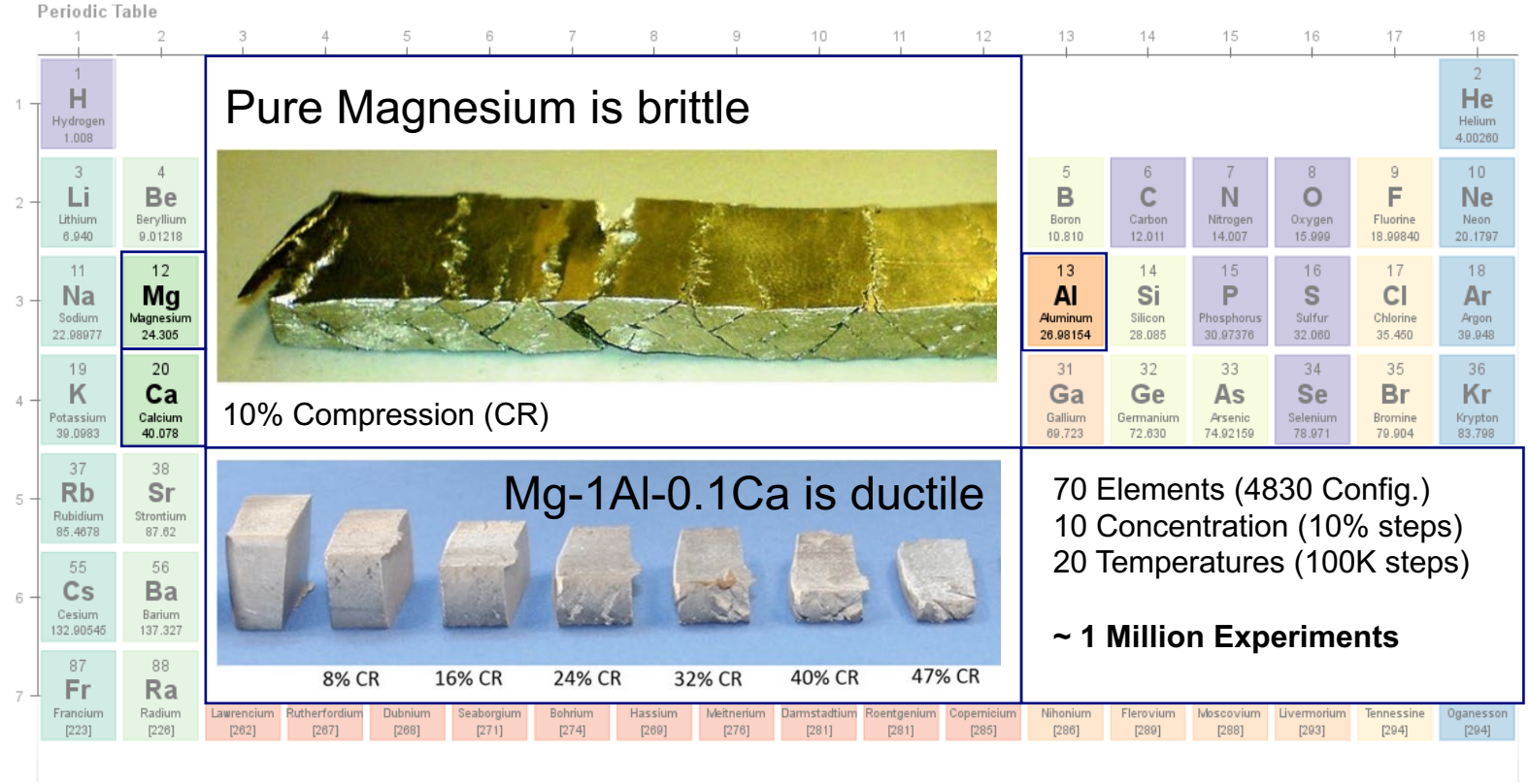
10% Compression (CR)

Mg-1Al-0.1Ca is ductile



S. Sandlöbes, et al. Sci. Rep. 7, 10458 (2017)

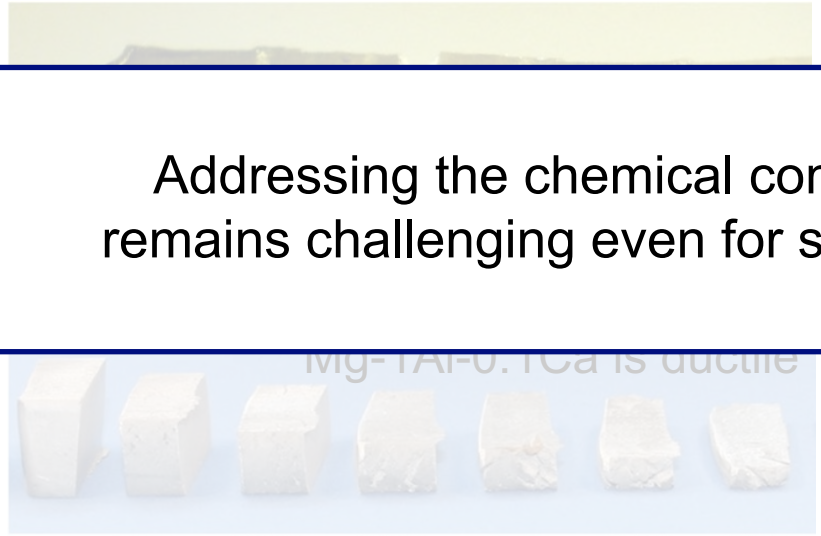
Materials Science



S. Sandlöbes, et al. Sci. Rep. 7, 10458 (2017)

Materials Science

Pure Magnesium is brittle



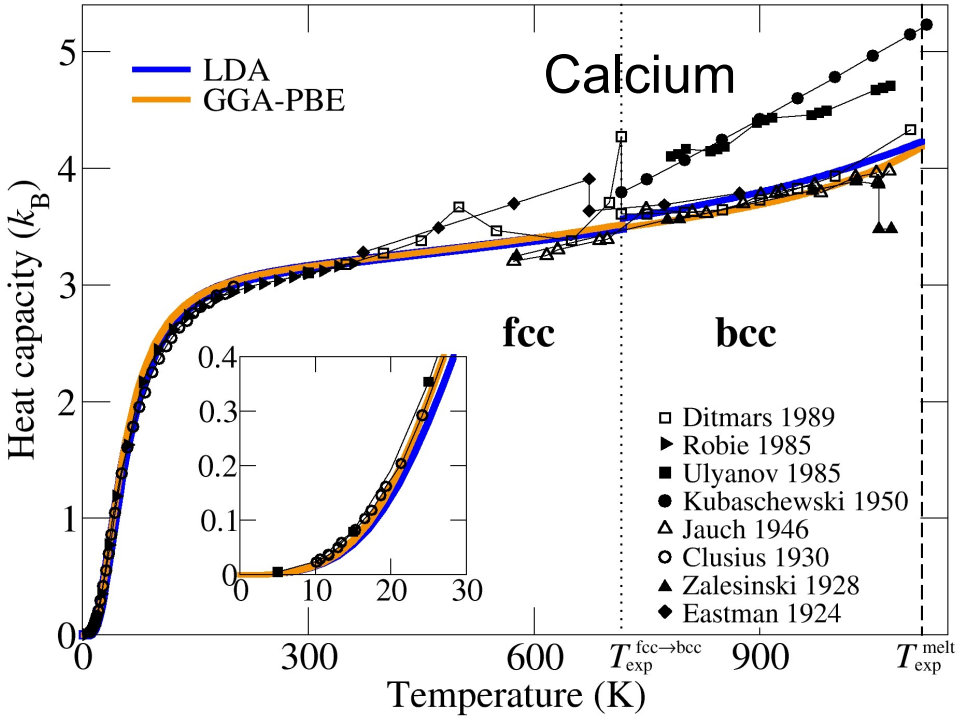
10 Concentration (10% steps)
20 Temperatures (100K steps)

~ 1 Million Experiments

Addressing the chemical complexity
remains challenging even for simulation.

S. Sandlöbes, et al. Sci. Rep. 7, 10458 (2017)

Ab-initio Thermodynamics

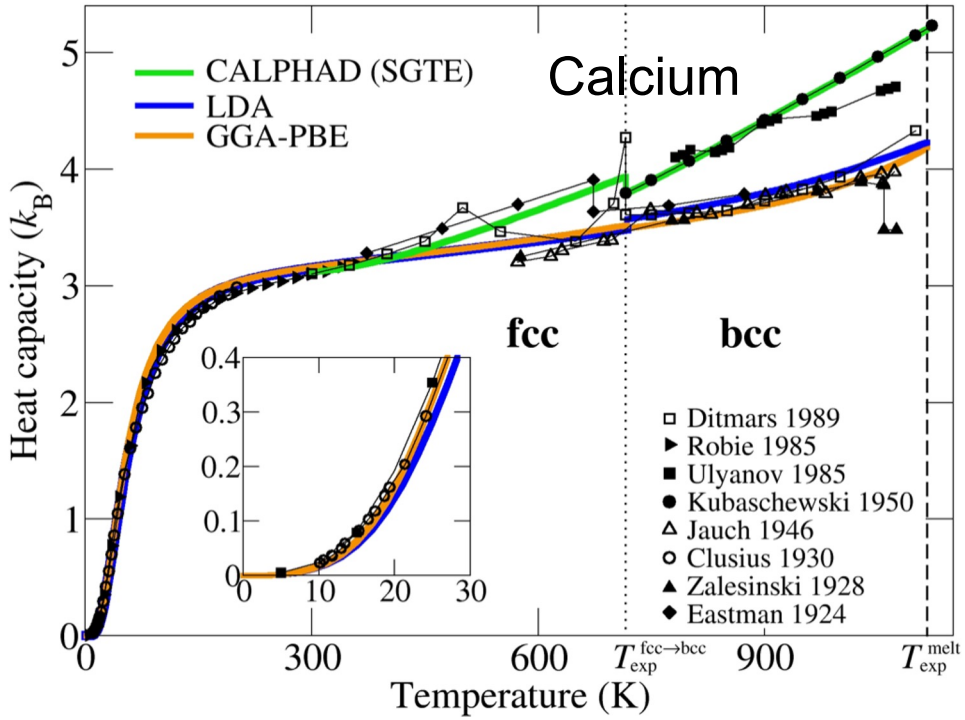


Adiabatic Approach:

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$

B. Grabowski, et al. PRB 84, 214107 (2011)

Ab-initio Thermodynamics

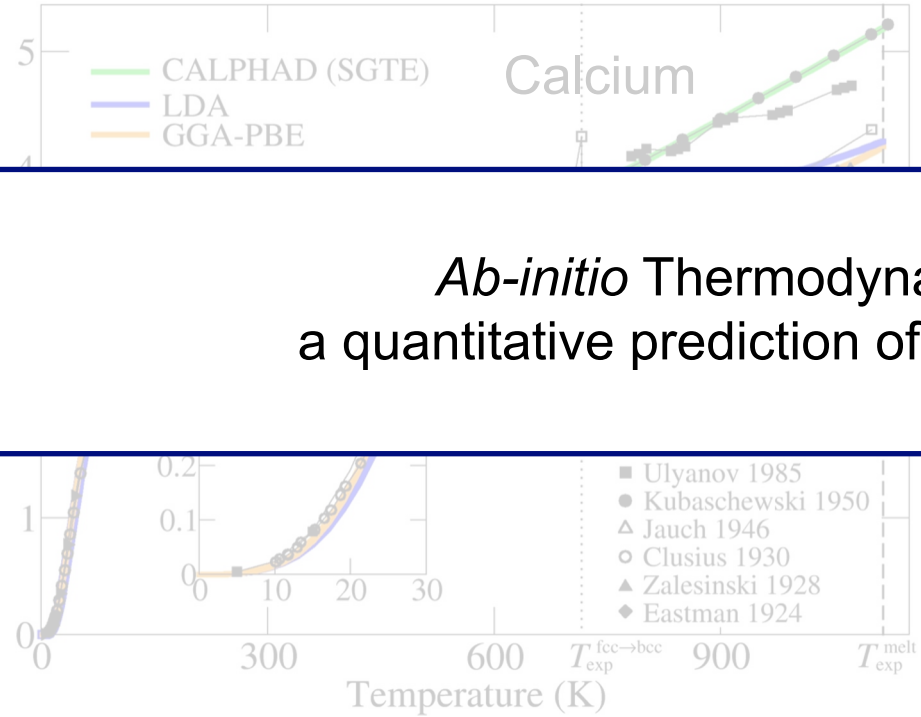


Adiabatic Approach:

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$

B. Grabowski, et al. PRB 84, 214107 (2011)

Ab-initio Thermodynamics



Adiabatic Approximation:

$$F(V,T) = E^{tot}(V) + F^{el}(V,T)$$

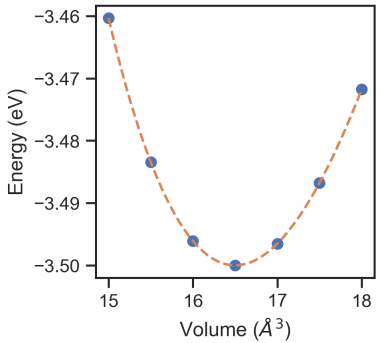
Ab-initio Thermodynamics enables
a quantitative prediction of material properties.

B. Grabowski, et al. PRB 84, 214107 (2011)

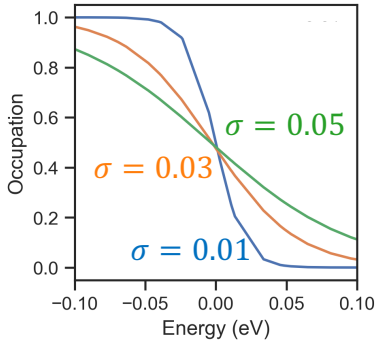
Adiabatic Approach

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$

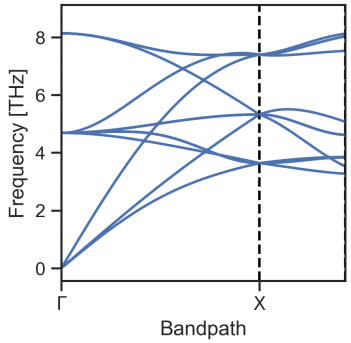
E(V) Curve



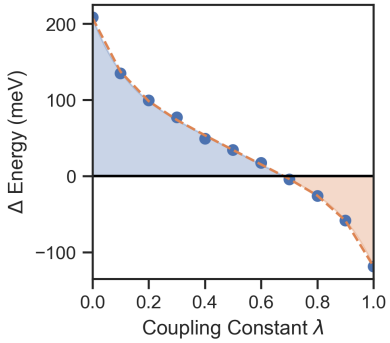
El. Temperature



Phonons



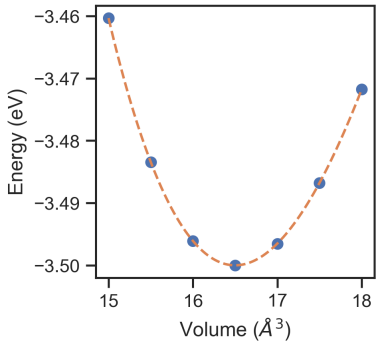
Therm. Integration



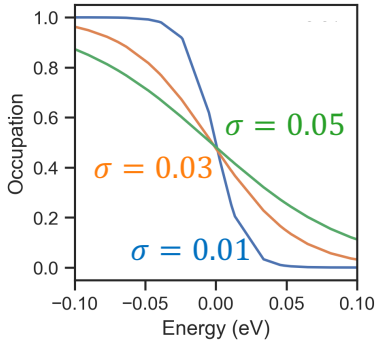
Adiabatic Approach

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$

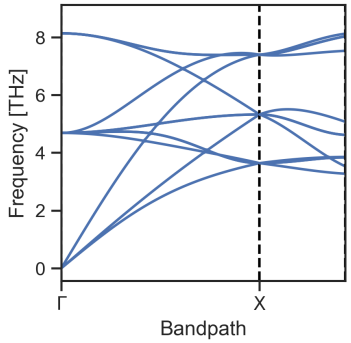
E(V) Curve



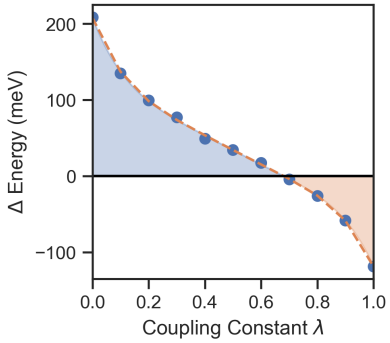
El. Temperature



Phonons



Therm. Integration



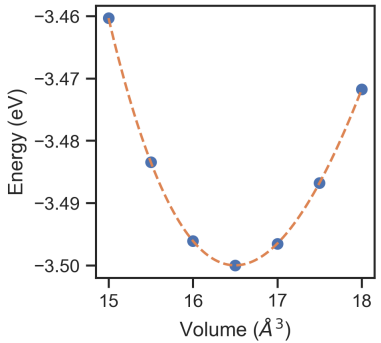
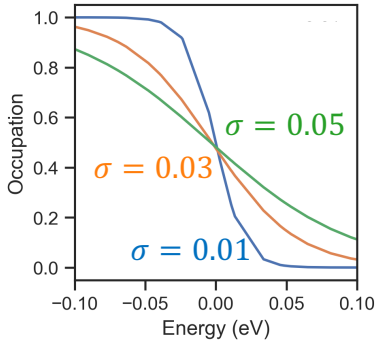
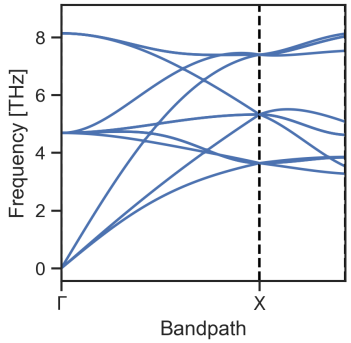
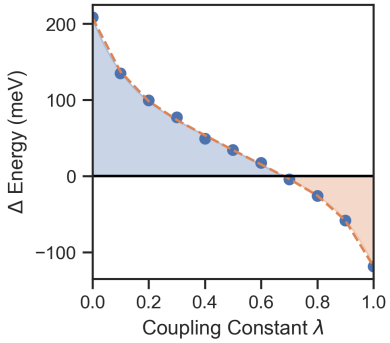
DFT calculation with different volumes and electronic temperatures

Finite Displacements

MD, Fit Potential, Integration

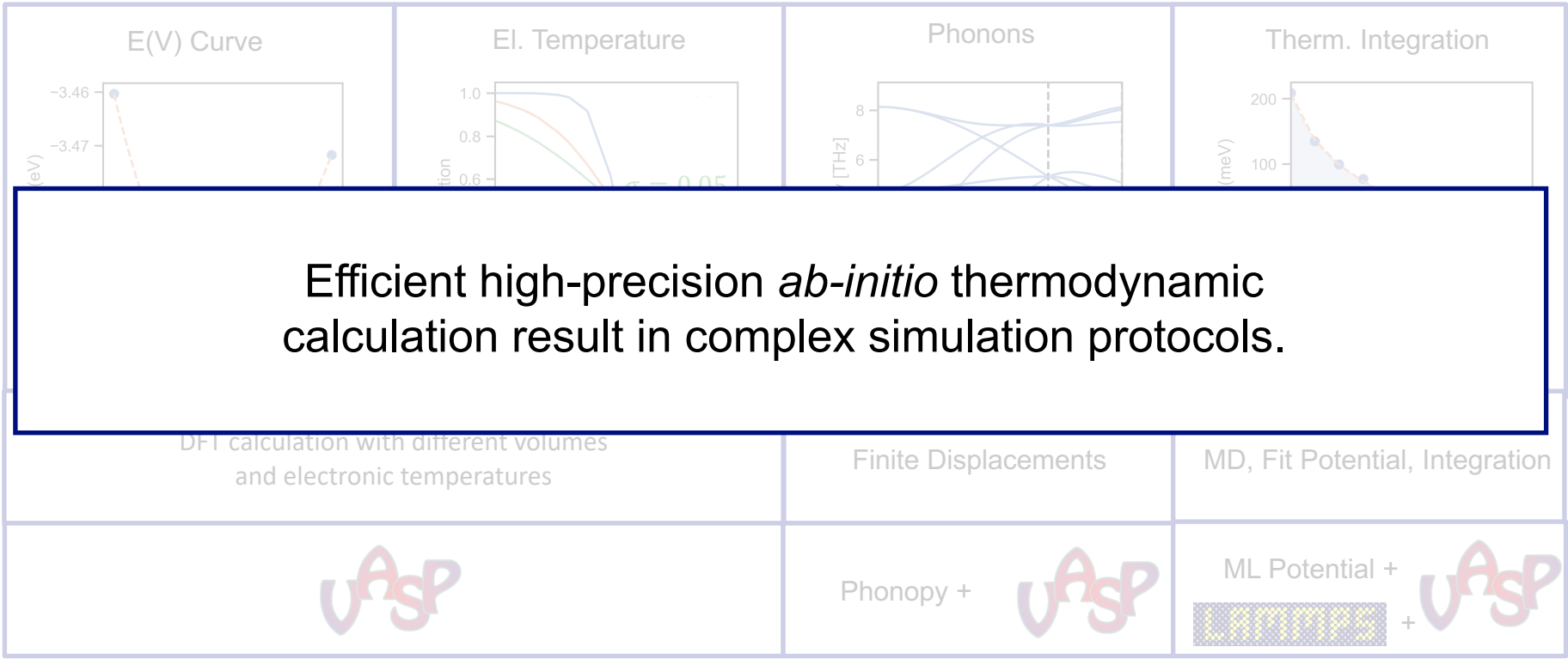
Adiabatic Approach

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$

E(V) Curve	El. Temperature	Phonons	Therm. Integration
			
DFT calculation with different volumes and electronic temperatures		Finite Displacements	MD, Fit Potential, Integration
VASP		Phonopy + VASP	ML Potential + LAMMPS + VASP

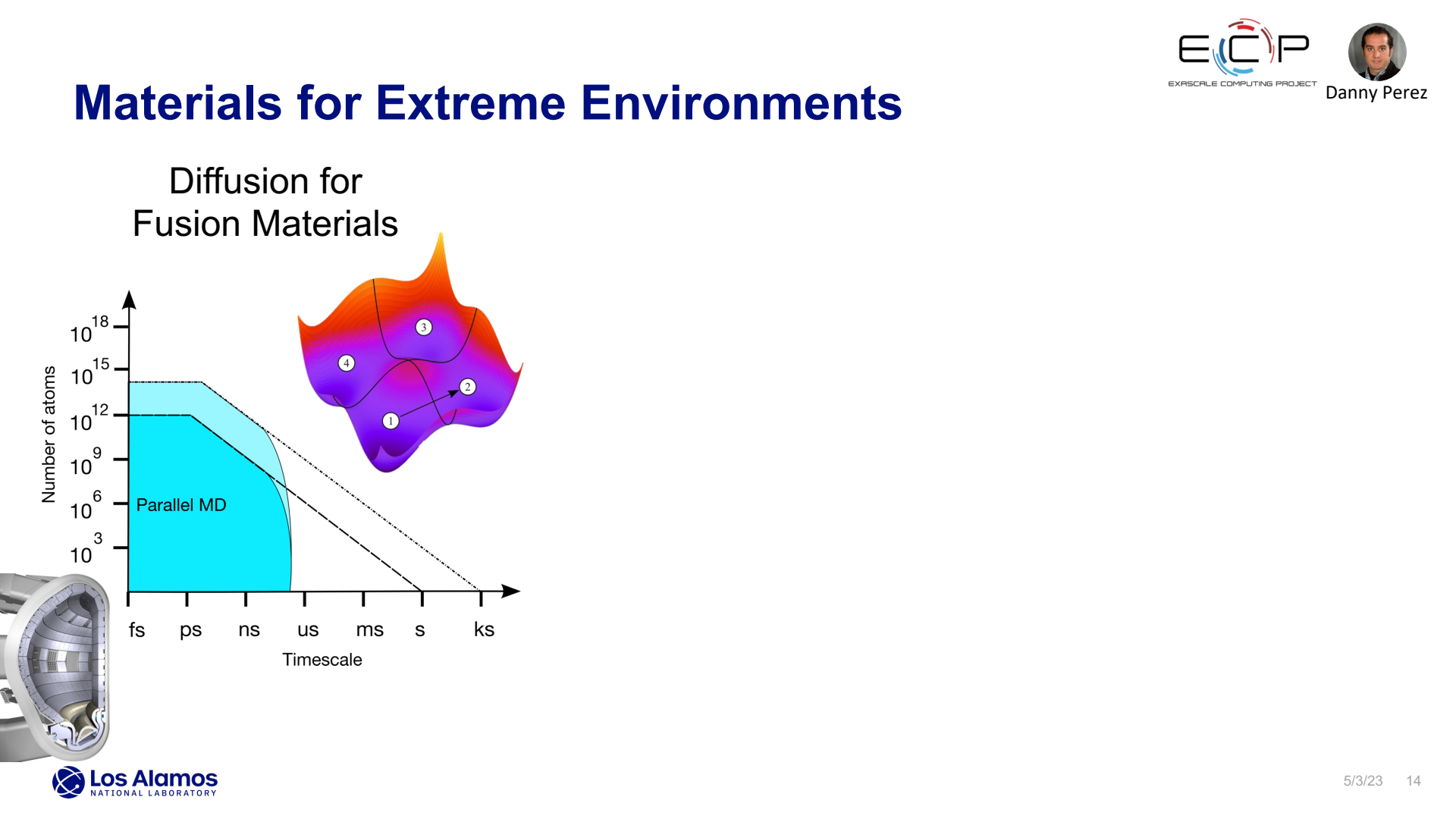
Adiabatic Approach

$$\mathrm{F(V,T)} = E^{tot}(V) + F^{el}(V,T) + F^{qh}(V,T) + F^{ah}(V,T)$$



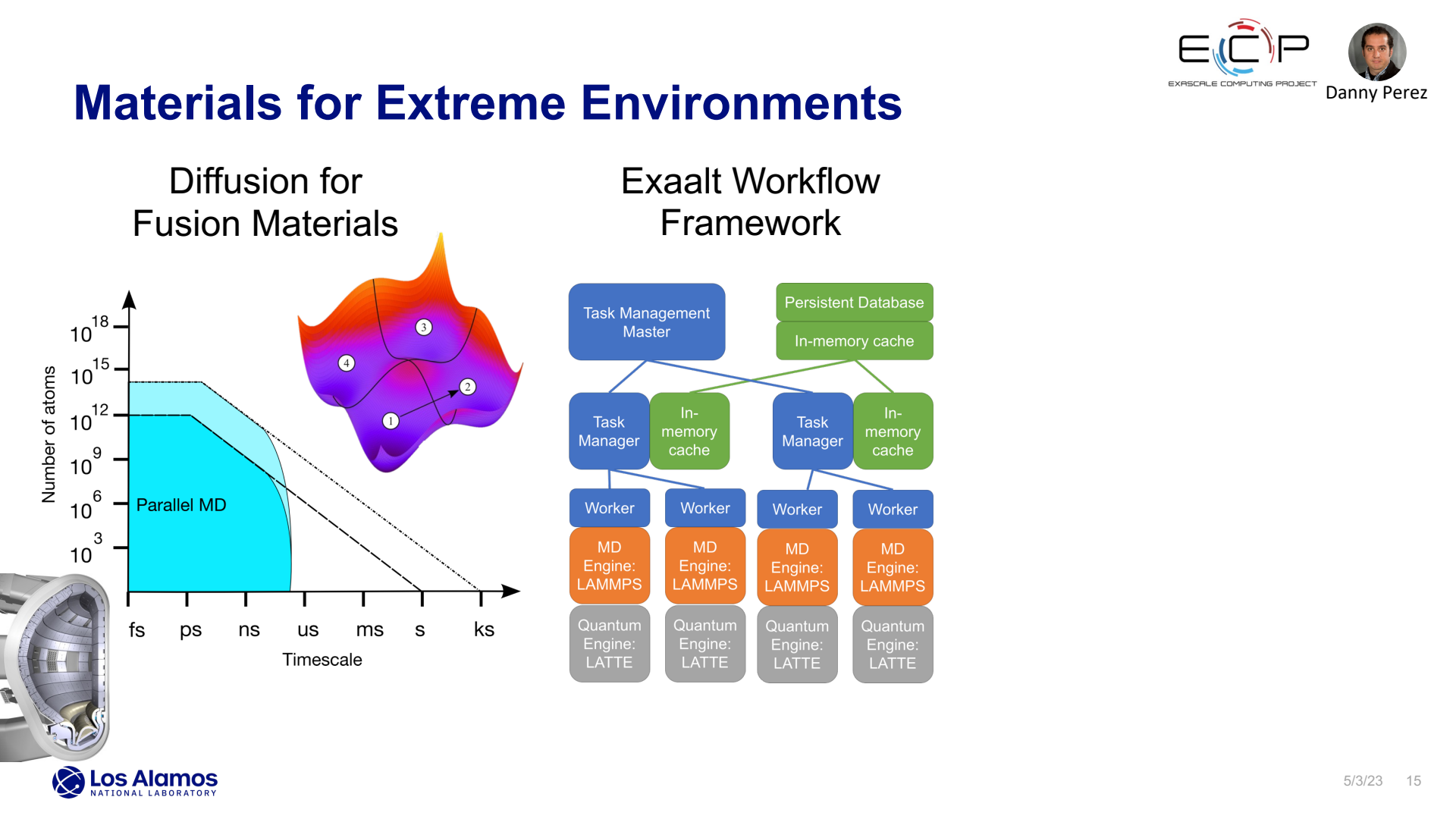
Efficient high-precision *ab-initio* thermodynamic calculation result in complex simulation protocols.

Materials for Extreme Environments



Danny Perez

Materials for Extreme Environments



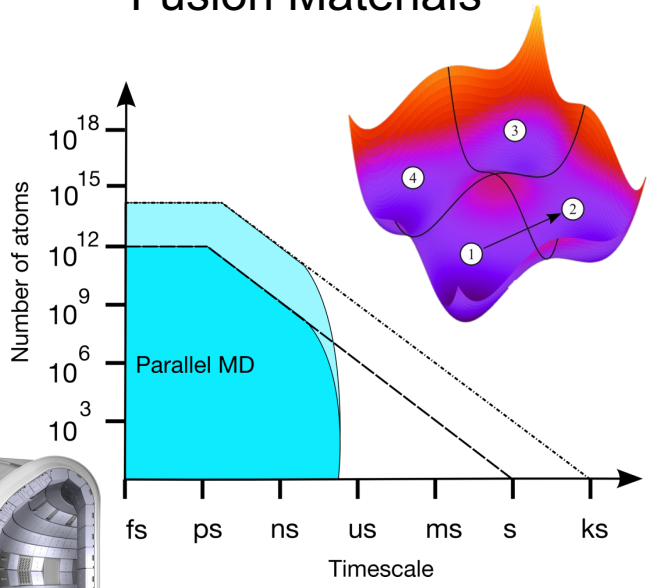
Diffusion for Fusion Materials

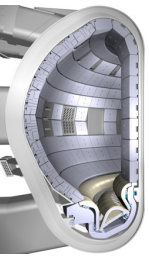
Exaalt Workflow Framework

Materials for Extreme Environments

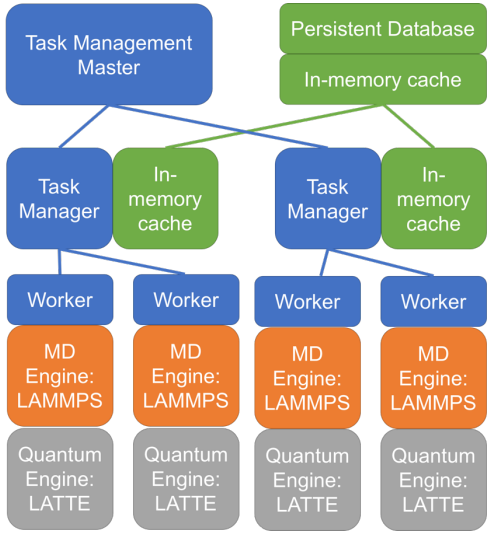
Danny Perez

Diffusion for Fusion Materials

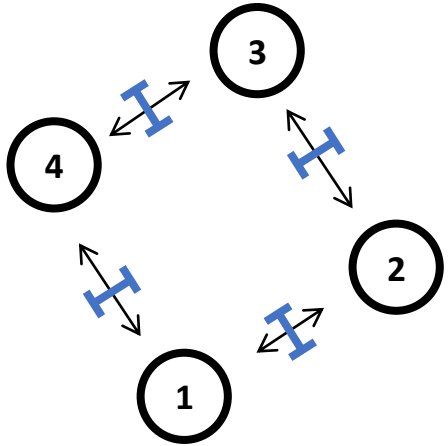




Exaalt Workflow Framework

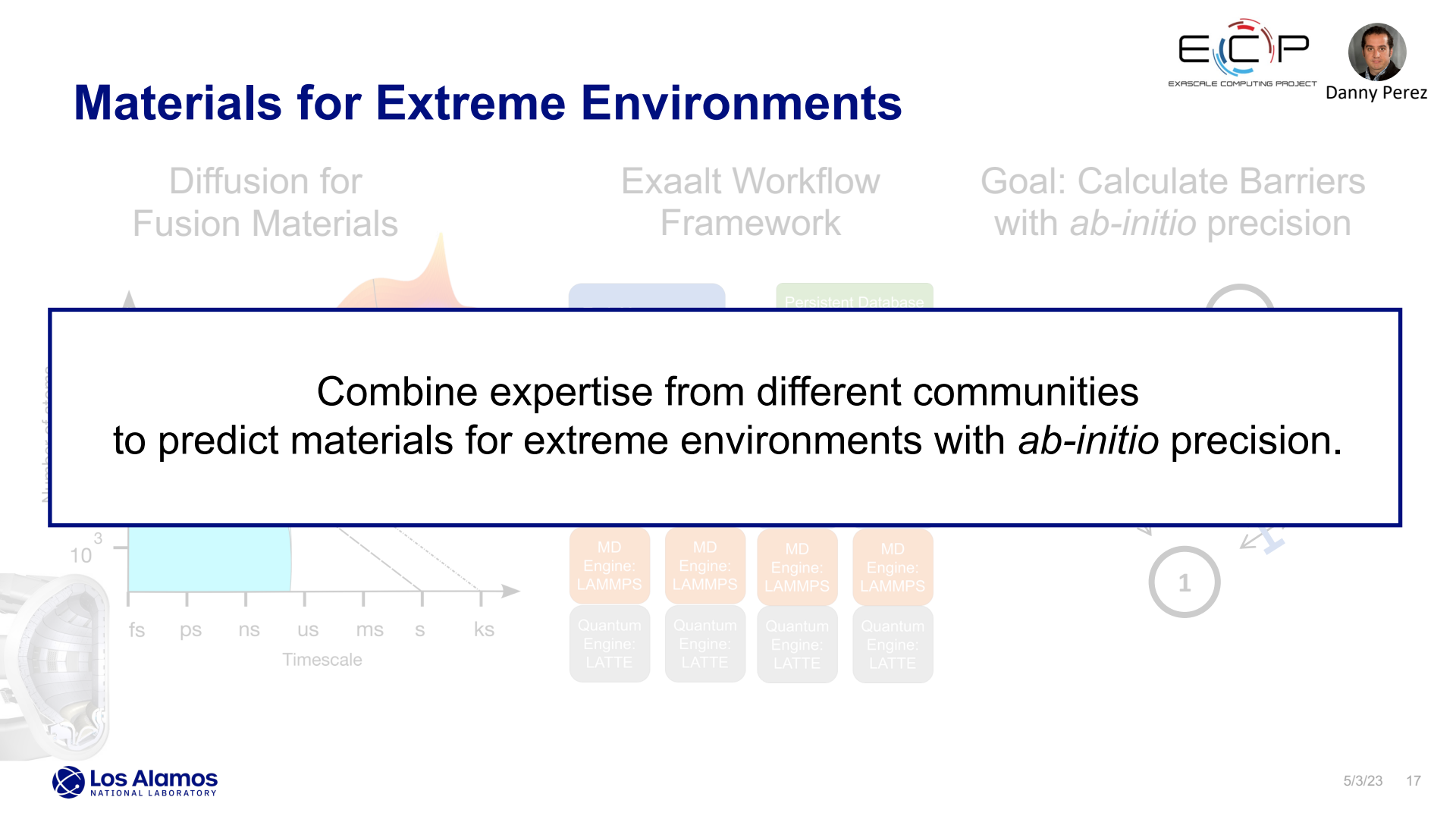


Goal: Calculate Barriers with *ab-initio* precision



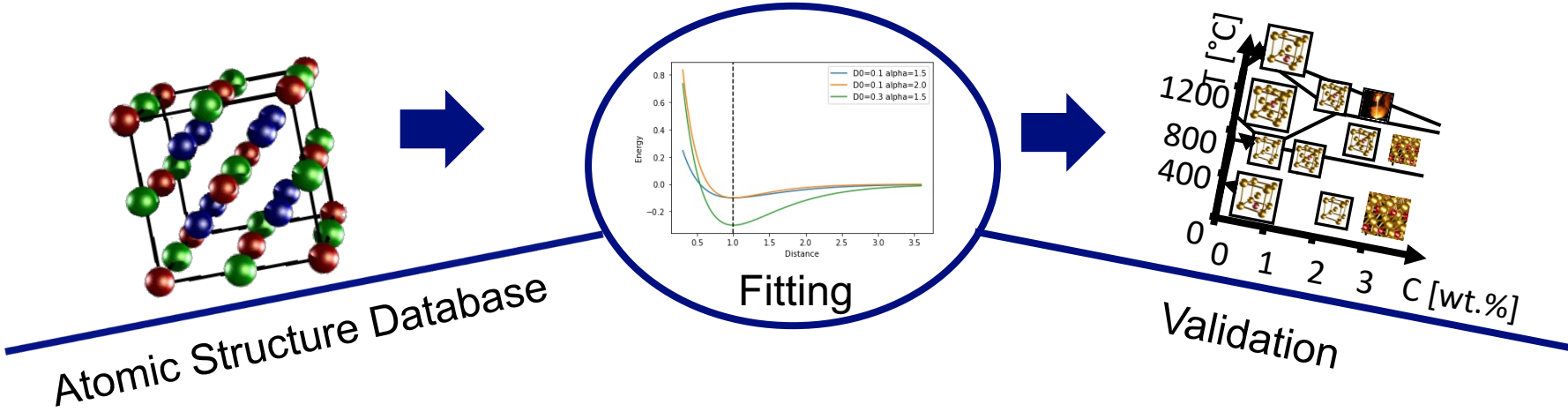
Los Alamos NATIONAL LABORATORY

Materials for Extreme Environments

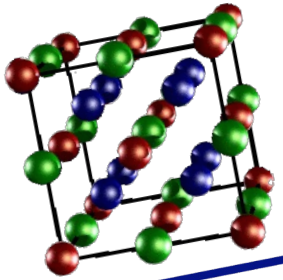


Combine expertise from different communities
to predict materials for extreme environments with *ab-initio* precision.

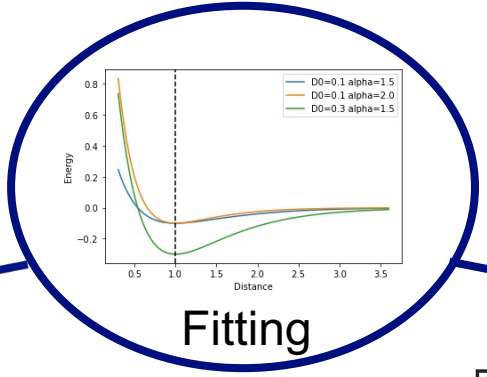
Machine-Learned Interatomic Potentials



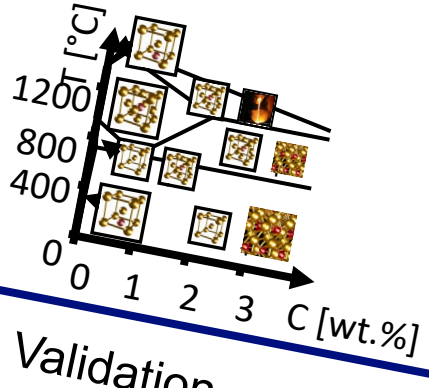
Machine-Learned Interatomic Potentials



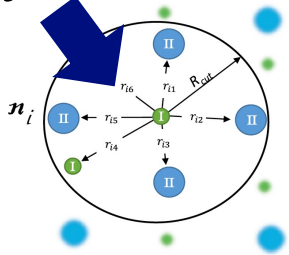
Atomic Structure Database



Fitting



Validation



MTP

SNAP

EAM

ACE

Descriptors for the atomic environment

PyTorch

Neural network

SciPy

Linear regression

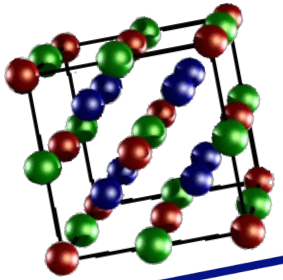
Gaussian process

TensorFlow 2.0

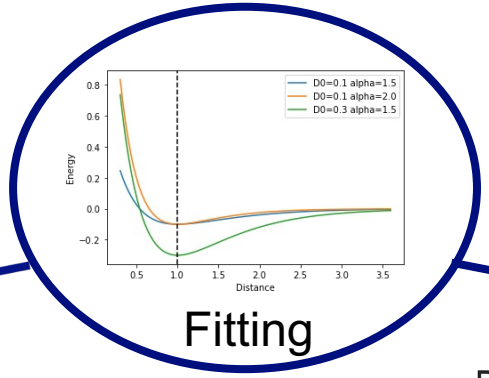
Fitting to energies and forces

Andrew Rohskopf

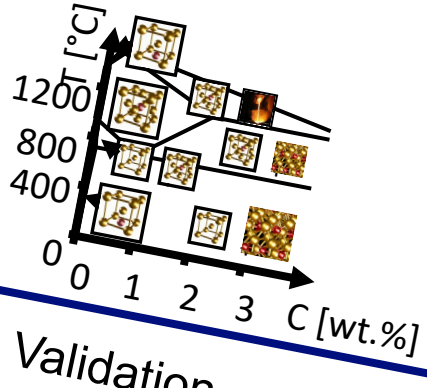
Machine-Learned Interatomic Potentials



Atomic Structure Database



Fitting



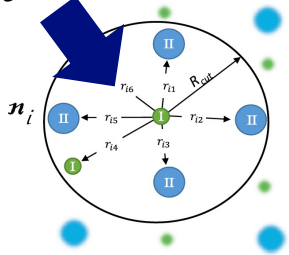
Validation

MTP

SNAP

EAM

ACE



Descriptors for the atomic environment

PyTorch

Neural network

SciPy

Linear regression

Gaussian process

TensorFlow 2.0

Fitting to energies and forces

A. Rohskopf, C. Sievers, N. Lubbers, M.a. Cusentino, J. Goff, **J. Janssen**, et al. JOSS, 8 (2023)

Andrew Rohskopf

Machine-Learned Interatomic Potentials

Challenge: Construct **transferable, computationally efficient** and **precise** potentials for long-time scale simulation.

EAM

ACE

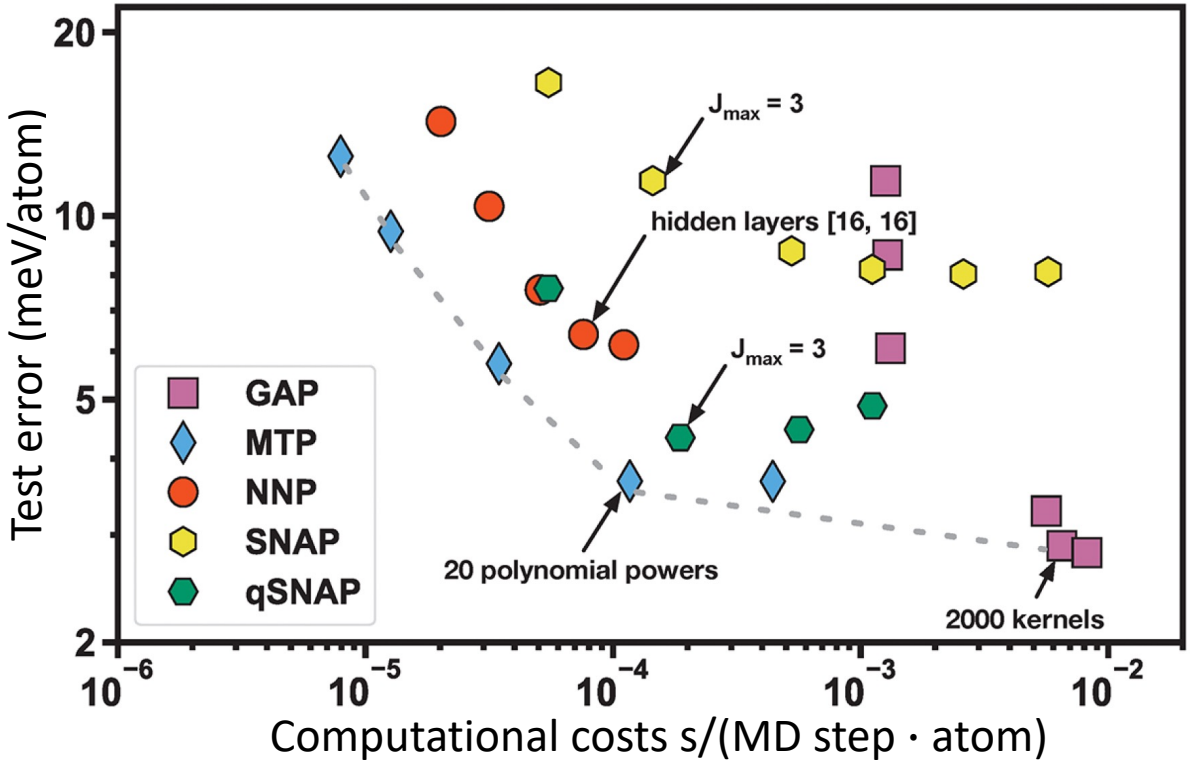
Descriptors for the atomic environment

SNAP

Linear regression

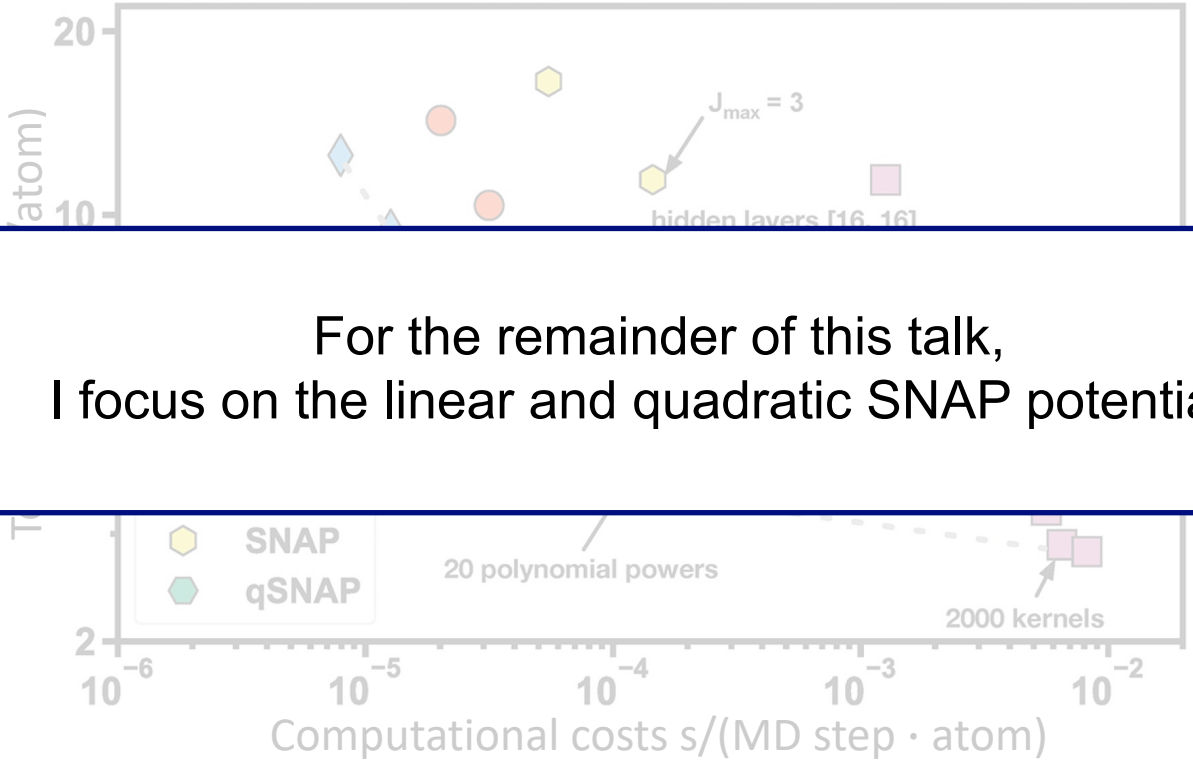
Fitting to energies and forces

Benchmark of Machine-Learned Interatomic Potentials



Y. Zuo, et. al. J. Phys. Chem. A 2020, 124, 4, 731-745

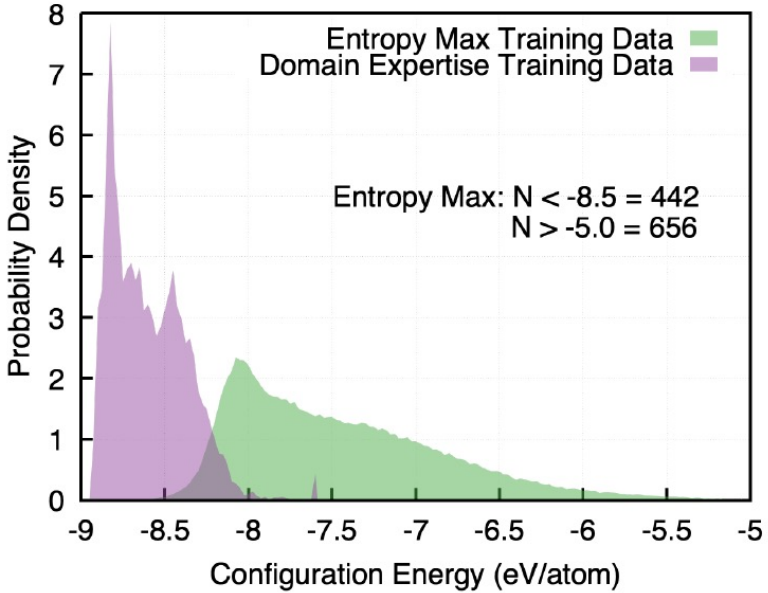
Benchmark of Machine-Learned Interatomic Potentials



For the remainder of this talk,
I focus on the linear and quadratic SNAP potentials.

Y. Zuo, et. al. J. Phys. Chem. A 2020, 124, 4, 731-745

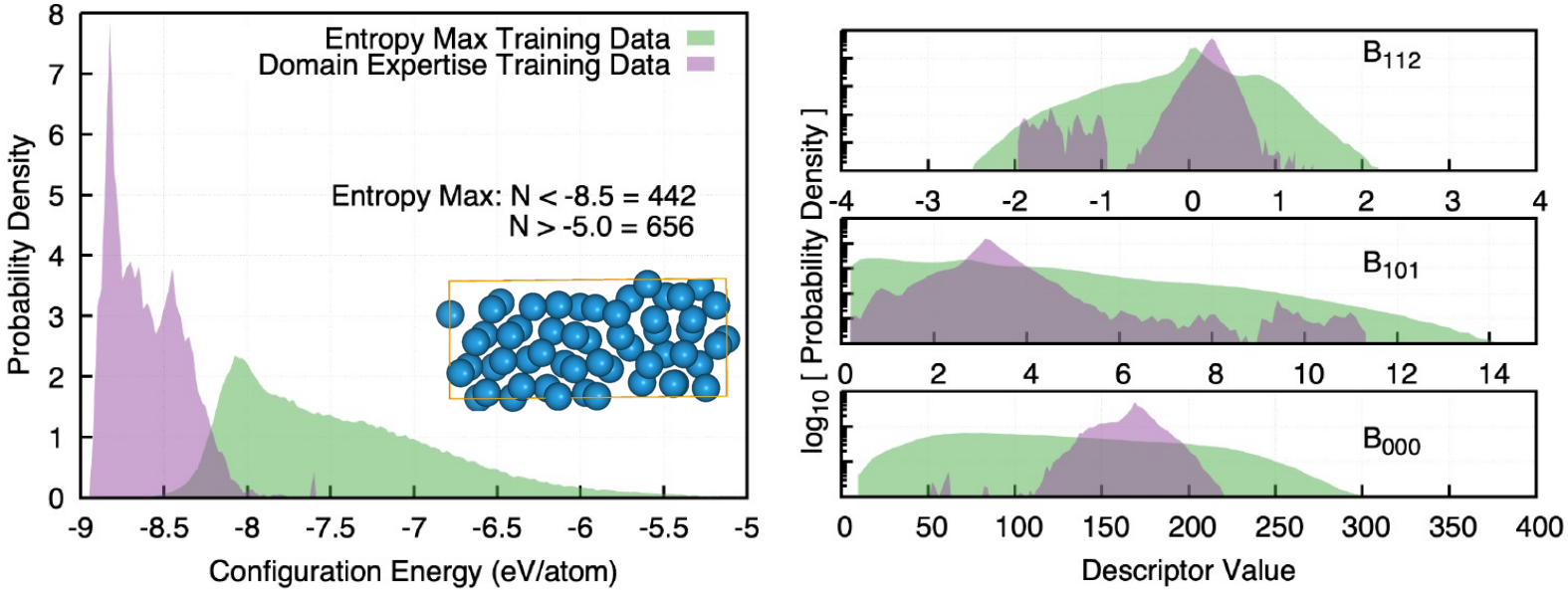
Transferable Machine-Learned Interatomic Potentials



M. Karabin, et. al. J. Chem. Phys. 153, 094110 (2020)

Montes de Oca Zapiain, D., et al. npj Comput Mater 8, 189 (2022).

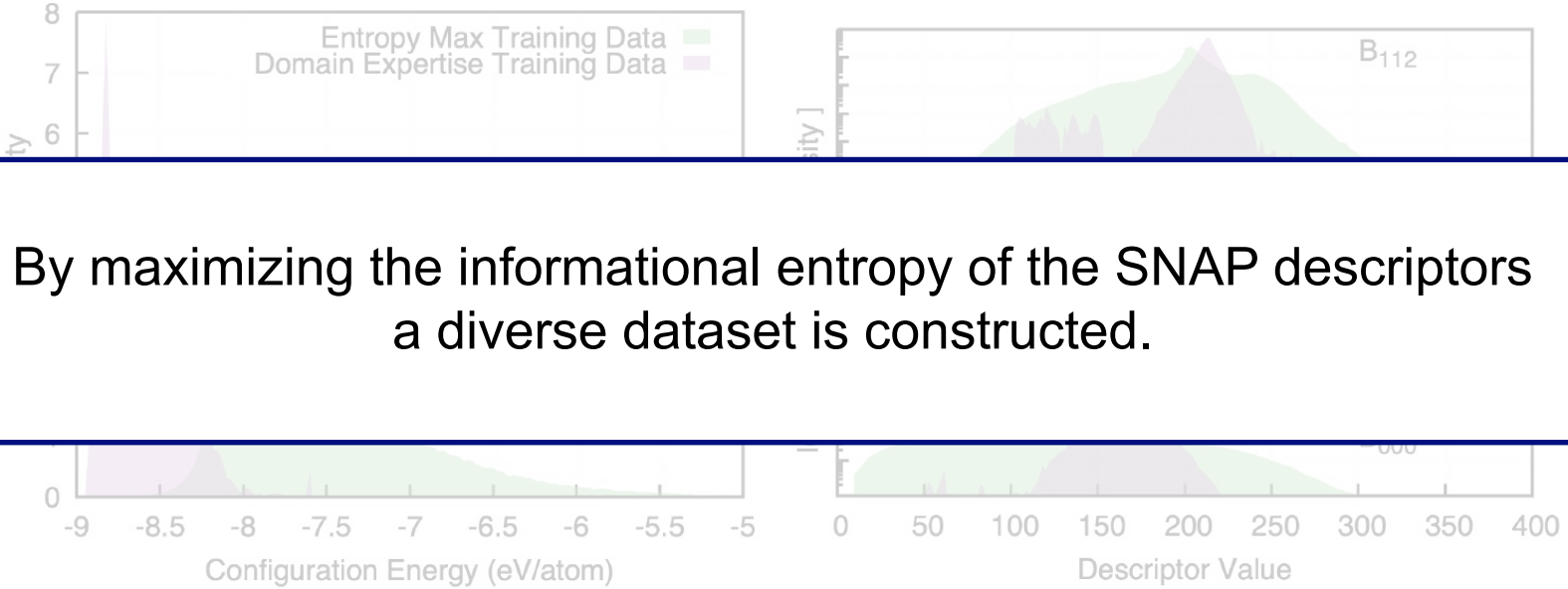
Transferable Machine-Learned Interatomic Potentials



M. Karabin, et. al. J. Chem. Phys. 153, 094110 (2020)

Montes de Oca Zapiain, D., et al. npj Comput Mater 8, 189 (2022).

Transferable Machine-Learned Interatomic Potentials

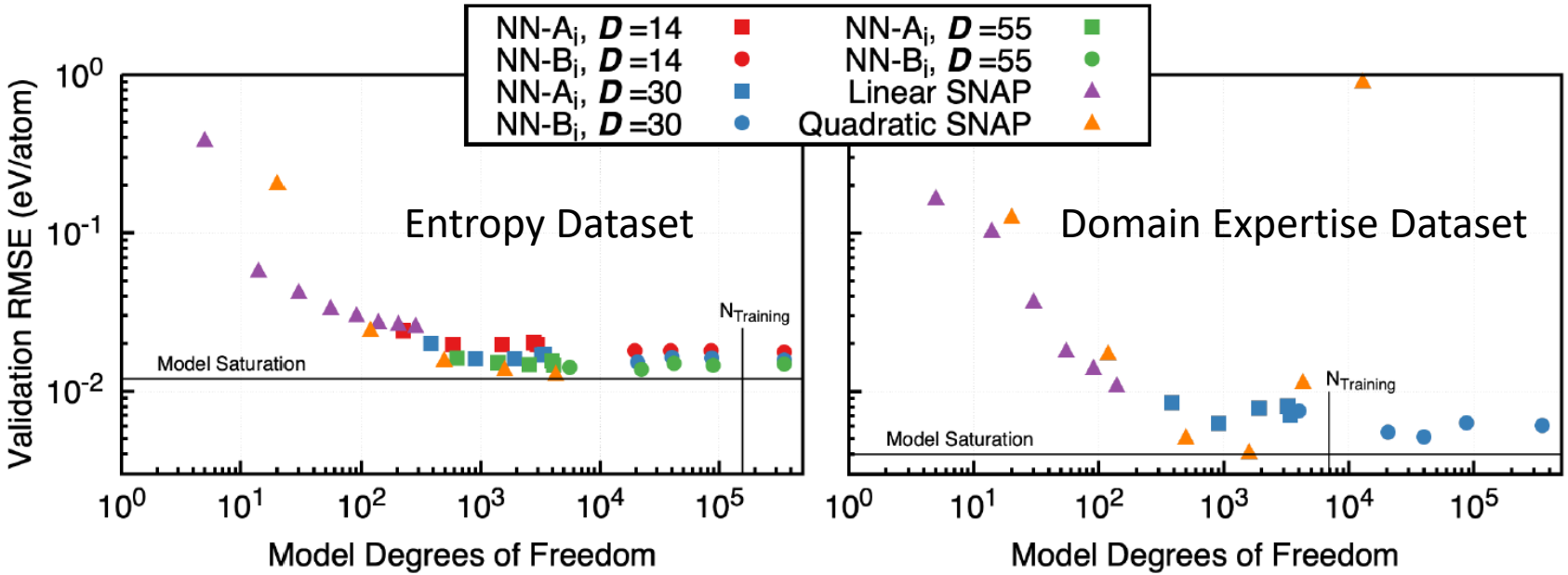


By maximizing the informational entropy of the SNAP descriptors a diverse dataset is constructed.

M. Karabin, et. al. J. Chem. Phys. 153, 094110 (2020)

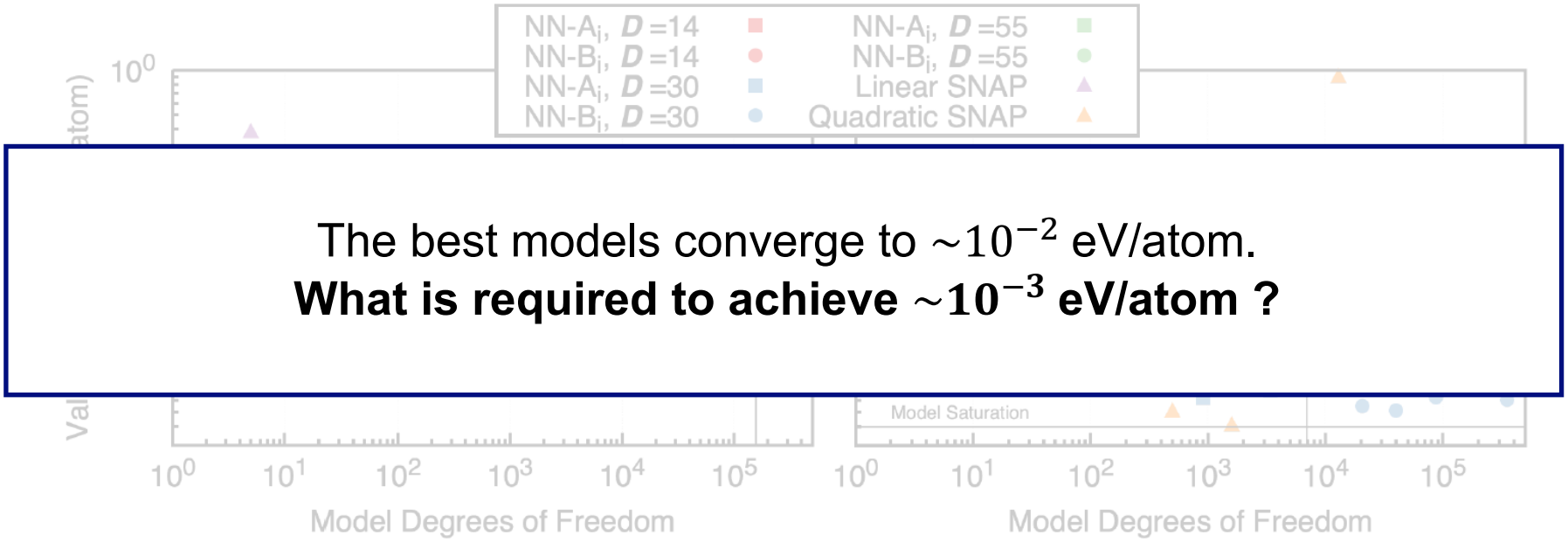
Montes de Oca Zapiain, D., et al. npj Comput Mater 8, 189 (2022).

Benchmark Different Types of Interatomic Potentials



Montes de Oca Zapiain, D., et al. npj Comput Mater 8, 189 (2022).

Benchmark Different Types of Interatomic Potentials

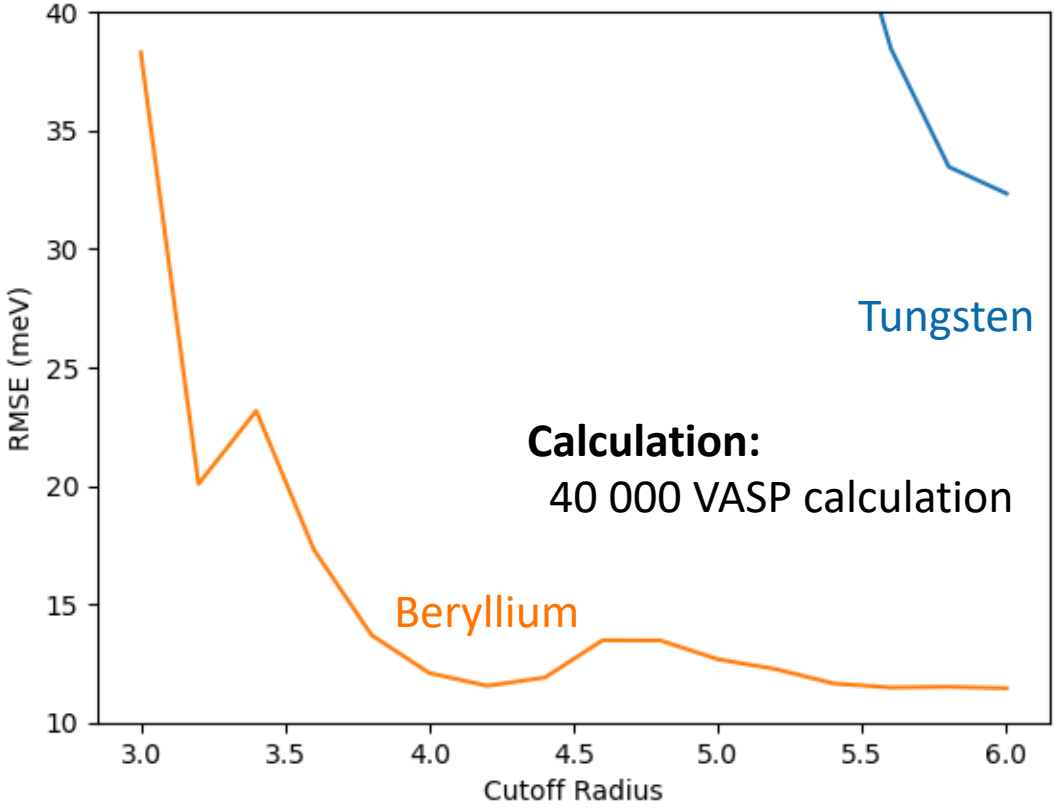


The best models converge to $\sim 10^{-2}$ eV/atom.

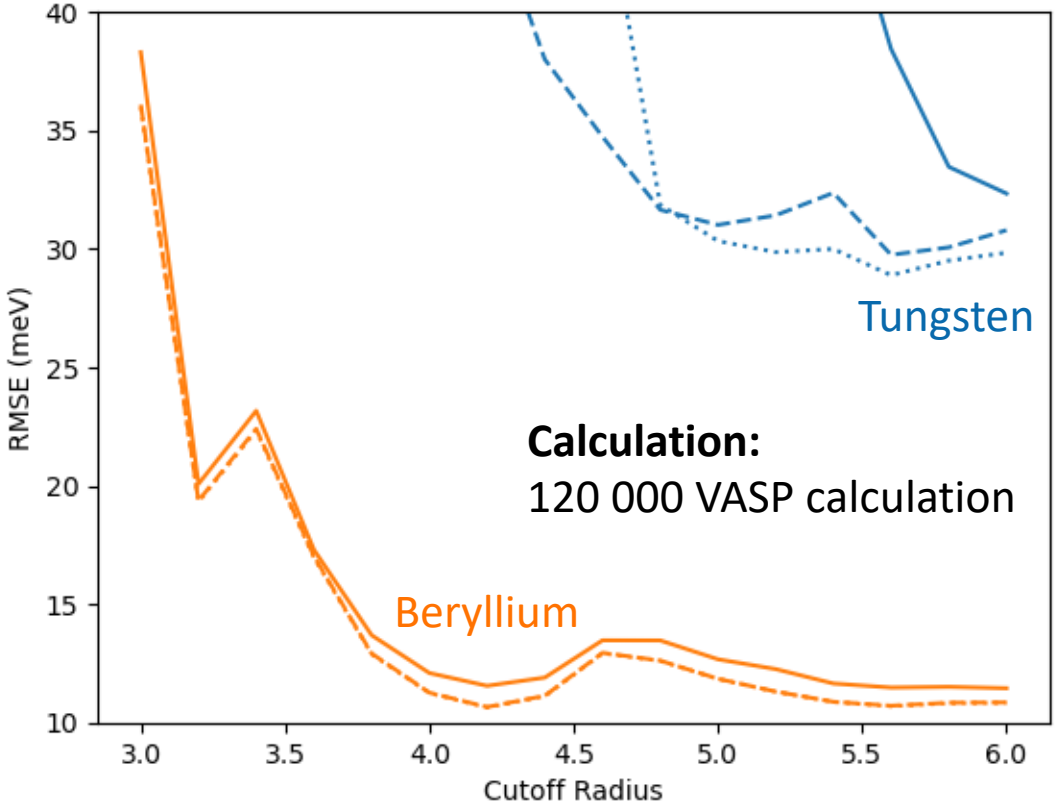
What is required to achieve $\sim 10^{-3}$ eV/atom ?

Montes de Oca Zapiain, D., et al. npj Comput Mater 8, 189 (2022).

Compare Tungsten and Beryllium

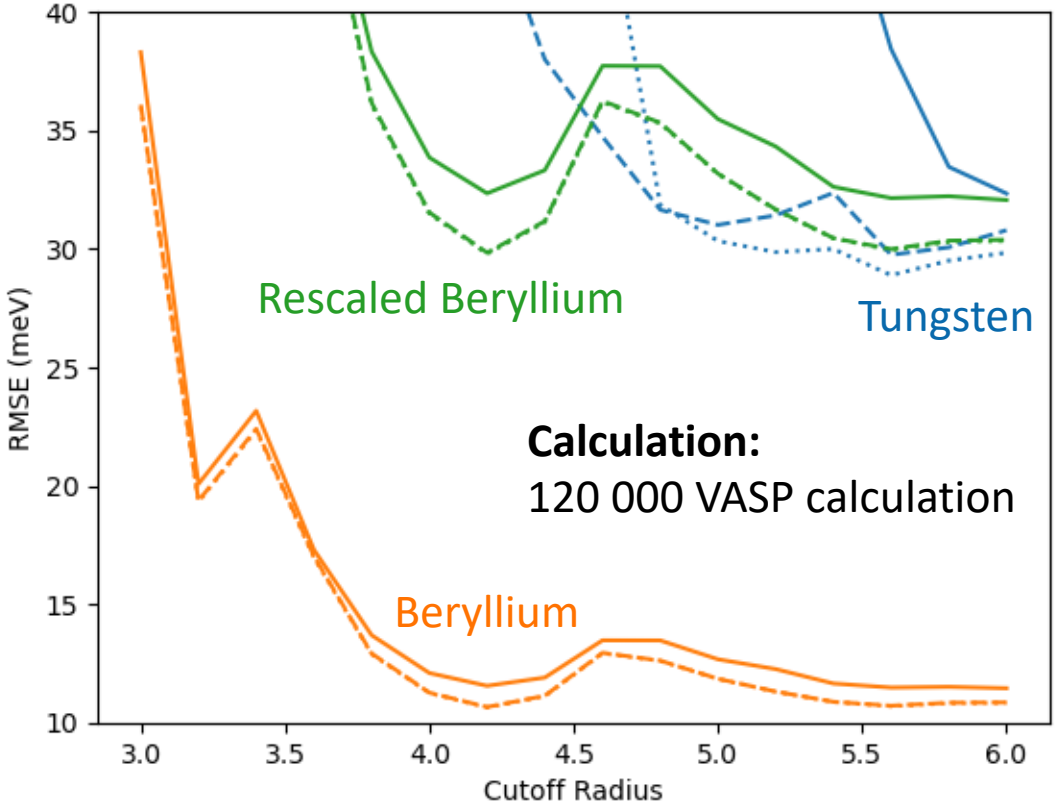


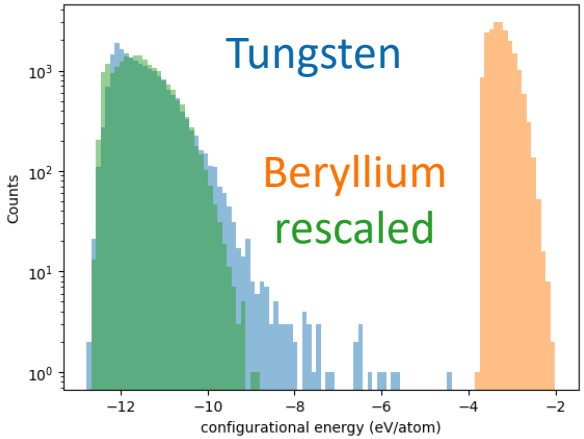
Compare Tungsten and Beryllium



DFT precision:
solid: 0.5 kspacing 500 eV encut
dashed: 0.25 kspacing 700 eV encut
dotted: 0.1 kspacing 900 eV encut

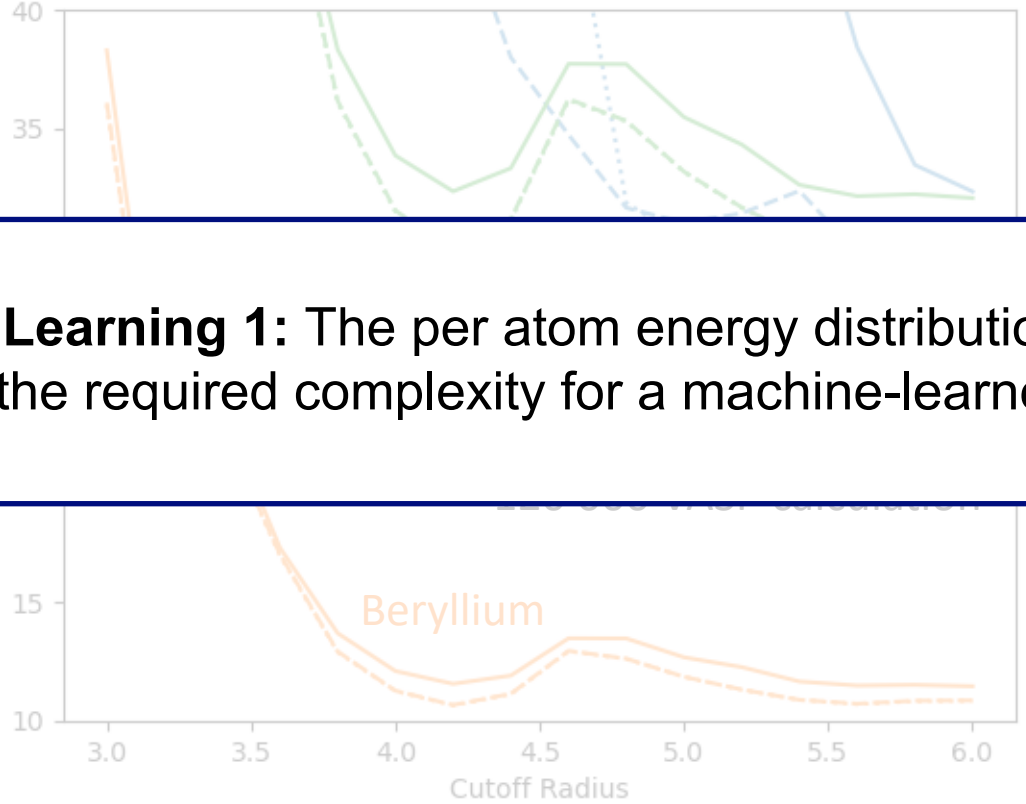
Compare Tungsten and Beryllium





DFT precision:
solid: 0.5 kspacing 500 eV encut
dashed: 0.25 kspacing 700 eV encut
dotted: 0.1 kspacing 900 eV encut

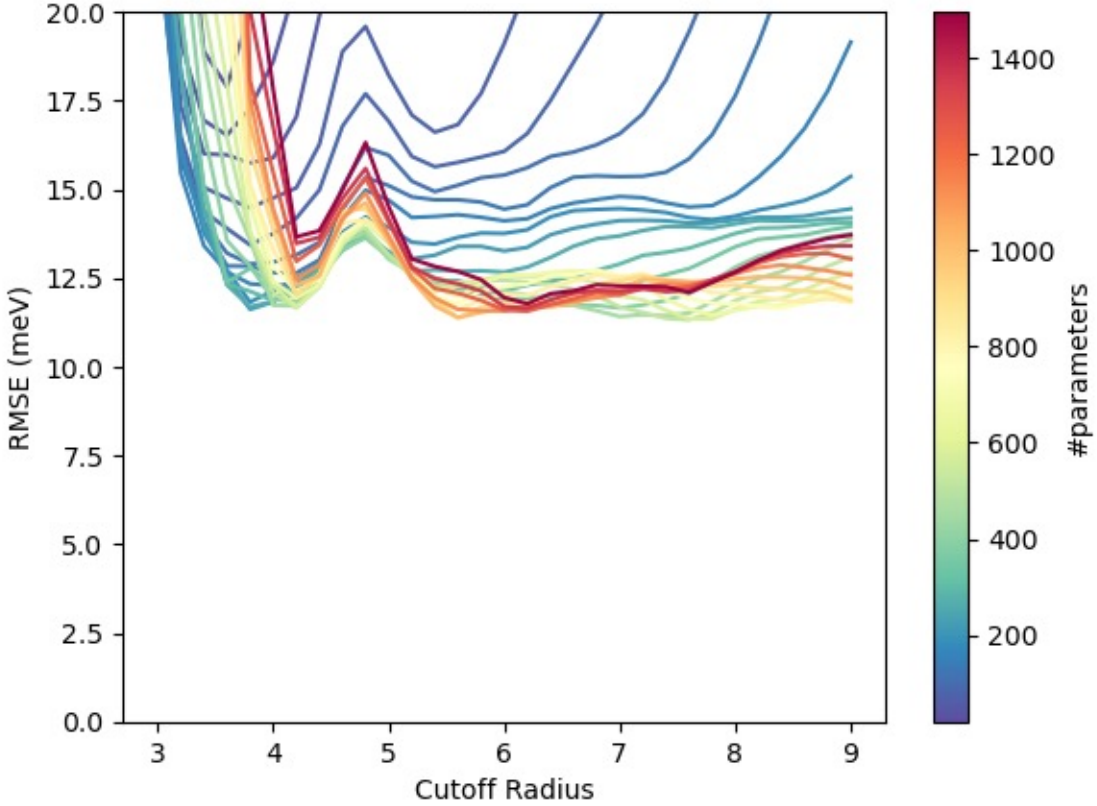
Compare Tungsten and Beryllium



Learning 1: The per atom energy distribution is a good indicator of the required complexity for a machine-learned interatomic potential.

DFT precision:
solid: 0.5 kspacing 500 eV encut
dashed: 0.25 kspacing 700 eV encut
dotted: 0.1 kspacing 900 eV encut

Optimal Cut-Off Radius



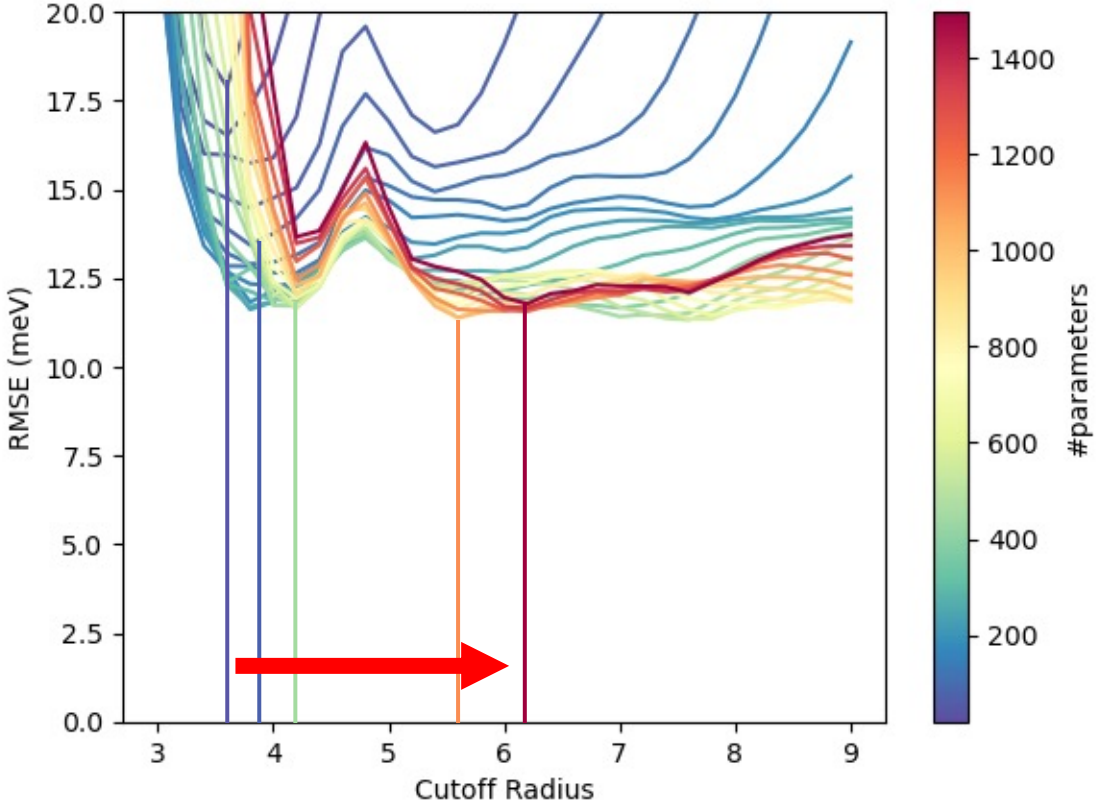
Calculation:

- 26 $2j_{max}$
- 31 radii

Total:

806 calculation

Optimal Cut-Off Radius



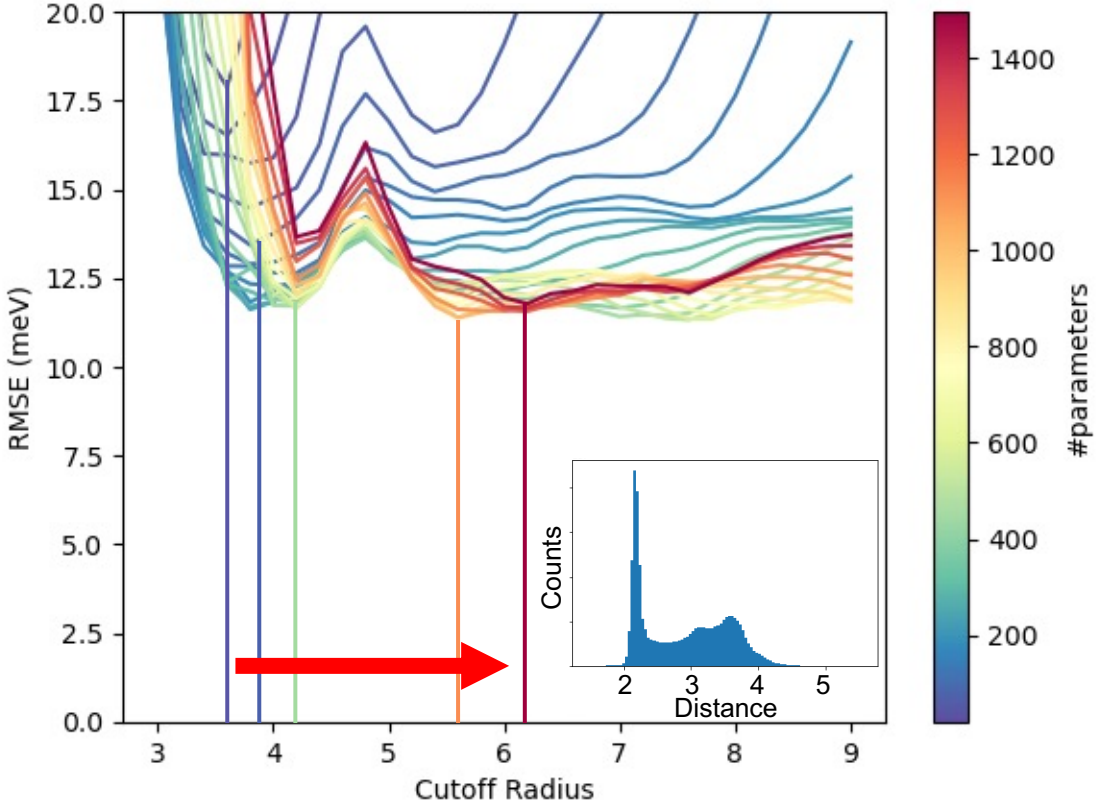
Calculation:

- 26 $2j_{max}$
- 31 radii

Total:

806 calculation

Optimal Cut-Off Radius



Calculation:

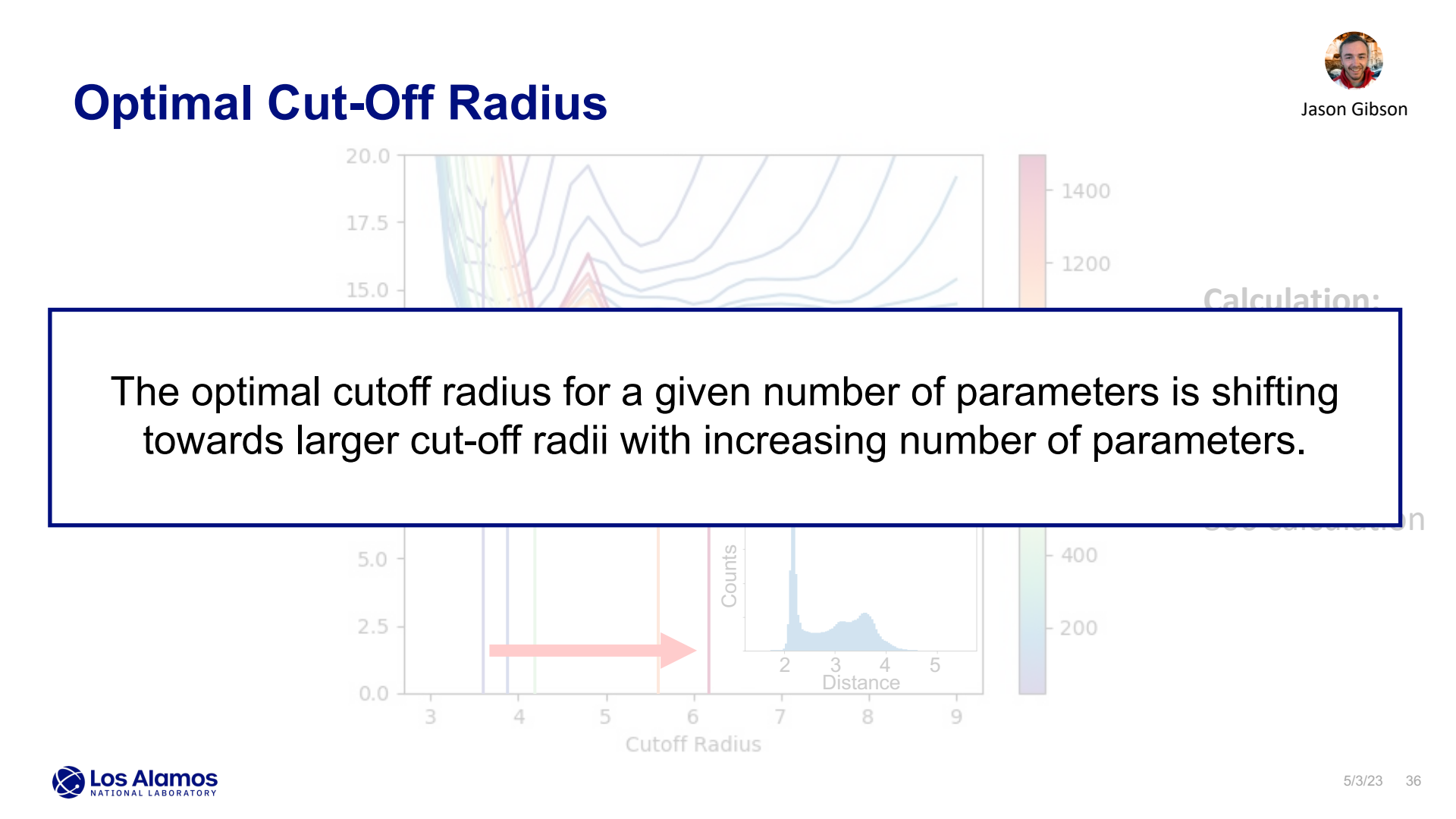
- 26 $2j_{max}$
- 31 radii

Total:

806 calculation

Jason Gibson

Optimal Cut-Off Radius

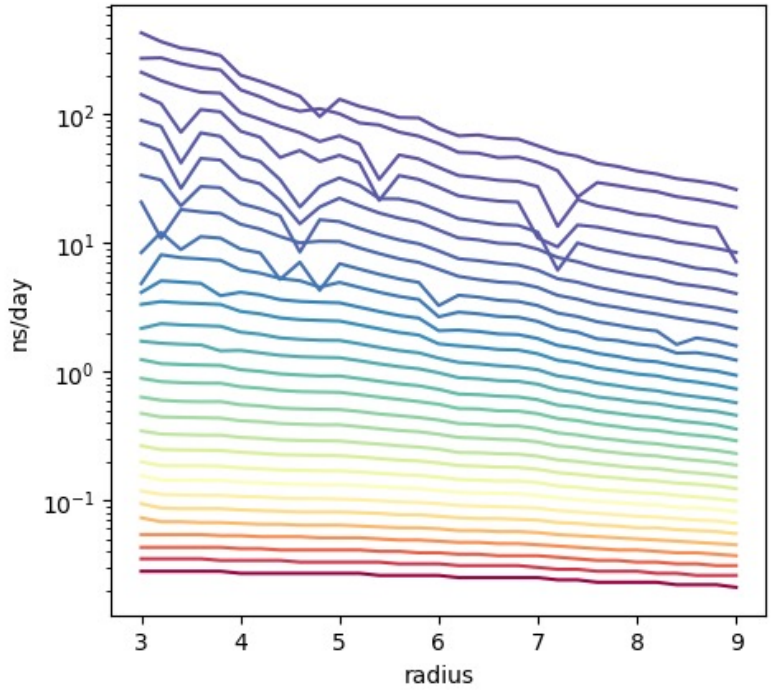


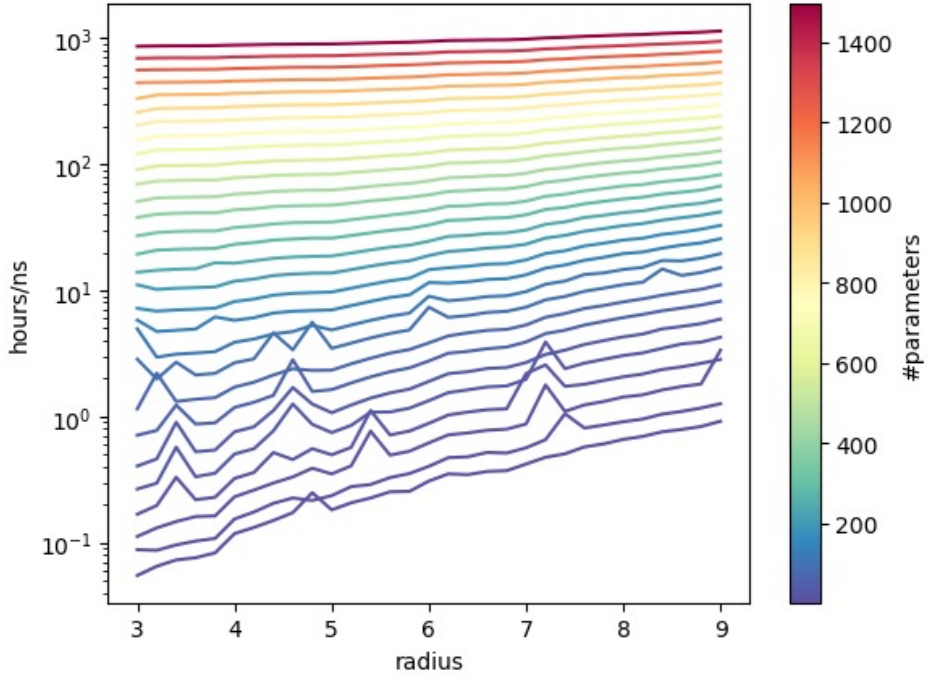
Jason Gibson

The optimal cutoff radius for a given number of parameters is shifting towards larger cut-off radii with increasing number of parameters.

Computational Cost on an Intel Xeon 6152

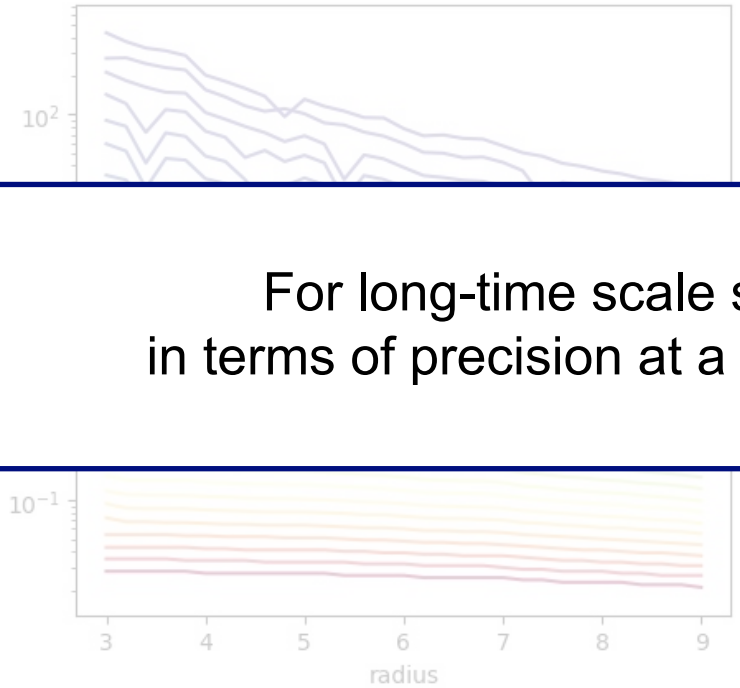
Jason Gibson

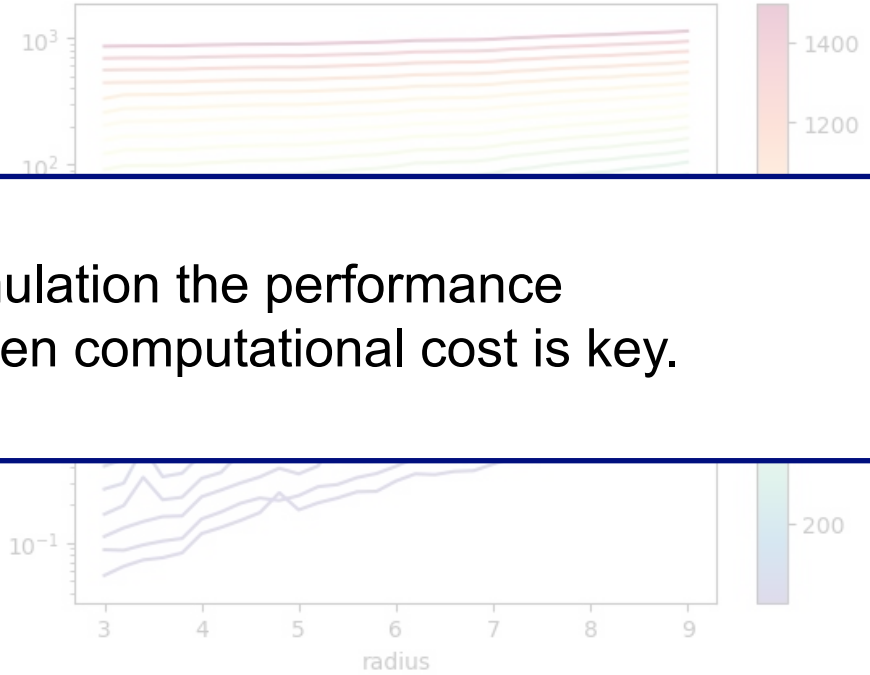




Computational Cost on an Intel Xeon 6152

Jason Gibson

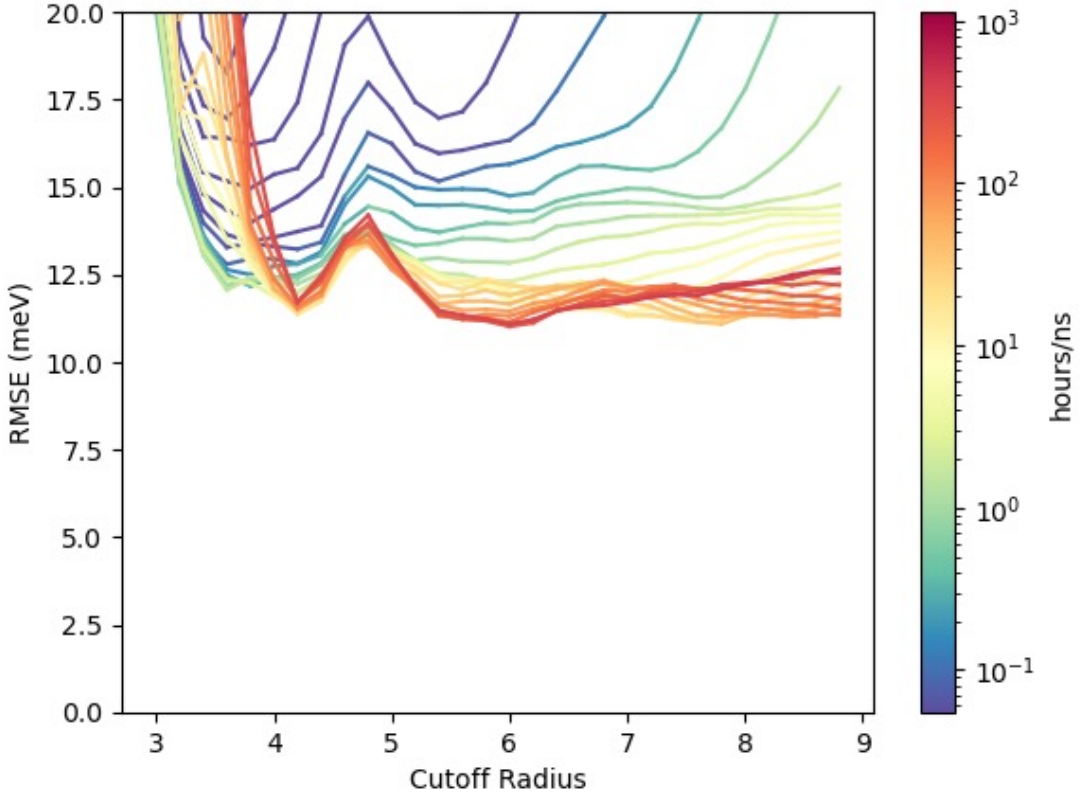




For long-time scale simulation the performance
in terms of precision at a given computational cost is key.

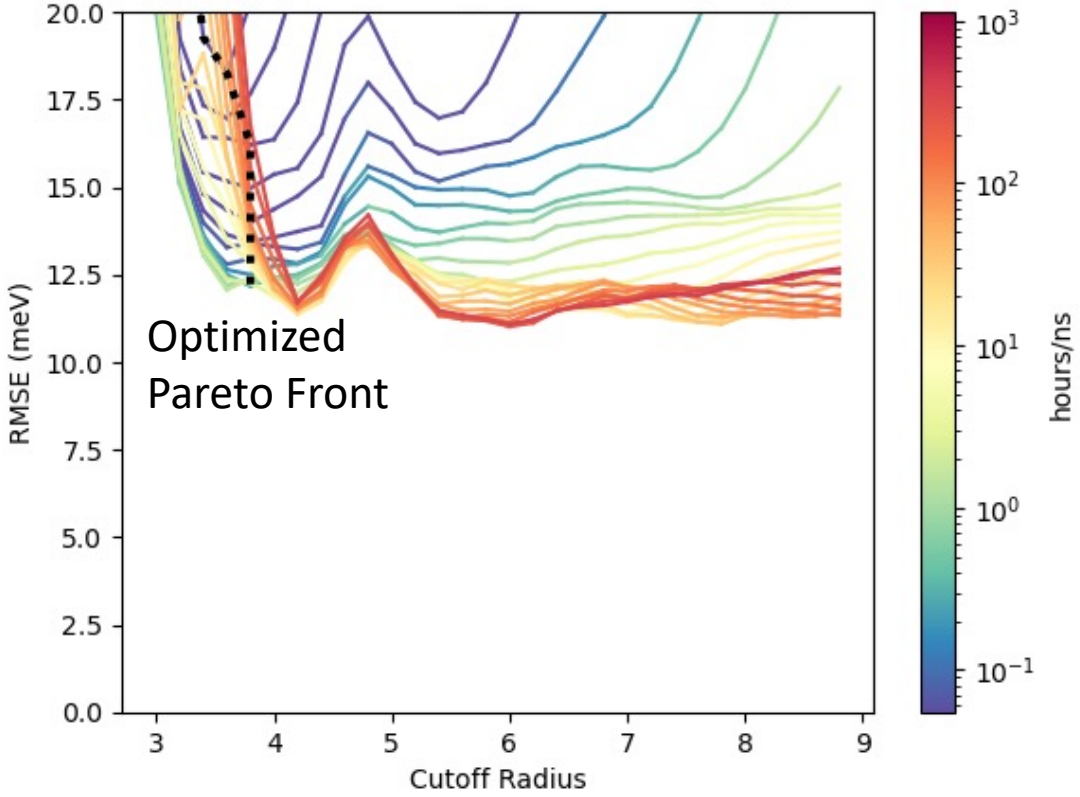
Pareto Front Including Computational Cost

Jason Gibson

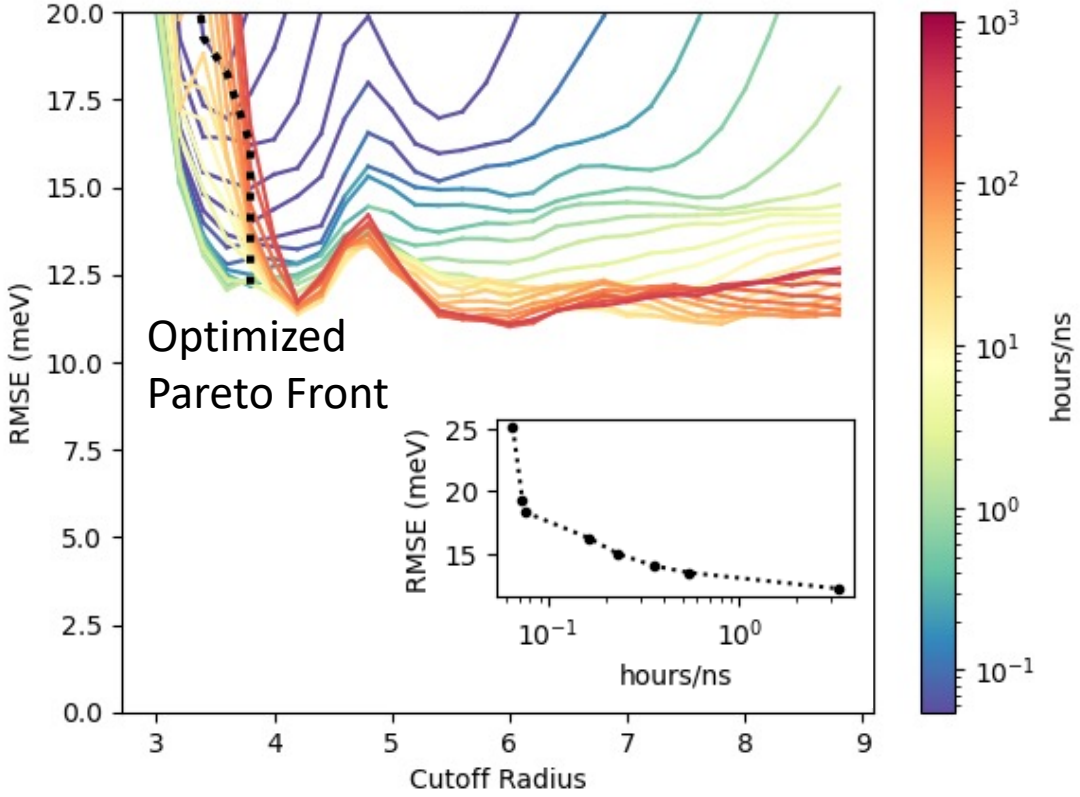


Pareto Front Including Computational Cost

Jason Gibson

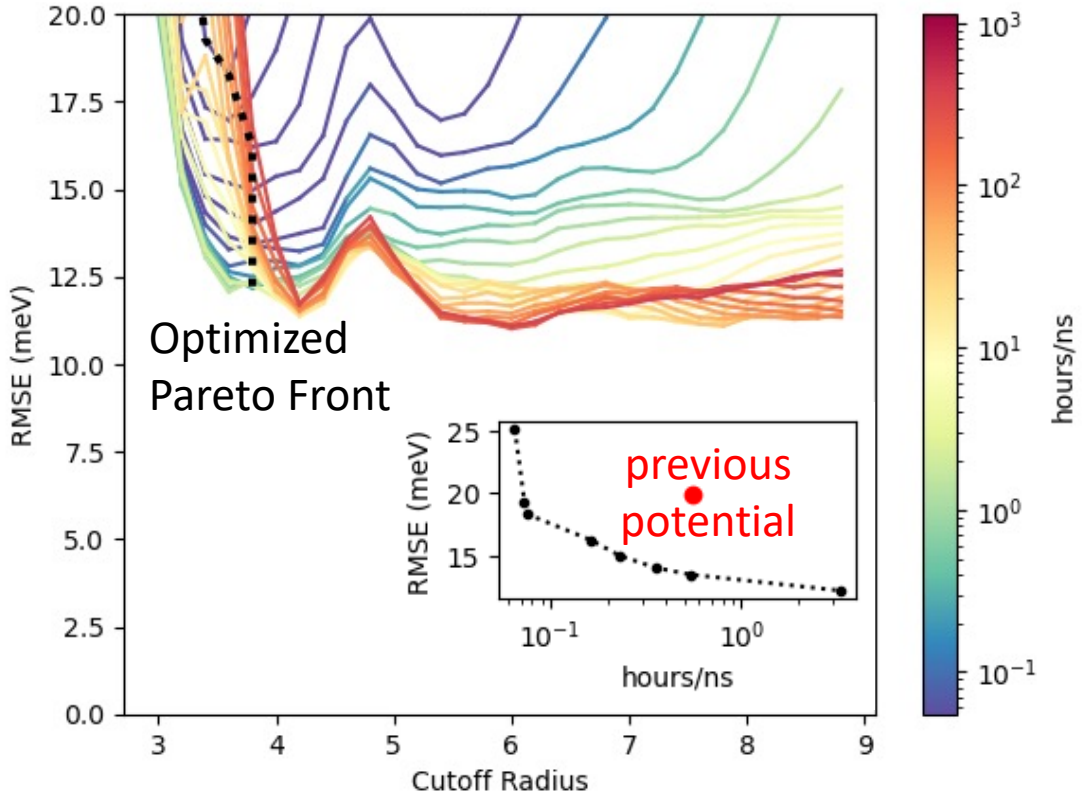


Pareto Front Including Computational Cost



Jason Gibson

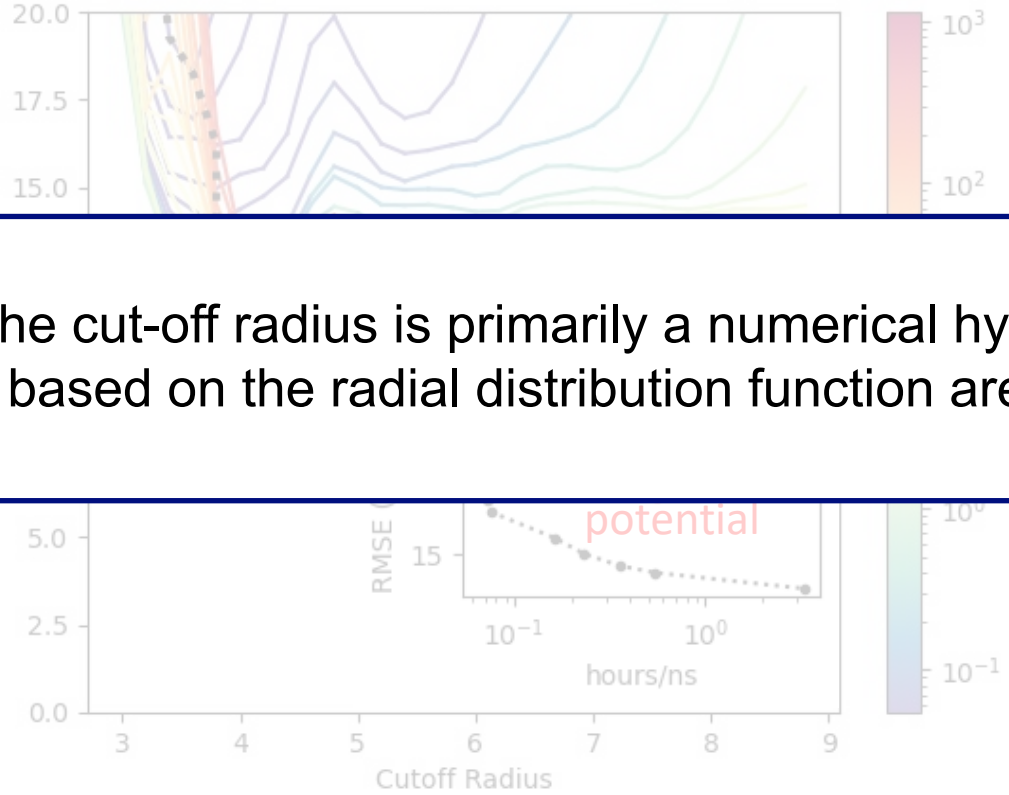
Pareto Front Including Computational Cost



Jason Gibson

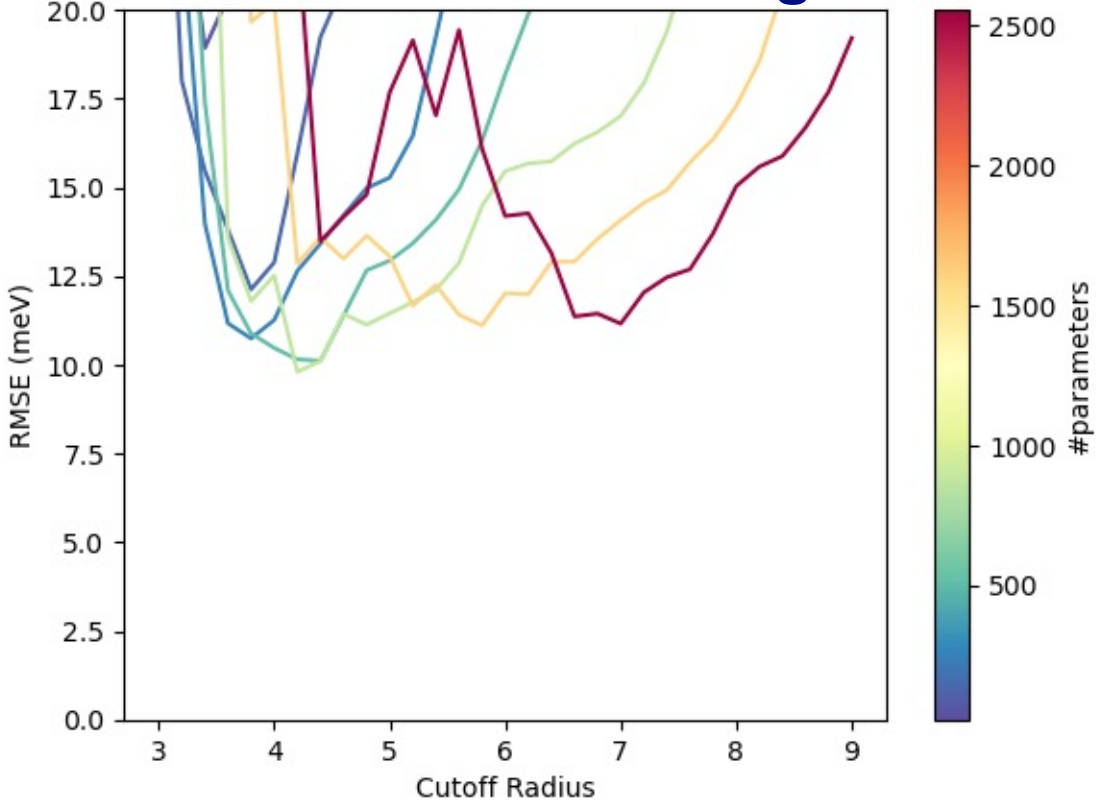
Pareto Front Including Computational Cost

Jason Gibson

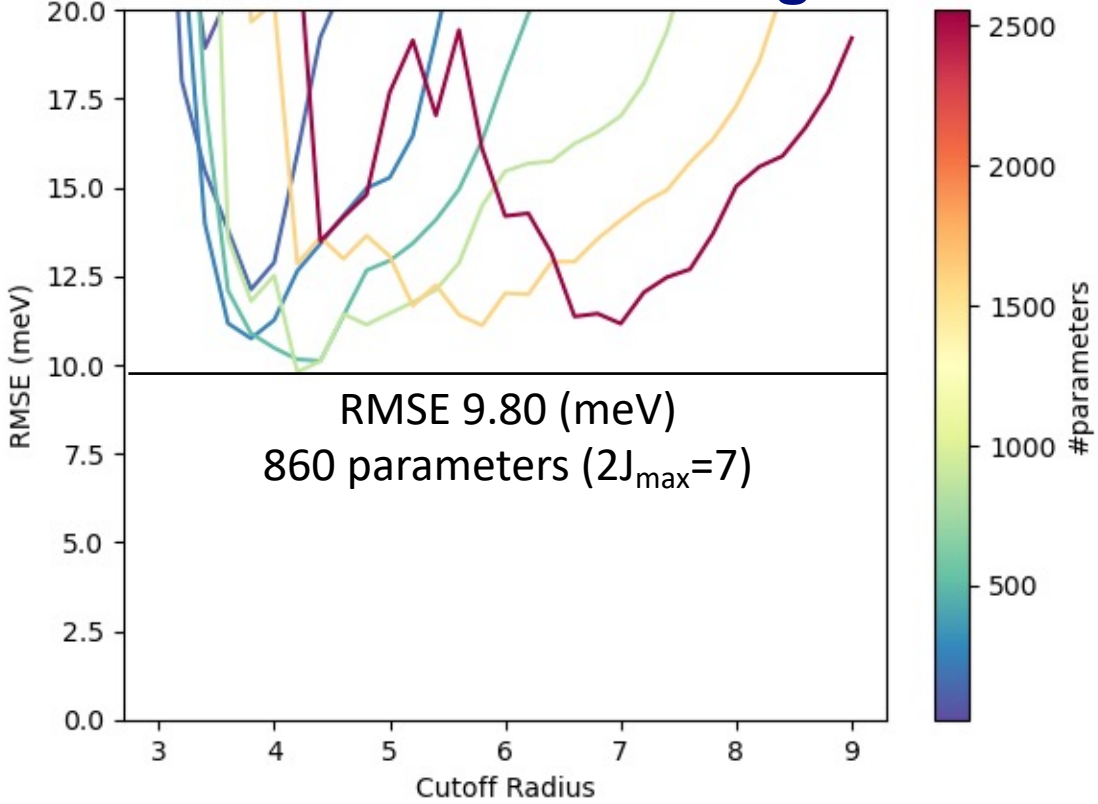


Learning 2: The cut-off radius is primarily a numerical hyperparameter, considerations based on the radial distribution function are less relevant.

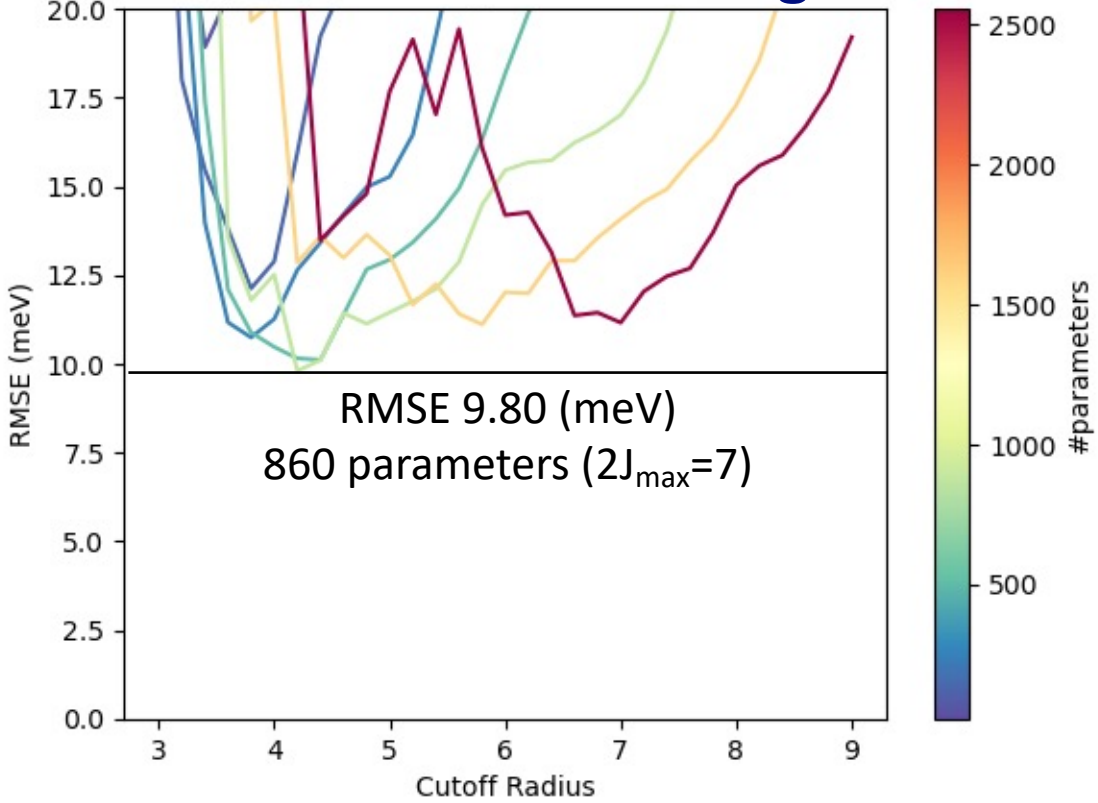
Quadratic SNAP Fit to Energies

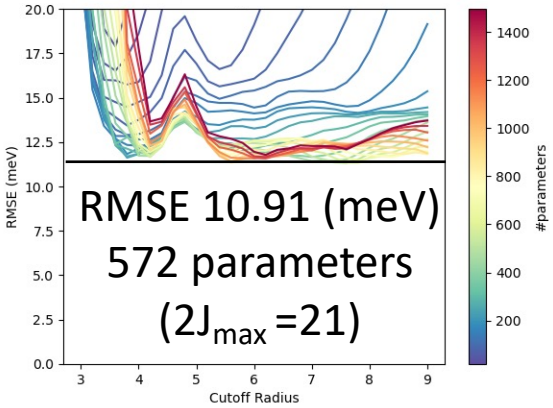


Quadratic SNAP Fit to Energies

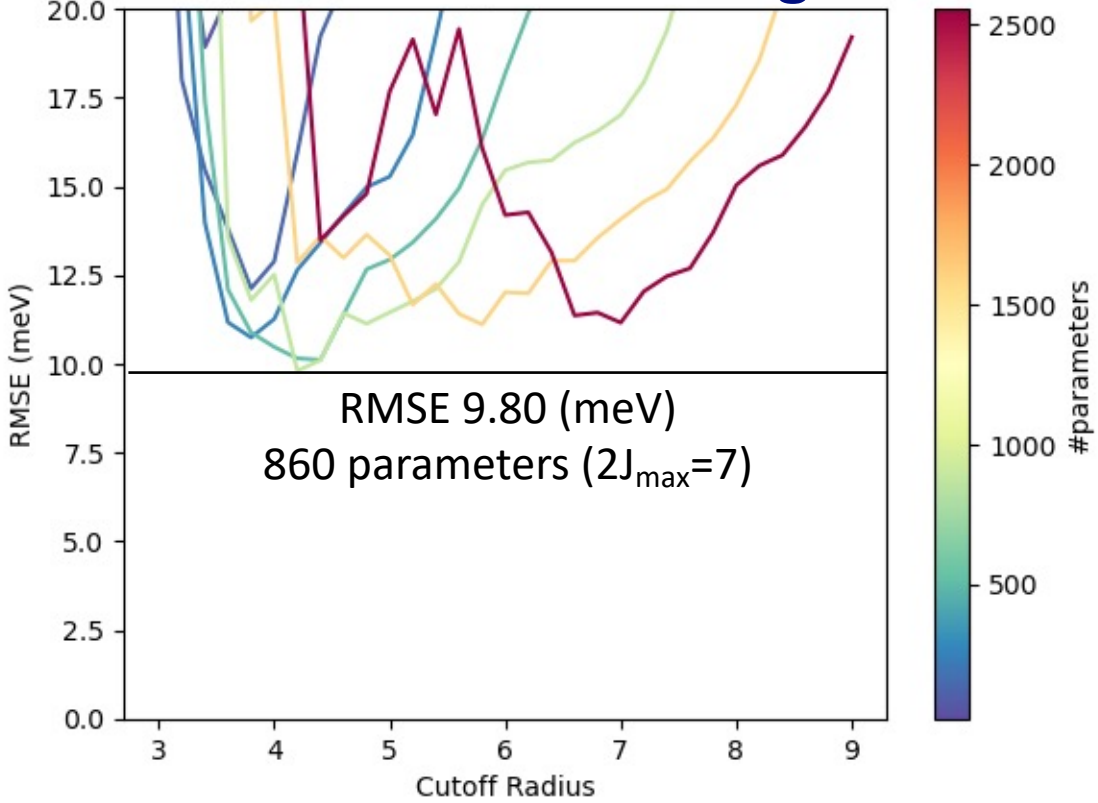


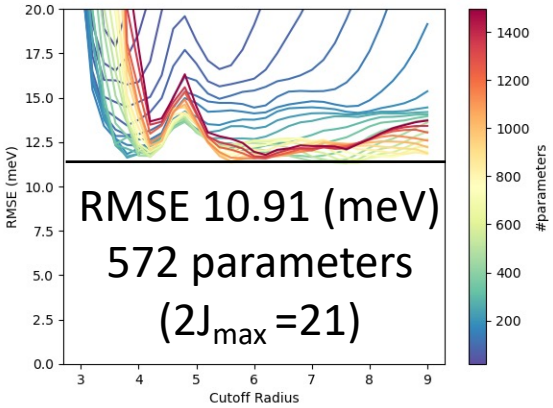
Quadratic SNAP Fit to Energies

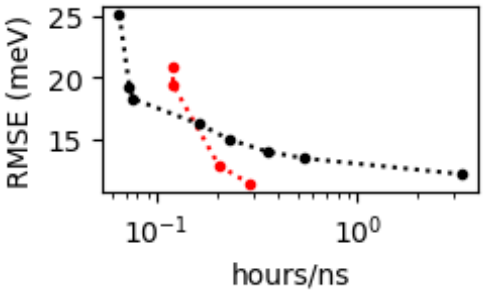




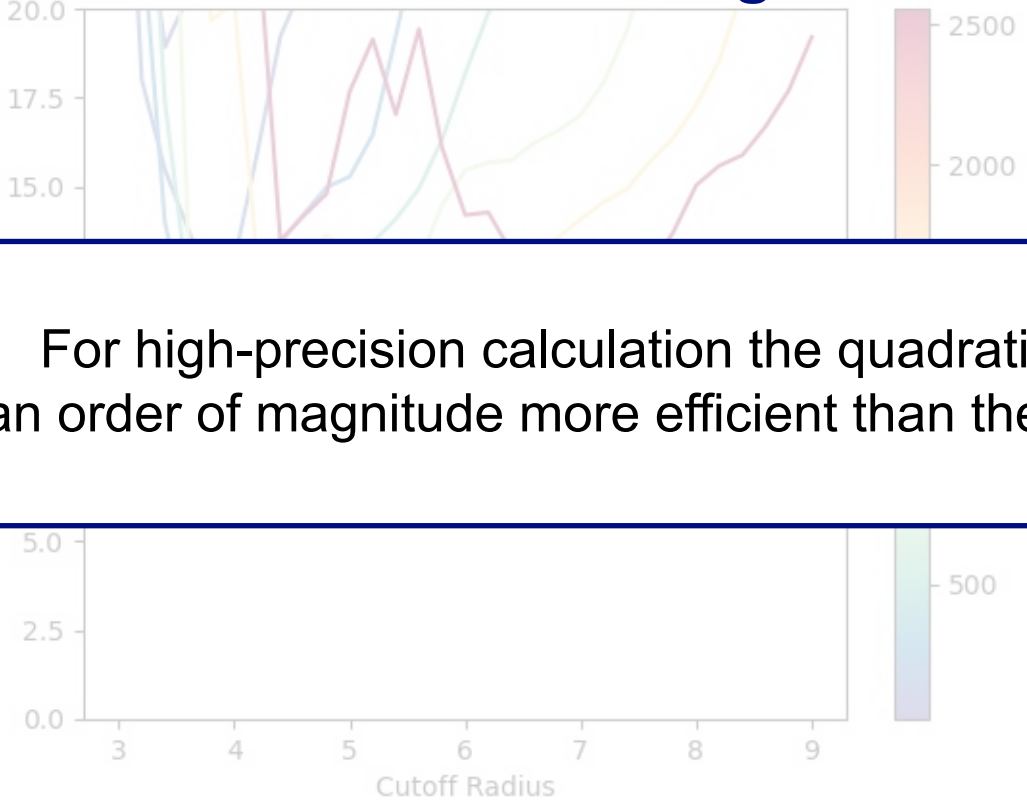
Quadratic SNAP Fit to Energies





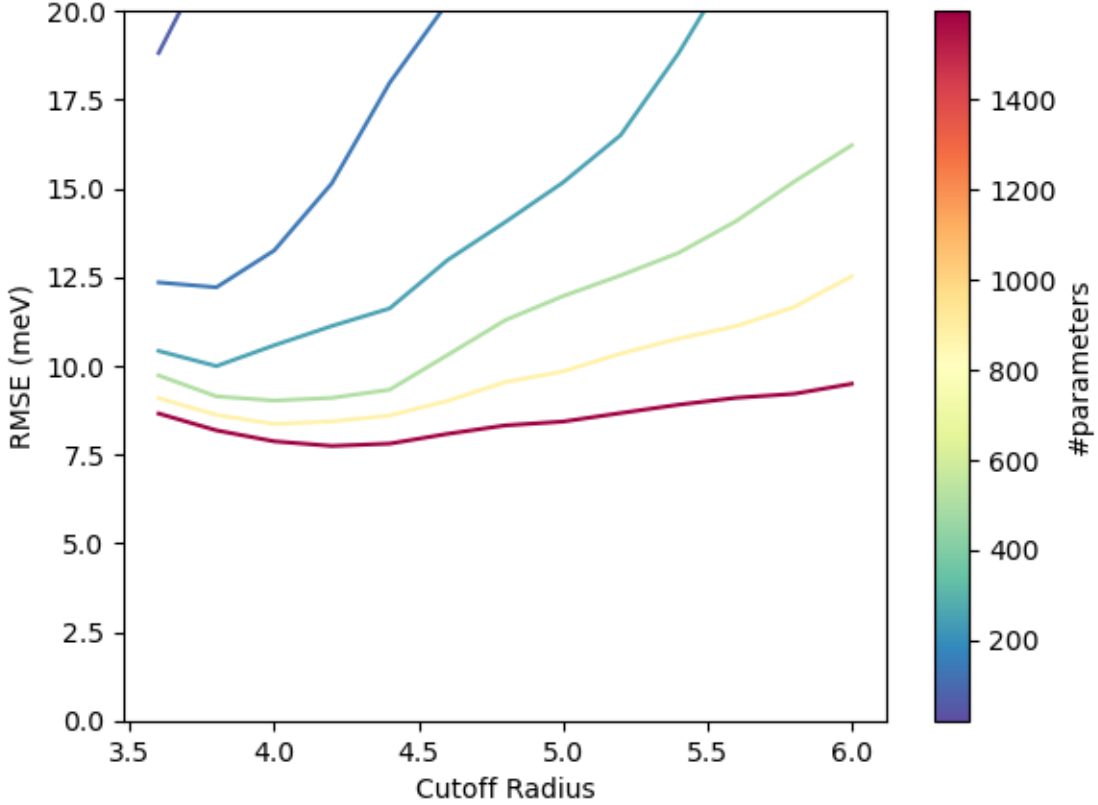


Quadratic SNAP Fit to Energies

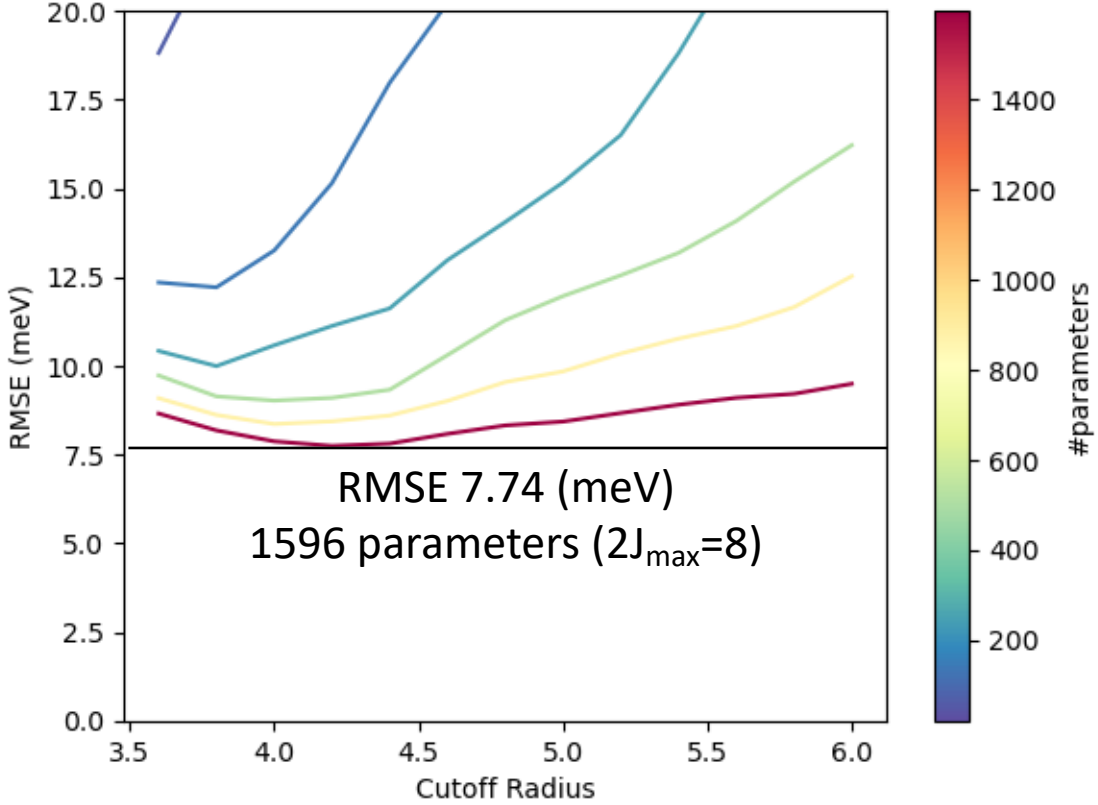


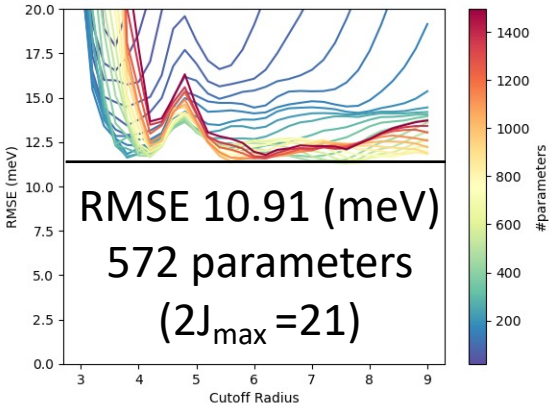
For high-precision calculation the quadratic SNAP models are an order of magnitude more efficient than the linear SNAP models.

Quadratic SNAP Fit to Forces

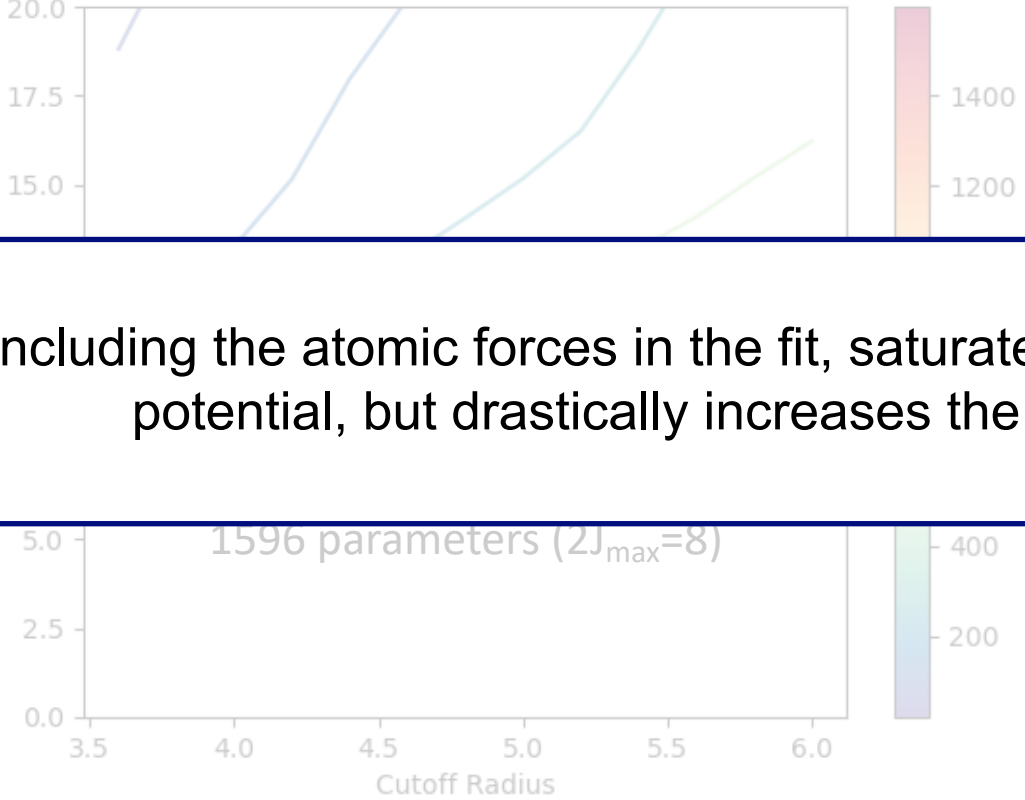


Quadratic SNAP Fit to Forces



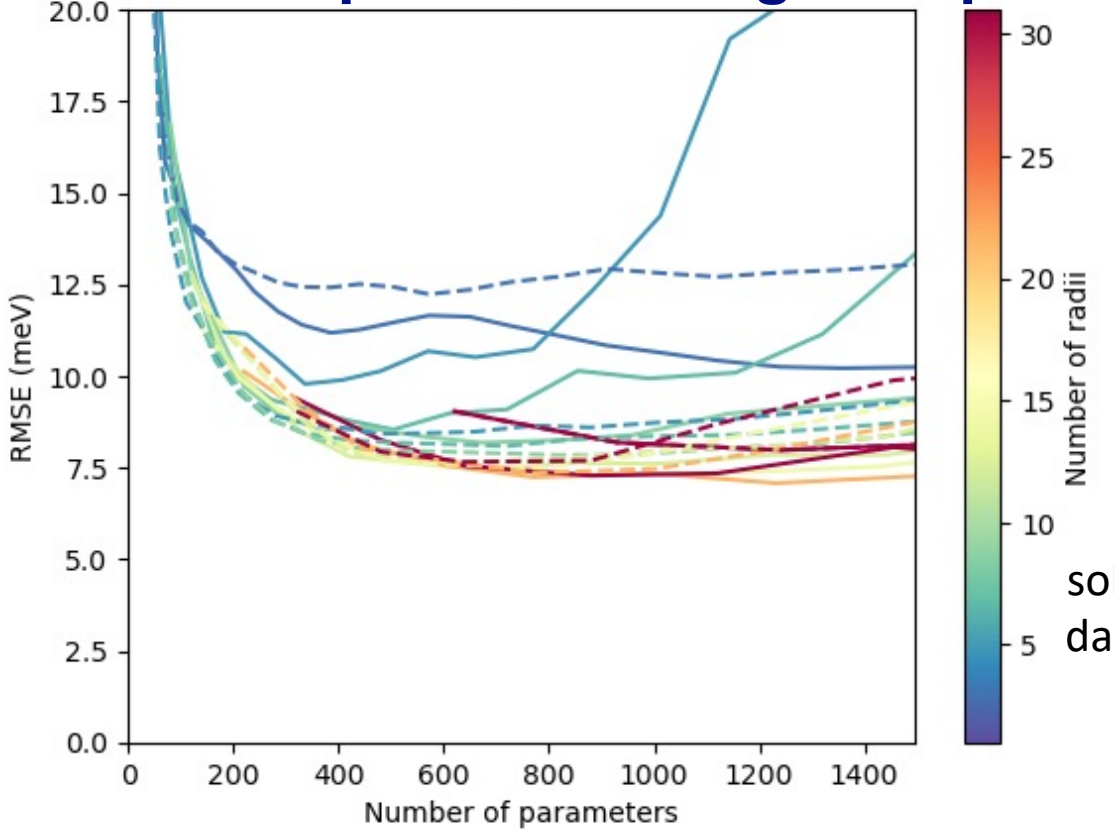


Quadratic SNAP Fit to Forces



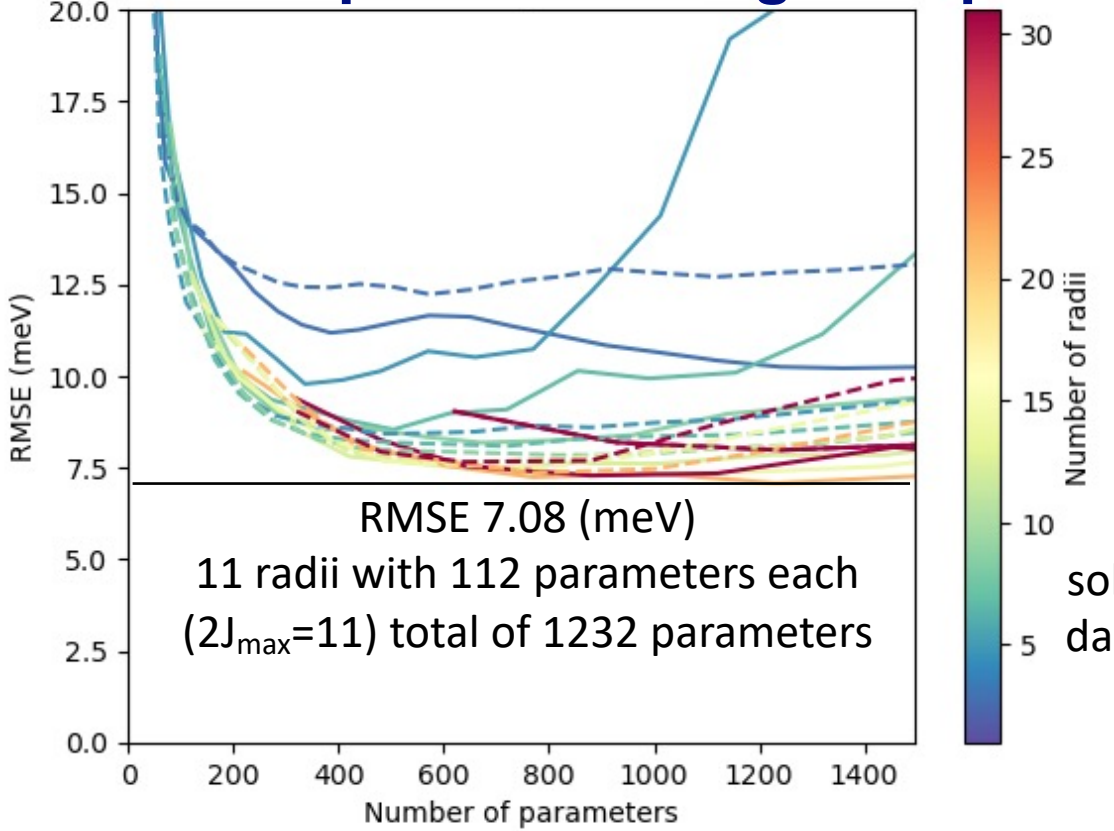
Including the atomic forces in the fit, saturates the quadratic SNAP potential, but drastically increases the memory usage.

SNAP Descriptors Combing Multiple Radii



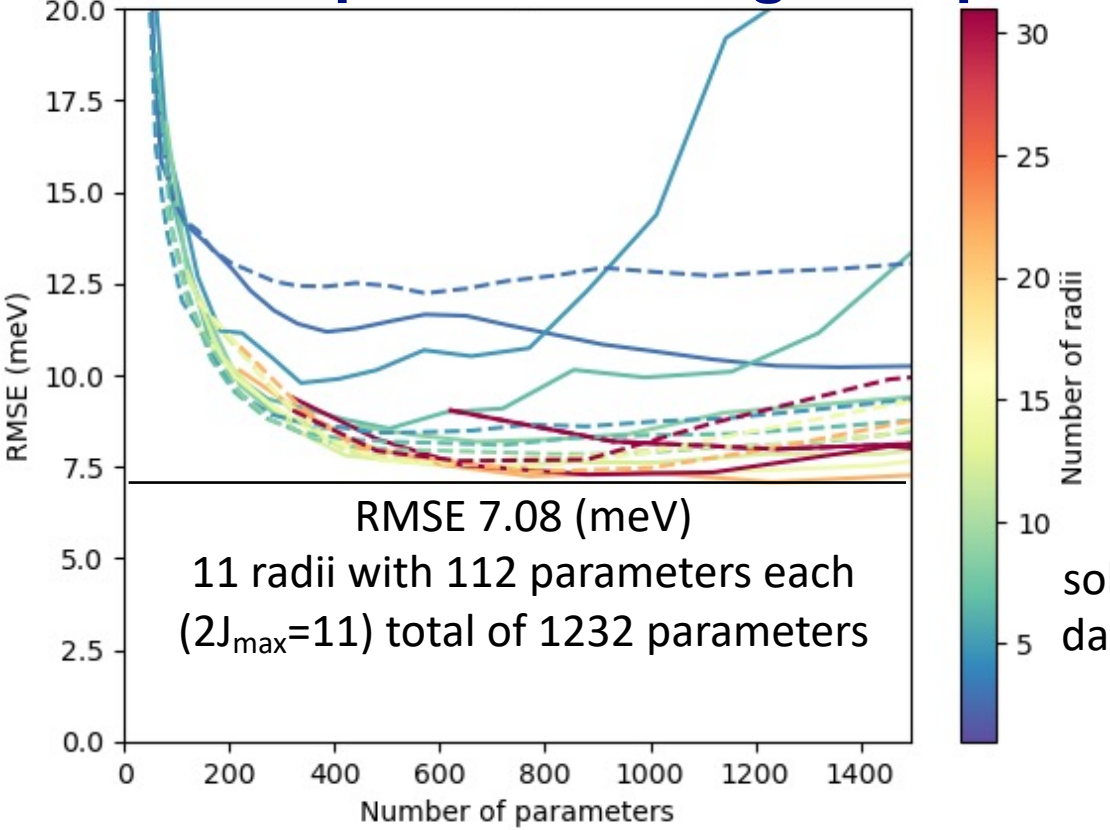
solid line – fit range from 3.0 to 9.0
dashed line – fit radii around 4.6

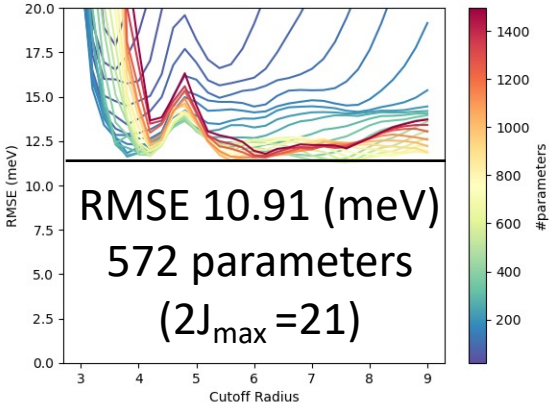
SNAP Descriptors Combing Multiple Radii



solid line – fit range from 3.0 to 9.0
dashed line – fit radii around 4.6

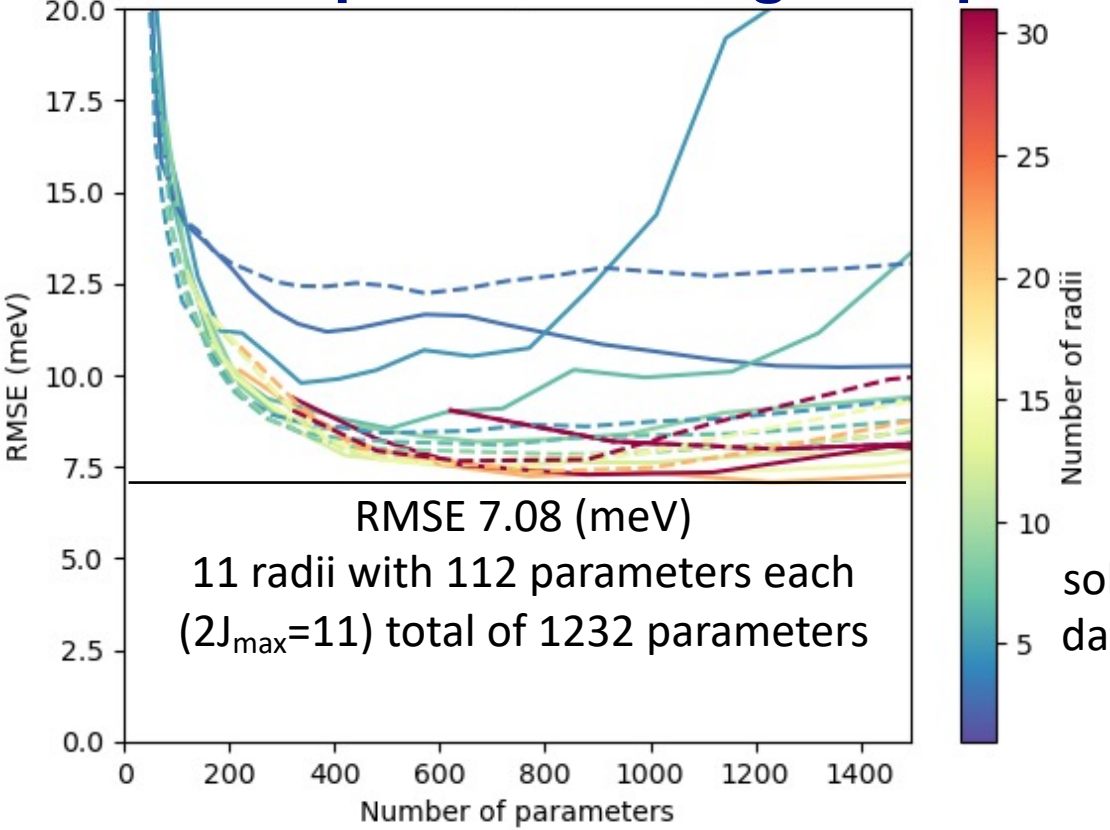
SNAP Descriptors Combing Multiple Radii

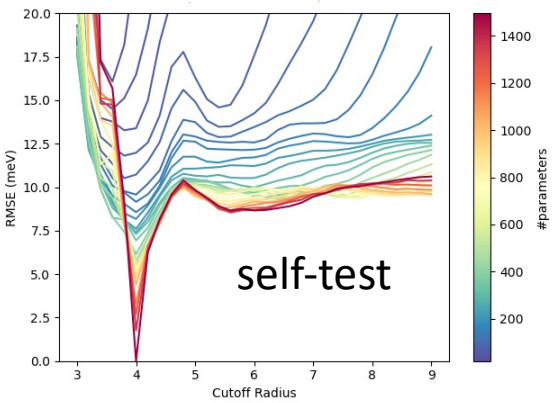




solid line – fit range from 3.0 to 9.0
dashed line – fit radii around 4.6

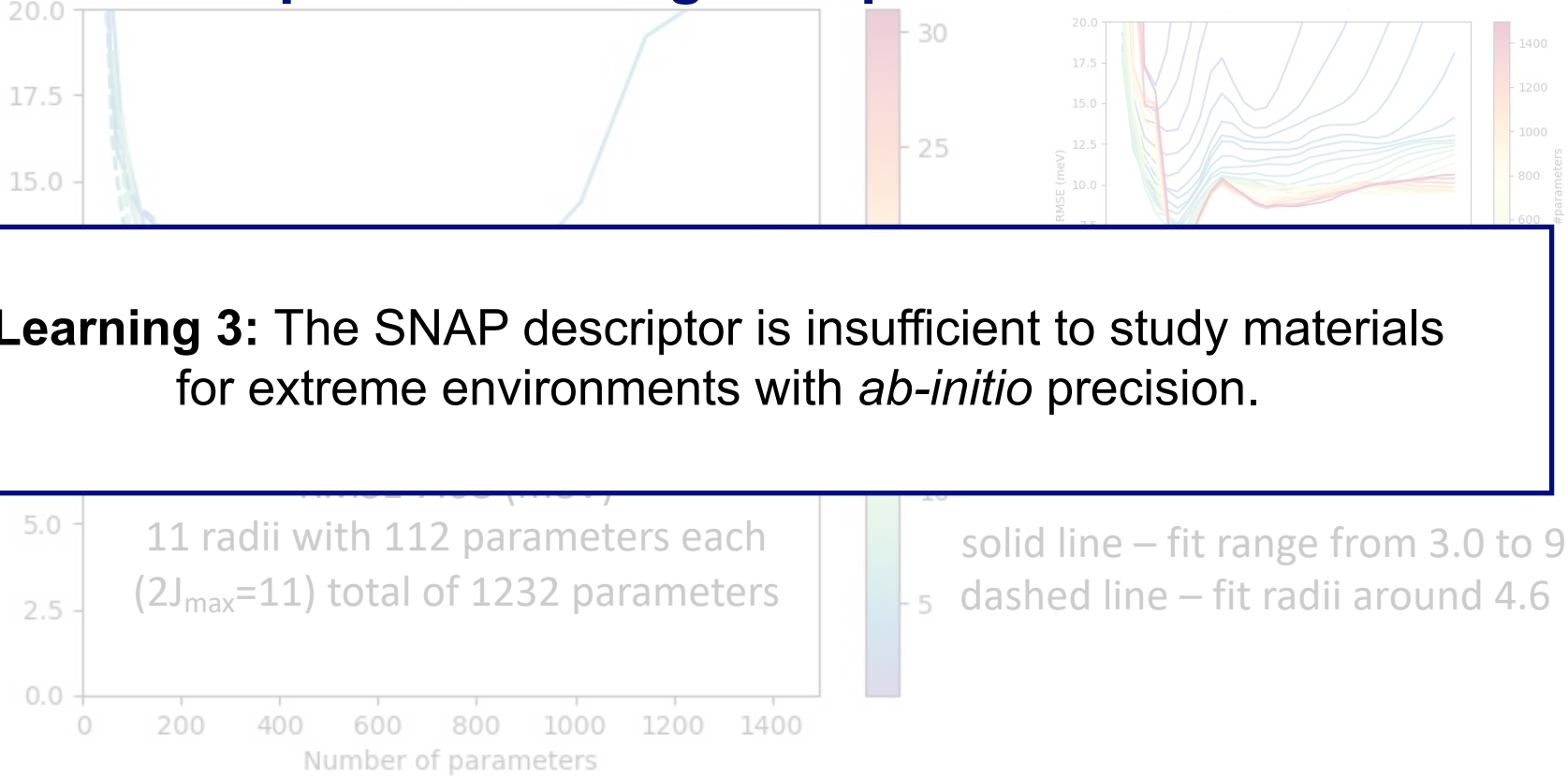
SNAP Descriptors Combing Multiple Radii





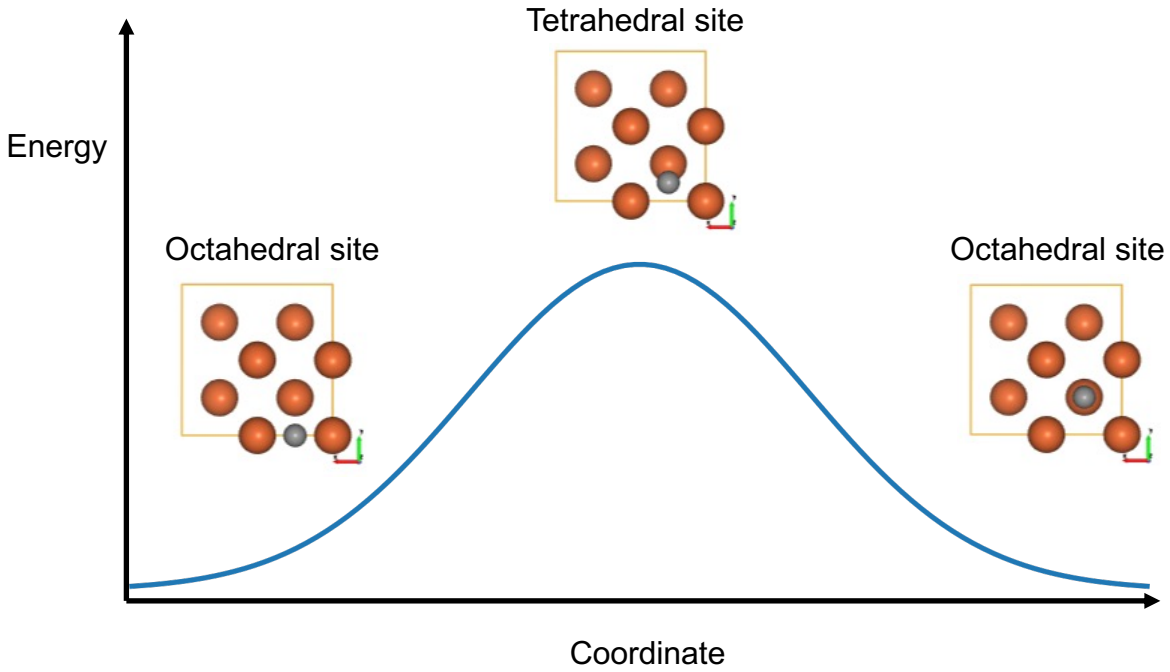
solid line – fit range from 3.0 to 9.0
dashed line – fit radii around 4.6

SNAP Descriptors Combing Multiple Radii

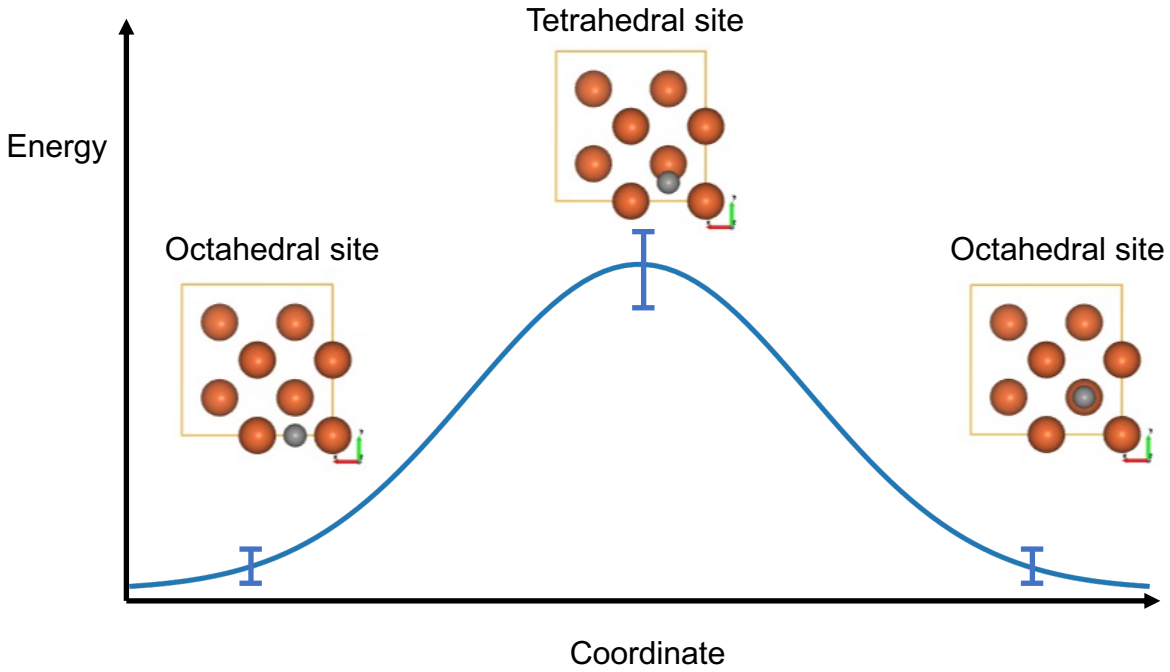


Learning 3: The SNAP descriptor is insufficient to study materials for extreme environments with *ab-initio* precision.

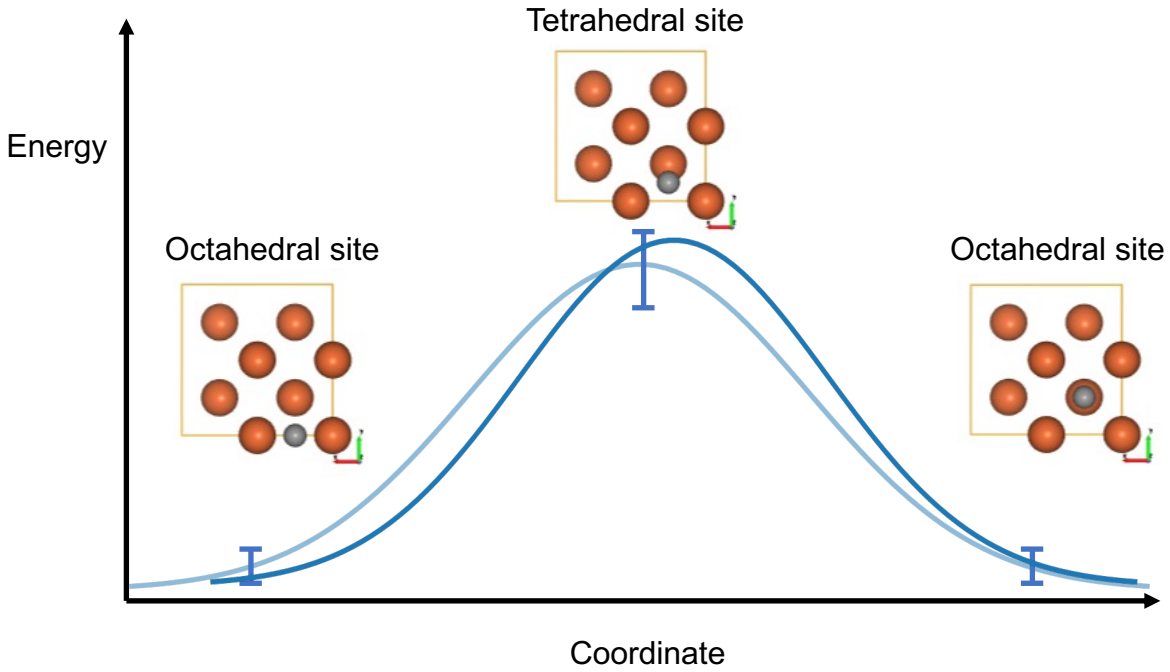
Extension to Transition States with ab-initio Precision



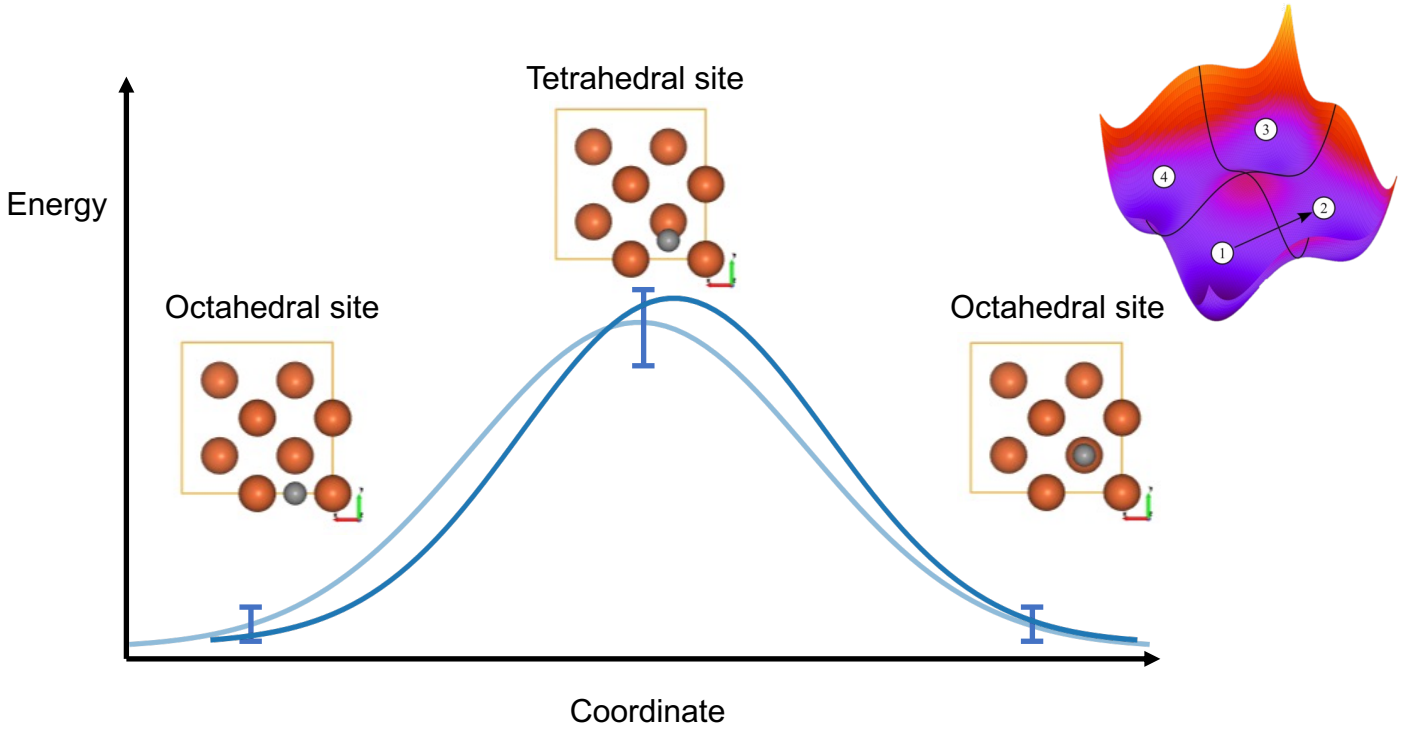
Extension to Transition States with ab-initio Precision



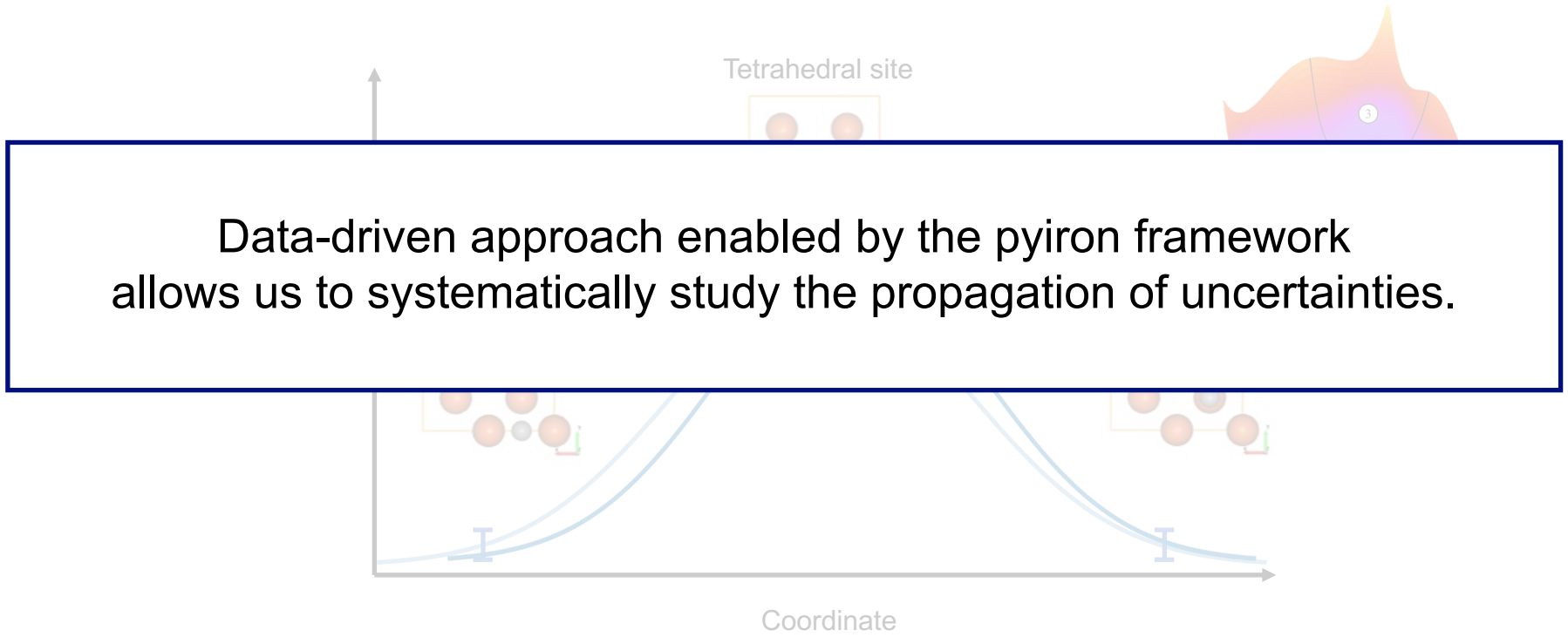
Extension to Transition States with ab-initio Precision



Extension to Transition States with ab-initio Precision



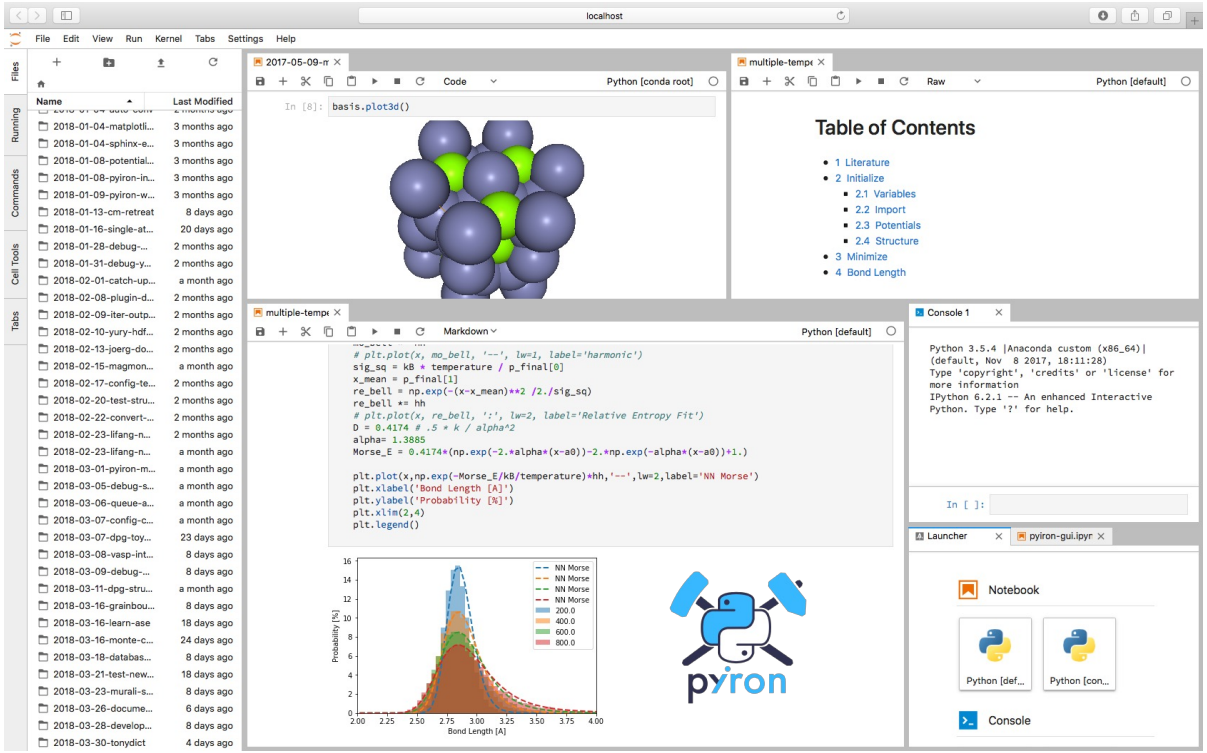
Extension to Transition States with ab-initio Precision



Data-driven approach enabled by the pyiron framework
allows us to systematically study the propagation of uncertainties.

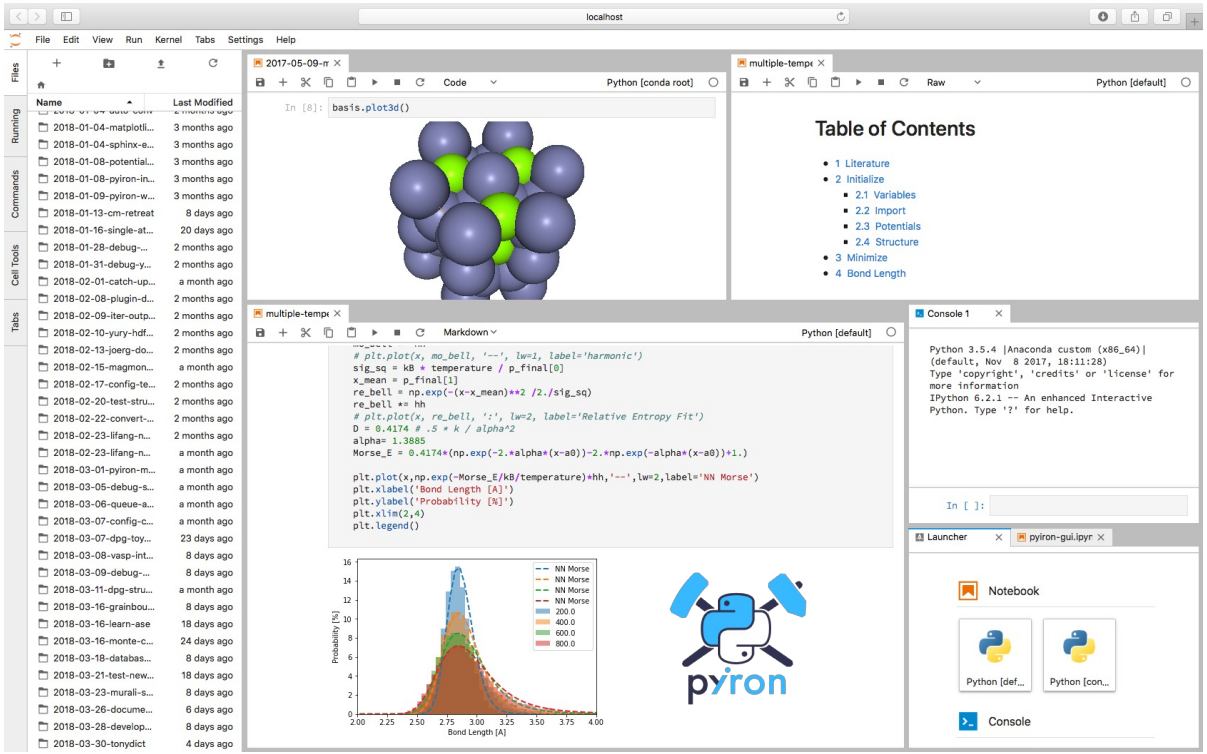
Rapid Prototyping and Up-scaling with pyiron

Jupyterlab



Rapid Prototyping and Up-scaling with pyiron

Jupyterlab



MAX-PLANCK-INSTITUT FÜR EISENFORSCHUNG

ICAMS INTERDISCIPLINARY CENTRE FOR ADVANCED MATERIALS SIMULATION

FACHGEBIET MATERIALMODELLIERUNG

Skoltech Skolkovo Institute of Science and Technology

THE UNIVERSITY OF BRITISH COLUMBIA

CENTER FOR MOLECULAR MODELING

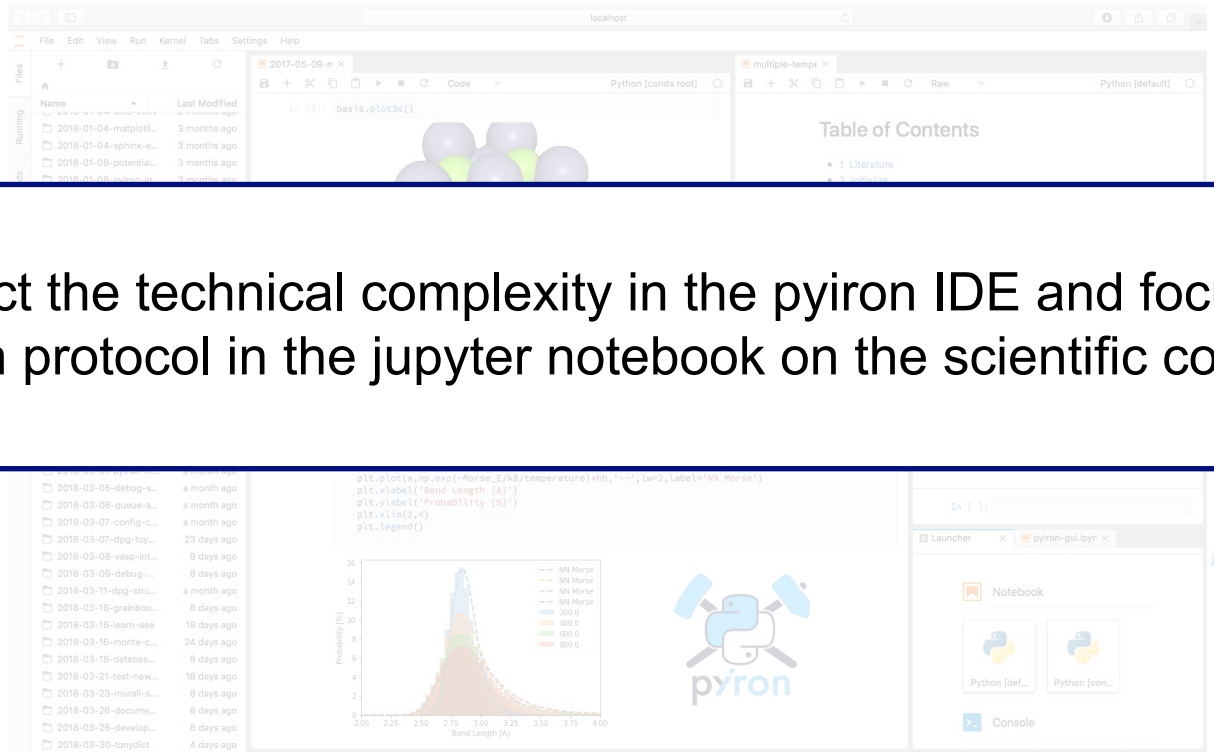
MONTAN UNIVERSITÄT LEOBEN

Los Alamos NATIONAL LABORATORY

J. Janssen, et al., Comp. Mat. Sci. 161 (2019) - http://pyiron.org

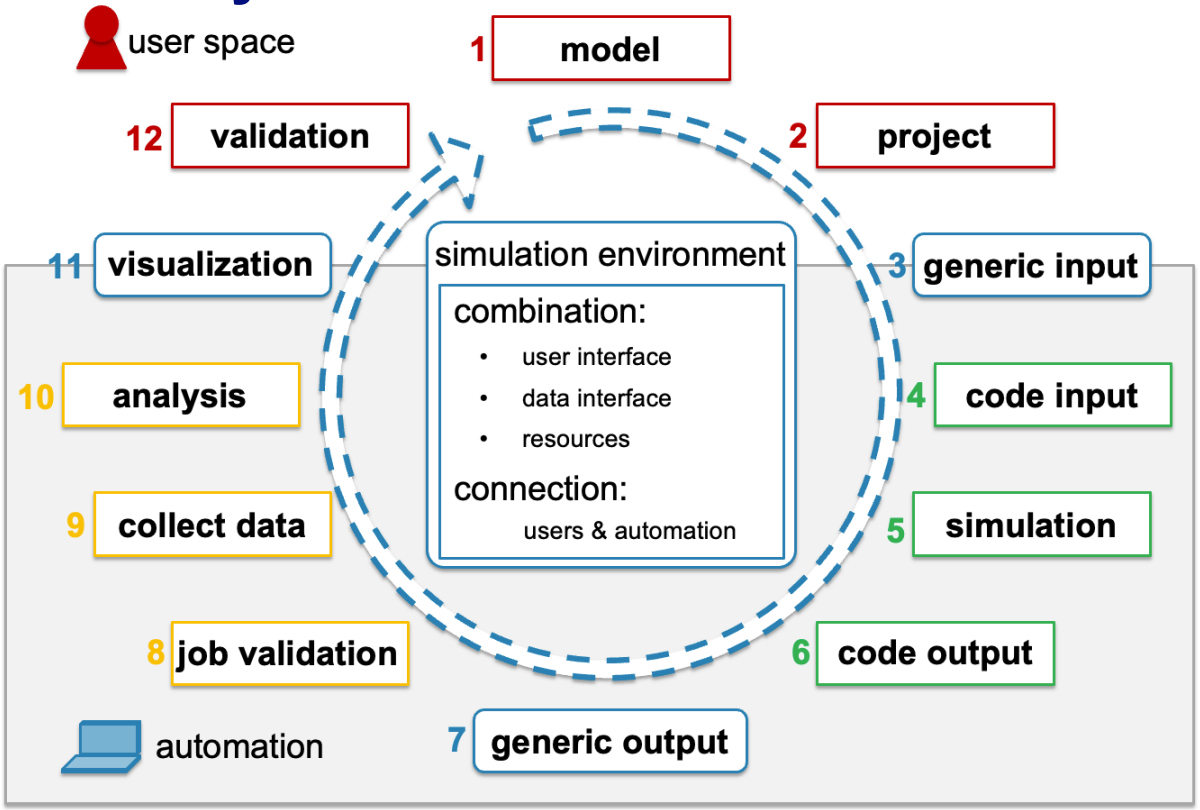
Rapid Prototyping and Up-scaling with pyiron

Jupyterlab

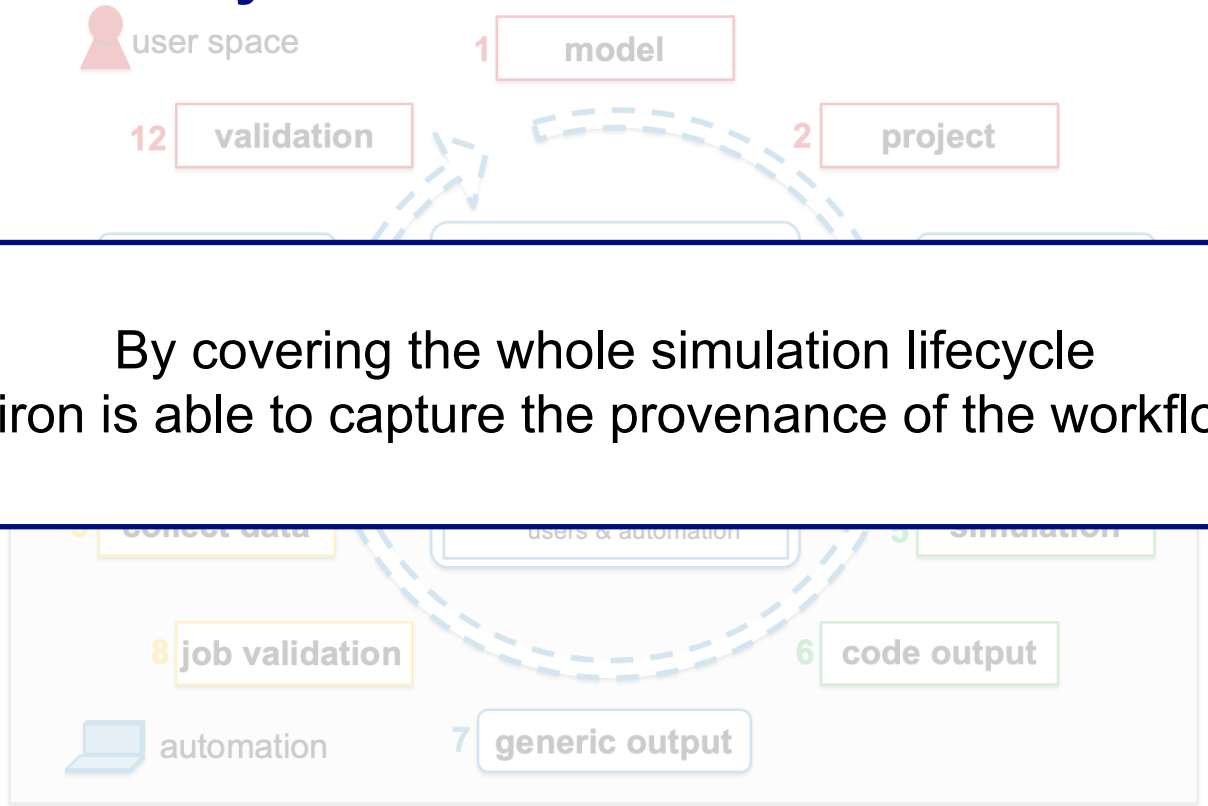


Abstract the technical complexity in the pyiron IDE and focus the simulation protocol in the jupyter notebook on the scientific complexity.

Simulation Lifecycle

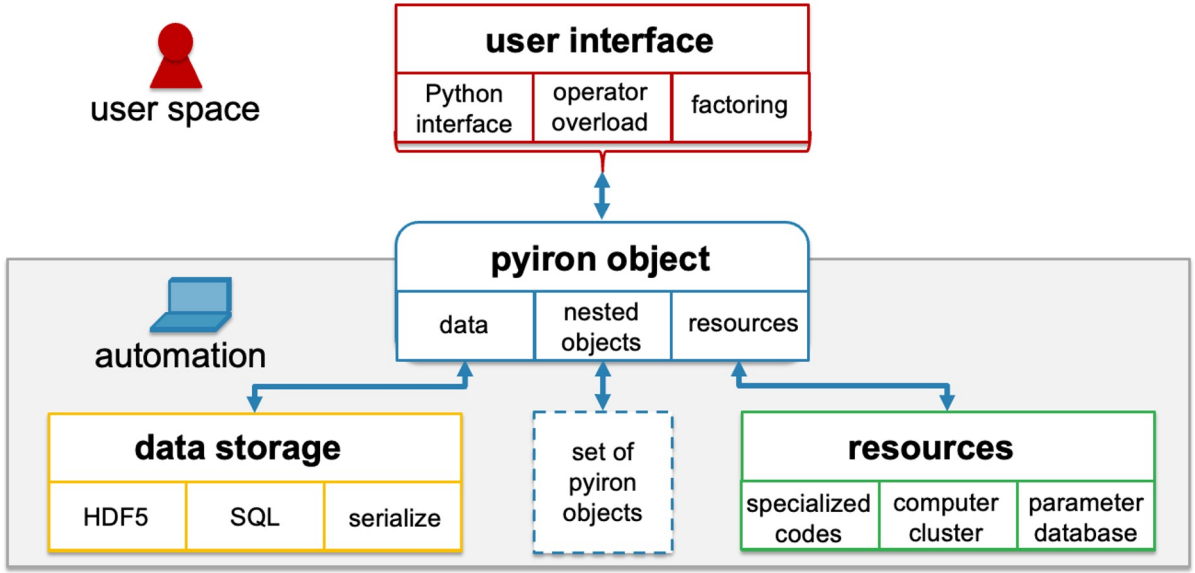


Simulation Lifecycle



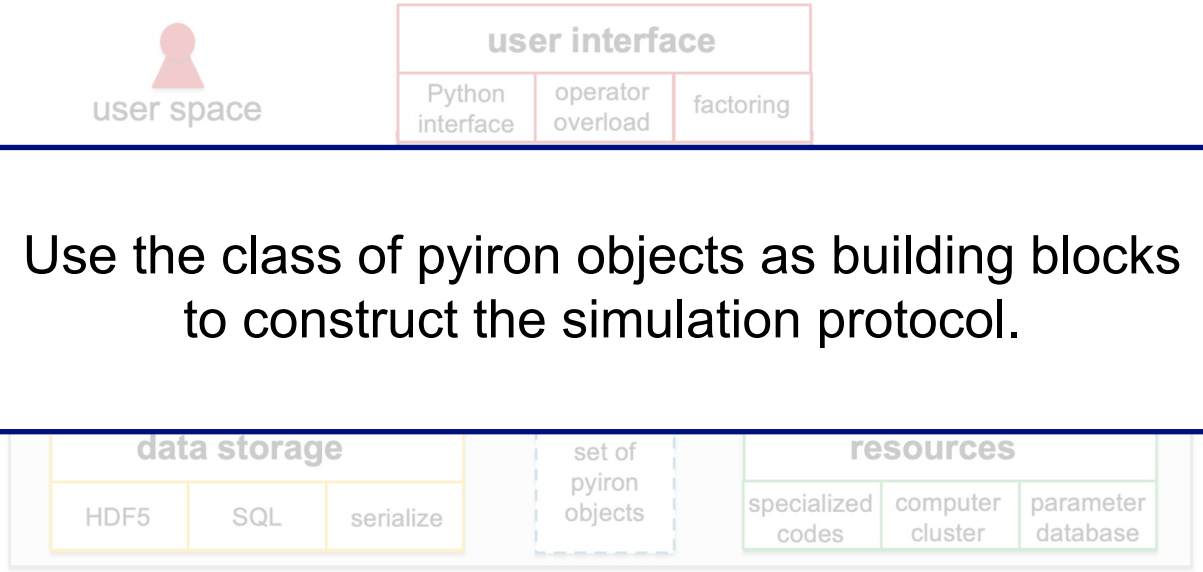
By covering the whole simulation lifecycle
pyiron is able to capture the provenance of the workflow.

Building Blocks



J. Janssen, et al., Comp. Mat. Sci. 161 (2019) - http://pyiron.org

Building Blocks



Use the class of pyiron objects as building blocks to construct the simulation protocol.

Example Calculation

```
In [1]: from pyiron import Project

In [2]: pr = Project("demonstration")

In [3]: job = pr.create.job.Lammps("my_calculation")

In [4]: job.structure = pr.create.structure.ase.bulk("Cu")

In [5]: # job.list_potentials()

In [6]: job.potential = "2012--Mendelev-M-I--Cu--LAMMPS--ipr1"

In [7]: job.server.cores = 2

In [8]: job.run()
The job my_calculation was saved and received the ID: 263
```

J. Janssen, et al., Comp. Mat. Sci. 161 (2019) - http://pyiron.org

Example Calculation

```
In [1]: from pyiron import Project

In [2]: pr = Project("demonstration")

In [3]: job = pr.create.job.Lammps("my_calculation")

In [4]: job.structure = pr.create.structure.ase.bulk("Cu")

In [5]: # job.list_potentials()

In [6]: job.potential = "2012--Mendelev-M-I--Cu--LAMMPS--ipr1"

In [7]: job.server.cores = 2

In [8]: job.run()
The job my_calculation was saved and received the ID: 263
```

J. Janssen, et al., Comp. Mat. Sci. 161 (2019) - http://pyiron.org

Example Calculation

```
In [1]: from pyiron import Project

In [2]: pr = Project("demonstration")

In [6]: job.potential = "2012--Mendelev-M-1--Cu--LAMMPS--ipr1"

In [7]: job.server.cores = 2

In [8]: job.run()
        The job my_calculation was saved and received the ID: 263
```

The resources (executable, interatomic potentials, queuing system, …) are configured once and can be used in every simulation protocol.

Equation of State

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve")
element = "Al"
```

In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

```
The job calc_0 was saved and received the ID: 60
The job calc_1 was saved and received the ID: 61
The job calc_2 was saved and received the ID: 62
The job calc_3 was saved and received the ID: 63
The job calc_4 was saved and received the ID: 64
The job calc_5 was saved and received the ID: 65
```

Equation of State

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve")
element = "Al"
```

In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

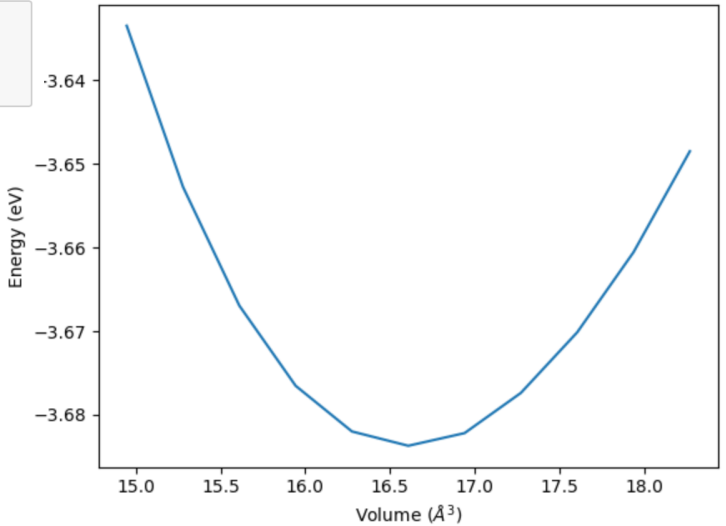
```
The job calc_0 was saved and received the ID: 60
The job calc_1 was saved and received the ID: 61
The job calc_2 was saved and received the ID: 62
The job calc_3 was saved and received the ID: 63
The job calc_4 was saved and received the ID: 64
The job calc_5 was saved and received the ID: 65
```

Equation of State

```
In [4]: pr["calc_0/output/generic/energy_tot"], pr["calc_0/output/generic/volume"]
Out[4]: (array([-3.63346467]), array([14.94677804]))
```

Equation of State

```
In [4]: pr["calc_0/output/generic/energy_tot"], pr["calc_0/output/generic/volume"]
Out[4]: (array([-3.63346467]), array([14.94677804]))

In [5]: volume_lst = [job["output/generic/volume"] for job in pr.iter_jobs()]
        energy_lst = [job["output/generic/energy_tot"] for job in pr.iter_jobs()]

In [6]: plt.plot(volume_lst, energy_lst)
        plt.xlabel("Volume $(\AA^3)$")
        plt.ylabel("Energy (eV)");
```



Equation of State

```
In [4]: pr["calc_0/output/generic/energy_tot"], pr["calc_0/output/generic/volume"]
Out[4]: (array([-3.63346467]), array([14.94677804]))

In [5]: volume_lst = [job["output/generic/volume"] for job in pr.iter_jobs()]
        energy_lst = [job["output/generic/energy_tot"] for job in pr.iter_jobs()]
```

Data management is automatically handled in the background for scientists to focus on the scientific complexity.

Up-scaling Simulation Protocols

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve_queue")
element = "Al"
```

In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.queue = "cm"
    job.server.cores = 40
    job.server.run_time = 600
    job.run()
```

```
The job calc_0 was saved and received the ID: 19674003
Queue system id:  5430087
The job calc_1 was saved and received the ID: 19674004
Queue system id:  5430088
The job calc_2 was saved and received the ID: 19674005
Queue system id:  5430089
```

Up-scaling Simulation Protocols

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve_queue")
element = "Al"
```

In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.queue = "cm"
    job.server.cores = 40
    job.server.run_time = 600
    job.run()
```

```
The job calc_0 was saved and received the ID: 19674003
Queue system id:  5430087
The job calc_1 was saved and received the ID: 19674004
Queue system id:  5430088
The job calc_2 was saved and received the ID: 19674005
Queue system id:  5430089
```

Up-scaling Simulation Protocols

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve_queue")
element = "Al"
```

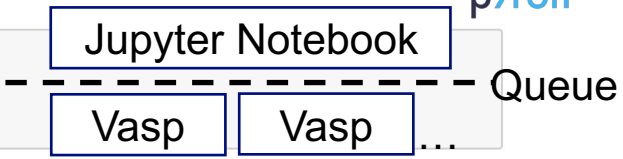
In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.queue = "cm"
    job.server.cores = 40
    job.server.run_time = 600
    job.run()
```

```
job.server.view_queues()
```

	cores_max	cores_min	run_time_max	memory_max
cm	1200	1	345600	None
cmfe	1200	1	345600	None
cmti	1200	1	345600	None
s_cmfe	40	1	345600	None
p_cmfe	1200	40	345600	None
cmti_large	1200	40	345600	None

```
The job calc_0 was saved and received the ID: 19674003
Queue system id:  5430087
The job calc_1 was saved and received the ID: 19674004
Queue system id:  5430088
The job calc_2 was saved and received the ID: 19674005
Queue system id:  5430089
```

Up-scaling Simulation Protocols



In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve_queue")
element = "Al"
```

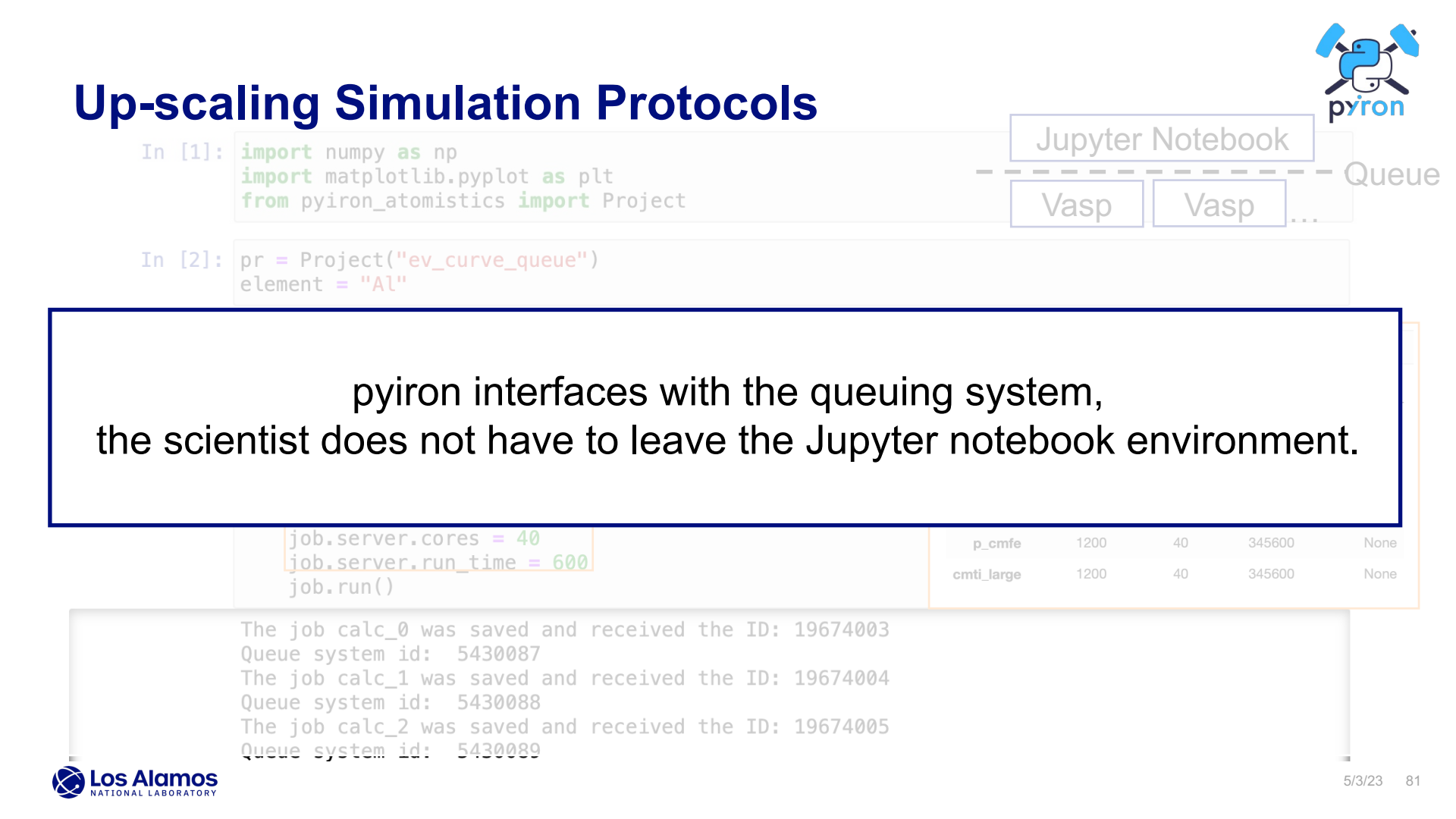
In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.queue = "cm"
    job.server.cores = 40
    job.server.run_time = 600
    job.run()
```

```
job.server.view_queues()
```

	cores_max	cores_min	run_time_max	memory_max
cm	1200	1	345600	None
cmfe	1200	1	345600	None
cmti	1200	1	345600	None
s_cmfe	40	1	345600	None
p_cmfe	1200	40	345600	None
cmti_large	1200	40	345600	None

```
The job calc_0 was saved and received the ID: 19674003
Queue system id:  5430087
The job calc_1 was saved and received the ID: 19674004
Queue system id:  5430088
The job calc_2 was saved and received the ID: 19674005
Queue system id:  5430089
```

Up-scaling Simulation Protocols



pyiron interfaces with the queuing system,
the scientist does not have to leave the Jupyter notebook environment.

Built-in Caching

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("ev_curve")
element = "Al"
```

In [3]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

```
2023-05-01 09:18:05,303 - pyiron_log - WARNING - The job calc_0 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:06,097 - pyiron_log - WARNING - The job calc_1 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:06,948 - pyiron_log - WARNING - The job calc_2 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:07,781 - pyiron_log - WARNING - The job calc_3 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:08,588 - pyiron_log - WARNING - The job calc_4 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:09,383 - pyiron_log - WARNING - The job calc_5 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
```

Built-in Caching

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from pyiron_atomistics import Project

In [2]: pr = Project("ev_curve")
        element = "Al"
```

Based on the semantic path existing calculations can be reloaded to reuse the same Jupyter notebook for submission and analysis of calculation.

```
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:06,097 - pyiron_log - WARNING - The job calc_1 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:06,948 - pyiron_log - WARNING - The job calc_2 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:07,781 - pyiron_log - WARNING - The job calc_3 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:08,588 - pyiron_log - WARNING - The job calc_4 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
2023-05-01 09:18:09,383 - pyiron_log - WARNING - The job calc_5 is being loaded instead of
running. To re-run use the argument 'delete_existing_job=True in create_job'
```

Data Aggregation

```
In [7]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.run()
        df = table.get_dataframe()
        df
```

```
The job table was saved and received the ID: 71
```

Out[7]:

	job_id	energy_tot	volume
0	60	-3.633465	14.946778
1	61	-3.652753	15.278929
2	62	-3.666954	15.611079
3	63	-3.676548	15.943230
4	64	-3.682013	16.275381
5	65	-3.683725	16.607531
6	66	-3.682209	16.939682
7	67	-3.677397	17.271832

Data Aggregation

```
In [7]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.run()
        df = table.get_dataframe()
        df
```

```
The job table was saved and received the ID: 71
```

Out[7]:

	job_id	energy_tot	volume
0	60	-3.633465	14.946778
1	61	-3.652753	15.278929
2	62	-3.666954	15.611079
3	63	-3.676548	15.943230
4	64	-3.682013	16.275381
5	65	-3.683725	16.607531
6	66	-3.682209	16.939682
7	67	-3.677397	17.271832

Data Aggregation

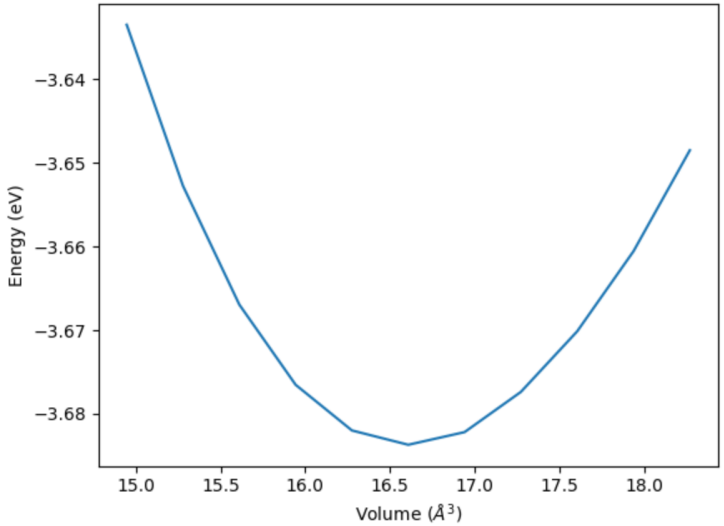
```
In [7]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.run()
        df = table.get_dataframe()
        df
```

```
The job table was saved and re
```

Out[7]:

	job_id	energy_tot	volume
0	60	-3.633465	14.946778
1	61	-3.652753	15.278929
2	62	-3.666954	15.611079
3	63	-3.676548	15.943230
4	64	-3.682013	16.275381
5	65	-3.683725	16.607531
6	66	-3.682209	16.939682
7	67	-3.677397	17.271832

```
In [8]: plt.plot(df["volume"], df["energy_tot"])
        plt.xlabel("Volume $(\AA^3)$")
        plt.ylabel("Energy (eV)");
```

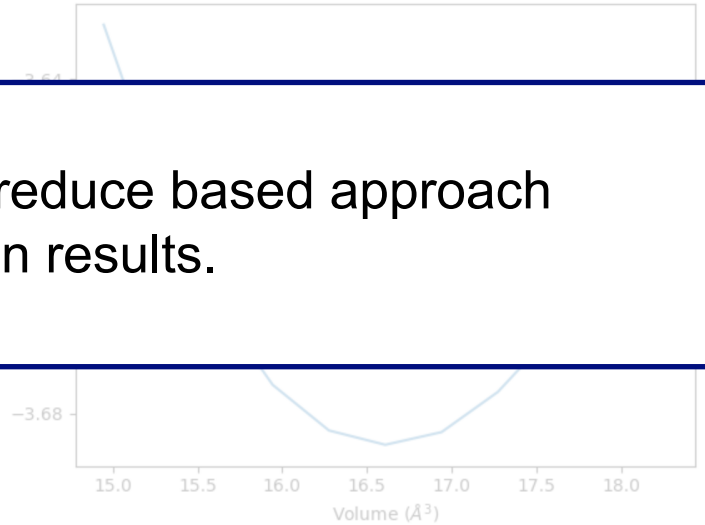


Data Aggregation

```
In [7]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.run()
        df = table.get_dataframe()
        df
```

3	63	-3.676548	15.943230
4	64	-3.682013	16.275381
5	65	-3.683725	16.607531
6	66	-3.682209	16.939682
7	67	-3.677397	17.271832

```
In [8]: plt.plot(df["volume"], df["energy_tot"])
        plt.xlabel("Volume $(\AA^3)$")
        plt.ylabel("Energy (eV)");
```



The pyiron table implements a map-reduce based approach to aggregate simulation results.

Next Layer of Abstraction

In [1]:
```
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("parallel_master")
element_lst = ["Al", "Cu"]
```

In [3]:
```
for element in element_lst:
    job = pr.create.job.Vasp("calc")
    job.structure = pr.create.structure.ase.bulk(element)
    job.server.cores = 8
    murn = job.create_job(pr.job_type.Murnaghan, element)
    murn.server.queue = "cm"
    murn.server.cores = 40
    murn.run()
```

```
The job Al was saved and received the ID: 19675056
Queue system id:  5431127
The job Cu was saved and received the ID: 19675057
Queue system id:  5431128
```

Next Layer of Abstraction

In [1]:
```
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("parallel_master")
element_lst = ["Al", "Cu"]
```

In [3]:
```
for element in element_lst:
    job = pr.create.job.Vasp("calc")
    job.structure = pr.create.structure.ase.bulk(element)
    job.server.cores = 8
    murn = job.create_job(pr.job_type.Murnaghan, element)
    murn.server.queue = "cm"
    murn.server.cores = 40
    murn.run()
```

```
The job Al was saved and received the ID: 19675056
Queue system id:  5431127
The job Cu was saved and received the ID: 19675057
Queue system id:  5431128
```

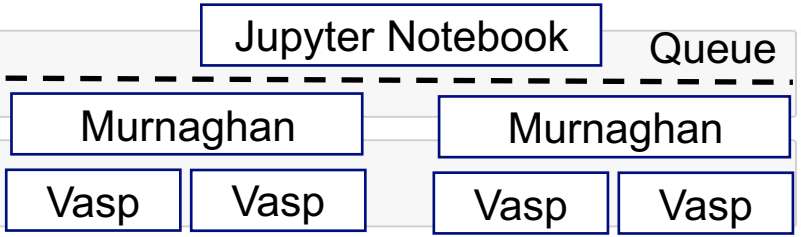
Next Layer of Abstraction

In [1]:
```
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("parallel_master")
element_lst = ["Al", "Cu"]
```

In [3]:
```
for element in element_lst:
    job = pr.create.job.Vasp("calc")
    job.structure = pr.create.structure.ase.bulk(element)
    job.server.cores = 8
    murn = job.create_job(pr.job_type.Murnaghan, element)
    murn.server.queue = "cm"
    murn.server.cores = 40
    murn.run()
```

```
The job Al was saved and received the ID: 19675056
Queue system id:  5431127
The job Cu was saved and received the ID: 19675057
Queue system id:  5431128
```



Next Layer of Abstraction

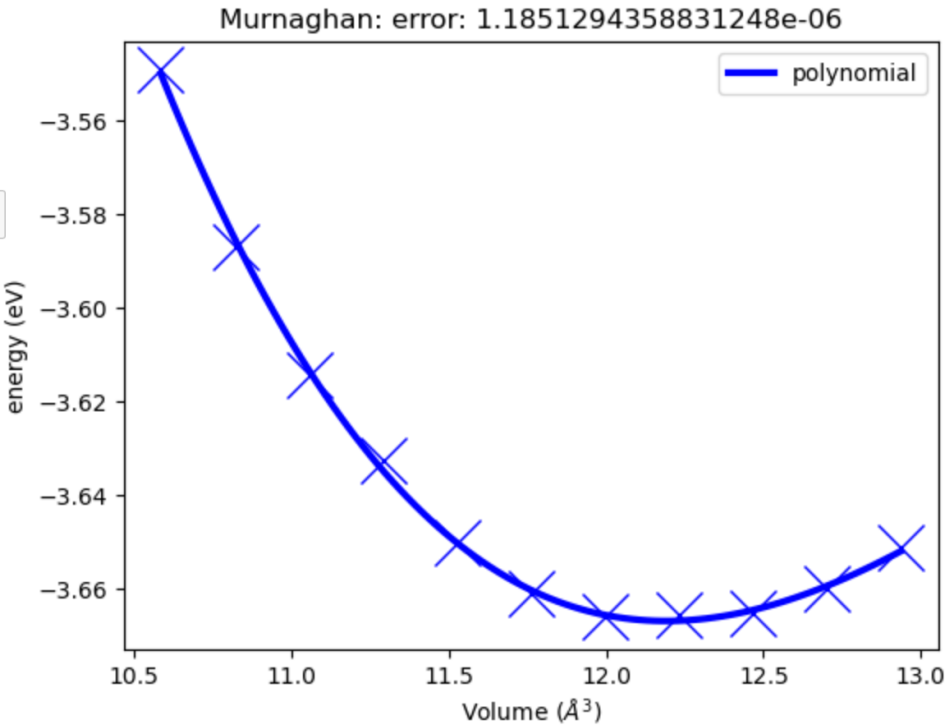
```
In [4]: murn["output/volume"], murn["output/energy"]

Out[4]: (array([10.58532324, 10.82055261, 11.05578211, 11.29101148, 11.52624083,
                11.76147025, 11.99669964, 12.23192898, 12.46715837, 12.70238786,
                12.93761731]),
         array([-3.54904145, -3.587095  , -3.61442632, -3.6328282 , -3.65022603,
                -3.66104209, -3.66623825, -3.66568193, -3.66544783, -3.66029473,
                -3.65151576]))
```

Next Layer of Abstraction

```
In [4]: murn["output/volume"], murn["output/energy"]
```

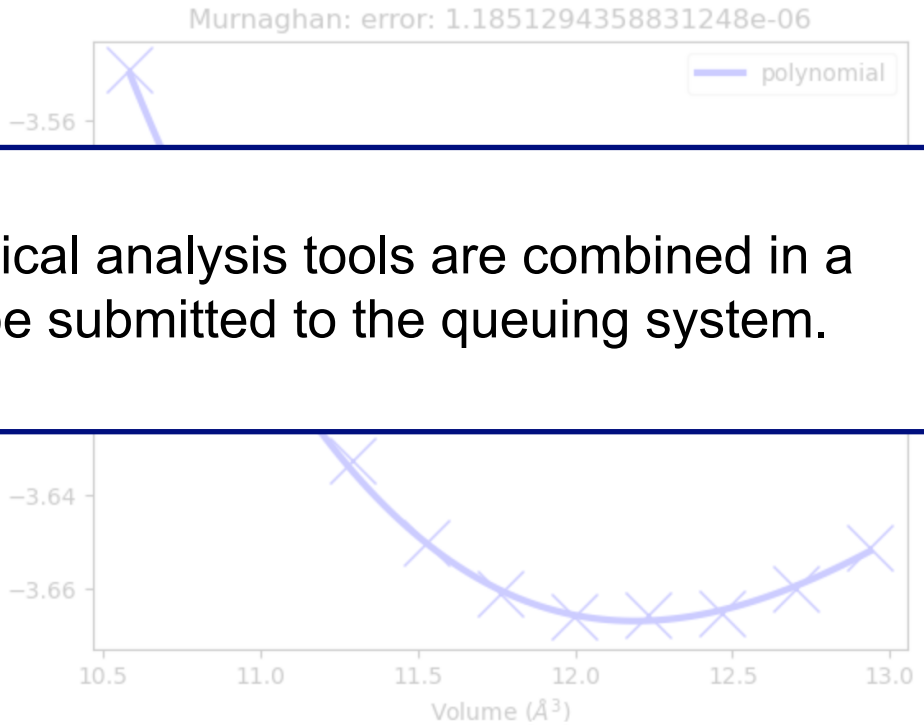
```
Out[4]: (array([10.58532324, 10.82055261,
                11.76147025, 11.99669964,
                12.93761731]),
         array([-3.54904145, -3.587095  ,
                -3.66104209, -3.66623825,
                -3.65151576]))
```

```
In [5]: murn.plot()
```



Next Layer of Abstraction

```
In [4]: murn["output/volume"], murn["output/energy"]

Out[4]: (array([10.58532324, 10.82055261,
                11.76147025, 11.99669964,
                12.93761731]),
         array([-3.54904145, -3.587095  ,
                -3.66104209, -3.66623825,
                -3.65151576]))
```



Job management as well as typical analysis tools are combined in a reusable job class which can be submitted to the queuing system.

Reusable Templates

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [ ]:
```
pr = Project("ev_curve")
element = pr.get_external_input()["element"]
```

In [ ]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

Reusable Templates

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [ ]:
```
pr = Project("ev_curve")
element = pr.get_external_input()["element"]
```

In [ ]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

Reusable Templates

In [1]:
```
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [2]:
```
pr = Project("parameter_study")
element_lst = ["Al", "Cu"]
```

In [3]:
```
for element in element_lst:
    job = pr.create.job.ScriptJob(element)
    job.input["element"] = element
    job.script_path = "equation_of_state_element_v1.ipynb"
    job.run()
```

```
The job Al was saved and received the ID: 72

100% 1/1 [00:00<00:00, 23.03it/s]

The job Cu was saved and received the ID: 84

100% 2/2 [00:00<00:00, 67.72it/s]
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [ ]:
```
pr = Project("ev_curve")
element = pr.get_external_input()["element"]
```

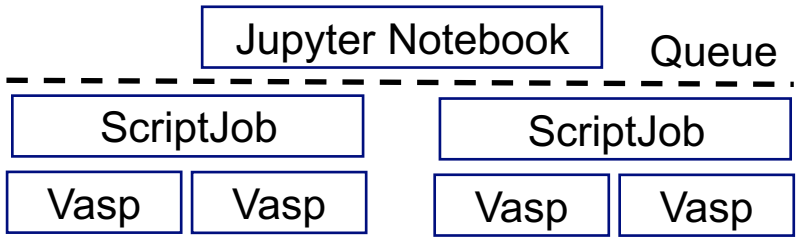
In [ ]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.run()
```

Reusable Templates

```
In [1]: import matplotlib.pyplot as plt
        from pyiron_atomistics import Project

In [2]: pr = Project("parameter_study")
        element_lst = ["Al", "Cu"]

In [3]: for element in element_lst:
            job = pr.create.job.ScriptJob(element)
            job.input["element"] = element
            job.script_path = "equation_of_state_element_v1.ipynb"
            job.run()
```

```
The job Al was saved and received the ID: 72
100% ████████████████████████████████████████
The job Cu was saved and received the ID: 84
100% ████████████████████████████████████████
```

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        from pyiron_atomistics import Project

In [ ]: pr = Project("ev_curve")
        element = pr.get_external_input()["element"]

In [ ]: structure = pr.create.structure.ase.bulk(element)
        for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
            volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
            structure_strain = structure.copy()
            structure_strain.apply_strain(epsilon=volume_strain)
            job = pr.create.job.Vasp("calc_" + str(i))
            job.structure = structure_strain
            job.run()
```



Reusable Templates

In [4]:
```
table = pr.create.table()
table.add.get_volume
table.add.get_energy_tot
table.add.get_elements
table.run()
```

```
The job table was saved and received the ID: 96
```

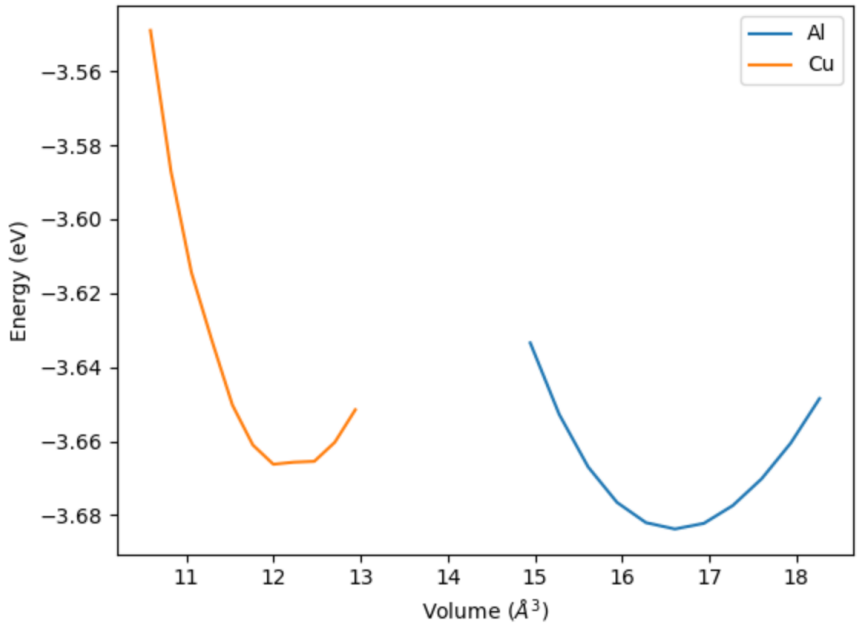
Loading and filtering jobs: 100% 25/25 [00:00<00:00, 297.41it/s]

Processing jobs: 100% 24/24 [00:00<00:00, 47.29it/s]

Reusable Templates

In [4]:
```
table = pr.create.table()
table.add.get_volume
table.add.get_energy_tot
table.add.get_elements
table.run()
```

In [5]:
```
df = table.get_dataframe()
for element in element_lst:
    df_el = df[df[element]==1.0]
    plt.plot(df_el["volume"], df_el["energy_tot"], label=element)
plt.xlabel("Volume $(\AA^3)$")
plt.ylabel("Energy (eV)")
plt.legend();
```

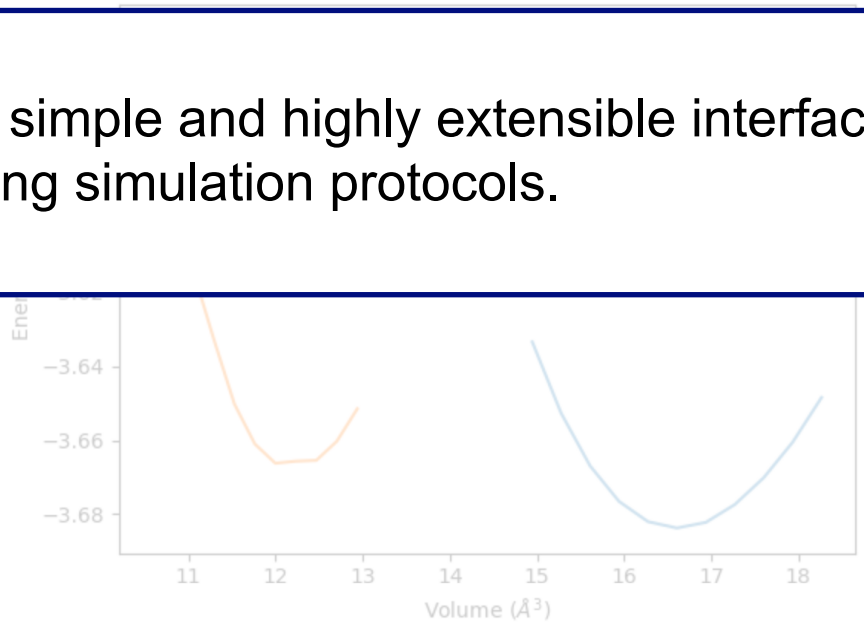


Reusable Templates

```
In [4]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.add.get_elements
        table.run()
```

```
In [5]: df = table.get_dataframe()
        for element in element_lst:
            df_el = df[df[element]==1.0]
            plt.plot(df_el["volume"], df_el["energy_tot"], label=element)
        plt.xlabel("Volume $(\AA^3)$")
        plt.ylabel("Energy (eV)")
        plt.legend();
```

The ScriptJob provides a simple and highly extensible interface for up-scaling simulation protocols.



Task Management within the Allocation

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [ ]:
```python
pr = Project("ev_curve")
element = pr.get_external_input()["element"]
```

In [ ]:
```python
pr_work = Project("worker")
job_worker = pr_work.create.job.WorkerJob("runner")
job_worker.server.run_mode.thread = True
job_worker.project_to_watch = pr
job_worker.run()
```

In [ ]:
```python
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.run_mode.worker = True
    job.master_id = job_worker.job_id
    job.run()
```

Task Management within the Allocation

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyiron_atomistics import Project
```

In [ ]:
```
pr = Project("ev_curve")
element = pr.get_external_input()["element"]
```

In [ ]:
```
pr_work = Project("worker")
job_worker = pr_work.create.job.WorkerJob("runner")
job_worker.server.run_mode.thread = True
job_worker.project_to_watch = pr
job_worker.run()
```

In [ ]:
```
structure = pr.create.structure.ase.bulk(element)
for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.run_mode.worker = True
    job.master_id = job_worker.job_id
    job.run()
```

Task Management within the Allocation

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        from pyiron_atomistics import Project

In [ ]: pr = Project("ev_curve")
        element = pr.get_external_input()["element"]
```

The WorkerJob distributes the individual calculation within the queuing system allocation.

```
    volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
    structure_strain = structure.copy()
    structure_strain.apply_strain(epsilon=volume_strain)
    job = pr.create.job.Vasp("calc_" + str(i))
    job.structure = structure_strain
    job.server.run_mode.worker = True
    job.master_id = job_worker.job_id
    job.run()
```

Function Container

In [3]:
```python
def create_worker(pr_work, pr_to_watch):
    pr_work = Project(pr_work)
    job_worker = pr_work.create.job.WorkerJob("runner")
    job_worker.server.run_mode.thread = True
    job_worker.project_to_watch = pr_to_watch
    job_worker.run()
    return job_worker.job_id
```

In [4]:
```python
def submit_energy_volume_curve(pr, element, worker_id):
    structure = pr.create.structure.ase.bulk(element)
    for i, strain in enumerate(np.linspace(-0.1, 0.1, 11)):
        volume_strain = ((1 + strain) ** (1.0 / 3) - 1)
        structure_strain = structure.copy()
        structure_strain.apply_strain(epsilon=volume_strain)
        job = pr.create.job.Vasp("calc_" + str(i))
        job.structure = structure_strain
        job.server.run_mode.worker = True
        job.master_id = worker_id
        job.run()
```

In [5]:
```python
def calc_energy_volume_curve(pr, element):
    pr_element = pr.create_group(element)
    pr_worker = pr.create_group(element + "_worker")
    worker_id = create_worker(pr_work=pr_worker, pr_to_watch=pr_element)
    submit_energy_volume_curve(pr=pr_element, element=element, worker_id=worker_id)
    pr_element.wait_for_jobs()
```

Function Container

```
In [6]: fc = pr.create.job.FunctionContainer("fc")
        fc.function = calc_energy_volume_curve
        fc.input = [{"element": element, "pr": pr} for element in element_lst]
        fc.run()
```

```
The job fc was saved and received the ID: 60
The job runner was saved and received the ID: 61
The job calc_0 was saved and received the ID: 62
The job calc_1 was saved and received the ID: 63
The job calc_2 was saved and received the ID: 64
The job calc_3 was saved and received the ID: 65
The job calc_4 was saved and received the ID: 66
The job calc_5 was saved and received the ID: 67
The job calc_6 was saved and received the ID: 68
The job calc_7 was saved and received the ID: 69
The job calc_8 was saved and received the ID: 70
The job calc_9 was saved and received the ID: 71
The job calc_10 was saved and received the ID: 72
The job runner was saved and received the ID: 73
The job calc_0 was saved and received the ID: 74
The job calc_1 was saved and received the ID: 75
The job calc_2 was saved and received the ID: 76
The job calc_3 was saved and received the ID: 77
The job calc_4 was saved and received the ID: 78
```

Function Container

```
In [6]: fc = pr.create.job.FunctionContainer("fc")
        fc.function = calc_energy_volume_curve
        fc.input = [{"element": element, "pr": pr} for element in element_lst]
        fc.run()
```

```
The job fc was saved and received the ID: 60
The job runner was saved and received the ID: 61
The job calc_0 was saved and received the ID: 62
The job calc_1 was saved and received the ID: 63
The job calc_2 was saved and received the ID: 64
The job calc_3 was saved and received the ID: 65
The job calc_4 was saved and received the ID: 66
The job calc_5 was saved and received the ID: 67
The job calc_6 was saved and received the ID: 68
The job calc_7 was saved and received the ID: 69
The job calc_8 was saved and received the ID: 70
The job calc_9 was saved and received the ID: 71
The job calc_10 was saved and received the ID: 72
The job runner was saved and received the ID: 73
The job calc_0 was saved and received the ID: 74
The job calc_1 was saved and received the ID: 75
The job calc_2 was saved and received the ID: 76
The job calc_3 was saved and received the ID: 77
The job calc_4 was saved and received the ID: 78
```

Function Container

In [6]:
```
fc = pr.create.job.FunctionContainer("fc")
fc.function = calc_energy_volume_curve
fc.input = [{"element": element, "pr": pr} for element in element_lst]
fc.run()
```

In [7]:
```
table = pr.create.table()
table.add.get_volume
table.add.get_energy_tot
table.add.get_elements
table.run()
```

```
The job table was saved and received the ID: 85
```

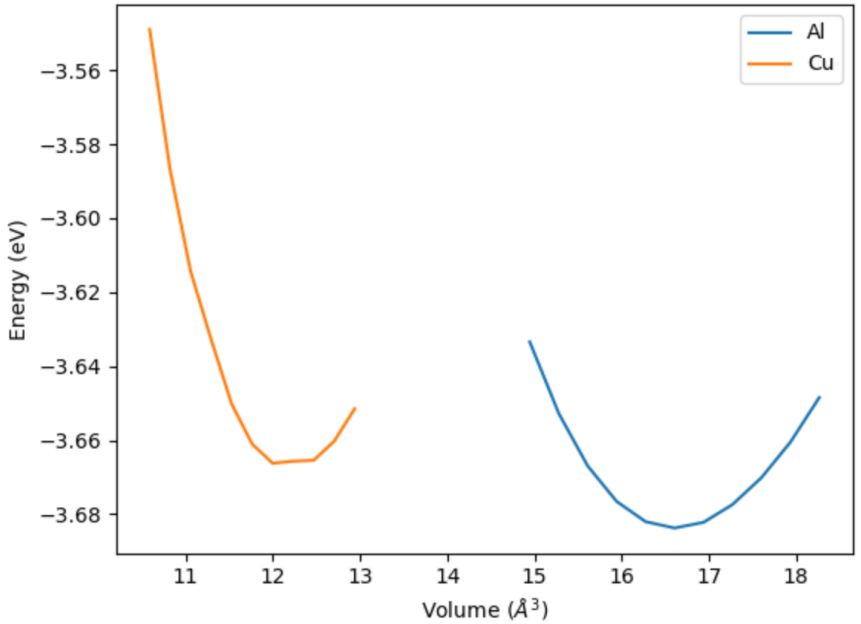
Loading and filtering jobs: 100% 26/26 [00:00<00:00, 471.46it/s]

Processing jobs: 100% 22/22 [00:00<00:00, 107.88it/s]

Function Container

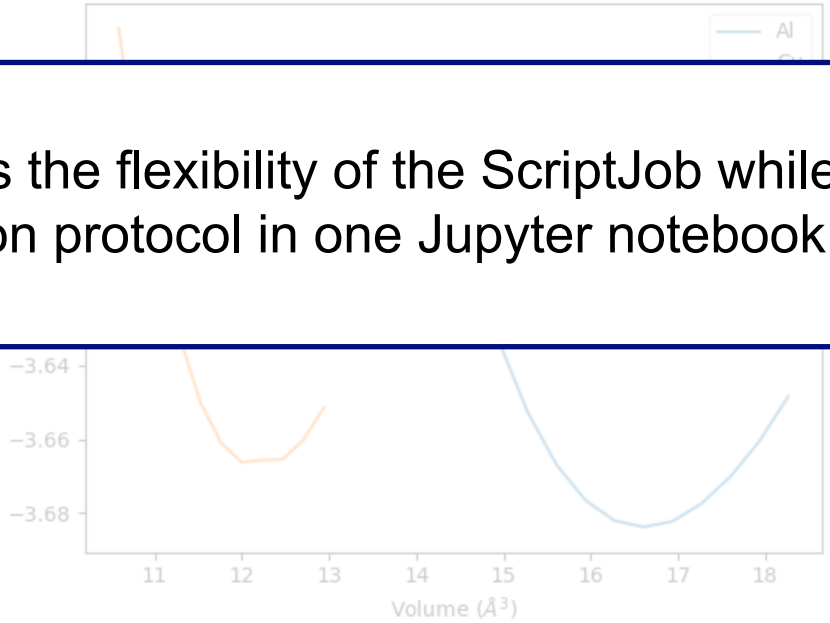
```
In [6]: fc = pr.create.job.FunctionContainer("fc")
        fc.function = calc_energy_volume_curve
        fc.input = [{"element": element, "pr": pr} for element in element_lst]
        fc.run()
```

```
In [7]: table = pr.create.table()
        table.add.get_volume
        table.add.get_energy_tot
        table.add.get_elements
        table.run()
```



Function Container

```
In [6]: fc = pr.create.job.FunctionContainer("fc")
        fc.function = calc_energy_volume_curve
        fc.input = [{"element": element, "pr": pr} for element in element_lst]
        fc.run()

In [7]: table = pr.create.table()
        table.add.get_volume
```



The FunctionContainer extends the flexibility of the ScriptJob while maintaining the whole simulation protocol in one Jupyter notebook.

Communication Overhead

A NERSC working group review and refresh of this content is currently in progress; we are actively updating these docs pages with new information as we evaluate new tools. In the meantime we request the following of users considering workflow management solutions:

- Before you begin developing a codebase which requires a particular workflow manager, please contact NERSC consultants via help.nersc.gov to confirm it can be effectively used at NERSC. Some tools have infrastructure needs or operate in a manner which is fundamentally incompatible with NERSC systems and we'd like to protect users from wasting effort if we can.
- Please do not write your own workflow manager. More than 200 such solutions already exist and almost certainly one of them can be found which will fit your needs and our infrastructure.
- Please don't do this!!!

```
For i=1=10,000
   srun -n 1 a.out
```

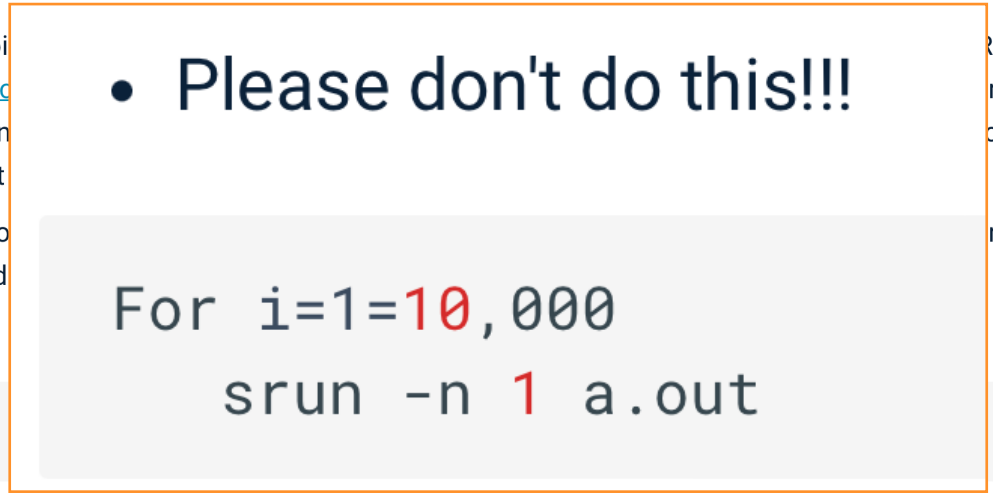
Issuing many `srun`s in a short period of time really stresses our SLURM scheduler. It will ruin not only your own job performance but also the performance for all other NERSC users, too. If this is what you need for your application, please consider a workflow tool. This is what they were designed to do!

Communication Overhead

A NERSC working group review and refresh of this content is currently in progress; we are actively updating these docs pages with new information as we evaluate new tools. In the meantime we request the following of users considering workflow management solutions:

- Before you begin developi… consultants via help.nersc… needs or operate in a man… users from wasting effort…
- Please do not write your o… one of them can be found…
- Please don't do this!!!

```
For i=1=10,000
   srun -n 1 a.out
```



Issuing many `srun`s in a short period of time really stresses our SLURM scheduler. It will ruin not only your own job performance but also the performance for all other NERSC users, too. If this is what you need for your application, please consider a workflow tool. This is what they were designed to do!

Communication Overhead

A NERSC working group review and refresh of this content is currently in progress; we are actively updating these docs pages with new information as we evaluate new tools. In the meantime we request the following of users considering workflow management solutions:

- Before you begin developi

Rather than using SLURM to manage individual calculation the WorkerJob is currently being transitioned to the flux-framework.

```
For i=1=10,000
   srun -n 1 a.out
```

```
srun -n 1 a.out
```

Issuing many `srun`s in a short period of time really stresses our SLURM scheduler. It will ruin not only your own job performance but also the performance for all other NERSC users, too. If this is what you need for your application, please consider a workflow tool. This is what they were designed to do!

Implement New Classes

In [1]:
```
from os.path import join
from pyiron_base import TemplateJob, Project
```

In [2]:
```
class ToyJob(TemplateJob):
    def __init__(self, project, job_name):
        super().__init__(project, job_name)
        self.input['input_energy'] = 100
        self.executable = "cat input > output"

    def write_input(self):
        self.input.write_file(
            file_name="input",
            cwd=self.working_directory
        )

    def collect_output(self):
        file = join(self.working_directory, "output")
        with open(file) as f:
            line = f.readlines()[0]
        energy = float(line.split()[1])
        with self.project_hdf5.open("output/generic") as h5out:
            h5out["energy_tot"] = energy
```

Implement New Classes

In [1]:
```
from os.path import join
from pyiron_base import TemplateJob, Project
```

In [2]:
```
class ToyJob(TemplateJob):
    def __init__(self, project, job_name):
        super().__init__(project, job_name)
        self.input['input_energy'] = 100
        self.executable = "cat input > output"

    def write_input(self):
        self.input.write_file(
            file_name="input",
            cwd=self.working_directory
        )

    def collect_output(self):
        file = join(self.working_directory, "output")
        with open(file) as f:
            line = f.readlines()[0]
        energy = float(line.split()[1])
        with self.project_hdf5.open("output/generic") as h5out:
            h5out["energy_tot"] = energy
```

`write_input()`:

input: python dictionary

output: write Input files

`collect_output()`:

input: directory of output

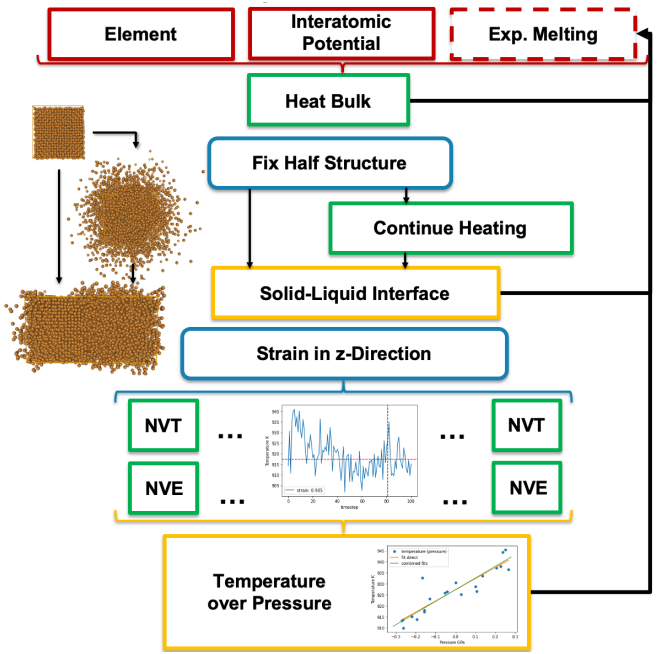
output: python dictionary

Implement New Classes

```
In [1]: from os.path import join
        from pyiron_base import TemplateJob, Project

In [2]: class ToyJob(TemplateJob):
            def __init__(self, project, job_name):
                write_input():
                input: directory of output
                output: python dictionary

            def collect_output(self):
                file = join(self.working_directory, "output")
                with open(file) as f:
                    line = f.readlines()[0]
                energy = float(line.split()[1])
                with self.project_hdf5.open("output/generic") as h5out:
                    h5out["energy_tot"] = energy
```

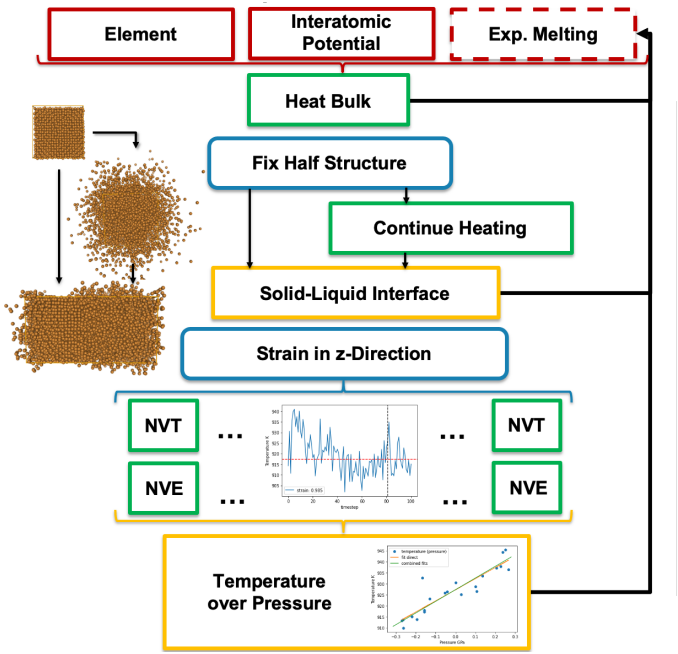
In the most basic case only a `write_input()` function and a `collect_output()` function are required to leverage pyiron.

Autonomous Calculation of Melting Temperatures



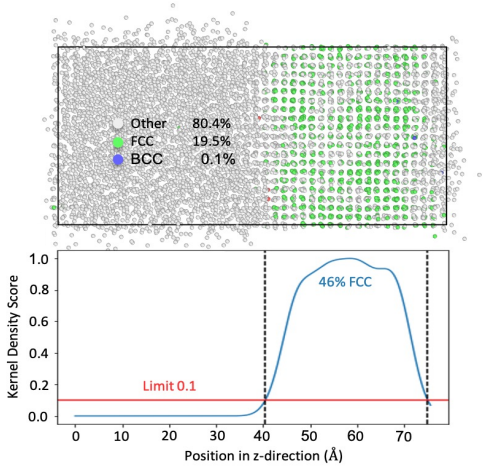
L.F. Zhu, **J. Janssen**, et al. Comp. Mat. Sci. 187 (2021)

Autonomous Calculation of Melting Temperatures



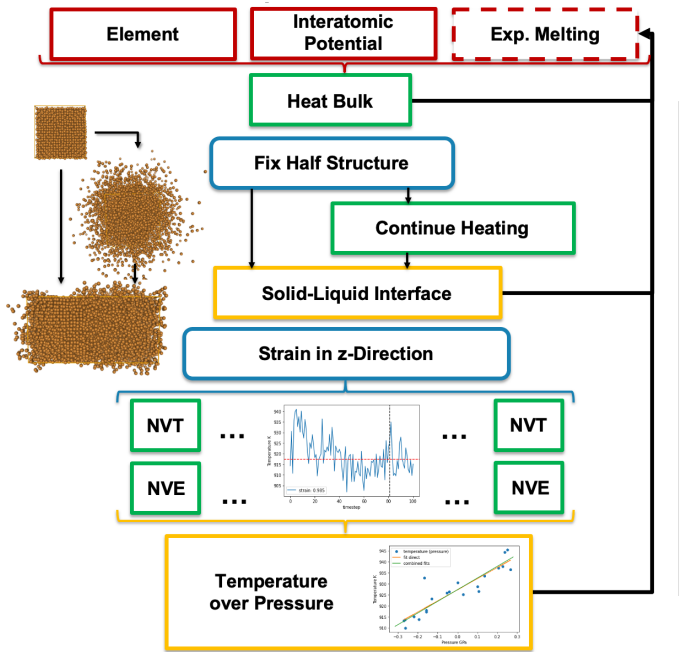
Challenges:

Solid Liquid Detector



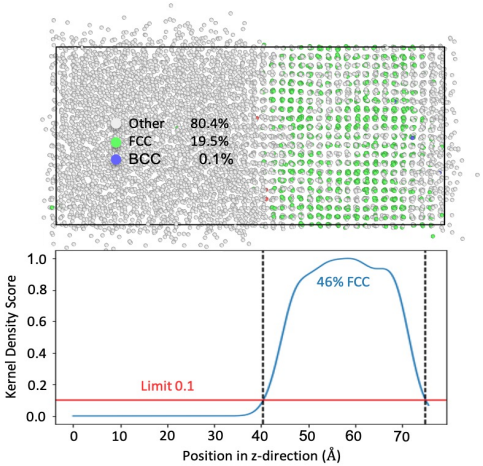
L.F. Zhu, **J. Janssen**, et al. Comp. Mat. Sci. 187 (2021)

Autonomous Calculation of Melting Temperatures

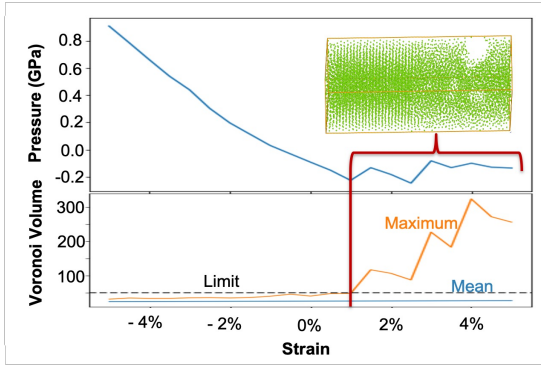


Challenges:

Solid Liquid Detector

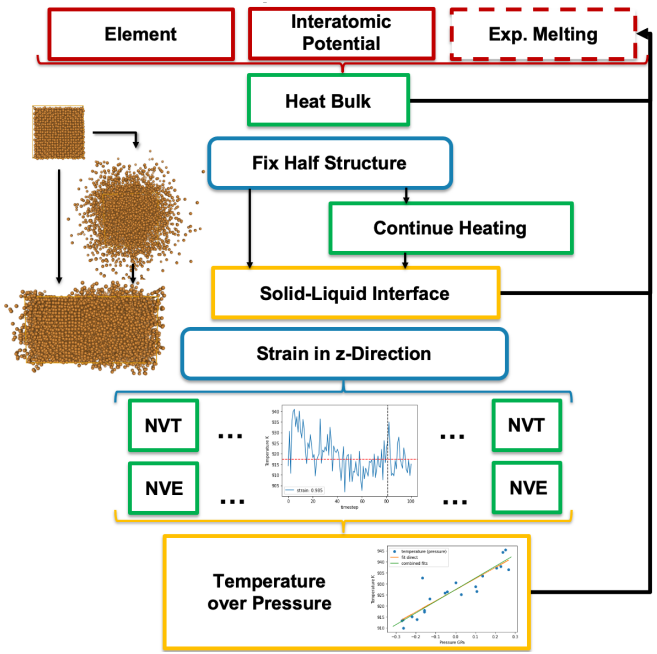


Void Detector

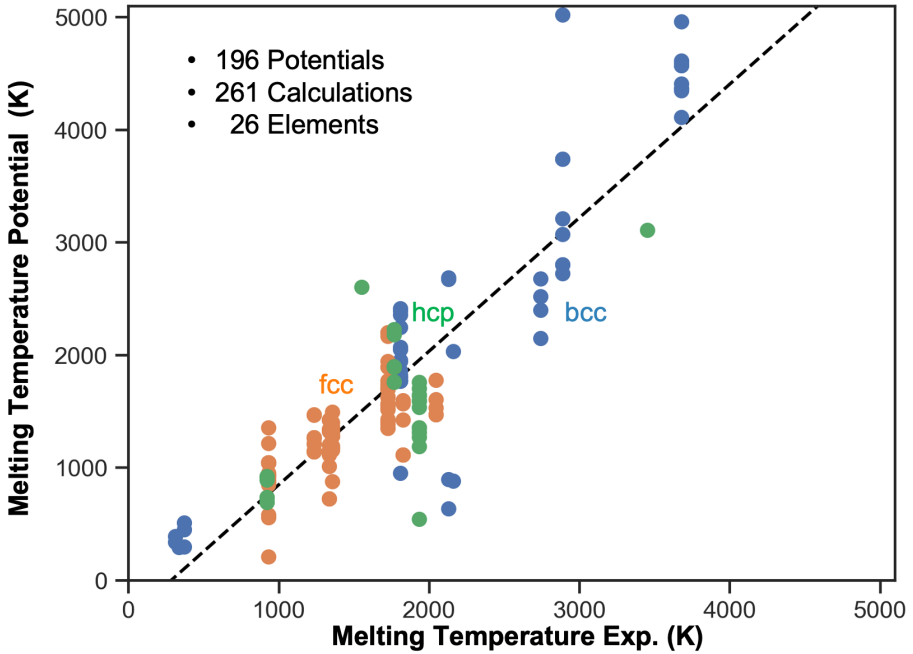


L.F. Zhu, **J. Janssen**, et al. Comp. Mat. Sci. 187 (2021)

Autonomous Calculation of Melting Temperatures

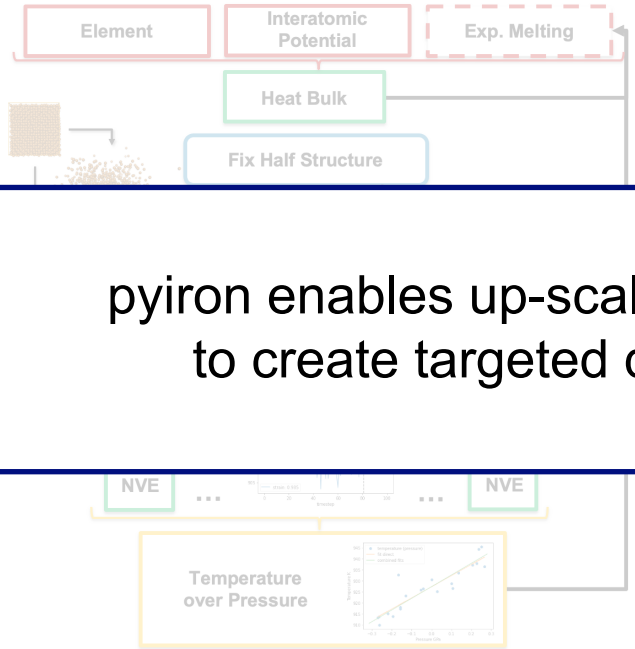


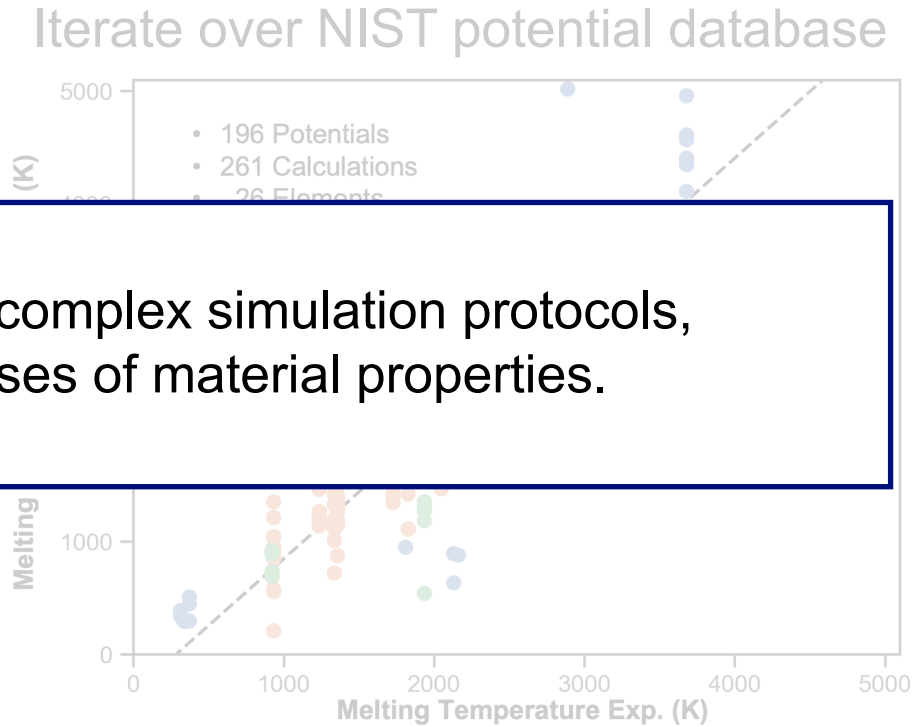
Iterate over NIST potential database



L.F. Zhu, **J. Janssen**, et al. Comp. Mat. Sci. 187 (2021)

Autonomous Calculation of Melting Temperatures

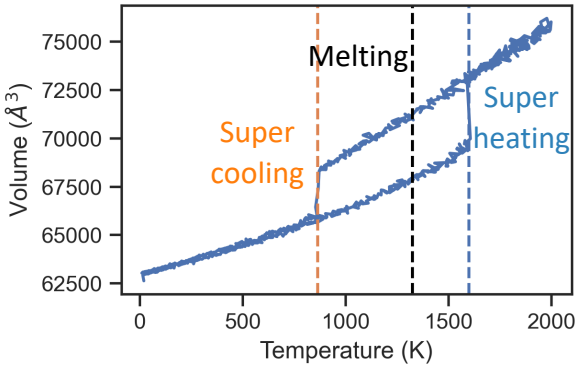




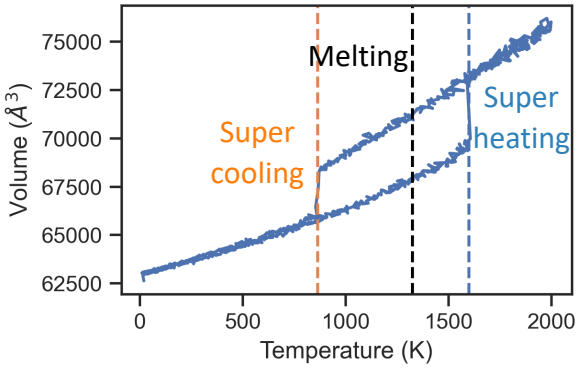
pyiron enables up-scaling of complex simulation protocols,
to create targeted databases of material properties.

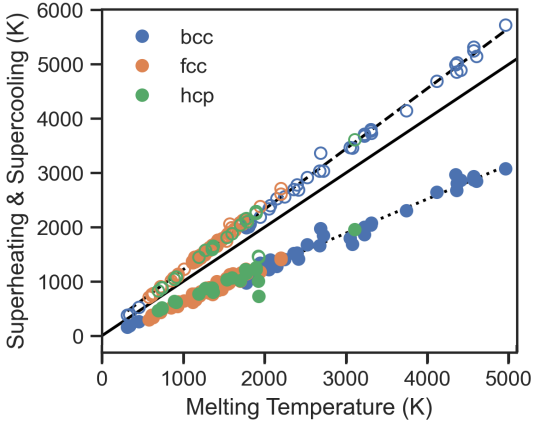
L.F. Zhu, **J. Janssen**, et al. Comp. Mat. Sci. 187 (2021)

Prediction of melting temperatures

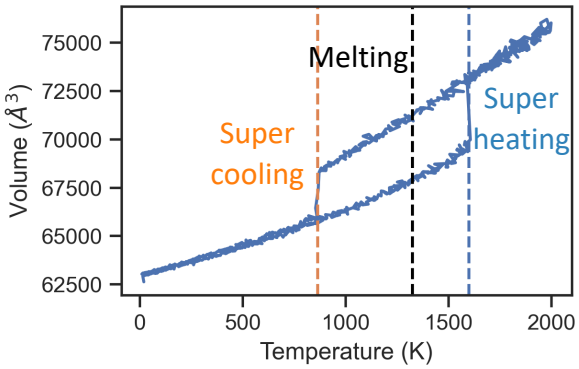


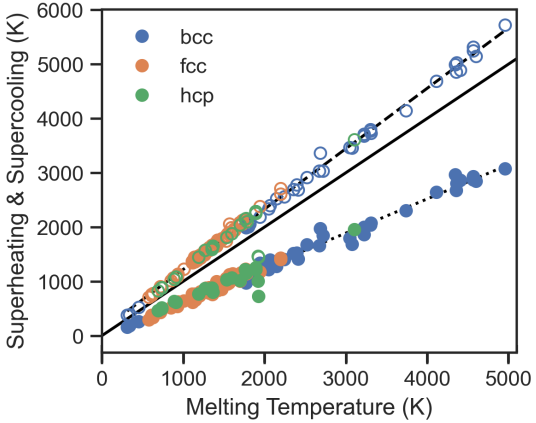
Prediction of melting temperatures

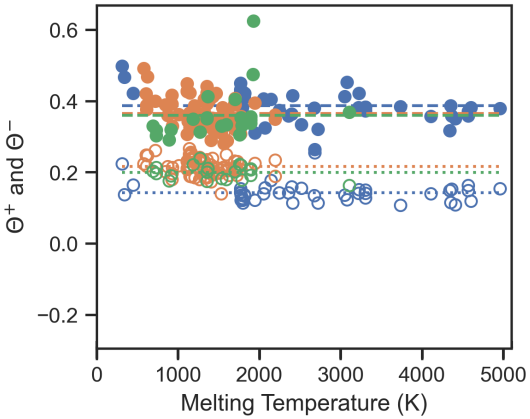




Prediction of melting temperatures



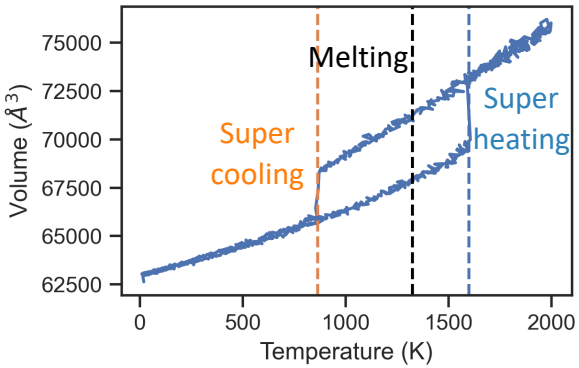




$$\Theta^- = 1 - \frac{T_-}{T_M}$$

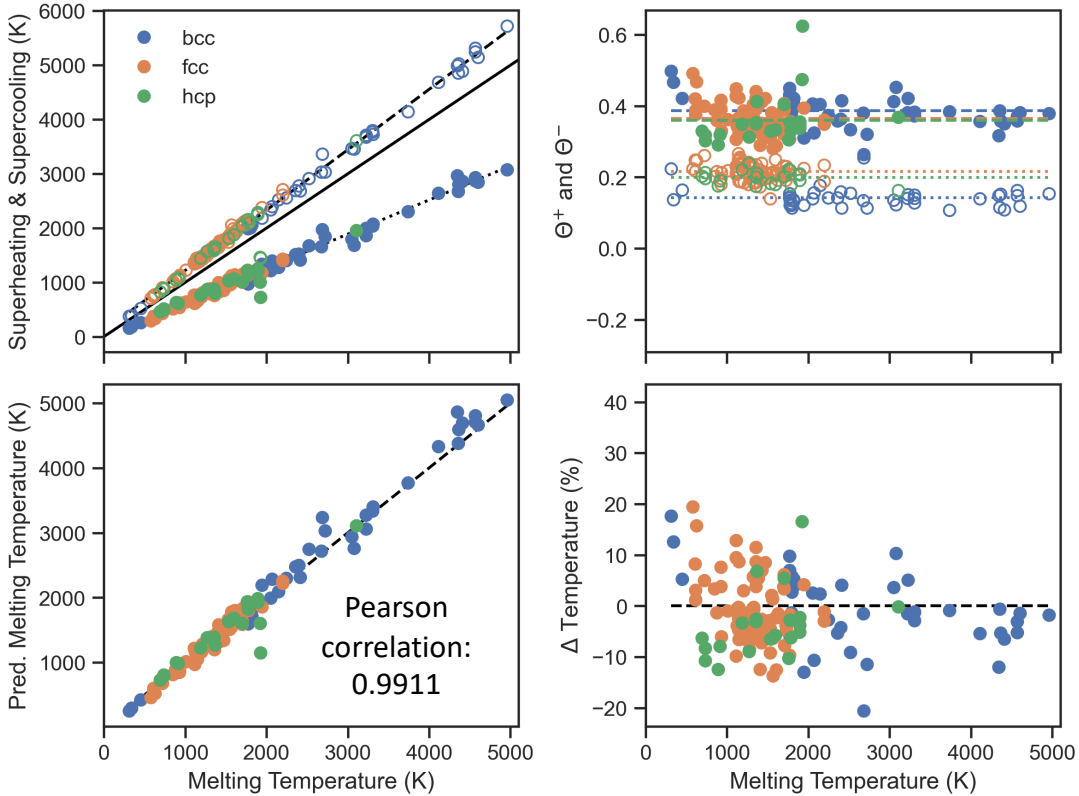
$$\Theta^+ = \frac{T_+}{T_M} - 1$$

Prediction of melting temperatures

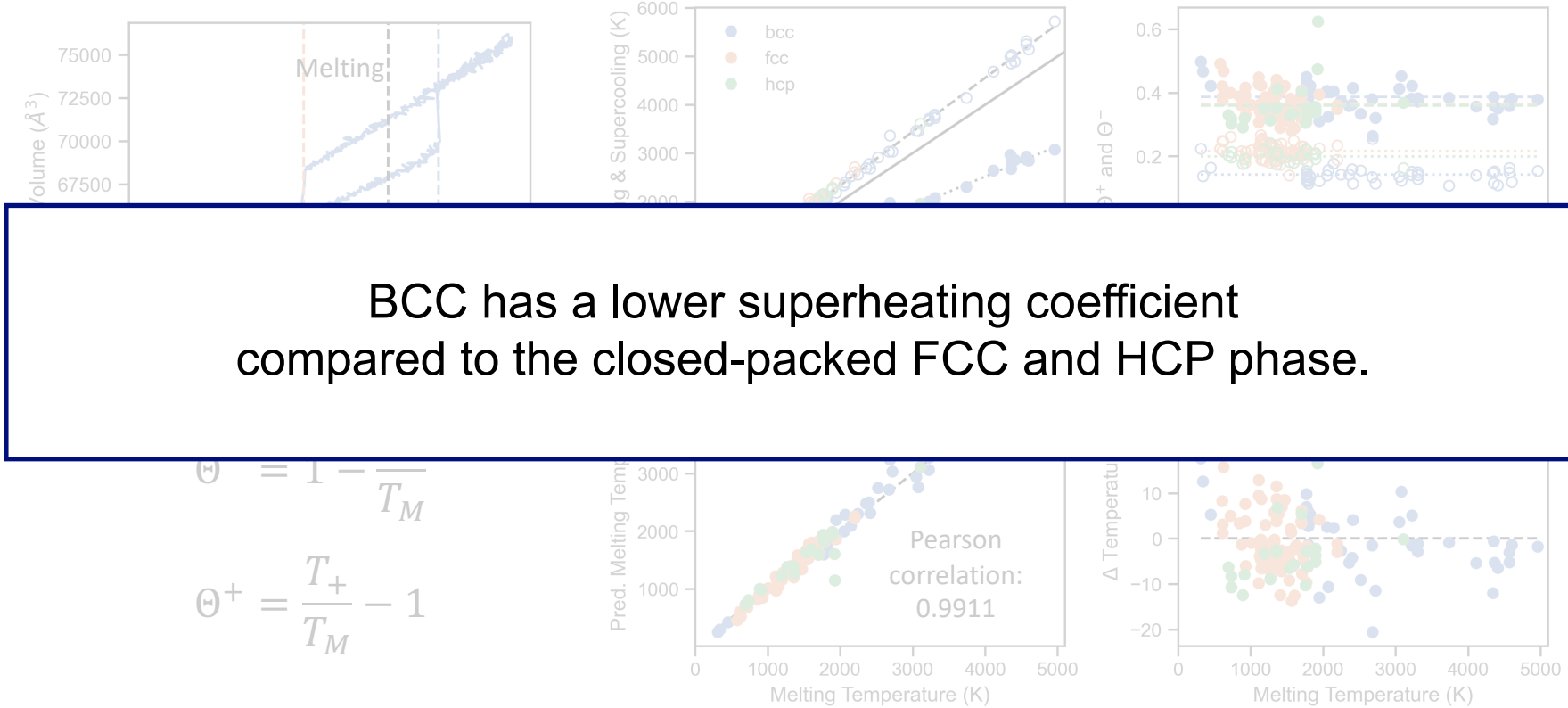


$$\Theta^- = 1 - \frac{T_-}{T_M}$$

$$\Theta^+ = \frac{T_+}{T_M} - 1$$



Prediction of melting temperatures



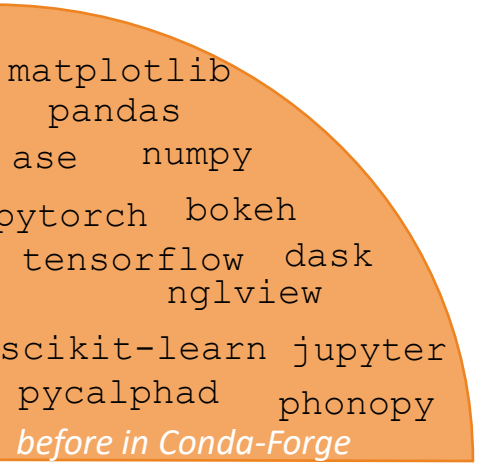
BCC has a lower superheating coefficient
compared to the closed-packed FCC and HCP phase.

Distribution via Conda-Forge

CONDA-FORGE

Easy installation of scientific software:

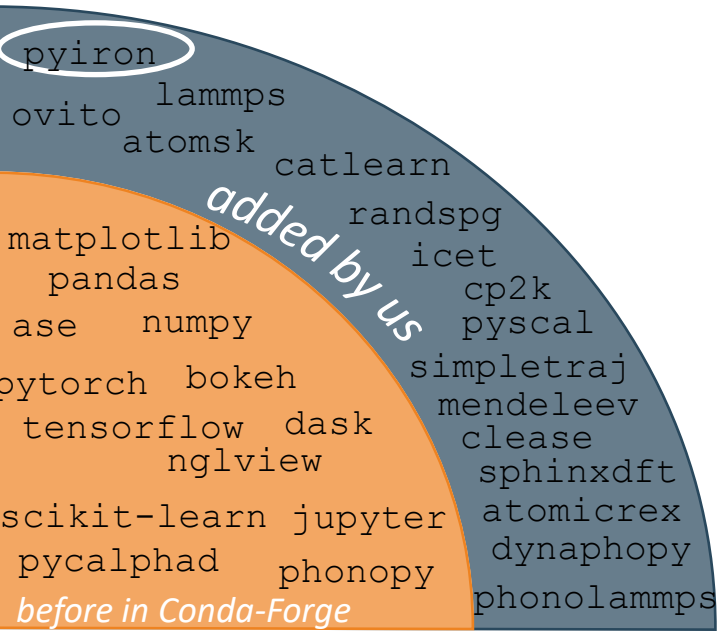
```
conda install -c conda-forge pyiron
```



Advantages:

- Over 500 materials science package
- Distribution of precompiled packages
  - Linked to defined Python version
  - Dependency of packages clarified
  - Operating system independence

Distribution via Conda-Forge



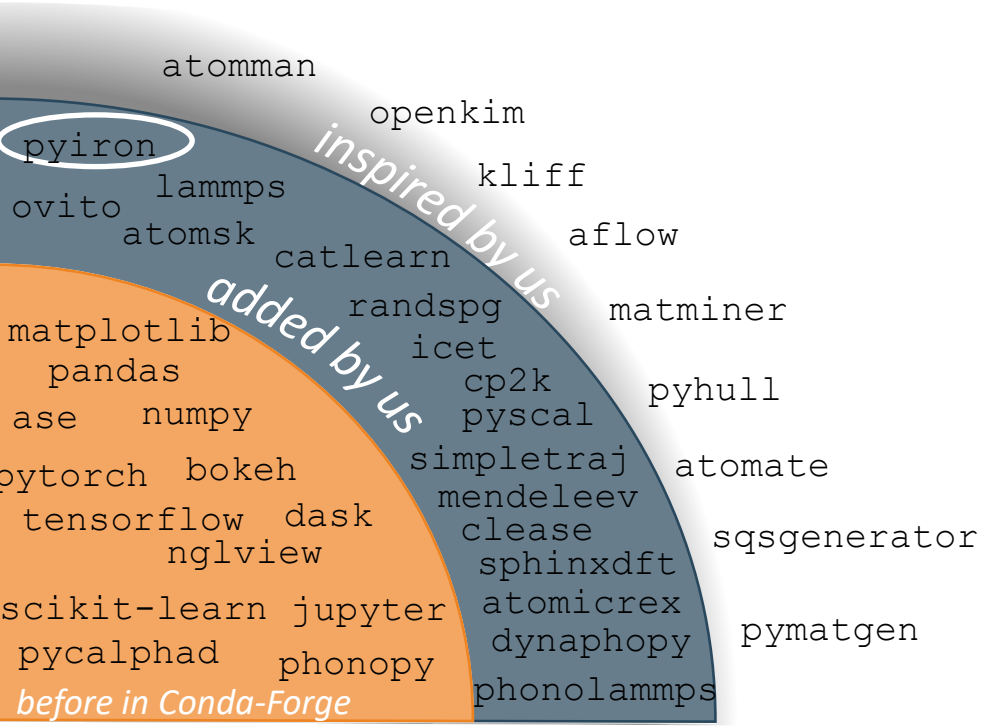
Easy installation of scientific software:

```
conda install -c conda-forge pyiron
```

Advantages:

- Over 500 materials science package
- Distribution of precompiled packages
  - Linked to defined Python version
  - Dependency of packages clarified
  - Operating system independence

Distribution via Conda-Forge



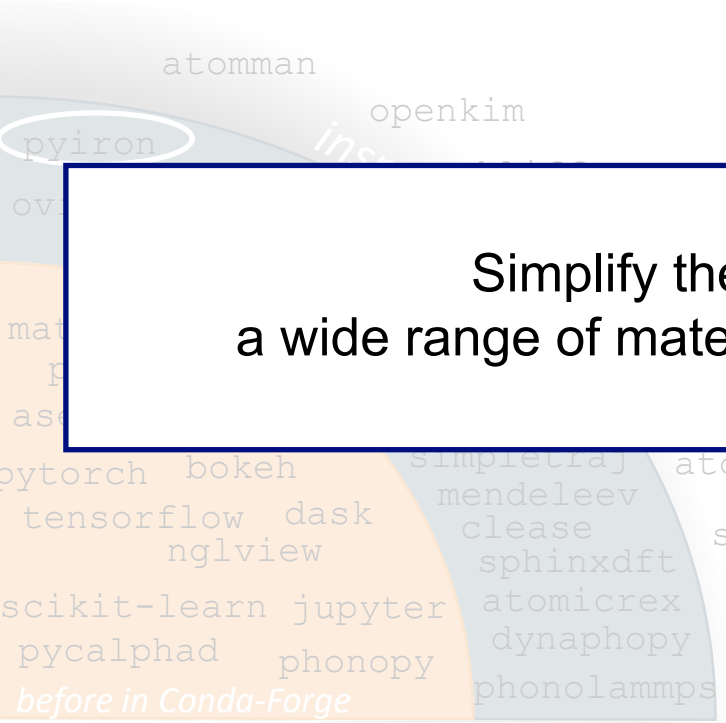
Easy installation of scientific software:

```
conda install -c conda-forge pyiron
```

Advantages:

- Over 500 materials science package
- Distribution of precompiled packages
  - Linked to defined Python version
  - Dependency of packages clarified
  - Operating system independence

Distribution via Conda-Forge



Simplify the installation and distribution of
a wide range of materials science simulation codes and tools.

Easy installation of scientific software:

- Distribution of precompiled packages
  - Linked to defined Python version
  - Dependency of packages clarified
  - Operating system independence

Abstraction Levels

High-level User Interface

Use predefined simulation protocols, only change a predefined set of parameters.

Scientific Level

Develop simulation protocols in Jupyter notebooks.

Technical Implementation

Job Management

Data Management

Parser/ Interface

Package Manager for Scientific Software

Distributing binaries, supporting all major operation systems.

Abstraction Levels

High-level User Interface

Use predefined simulation protocols, only change a predefined set of parameters.

The same workflow can be used by users
with different levels of experience.

Job Management | Data Management | Parser/ Interface

pyiron

Package Manager for Scientific Software

Distributing binaries, supporting all major operation systems.

CONDA-FORGE

Publication Template

Fork the Github Repository:

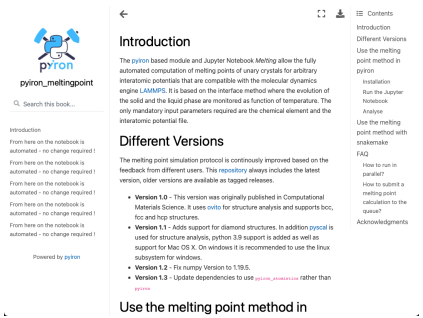
https://github.com/pyiron/pyiron-publication-template

Update conda Environment

Replace the `environment.yml` file in the repository.

Include additional resources

Include additional classes, results or parameter database

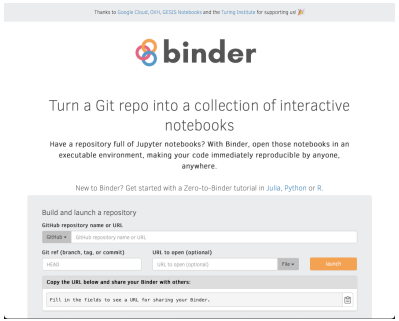


Publish the Jupyter Notebook

Render Notebook for Website and mybinder.org

Melting Temperature Example:

https://github.com/pyiron/pyiron_meltingpoint



Publication Template

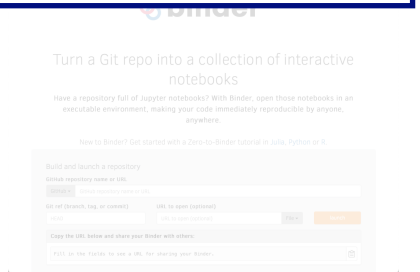
Fork the Github Repository:

https://github.com/pyiron/pyiron-publication-template

With the publication pyiron covers all steps of the simulation lifecycle from the initial idea to the sharing of the workflow.

Publish the Jupyter Notebook

Render Notebook for Website and mybinder.org



Melting Temperature Example:

https://github.com/pyiron/pyiron_meltingpoint

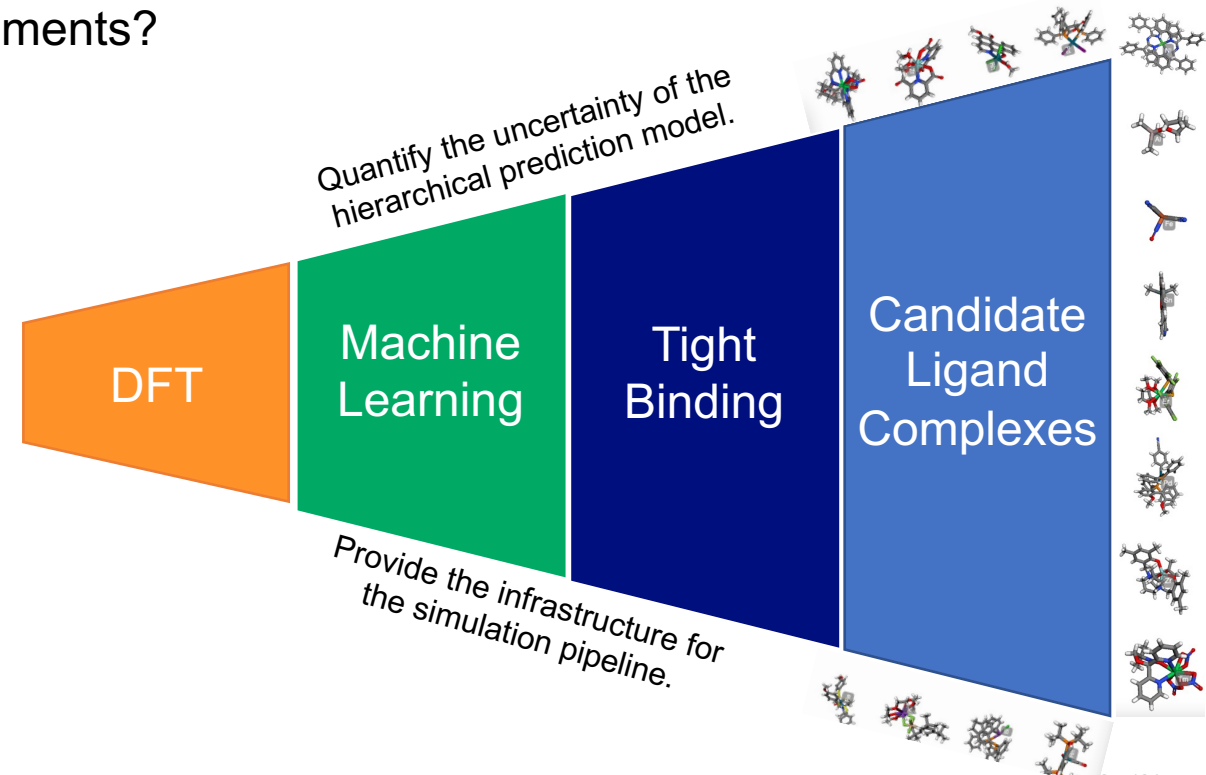

Ping Yang


Michael G. Taylor

Autonomous Discovery

Can we use different levels of theory to reduce the number of required experiments?

Experimental Robot




Ping Yang


Michael G. Taylor

Autonomous Discovery

Can we use different levels of theory to reduce the number of required experiments?

To efficiently sub-sample the chemical configuration space we use a hierarchical simulation approach.

Experimental Robot

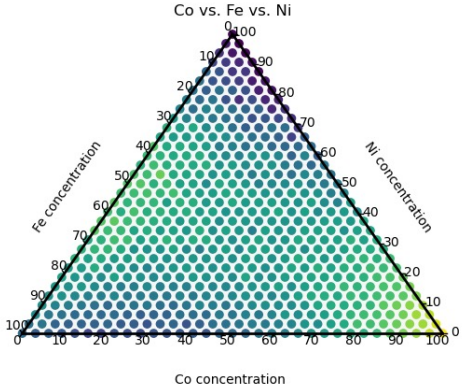
tify the uncertainty of the diction model.

Provide the infrastructure for the simulation pipeline.

Materials for the Future

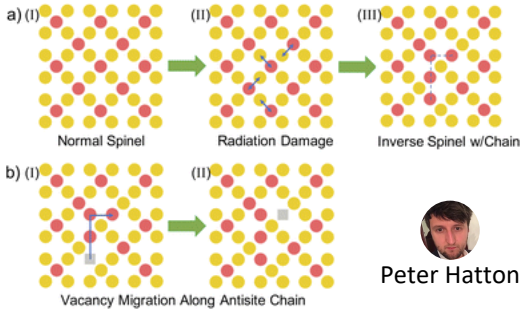
High-throughput parameter studies provide a systematic understanding of:

Elastic properties for complex alloys



Collaboration with Texas A&M University

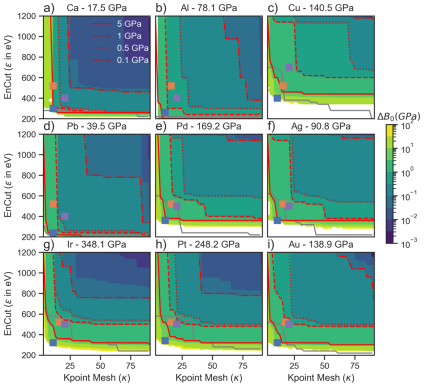
Vacancy transport in disordered spinels



Peter Hatton

Materials Science and Technology Division LANL

Uncertainty propagation for plane-wave DFT



For material properties and machine learned potentials

Materials for the Future

High-throughput parameter studies provide a systematic understanding of:

The pyiron approach of rapid-prototyping, up-scaling and coarse graining is applicable to a wide range of materials science challenges.

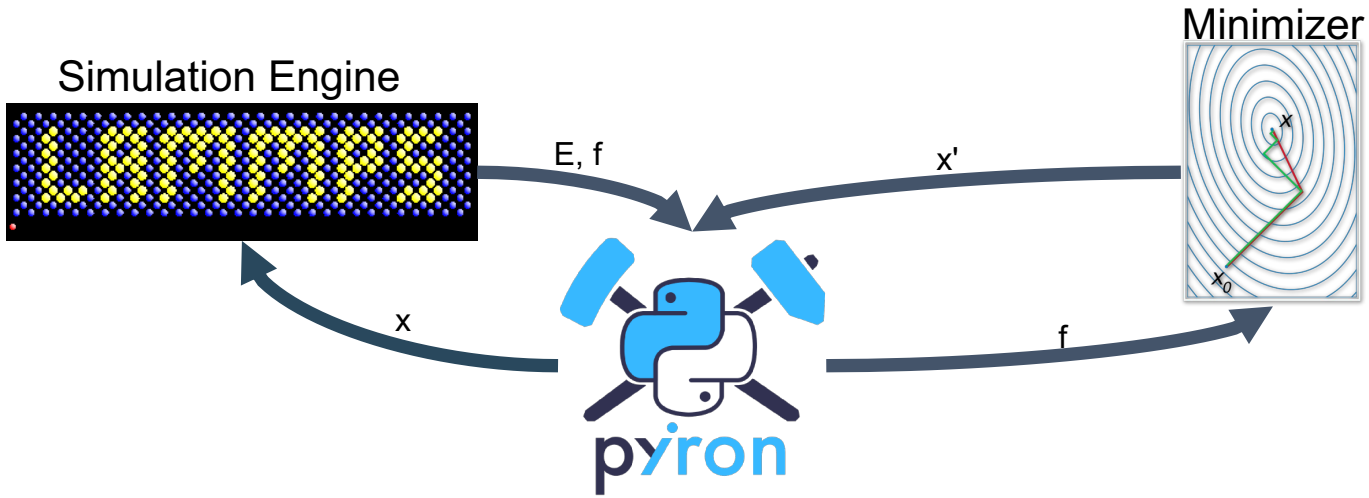
Collaboration with Texas A&M University

Peter Hatton

Materials Science and Technology Division LANL

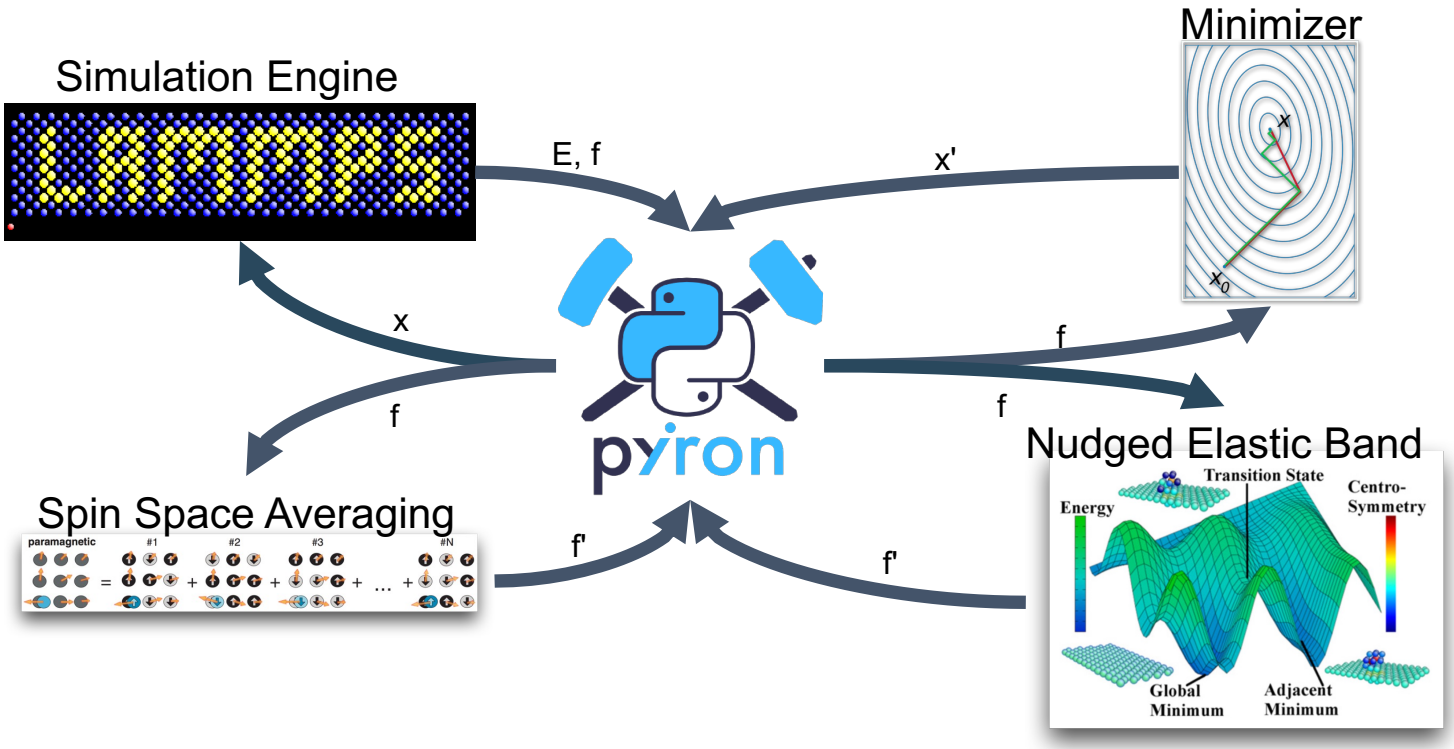
For material properties and machine learned potentials

Interactive Coupling



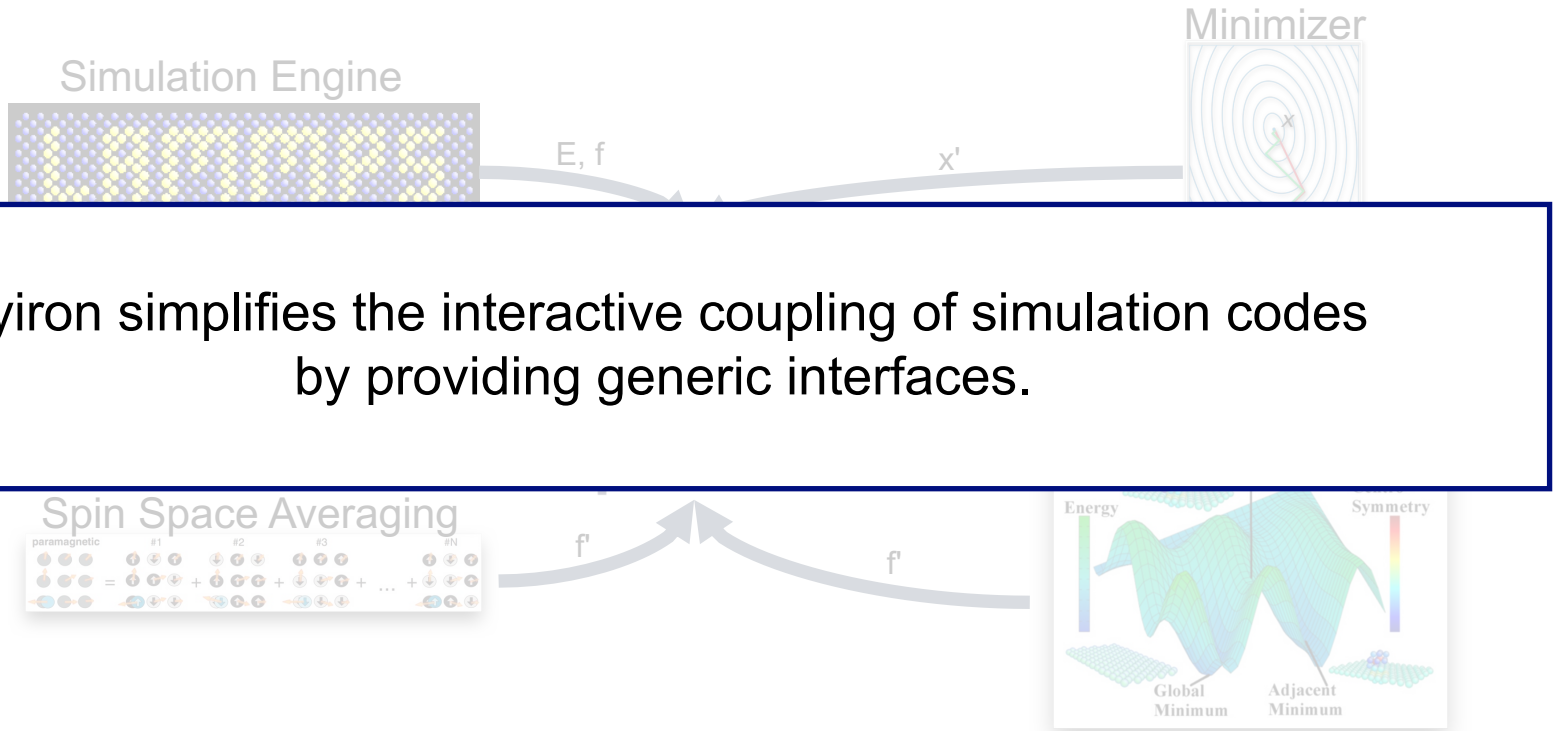
O. Hegde et al. Phys. Rev. B 102, 144101 (2020)

Interactive Coupling



O. Hegde et al. Phys. Rev. B 102, 144101 (2020)

Interactive Coupling



O. Hegde et al. Phys. Rev. B 102, 144101 (2020)

Summary

Materials Science at Scale:

Scientific Software

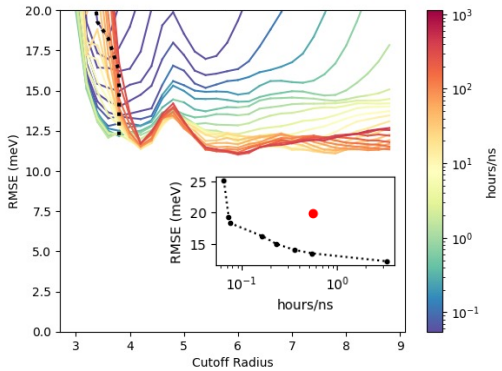
pyron

CONDA-FORGE

SNAP

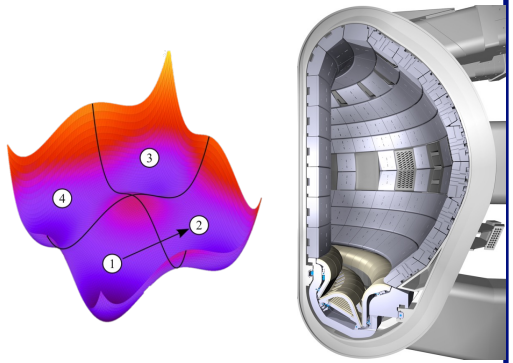
Scientific software is a building block for systematic and reproducible research.

Parameter Studies



Parameter studies provide new insights and remove human bias.

Applications



The goal is to predict materials for extreme environments with ab-initio precision.

Summary – Thank you

Materials Science at Scale:

Scientific Software

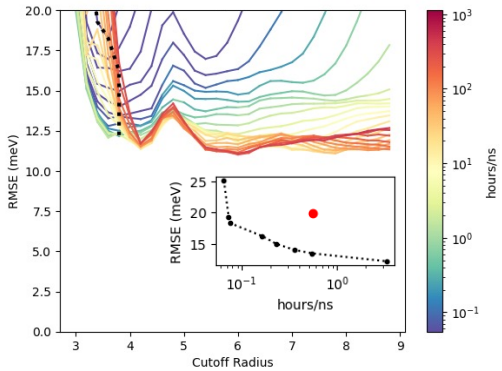
pyron

CONDA-FORGE

SNAP

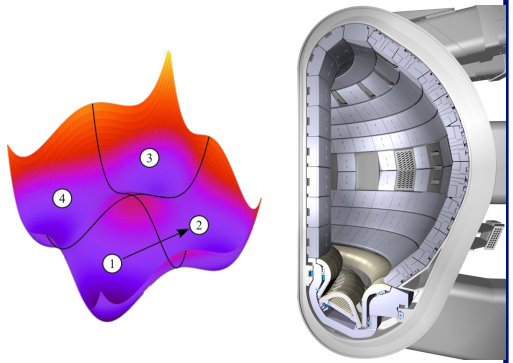
Scientific software is a building block for systematic and reproducible research.

Parameter Studies



Parameter studies provide new insights and remove human bias.

Applications



The goal is to predict materials for extreme environments with ab-initio precision.