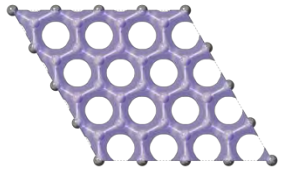
The Quantum Accelerator: Accessible and Scalable Materials Science Workflows

Andrew S. Rosen

Assistant Professor, Chemical and Biological Engineering
Associated Faculty, Princeton Materials Institute
Associated Faculty, Princeton Institute for Computational Science and Engineering
Affiliated Faculty, Center for Statistics and Machine Learning

A Day in the Life of a Computational Materials Scientist

High-Throughput Quantum Chemistry…



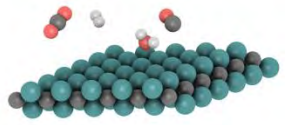
Electronic structure methods

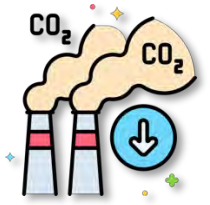
Automated workflows

Machine learning

Virtual screening

… to Unlock a More Sustainable Future

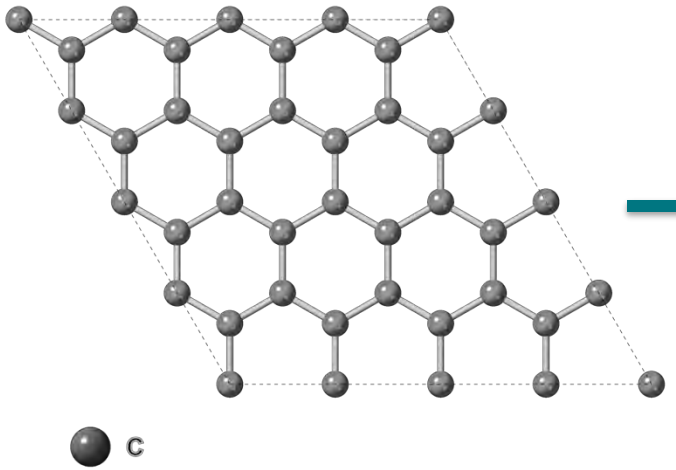




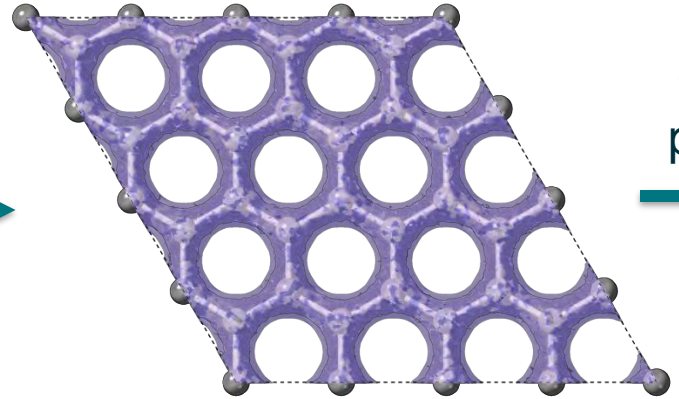
Next-generation heterogeneous catalysts, adsorbents, energy storage technologies, and electronic devices

A Brief Look Inside the Computational Materials Chemistry Box

Inputs:
Atom identities (Z)
and coordinates ($\mathbf{r}$)

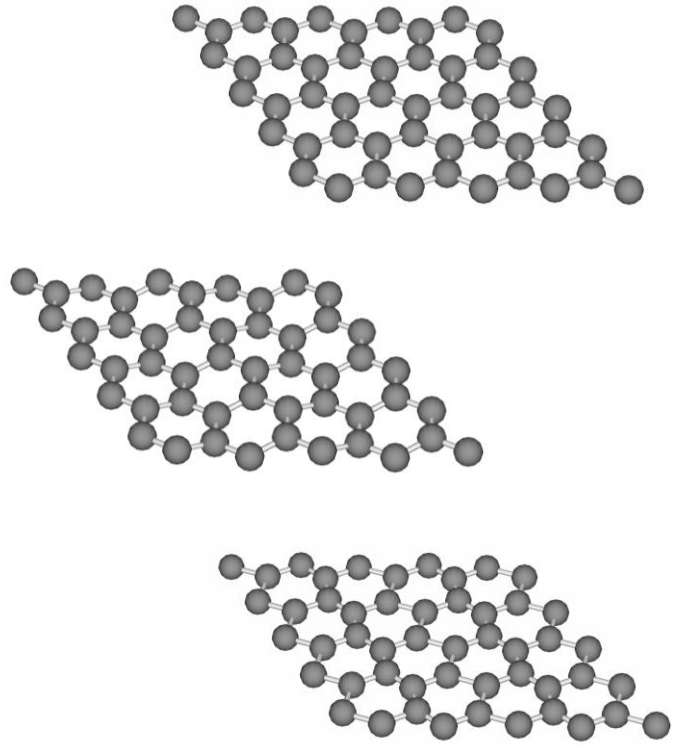


Solving for the ground-state
electron density, $\rho(\mathbf{r})$



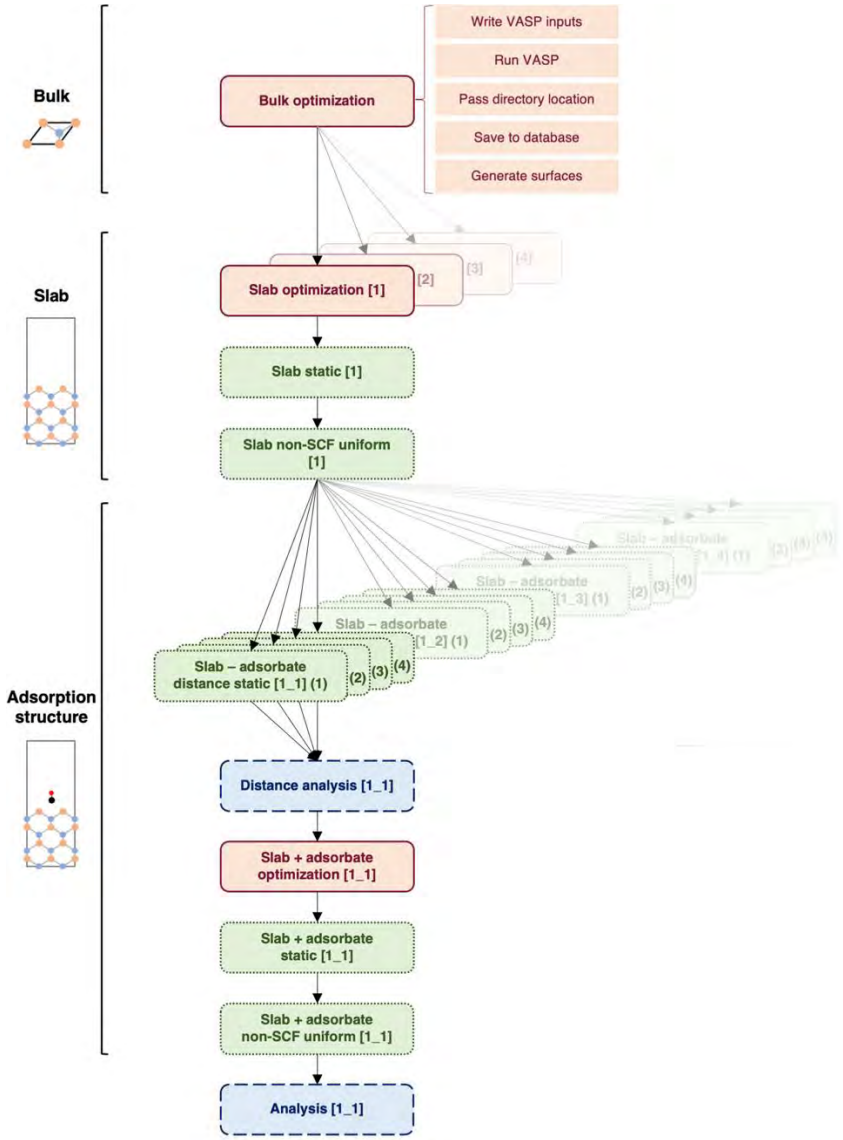
Compute
properties

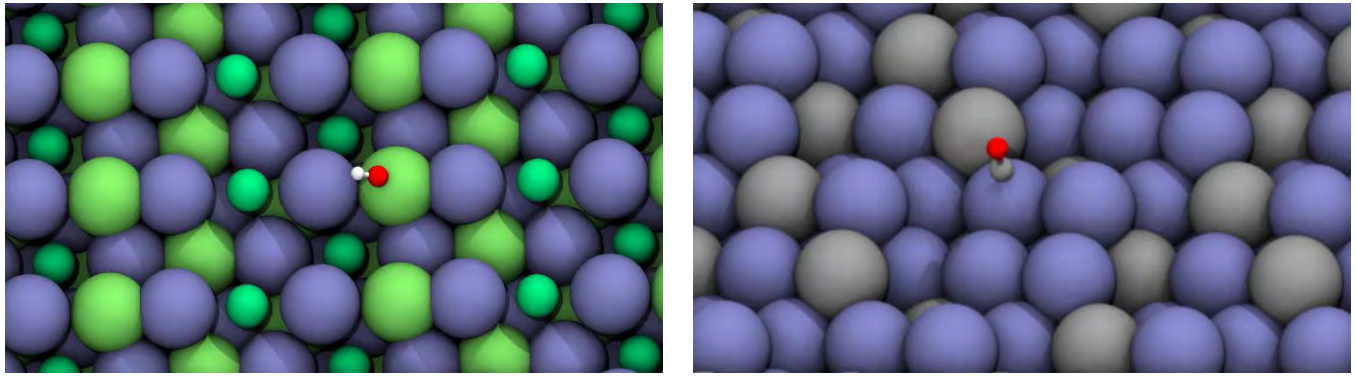
$E[\rho(\boldsymbol{r})]$



- We often need to simulate ***many physical and chemical properties*** for a given material
- We need to simulate the properties of ***many material candidates***
- We need to so in a ***highly scalable*** manner with robust error handling and logging

The Age-Old Problem: What's Simple for 1 is Rarely Simple for 100000





Credit: Open Catalyst Project from Meta AI and CMU

Why it's not trivial:

- Complex job connectivity
- Dynamic workflow that is ill-defined until runtime
- If one job fails, you don't want the workflow to fail
- Need to effectively monitor many calculations
- Need to be able to make adjustments on-the-fly
- Need to ensure output data is easily query-able
- …

O. Andriuc, M. Siron, J. Montoya, M. Horton, K.A. Persson, *J. Chem. Inf. Model.*, 61, 3908-3916 (2021)
R. Tran, …, Z. Ulissi, C.L. Zitnick, *ACS Catal.*, 13, 3066-3084 (2023)

The Quantum Accelerator: Efficient DFT Workflows at Extreme Scales

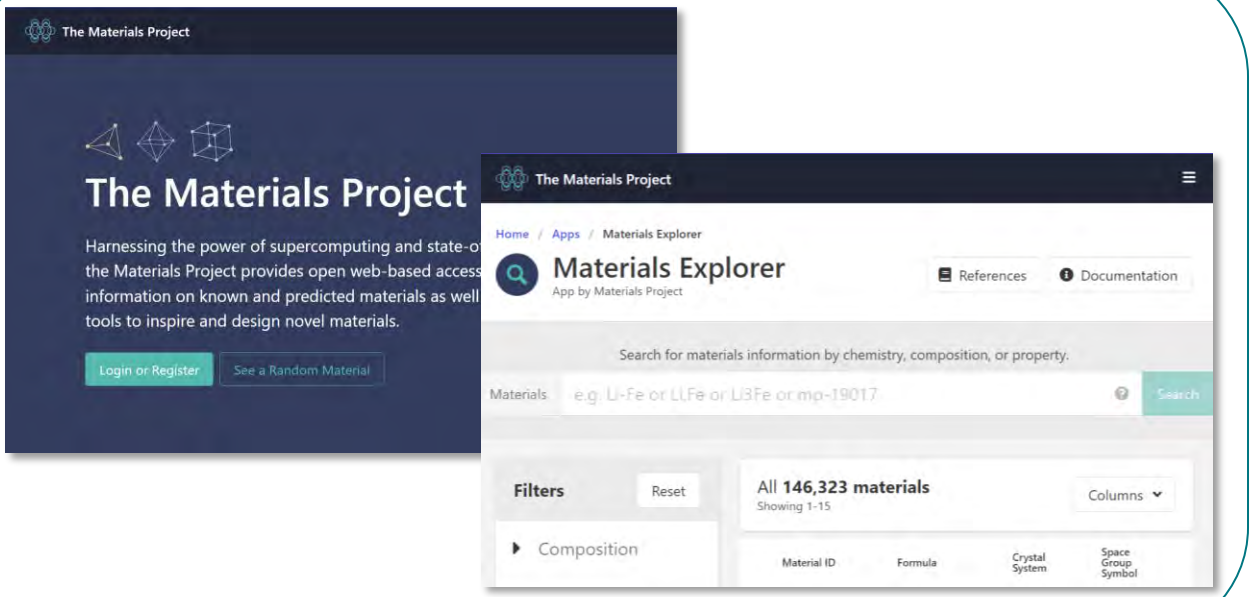
High-throughput Quantum Chem.

QuAcc

Reproducible, modular workflows

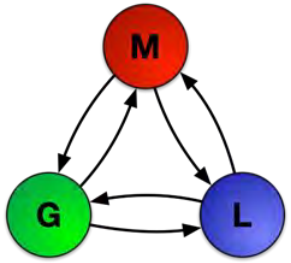
Name	Decorator	Documentation
VASP Static	@job	quacc.recipes.vasp.core.static_job
VASP Relax	@job	quacc.recipes.vasp.core.relax_job
VASP Double Relax	@job	quacc.recipes.vasp.core.double_relax_job
VASP Slab Static	@job	quacc.recipes.vasp.slabs.slab_static_job
VASP Slab Relax	@job	quacc.recipes.vasp.slabs.slab_relax_job
VASP Bulk to Slabs	@flow	quacc.recipes.vasp.slabs.bulk_to_slabs_flow
VASP Slab to Adsorbates	@flow	quacc.recipes.vasp.slabs.slab_to_ads_flow
VASP MP Prerelax	@job	quacc.recipes.vasp.mp.mp_relax_job
VASP MP Relax	@job	quacc.recipes.vasp.mp.mp_relax_job
VASP MP Relax Workflow	@flow	quacc.recipes.vasp.mp.mp_relax_flow
VASP QMOF Relax	@job	quacc.recipes.vasp.qmof.qmof_relax_job

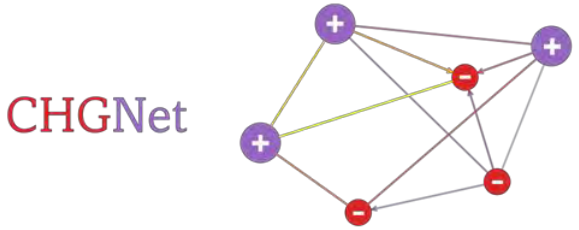
Open-Access Data Sharing

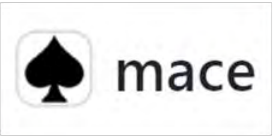


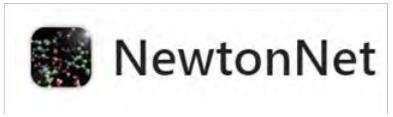
Adding Recipes for New Codes is Quick and Simple

- There are “recipes” for many codes out-of-the box, and it is trivial to add new ones
  - Built around the interoperable Atomic Simulation Environment (ASE) that I co-maintain





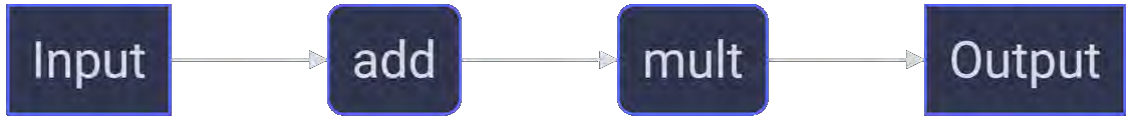




All of these interfaces are regularly tested *in production* on Princeton HPC resources with each pull request (via Jenkins)

Embracing Convergent Evolution of Workflow Technologies

- Many modern workflow orchestration tools have converged to a **similar decorator approach**

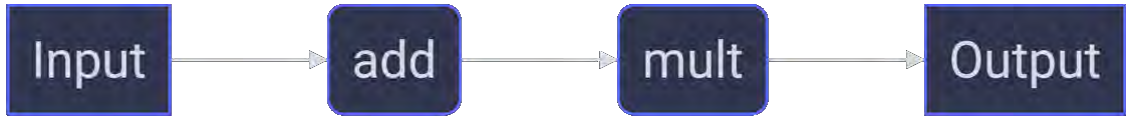



```python
from parsl import python_app

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

@python_app
def mult(a, b):
    return a * b

def workflow(a, b, c):
    return mult(add(a, b), c)

result = workflow(1, 2, 3).result() # 9
```

Parsl

Embracing Convergent Evolution of Workflow Technologies

- Many modern workflow orchestration tools have converged to a **similar decorator approach**




```python
from prefect import flow, task

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

@task
def mult(a, b):
    return a * b

@flow
def workflow(a, b, c):
    return mult.submit(add.submit(a, b), c)

result = workflow(1, 2, 3).result() # 9
```

```python
import covalent as ct

@ct.electron
def add(a, b):
    return a + b

@ct.electron
def mult(a, b):
    return a * b

@ct.lattice
def workflow(a, b, c):
    return mult(add(a, b), c)

result = workflow(1, 2, 3) # 9
```

```python
from redun import task, Scheduler

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

@task
def mult(a, b):
  return a * b

@task
def workflow(a, b, c):
  return mult(add(a, b), c)

Scheduler = scheduler()

result = Scheduler.run(workflow(1, 2, 3)) # 9
```

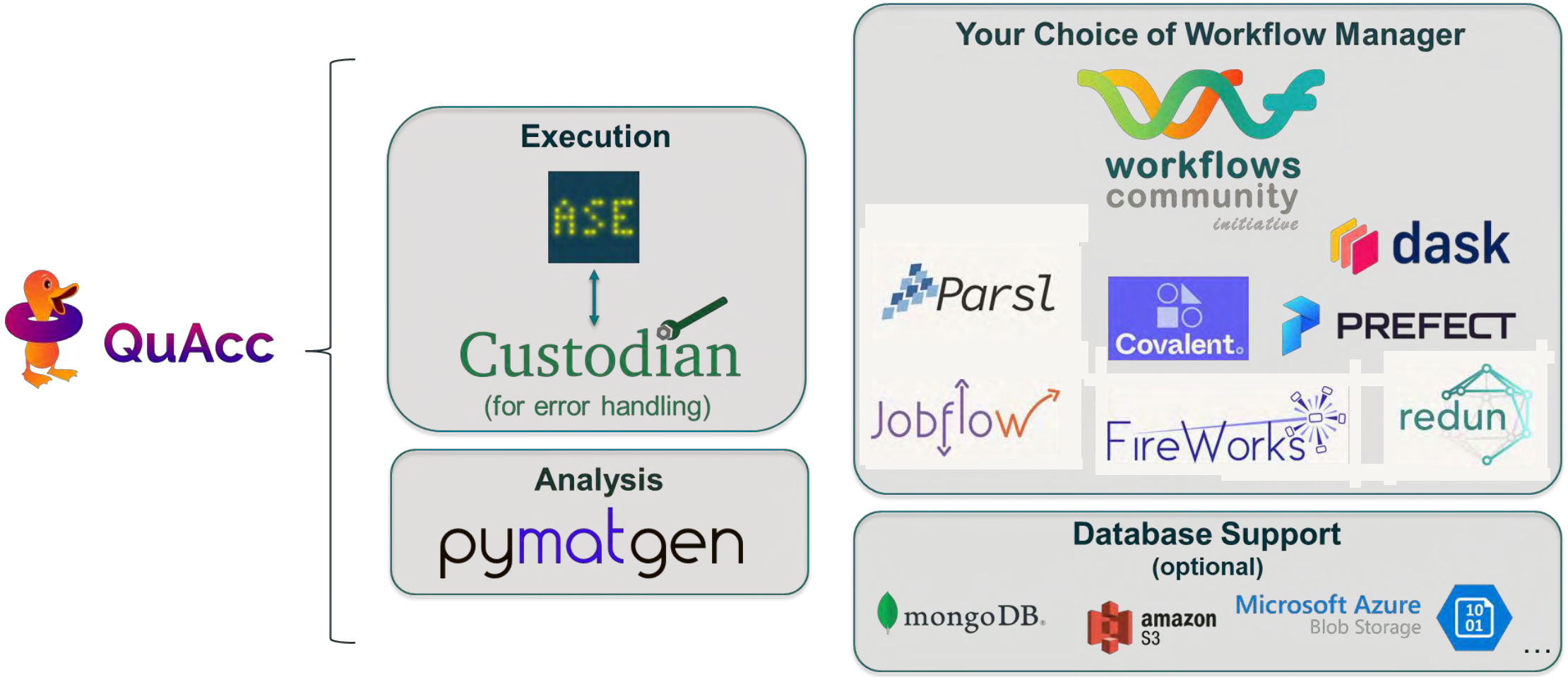
...

PREFECT

Covalent

redun

Quacc is Written to Be Both Familiar Yet Highly Flexible



Of course, there are many **major benefits for using Parsl**!

Parsl is the workflow engine typically chosen by new users (ease-of-setup, pilot job for MPI tasks)

Mutual Growth Between QuAcc and Parsl

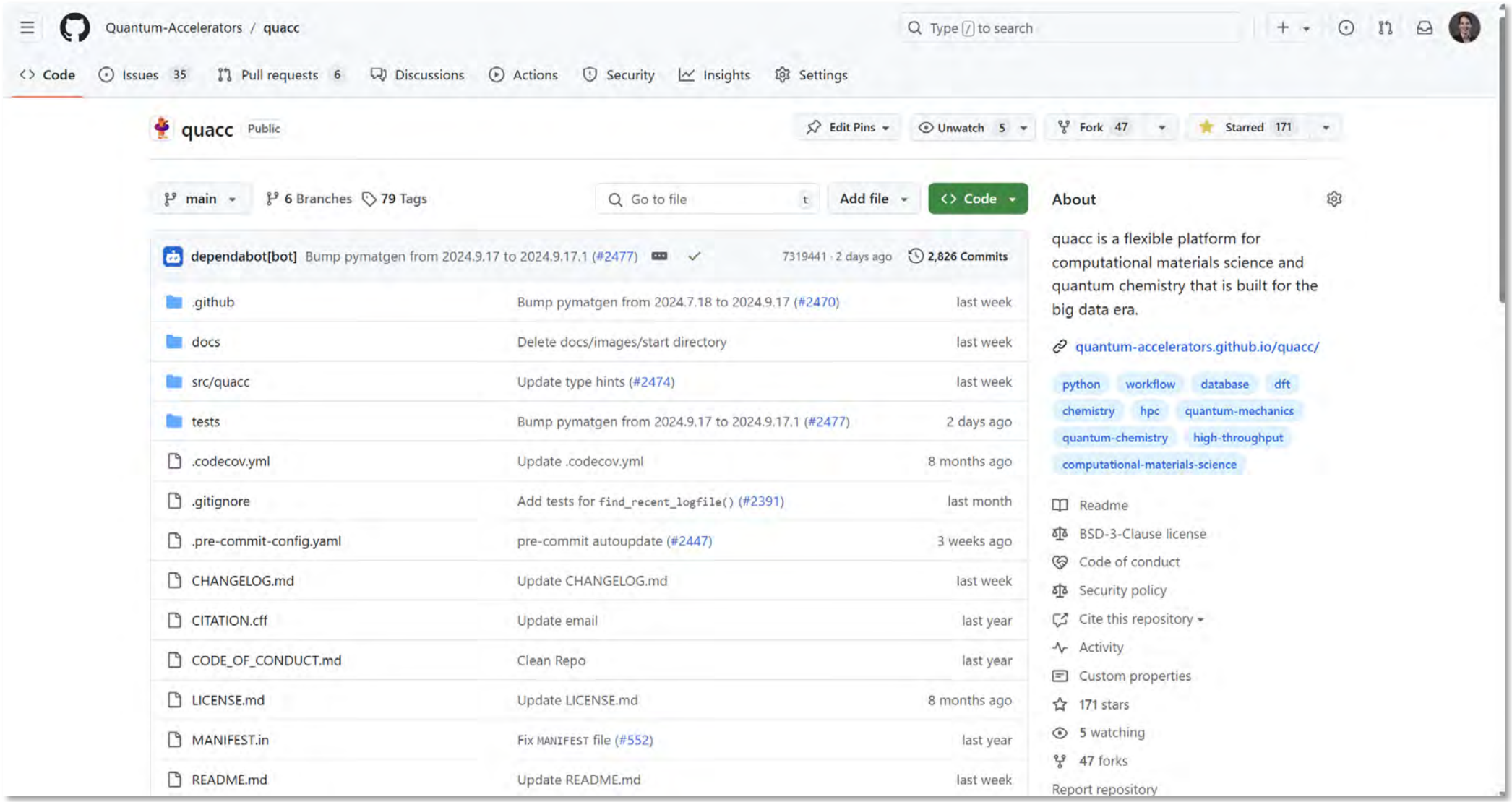
- Contributors to QuAcc are typically **domain experts**, not programmers
  - Workflows need to be as close to “pure Python” as possible (i.e. minimal injection of workflow logic)
  - QuAcc is shipped as a workflow library — has led to some re-thinking about Parsl

Example Parsl development: *Lifted operators*

```python
@python_app
def simulate(accuracy = "high", ...) -> dict:
    # do some complex simulation
    return output

@python_app
def get_max_force(forces) -> float:
    import numpy as np

    return np.linalg.norm(forces, axis=1).max()

output = simulate(...)
max_force = get_max_force(output["forces"])
```

Querying a key or index of an AppFuture

Many Thanks to My Collaborators and Colleagues (especially Ben Clifford!)



Thanks to Contributors: Dr. Zack Ulissi (Meta), Tom Demeyere (Southampton University), Dr. Anup Kumar (LBNL), Dr. Sam Blau (LBNL) + many more