# Streamlining Machine Learning for Molecular Dynamics by Interfacing Python with LAMMPS


Steven Anaya

HPC-DES

sanaya@lanl.gov

Mentor: Nicholas Lubbers (CCS-3)


## Motivation

- LAMMPS is a gold-standard molecular dynamics simulation program written in C++
- The ML-IAP (Machine-Learning Interatomic Potential) package provides a simplified interface for connecting external C++ codes into LAMMPS
- However, many advanced ML packages are written in Python
- While LAMMPS allows users to couple Python models with their interatomic potentials through its Python package, this is meant for prototype codes and only allows for simplified potentials
- **LAMMPS lacks a direct interface to arbitrary Python models**

## The ML-IAP Unified Interface

- The new "ML-IAP Unified" Interface unifies both model and descriptor functionalities into a single streamlined Python interface
  - Descriptors give features but neural net finds own
- **This allows users to construct elegant Python models that seamlessly interface with LAMMPS using familiar methods**
- Enables computational scientists to more quickly and easily construct Python models using GPU-accelerated machine-learning interatomic potentials for use with LAMMPS
- **Allows for easier usage of PyTorch-based machine learning models such as HIP-NN with LAMMPS**

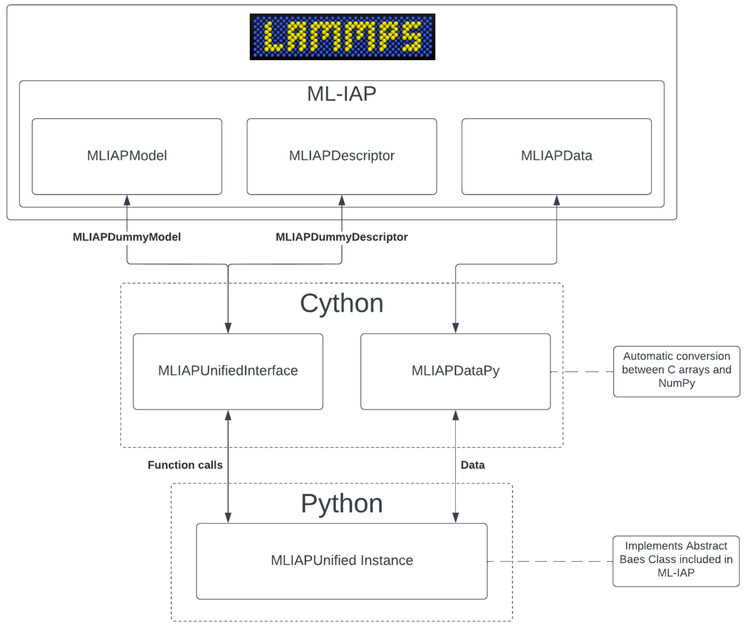
## Background: HIP-NN

- Convolutional neural network designed for interatomic potentials
- Highly-tunable parameters, performance
- Implemented in Python using PyTorch

## Challenges Encountered

- Creating direct, two-way Cython interface
- Implementing automatic data conversion between C++ and NumPy
- Supporting modified atom neighbor lists for models that require them
- Supporting multiple atom types (see InP Example)
- Identifying and addressing Python bottlenecks

## Unified Interface Architecture




## InP Example: Unified HIP-NN vs SNAP

- SNAP: machine-learning interatomic potential that uses bispectrum components
- Unified HIP-NN has approx. 9✕ shorter loop time
- Energy and pressure differ considerably, but neither model considered ground truth

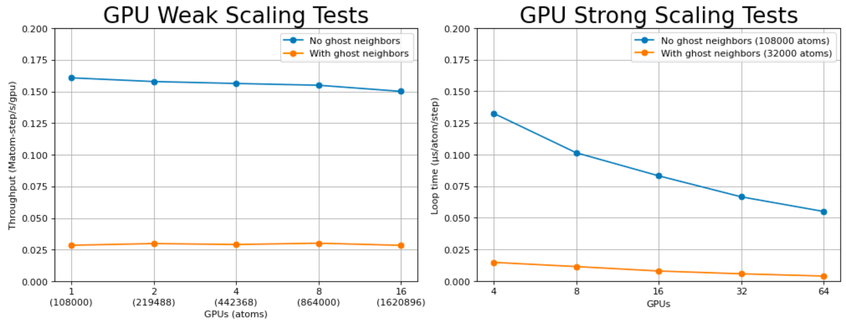
**Unified HIPNN** — *in.mliap.unified.hippynn.InP*

```
Per MPI rank memory allocation (min/avg/max) = 996.7 | 996.7 | 996.7 Mbytes
   Step          Temp          E_pair          E_mol          TotEng          Press
         0          300   -4.8412739               0   -4.8025716   -2305.4876
        10    289.4449   -4.8399121               0   -4.8025714   -2253.981
        20    259.98539   -4.836111               0   -4.8025709   -2005.1126
        30    218.06383   -4.8307021               0   -4.8025701   -1448.3016
        40    173.34948   -4.8249327               0   -4.8025693   -686.48167
        50    136.32353   -4.8201554               0   -4.8025686    13.292264
        60    114.93675   -4.8173961               0   -4.8025684    410.54349
        70    111.96058   -4.8170124               0   -4.8025686    404.16934
        80    124.36461   -4.8186132               0   -4.8025693    58.262582
        90    144.90639   -4.821264                0   -4.80257     -422.85179
       100    164.9825    -4.8238546               0   -4.8025705   -800.75432
Loop time of 2.42707 on 1 procs for 100 steps with 512 atoms

Performance: 1.780 ns/day, 13.484 hours/ns, 41.202 timesteps/s
99.4% CPU use with 1 MPI tasks x 1 OpenMP threads
```

**SNAP** — *in.snap.InP.JCPA2020*

```
Per MPI rank memory allocation (min/avg/max) = 6.027 | 6.027 | 6.027 Mbytes
   Step          Temp          E_pair          E_mol          TotEng          Press
         0          300   -3.4805794               0   -3.4418771    1353.5968
        10    285.84677   -3.4787531               0   -3.4418766    1611.7131
        20    248.14649   -3.4738884               0   -3.4418756    2312.0308
        30    198.94136   -3.4675394               0   -3.4418744    3168.1543
        40    152.74831   -3.4615791               0   -3.4418734    3903.5749
        50    121.9796    -3.4576091               0   -3.4418728    4387.1254
        60    113.27555   -3.4564863               0   -3.4418729    4556.3003
        70    125.68089   -3.4580873               0   -3.4418735    4431.2083
        80    151.47475   -3.4614159               0   -3.4418745    4107.2369
        90    179.18708   -3.4649919               0   -3.4418754    3739.5881
       100    197.50662   -3.4673559               0   -3.441876     3492.7778
Loop time of 21.9953 on 1 procs for 100 steps with 512 atoms

Performance: 0.196 ns/day, 122.196 hours/ns, 4.546 timesteps/s
99.8% CPU use with 1 MPI tasks x 1 OpenMP threads
```

## Unified HIP-NN GPU Scaling Tests




- Tested on Chicoma with NVIDIA A100 GPUs
  - 1 dual-socket CPU per node
  - Weak scaling: 1 GPU per node
  - Strong scaling: 4 GPUs per node
  - 1 process per GPU
  - 16 OpenMP threads per process
- **Higher is better; flat, horizontal line is ideal**
- Efficient scaling with modified (ghost) neighbor lists
- HIP-NN highly competitive with other ML interatomic potentials such as SNAP and ASE

## Conclusion

- LAMMPS enhanced with support for arbitrary, easy-to-make Python models
- **ML and GPU acceleration provide LAMMPS Python models with major performance advantage**
- ML-IAP Unified + HIP-NN highly competitive with other machine-learning interatomic potentials
- **ML driving HPC into the future!**

## Future Work

- Successfully merge code into LAMMPS GitHub repo
- Produce thorough documentation of each Unified Interface component and usage
- Perform scaling tests with more than 64 GPUs
- Implement other ML interatomic potential models

Los Alamos NATIONAL LABORATORY


LA-UR-22-28132