Coupled Simulation and AI Workflows for Science

Riccardo Balin, ALCF
Christine Simpson, ALCF

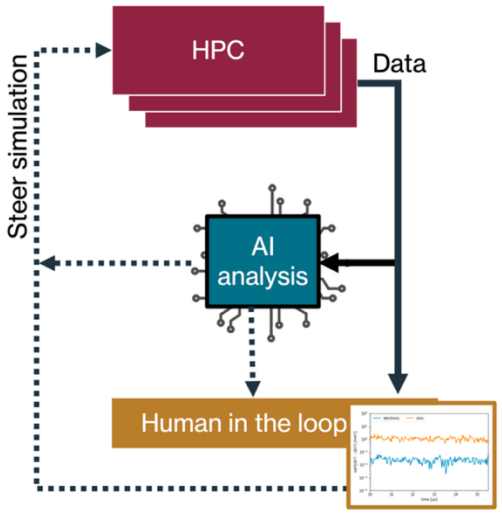
Tuesday August 4, 2026

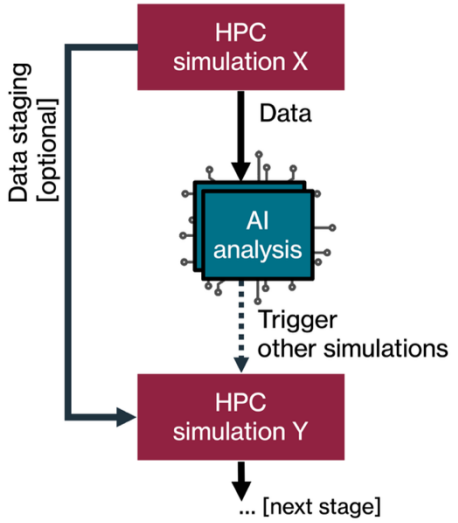
Why Couple HPC Simulations with AI/ML?

- Substitute inaccurate or expensive components of simulation with ML models
  - E.g.: Closure or surrogate modeling
- Optimize simulation parameters on-the-fly
  - E.g.: Select solver parameters at runtime based on AI inference
- Avoid IO bottleneck and disk storage issues during offline training
  - E.g.: In situ/online training through data streaming or in-memory staging
- Active learning and model fine-tuning
  - E.g.: Continuous fine-tuning and deployment of model
  - E.g.: Access training data not available during offline pre-training
- Steering of simulation ensembles
  - E.g.: Design space exploration or parameter optimization guided by AI

How to Couple HPC Simulations with AI/ML?

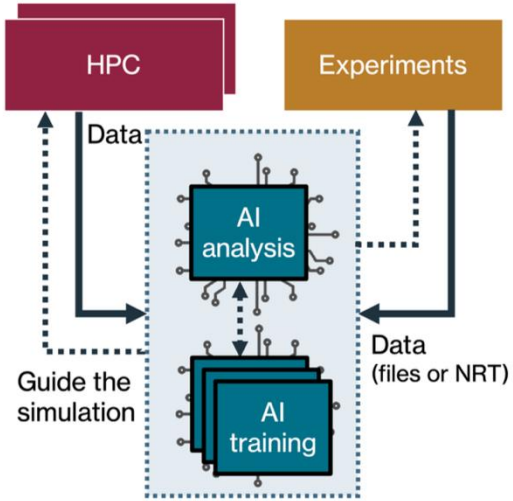
Steering of Ensembles

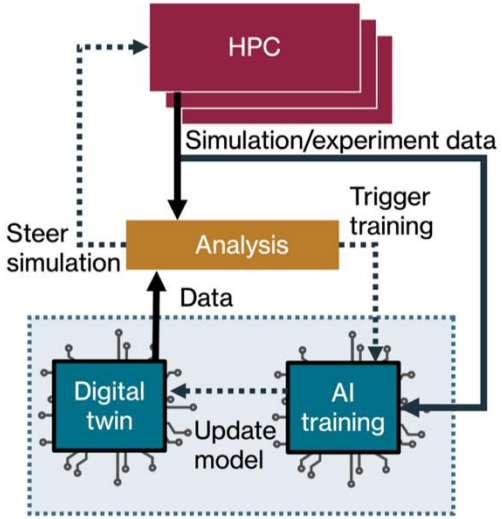




Optimize Simulation Parameters

Active Learning/ Online Fine-Tuning





Digital Twin

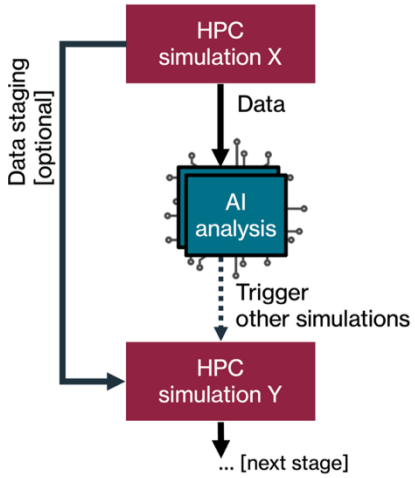
Brewer et al. "AI-coupled HPC workflow applications, middleware and performance." *arXiv:2406.14315* (2024)

How to Couple HPC Simulations with AI/ML?

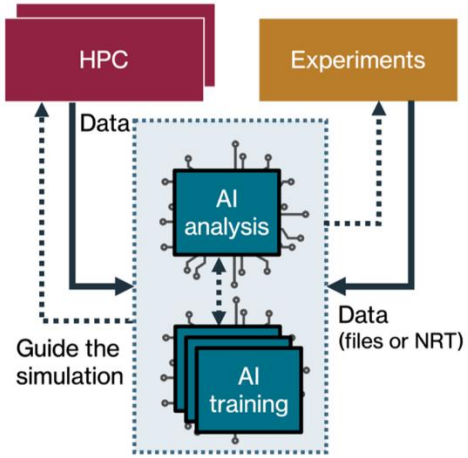
Type of coupling

- ❑Inference or training only
- ❑Both training and inference

Inference Only



Training and Inference



Programming languages and models

- ❑Simulation and AI/ML components often written in different languages
- ❑AI/ML relies heavily on vendor libraries
- ❑Separate parallelism strategies and/or libraries (MPI vs. NCCL)

Brewer et al. "AI-coupled HPC workflow applications, middleware and performance." *arXiv:2406.14315* (2024)

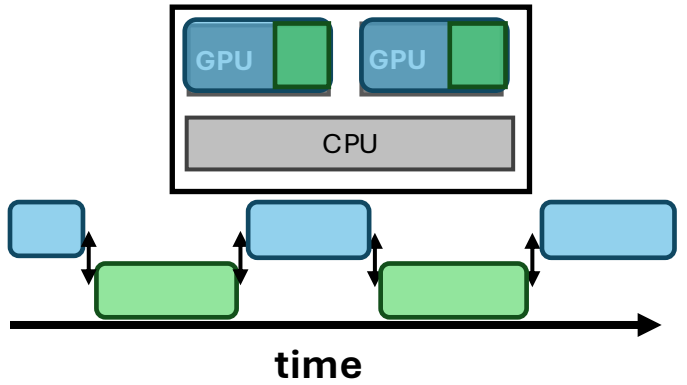
How to Couple HPC Simulations with AI/ML?

Execution Management

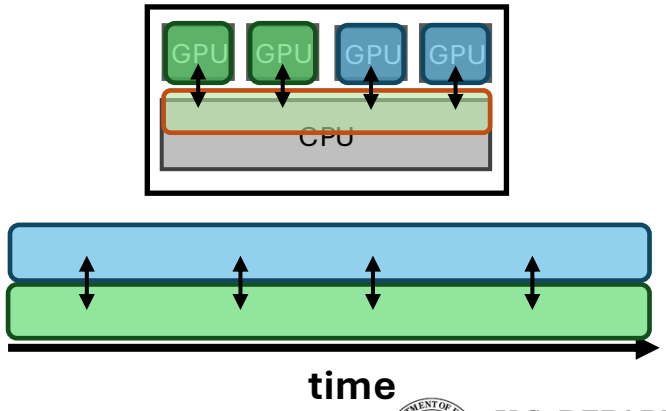
- Time division (tight coupling)
  - Components run on same compute resources (may even use same processes)
  - Staggered in time, execution of one component halts the other
  - May allow for direct memory access and no data copy/transfer
  - Idle time of individual components may be significant

- Space division (loose coupling)
  - Components run on separate compute resources
  - Concurrent in time, components run simultaneously
  - Minimal idle time of components for fast data copy/transfer
  - Usually requires indirect memory access with data copy/transfer

Time Division: Same Compute Hardware



Space Division: Separate Compute Hardware



Childs et al., "A terminology for in situ visualization and analysis systems", Intl. Journal of High Performance Computing Applications, 2020

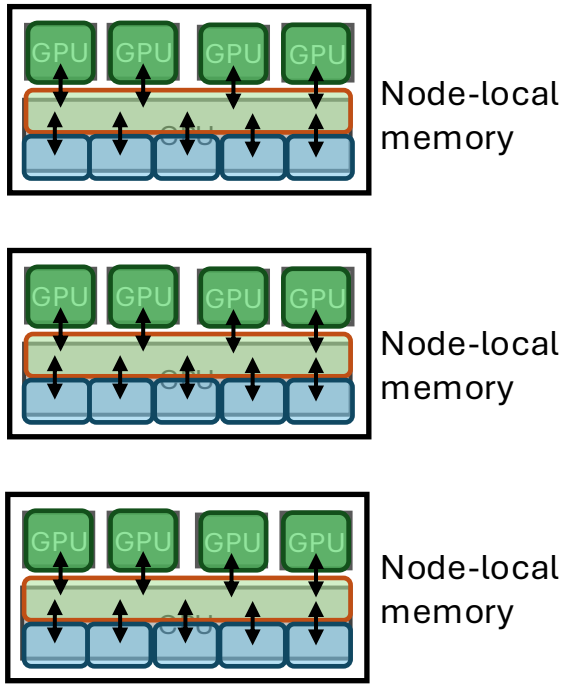
How to Couple HPC Simulations with AI/ML?

Simulation | ML component
Database | Data transfer

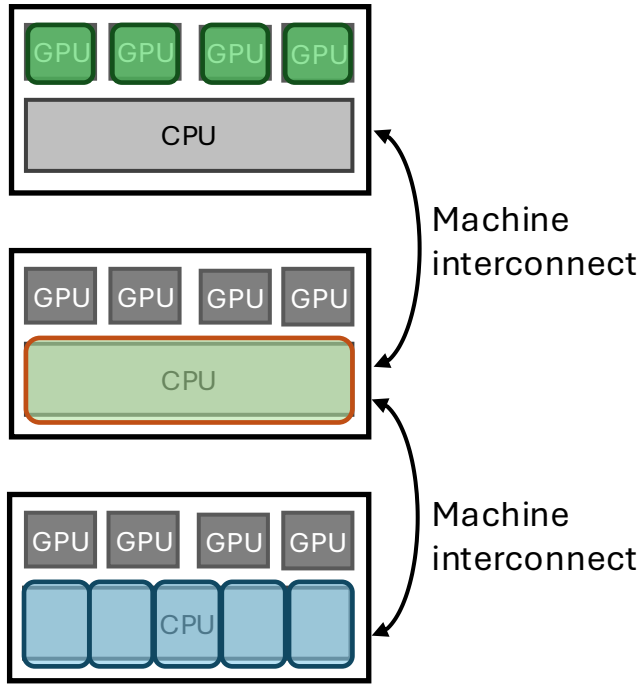
Physical Proximity

- **Colocation:** components share the same nodes
- **Node-level clustering:** components use different nodes on same system
- **Multi-system:** components run on separate specialized systems

Colocated Deployment



Node Clustered Deployment



System Clustered

How to Couple HPC Simulations with AI/ML?

Data Management

- Coupled workflows required frequent data sharing/transfer between components
- Data access:
  - **Direct:** components share same memory space (may allow zero-copy data use)
  - **Indirect:** components use distinct logical memory (requires data copy/transfers)
- Data movement:
  - **Streaming:** data is streamed directly between components (fast, but can increase idle time of components)
  - **Staging:** data is staged in memory or on disk (slower, but can reduce idle time and encourage data reuse)

Considerations for Sim-AI Coupled Workflows

Task Launching and Placement

Are the components run by the same or different processes?

Where does each component run?

How are components coupled? Any dependencies?

Do components need MPI? At what scale do components run?

Data Sharing and Transfer

Do components have data dependencies? How often is the data reused?

What is the size of the data? How far does it gave to move?

What data transfer latency is acceptable?

Elasticity

Is the problem size fixed or elastic?

Can the workflow run in a single batch job or needs many PBS/Slurm jobs?

Does the workflow need automation?

Do the workflow or external events generate new tasks?

Single vs. Multiple Systems

Does the workflow run on different systems? E.g., experimental facility + HPC, or Aurora + AI Testbed.

How is the work orchestrated across systems? Is co-scheduling needed?

How is data moved across systems?

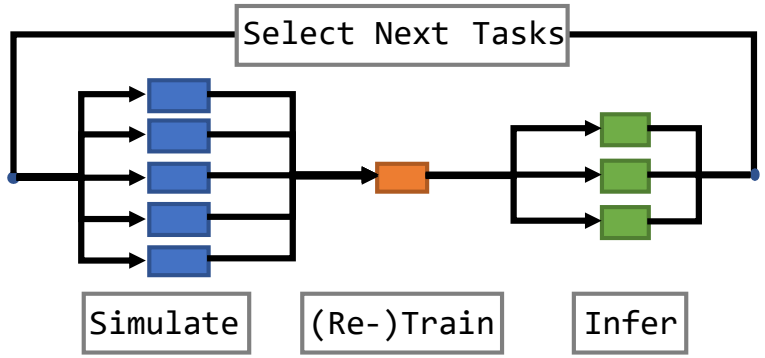
Software Tools for Sim-AI Workflows

- Tight coupling
  - LibTorch: C++ distribution of PyTorch
  - DLPack: Python & C API for zero-copy data exchange between many frameworks (NumPy, PyTorch, CuPy, dpnp, mpi4py, JAX, …)
- Loose coupling
  - Parsl: Python workflow tool for distributed, parallel task execution
  - DragonHPC: Runtime for managing process launching, memory and data at scale through high-performance RDMA transfers
  - ADIOS2: Unified I/O API for data management across different media (parallel file system, RDMA streaming, in-memory staging)
  - SmartSim/SmartRedis: Workflow and C/C++/Fortran client libraries for sim+AI workflows leveraging data staging through Redis database
  - And many more …

Hands-On Example

Model Fine-Tuning via Active Learning

- Simple molecular design application combining simulation with ML training and inference
- **Problem statement:** Identify molecules with largest ionization energy (IE) from large search space of candidates.
- **Solution:** Using simulation tools to screen through large sets of molecules can be very expensive. We can use ML surrogates and active learning to more efficiently screen through search space.
- **Active learning** (AL) is an iterative loop alternating simulation, training, inference and data acquisition
  1. Simulate initial batch of molecules to seed training data
  2. Train or fine-tune surrogate model
  3. Predict target quantity (IE) with surrogate model on entire search space
  4. Simulate the top-K candidates predicted by module to get truth value
  5. Augment training data with new simulation results
  6. Repeat steps 2-5 until model satisfies desired accuracy



Model Fine-Tuning via Active Learning

- **Hands-on material**

https://github.com/argonne-lcf/ATPESC_MachineLearning/tree/master/09_workflows_coupling_simulation_and_AI

- **Tools used**
  - Simulation: python-xtb -- Python API for the extended tight binding (xTB) program
  - Surrogate:
    - MoLFormer-XL open source model (available on Hugging Face)
    - Pre-trained on ~1.1B molecules with ~50M parameters
    - We freeze the encoder (backbone) and add a linear head to predict scalar IE values
  - Dataset: QM9 dataset with ~130,000 organic molecules
  - Workflow tools: Parsl and DragonHPC

Hands-On Exercises

1. Get an interactive session on Aurora with 2 nodes (1 node is also fine)

```
qsub -I -A ATPESC2026 -q ATPESC -l select=2 -l walltime=01:00:00 -l filesystems=home:flare
```

2. Clone the repository, cd to the example directory 09_workflows_coupling_simulation_and_AI and source the environment

```
source 0_activate_env.sh
```

Since we need a new conda env for this example, the script moves a prepackaged conda env to /tmp on all nodes. This is highly needed, working from Lustre when using large libraries like PyTorch can be very slow. Check it out!

Hands-On Exercises

3. **Q1:** To start, estimate how long it would take to run xTB simulations of all molecules in the QM9 dataset on 2 and 16 nodes of Aurora.

```
python 1_simulate_molecules.py
```

Hints:

- The script simulates 1020 molecules by default using Parsl
- Look at the Parsl config in utils/parsl_config.py to see how many workers per node are used

4. **Q2:** How long would it take to simulate the ZINC22 datasets containing ~5 billion molecules on all of Aurora?

Hands-On Exercises

5. Now let's dive into the active learning workflow, starting with the Parsl implementation which moves the data through Parsl's AppFutures.
6. Run the Parsl workflow with

```
python 2_parsl_futures.py
```

7. **Q3:** Based on the logs, which components of the workflow take the longest? Why?

Hints:

- Consider what data is transferred between components and how it is transferred

Fun fact, inference with the MolFormer surrogate is more than 10,000x faster than the xTB simulation!

Hands-On Exercises

8. **Q4:** To get around sending large datasets through the Parsl infrastructure, we can modify the Parsl apps to pass the model weights and chunked smiles for inference through the disk. Modify the `train_model_app` and `inference_app` functions, as well as the loop over SMILES chunks, to exchange data through Lustre then run the script.

```
python 3_parsl_io.py
```

9. **Q5:** What is the workflow overhead now for the training and inference components?

10. **Q6:** What can we expect from this data movement approach as the scale of the data and number of workers increases?

Lesson 1: Sim-AI workflows often involve large and frequent data movements between components. Considering the best approach for the application can be critical for performance.

Hands-On Exercises

11. **Q7:** Now let's look at the Dragon implementation, here we make use of the DDict to store model weights and SMILES in memory to avoid IO to Lustre. Modify the `train_model_app` and `inference_app` functions, as well as the loop over SMILES chunks, to exchange data through the DDict then run the script.

```
dragon 4_dragon.py
```

12. **Q8:** Why is the workflow overhead for the training and inference components larger than the Parsl I/O case? What can we do to reduce this overhead?

Hints:

- Consider how the components are launched in the Dragon implementation compared to Parsl

Hands-On Exercises

13. **Q9:** Now run the Dragon implementation from the solution directory, what is the workflow overhead now?

> **Lesson 2:** Iterative sim-AI workflows require components to be launched repeatedly across nodes. Process launching can be a significant bottleneck too and should be reduced as much as possible.

Conclusions

- Navigating the space of coupled simulation-AI workflows can be complex
- Many elements to consider when running on HPC
  - Process launching and task placement (tight vs. loose coupling)
  - Data management and transfer
  - Elasticity and job orchestration (possibly across systems)
  - Many workflow tools to choose from, each with specific characteristics
- Active learning (AL) is a useful strategy for efficient search space exploration and model training/fine-tuning combining simulation and AI/ML
- We used three implementations of an AL workflow to highlight two common bottlenecks:
  - Repetitive process launching can incur significant overhead, especially at scale
    - ✓ Consider reusing workers where possible!
  - Data movement can be expensive
    - ✓ Consider I/O vs. in-memory staging/streaming and co-locating data with workers

Optional Homework

- Optional homework invites you to look into the effectiveness of the workflow at fine-tuning the model
- **Goal:** Improve the model's selection of best candidates *and* prediction error by exploring different acquisition functions to more effectively augment the training dataset
- **Solution approach:**
  - Follow suggestions in README for other common acquisition functions or implement your own
  - Modify the workflow parameters to balance initial exploration with exploitation

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.