SCALING DISTRIBUTED MACHINE LEARNING INTERATOMIC POTENTIAL INFERENCE WITH DISTMLIP2

Anonymous authors
Paper under double-blind review

ABSTRACT

Atomistic simulation has been the primary workhorse in computational materials and drug discovery over the past 60 years. However, the quantum mechanical calculations pivotal for the effective simulation of atomistic systems, such as density functional theory (DFT), remain too slow for practical use in large scale systems. In recent years, machine learning interatomic potentials (MLIPs), machine learning models trained on DFT energy and force labels, are capable of reaching DFT-levels of accuracies while performing orders of magnitude faster inference than traditional DFT workflows. Distributing MLIP inference across many GPUs poses as a powerful method for scaling simulation sizes and time scales even further. In this work, we present DistMLIP2, where we introduce four elements enabling a new state of the art in distributed MLIP inference: **runtime skin recomputation, partial graph reconstruction, hierarchical partitioning and partition cycling,** and **graph computation pruning.** We implement four widely used MLIPs into DistMLIP2: UMA, MACE, Orbv3, and NequIP, and show DistMLIP2 pareto-dominates the previous state of the art distributed inference methods such as ALCHEMI, LAMMPS-KOKKOS, and DistMLIP when controlling for MLIP size, architecture, kernels, and inference settings. Most notably, DistMLIP2 achieves **2.4x faster inference time** than ALCHEMI while simulating **2.16x more atoms** compared to LAMMPS. With partition cycling enabled, we show that DistMLIP2 can simulate systems containing 4-5x the atom count of its previous limit.

1 INTRODUCTION

Simulating the motion of physical systems at the atomic level has been pivotal to the human understanding of the nanoscale world. Quantum mechanical methods, such as density functional theory (DFT), provide high-fidelity calculations on the movement of atoms. However, DFT is prohibitively slow and infeasible for simulations on realistic length and time scales, often only able to simulate on the order of 0.01 nanoseconds per day of wall-clock time for a few hundred atoms (Yokelson et al., 2022). In order to get around this computational bottleneck, machine learning interatomic potentials (MLIPs) – machine learning models trained on DFT energy and force labels – have been developed as computational surrogates to DFT. MLIPs, typically graph neural networks, achieve near-DFT level accuracy while performing inference thousands of times faster.

Modern MLIPs can achieve on the order of 1 nanosecond per day of wall-clock time for a few thousand atoms (Wood et al., 2025). However, many important biological and chemical phenomena of interest occur on the microsecond or millisecond timescales at the length scale of hundreds of thousands or millions of atoms (Henzler-Wildman et al., 2007; Voter et al., 2002). Therefore, efficient MLIP architectures, distillation methods, and custom CUDA kernels become pivotal to accelerating atomistic simulations. Efficient multi-GPU distributed inference also becomes crucially important for scaling MLIP inference to enable chemists and biologists to study physical phenomena that were previously unstudiable due to limitations in high-fidelity simulation capability.

Current distributed inference frameworks for MLIPs, such as DistMLIP, LAMMPS-KOKKOS, and ALCHEMI all follow the same flavor of distributed inference: partition the underlying space into separate GPUs, expand each partition to a 1-hop HALO region, perform a single graph convolution, and then communicate necessary HALO information to other GPUs for the next convolution (Han et al., 2025; Johansson et al., 2025; Tsai et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

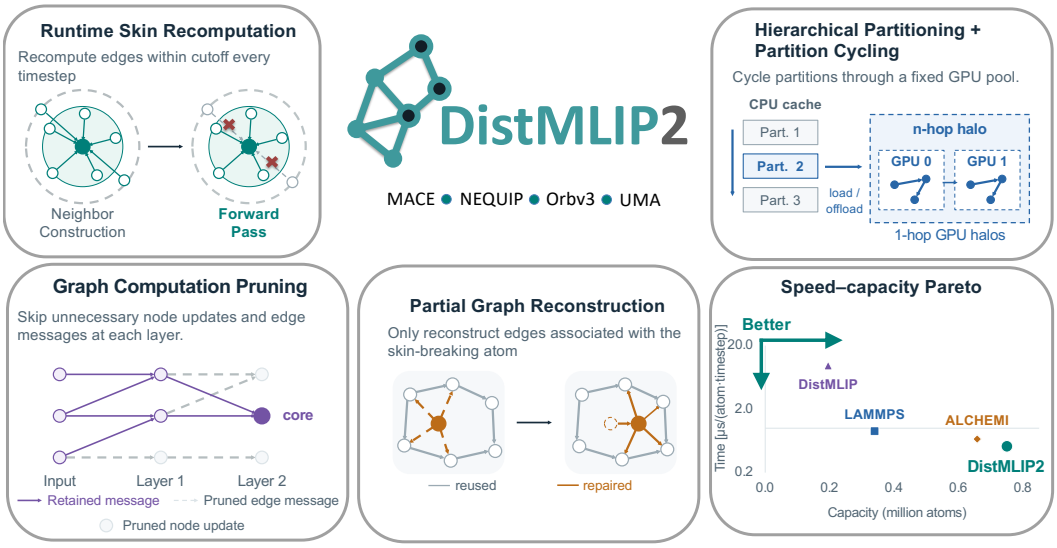

Figure 1: An overview of the DistMLIP2 platform, highlighting the four new components of the system: runtime skin recomputation, graph computation pruning, hierarchical partitioning and partition cycling, and partial graph reconstruction. DistMLIP2 outperforms current distributed MLIP inference systems in both simulation time and simulation capacity.

In this work, we introduce and rigorously benchmark DistMLIP2, a new state of the art distributed inference platform for MLIPs. On top of a major redesign from DistMLIP, DistMLIP2 also integrates four new components:

1. **Runtime skin recomputation**: During each inference step, we efficiently recompute the edges beyond the cutoff radius that were included in the atom graph for skin reasons. Effectively reducing the total number of edges required for inference.
2. **Partial graph reconstruction**: During skin breaks, instead of fully recomputing the neighbor list, we selectively only recompute the edges associated with the skin-breaking atom.
3. **Hierarchical partitioning** and **partition cycling**: Instead of expanding each partition by its 1-hop HALOs, we designate 2 tiers of partitions. One tier can expand at arbitrary n-hops while the second expands at 1 hop. This enables the first tier partition to be cycled on and off GPUs, allowing DistMLIP2 to beat the 1:1 simulation size to GPU memory ratio and simulate arbitrarily large systems using a fixed amount of GPU memory.
4. **Graph computation pruning**: We identify the node updates and edge messages required at each layer of the MLIP and skip unnecessary computation while preserving physically conservative energy and forces.

We implement 4 popular, state of the art MLIPs on DistMLIP2: UMA, MACE, Orbv3, and NequIP, and benchmark against DistMLIP, LAMMPS-KOKKOS, and ALCHEMI. We also benchmark the DistMLIP2 UMA implementation against the built-in UMA multi-GPU implementation (Wood et al., 2025). Benchmarking with the same model architectures, kernels, and inference settings, **DistMLIP2 pareto-dominates the previous state of the art distributed inference frameworks in both inference time and simulation capacity**. Notably, we show that DistMLIP2 is 2.4x faster than ALCHEMI while simulating 2.16x more atoms than LAMMPS. Furthermore, with hierarchical partitioning and partition cycling, we show that it is possible to break the 1:1 relationship between GPU memory and simulation capacity by scaling the atom count to 4-5x past the out-of-memory ceiling with only slight reduction in overall simulation throughput.

With this work, we establish DistMLIP2 as the latest state of the art distributed inference platform in both simulation speed and simulation capacity.

2 Related Works

Machine learning interatomic potentials (MLIPs) are graph neural networks (GNNs) that input atomic positions and element types and output the potential energy of the atomic system as well as the forces the atoms apply onto one another (Deringer et al., 2019). The underlying graph of the atomic system is a simple directed graph $\mathcal{G} = (V, E)$ where V, the nodes of the graph, represent the set of atoms in the system while E, the edges, are initialized when two atoms are within a cutoff distance r from each other. The MLIP is then trained on the energy and force labels calculated from high fidelity first principles quantum mechanical calculations such as density functional theory (DFT) or coupled cluster (CC) methods (Perdew et al., 1996; Raghavachari et al., 1989).

MLIPs such as MACE (Batatia et al., 2022), CHGNet (Deng et al., 2023), UMA (Wood et al., 2025), NequIP (Batzner et al., 2022), and Orbv3 (Rhodes et al., 2025) have been shown to deliver near-DFT levels of accuracy when trained on datasets such as OMol25 (Levine et al., 2025), OMat24 (Barroso-Luque et al., 2024), and MPTraj (Deng, 2023) while performing energy and force calculations several orders of magnitude faster than traditional DFT (Zhang et al., 2018; Smith et al., 2017; Xie et al., 2023). However, there still remains a large gap between the inference time and simulation capacity of MLIPs compared to classical molecular dynamics, where interatomic potential functions are defined by empirical or semi-empirical equations instead of GNNs (Kovács et al., 2025). Classical interatomic potential functions such as embedded atom methods (Daw & Baskes, 1984), CHARMM (Vanommeslaeghe et al., 2010) and Amber (Wang et al., 2004) are intuitive, explainable, and fast, but are often not sufficiently accurate or generalizable across chemical domains – forcing computational scientists to rely heavily on MLIPs as their primary workhorse for materials and drug discovery (Leimeroth et al., 2025).

There have been many recent efforts to accelerate MLIP inference as well as increase simulation capacity. Cuequivariance, OpenEquivariance, and FlashTP present CUDA kernels aimed at making the expensive tensor products and spherical convolutions used in architectures such as MACE and NequIP more memory and FLOP efficient (Geiger et al., 2024; Bharadwaj et al., 2025; Lee et al., 2025). The TensorNet, E2Former-V2, and Orbv3 MLIPs attempt to make model design decisions, such as graph convolutions without expensive tensor products, to reduce inference time during simulation (Simeon & De Fabritiis, 2023; Huang et al., 2026; Rhodes et al., 2025). Han et al. (2026) also introduces the notion of dynamic cutoff models, where the cutoff radius r is allowed to vary during a simulation to reduce the number of required edge computations during a graph convolution.

Highly accurate, but slow, foundational MLIPs that have been trained on the entire periodic table can be used to generate synthetic training labels for smaller, student models (Gardner et al., 2025). Work has also been done on using energy hessians to distill teacher model capabilities into smaller, faster MLIPs (Amin et al., 2025). Distillation methods result in faster, highly performant student models; however, student models are typically restricted to high accuracy only in specific chemical domains.

Distributed inference frameworks such as DistMLIP, LAMMPS KOKKOS ML-IAP, and ALCHEMI, are platforms built on top of various MLIPs designed to deliver fast multi-GPU inference of the MLIP (Han et al., 2025; Johansson et al., 2025; Tsai et al., 2026). They all incorporate multi-GPU inference by mutually exclusively partitioning the underlying atomic graph, expanding the graph by its 1-hop neighbors to perform as HALO nodes, and then placing each partitioned graph on its own GPU. Each GPU then performs a single convolution over their partitioned copy of the graph before communicating necessary HALO node features between GPU ranks and performing the next convolution.

Allegro (Musaelian et al., 2023) and SevenNet (Park et al., 2024) are MLIPs with native distributed inference capabilities built on top of LAMMPS domain decomposition (Johansson et al., 2025). Chemtrain-Deploy is another framework allowing the easy deployment of Jax-based MLIPs to LAMMPS (Fuchs et al., 2025). Ultimately, Chemtrain-Deploy also relies on LAMMPS domain decomposition as its distributed inference platform, which we benchmark heavily against. We also benchmark against the UMA MLIP's built-in distributed inference pipeline (Wood et al., 2025).

In this work, we introduce DistMLIP2, a novel distributed inference platform which achieves state of the art results in both simulation speed and simulation scale. We also show arbitrarily scaling of simulation size at small cost to overall throughput via hierarchical partitioning and partition cycling.

3 DISTMLIP2

DistMLIP2 utilizes efficient CUDA kernels for high throughput, high capacity distributed inference using MLIPs. DistMLIP2 implements runtime skin recomputation for more efficient skin usage during inference, partial graph reconstruction to minimize the overall time spent on neighbor list construction, hierarchical partitioning and partition cycling to enable a fixed GPU memory pool to simulate arbitrarily many atoms, and graph computation pruning to restrict MLIP computation to only the most required node feature and edge message operations.

Specific details on these algorithms and supporting subprocedures can be found in Appendix A.4 and A.5.

3.1 RUNTIME SKIN RECOMPUTATION

In DistMLIP2, every step of MLIP inference consists of two distance calculations over the same set of atom positions: the first calculation over distances is invoked by the neighbor list construction algorithm. The second calculation is invoked when calculating distances or bond vectors in order to featurize the MLIP. The second calculation is required in order to attach the atomic positions to the autograd computation graph when calculating conservative forces. We can take advantage of the two distance computations by utilizing the neighbor list distance calculation to first acquire all neighbors within a cutoff radius as well as an additional skin radius, $r_c + r_{\text{skin}}$. During each individual MD step, we can utilize the second calculation to **prune** edges from the neighbor list construction that are beyond our desired cutoff r_c.

The additional skin radius allows for the amortization of full neighbor list reconstruction over many timesteps without incurring additional MLIP-specific inference time or memory overhead – the only incurred overhead comes from the pruning operation, but this is typically minimal compared to the computation saved from MLIP inference. The set of edges E associated with the construction of the larger neighbor list with cutoff $r_c + r_{\text{skin}}$ is a superset of true edges E^* with cutoff r_c. E is cached for future timesteps until an atom with index i travels further than $r_{\text{skin}}/2$ with respect to its reference coordinate x_i^{ref}. At this point, the atoms becomes "dirty" and the graph must be reconstructed.

This redundant pruning mechanism during the autograd distance calculation should be interpreted as the MLIP equivalent to skins from classical MD literature (Colberg & Höfling, 2011).

3.2 PARTIAL GRAPH RECONSTRUCTION

Performing an exact full graph reconstruction is computationally expensive and inefficient. After a full graph construction is performed, we maintain a set of reference coordinates x^{ref} that refer to the coordinates of the atoms at the most recent full graph or partial graph reconstruction. Using these coordinates, subsequent graph constructions on skin breaks can then be done in a partial manner – only reconstructing the graph associated with atoms that break the skin cutoff.

Concretely, at every timestep, we construct

$$D = \left\{ i : \|x_i - x_i^{\text{ref}}\|_2 \geq \frac{1}{2} r_{\text{skin}} \right\} \tag{1}$$

where D represents the set of atoms which broke the skin barrier from the previous time the underlying graph was reconstructed (either fully or partially).

Given the stored superset of edges E, we then construct a subset of "clean" edges that can be reused for future timesteps and do not need to be modified for correctness

$$E_{\text{sub}} = \{(j, i, o, d) \in E \mid j \notin D \wedge i \notin D\} \tag{2}$$

Afterwards, we perform a cell-based neighbor search on only the skin-breaking, dirty atoms to create the updated set of edges for these atoms and concatenate them with E_{sub} to form the new list E' for the next timestep. Finally, we update x^{ref} for all dirty atoms.

We force a full graph reconstruction under two conditions: one, when the per-axis coordinate bounds need to be changed due to large-scale atomic movements; and two, when we need to perform load rebalancing across the nodes and subworkers.

Empirically, we find that only a small subset of atoms deviate by more than $\frac{1}{2}r_{\text{skin}}$ from the reference position. Therefore, this step notably decreases the scope and runtime of full graph construction, resulting in reduced end-to-end inference time.

3.3 Hierarchical Partitioning

When performing any graph construction, DistMLIP2 first organizes the atomic system into bins. We construct a grid of bins with side length $r_c + r_{\text{skin}}$ in Euclidean coordinates using the given crystal lattice as well as the per-axis bounds of the systems given by the coordinates. Then, we fit the bins onto a Z-curve using the (x, y, z) indices of the bins in the grid. For every atom in the system, we place it in a bin based on its fractional coordinates with respect to the lattice.

We perform hierarchical partitioning on the set of sorted bin indices, one for the worker-level partitions and one for the subworker-level partitions. After sorting all atoms by their bin index, we find cutoffs $c = \{0, c_1, c_2, \ldots, c_{n-1}, n_b\}$ such that the number of atoms in bins $[b_{c_i}, b_{c_{i+1}})$ is roughly equal for all i. Each range becomes a worker-level partition P_w.

To reduce inter-node network communication, which would otherwise occur at every layer during inference, we propose an "n-hop expansion" strategy, where we expand P_w by $n-1$ bins in all directions (with the last expansion "hop" occurring naturally during neighbor search in graph construction) to create Q_w. We call the original P_w the "core" portion of the partition and the expanded region $Q_w \setminus P_w$ the "expansion" portion.

After expansion, we partition P_w among the subworkers using a layered balancing approach. Let d be the "depth" of a partition based on the number of expansion hops used for that bin, starting with 0 for the core partition. First, partition the core section among the subworkers using the same algorithm as above. Then, for each successive layer of depth d, we estimate the work of bin b_i to be

$$w_{b_i} = c_{b_i}\left(1 + \sum_{b_j \in \mathcal{N}(b_i)} c_{b_j}\right) \tag{3}$$

where c_{b_i}, c_{b_j} is the number of atoms in bins b_i and b_j, respectively, and $N(b_i)$ is the set of neighbors of b_i. We then partition the layer into segments among the subworkers such that the estimated work is roughly equivalent across bands. Finally, we assign each segment t to an existing partition P_s. We assign segments to partitions in a way that maximizes the overall contact cost, calculated as

$$C_{t,p} = \sum_{b_i \in t} \sum_{b_j \in \mathcal{N}(b_i) \cap P_s} c_{b_i} c_{b_j} \tag{4}$$

for each pair (t, P_s). Performing this across all layers gives the subworker-level partitions. This layer-aware load balancing approach is required due to the layer-dependent computation pruning described in Section 3.5 – different layers of the MLIP have different associated computation costs which must be load balanced amongst HALO regions.

Expansion reduces global communication cost. Performing n convolutions on an atom a in expanded n-hop worker-level partition Q_w sets the interaction radius of a to nr. Since the width of a bin is at least r, the set of atoms in that interaction range is a subset of Q_w. Because MLIPs perform one convolution per layer, we can conclude that Q_w contains all atoms necessary for all n convolutions over a. Therefore, our hierarchical partitioning method reduces inter-node communication to only once every n layers at the cost of increased workload per node.

3.4 Partition Cycling

Hierarchical partitioning with multi-hop expansion enables partition cycling. In partition cycling, we can set an arbitrary number of logical nodes n_ℓ, which may exceed the number of physical nodes. For each node, we always perform a D-hop expansion, where D is the depth of the MLIP. Then, we cycle through all n_ℓ worker-level partitions by iteratively loading each partition onto any available physical node. Because we perform a D-hop expansion for each partition, the receptive field of each pure section is entirely contained within the expanded partition. Hence, communication between partitions is unnecessary until the end of the inference step, allowing each worker to sequentially and independently calculate energies and forces for multiple partitions. By decoupling molecular

system capacity from compute availability, DistMLIP2 enables MLIPs to simulate large systems on small distributed setups at the cost of slightly decreased throughput per step.

3.5 Graph Computation Pruning

We restrict each convolution layer within the MLIP to only the node updates and edge messages required by subsequent computation. The existence of unnecessary node feature updates and message passing is a direct result of the existence of, potentially multiple hops, of redundant HALOs within the partition. All updates that are not associated with calculating the energy and forces for the core atoms assigned to the GPU are subsequently pruned.

Concretely, let $\mathcal{E}_{\text{cand}}$ represent the set of all directed edges with a destination node within a GPU-level partition. Let $\mathcal{C}$ denote the core atoms owned by the GPU worker. For a block of B layers between inter-GPU communication points, we construct the required atom node sets, $\mathcal{V}_B$ backward from the core atoms we own:

$$\mathcal{V}_B = \mathcal{C}, \qquad \mathcal{V}_{\ell-1} = \mathcal{V}_\ell \cup \{u \mid (u,v) \in \mathcal{E}_{\text{cand}},\ v \in \mathcal{V}_\ell\}, \quad \ell = B, \ldots, 1. \tag{5}$$

This backward expansion is implemented as a parallel breadth first search and retains incoming message dependencies. At layer ℓ, each GPU also updates only the node features for its assigned nodes in $\mathcal{V}_\ell$. Note in the 1-hop HALO scenario, it is possible that every edge from the HALO region is retained. In this case, the effects of graph computation pruning primarily lie in the pruning of the node feature updates for the HALO nodes themselves.

Additionally, each $\mathcal{V}_\ell$ is computed and cached after each full graph and partial graph reconstruction – effectively amortizing the additional computational overhead associated with the breadth first search across potentially thousands of timesteps. The isolated effects of graph computation pruning can be found in Table 3.

4 Results

We implement four models in DistMLIP2: **MACE** (Batatia et al., 2022), an equivariant MLIP that explicitly models many-body interactions between atoms; **NequIP** (Batzner et al., 2022), a data-efficient equivariant MLIP; **Orbv3** (Rhodes et al., 2025), an invariant model designed with inference speed in mind; and **UMA** (Wood et al., 2025) a large mixture-of-experts style foundational MLIP achieving state of the art performance on many large datasets. We benchmark the pretrained variants of each MLIP: MACE-MP-0-small (3.8 million parameters), NequIP[1] (1.6 million parameters), Orb-v3-conservative-inf-omat (25.5 million parameters), and UMA-S 1.2.1 (291.5 million total parameters, 6.3 million active parameters). More details on the models can be found in Appendix B. We benchmark model inference for 100 steps after 10 steps of warmup on H100 GPUs. We also show real simulation stability of the DistMLIP2 platform in Figure 4 of Appendix A.2.

Table 1 outlines the MLIPs that are supported by each distributed inference platform. Native UMA-Distributed refers to the built-in distributed inference capabilities of the UMA model (Wood et al., 2025). For fair comparison, we benchmark DistMLIP2 against the supported models for each of the other inference platforms using the same inference settings, CUDA kernels, and model sizes. The only component being ablated is the choice of distributed inference platform. More details on the models themselves as well as the inference settings can be found in Appendix B.

Table 1: Model support among the current distributed inference platforms.

Inference platform	MACE	NequIP	Orbv3	UMA
DistMLIP	✓	—	—	—[2]
LAMMPS–KOKKOS	✓	✓	—	—
ALCHEMI	✓	—	—	—[2]
Native UMA-Distributed	—	—	—	✓
DistMLIP2	✓	✓	✓	✓

[1]For NequIP, we used a custom 1.6 million parameter variant. More details can be found in Appendix B.
[2]DistMLIP and ALCHEMI support UMA-S-1.0.0, but not UMA-S-1.2.1.

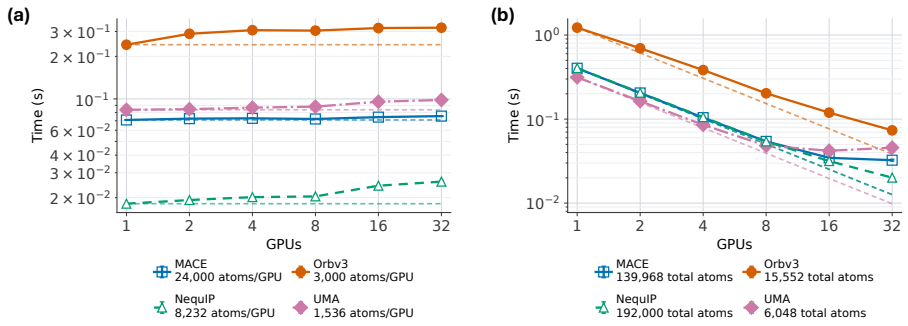

Figure 2: (a) The weak scaling behavior of DistMLIP2. The number of atoms/GPU is held constant while the SiO_2 supercell is expanded cubically. Measured inference time is end to end latency. For reference, ideal scaling is also plotted. (b) The strong scaling behavior of DistMLIP2. The total number of atoms is held constant while the number of GPUs increases. Divergence from ideal scaling at larger GPU counts indicate overheads dominating over computation performed by a single GPU worker.

4.1 Weak and Strong Scaling

We benchmark both the weak and strong scaling capabilities of DistMLIP2 with all four supported models in Figure 2 as well as curves showing ideal scaling behavior. We use perturbed supercells of a crystalline α-quartz SiO_2 system for benchmarking scaling. For benchmarking the weak scaling capabilities of DistMLIP2, we hold the number of atoms per GPU constant while increasing the total number of GPUs. Weak scaling measures the largest system size that is solvable within a reasonable amount of time as the number of GPUs increases. The SiO_2 systems were expanded cubically when increasing cell size to best represent real world simulation workloads.

For benchmarking strong scaling capabilities, we hold the total number of atoms in the SiO_2 system constant while increasing the number of GPUs. Strong scaling measures how much faster simulation can complete when more GPU resources are applied.

As seen in the figure, DistMLIP2 shows effective weak and strong scaling behavior when scaling to 32 GPUs. Divergences in the UMA model strong scaling at 32 GPUs can be explained by the amount of work performed by each GPU being dominated by inter-GPU communication overhead once the number of GPUs causes a small number of atoms to be scheduled on each device.

4.2 Real World Simulation

We benchmark the performance of DistMLIP2 compared to other distributed inference platforms in Table 2 on real world systems. For fair comparison, we fix the model architecture, CUDA kernels, compilation settings, and other inference settings for all benchmarking. The only difference is the distributed inference platform itself. Each timing value is averaged across 100 steps of MD after 10 GPU warmup steps. All systems are cubic or near-cubic.

As seen from the table, DistMLIP2 consistently displays state of the art performance across multiple real-world scenarios and GPU counts. We also show real world DistMLIP2 results in Table 4 in Appendix A.3.

4.3 Partition Cycling

Simulation size is typically bottlenecked by the MLIP's requirement to hold intermediate activations for the backpropagation necessary to calculate conservative forces. Hierarchical partitioning combined with partition cycling (where a large amount of independent hierarchical partitions run serially on a small amount of GPU nodes) promises to break the one to one relationship between simulation capacity and GPU memory. In other words, given the same amount of GPU memory, partition cycling enables the simulation of an arbitrarily large amount of atoms expensed via a small decline in overall throughput.

Table 2: Real world comparisons grouped by model on 4, 8, 16, and 32 GPUs. Throughput is reported as T in μs/(atom·timestep) (lower is better); C is demonstrated capacity in millions of atoms, rounded to four significant figures (higher is better). Capsid (pdb: AAV7) is a large biomolecule protein, consisting of 249,360 atoms. Bold denotes the best displayed value within each model and GPU count. DistMLIP only supports inference up to 8 GPUs.

		H_2O ice		Li_3PO_4		GaN		MOF		Capsid
Implementation	GPUs	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$
MACE										
DistMLIP2	4	**1.022**	**0.3776**	**1.006**	**0.3869**	**0.860**	**0.4189**	**0.881**	**0.4560**	**0.601**
	8	**0.527**	0.6273	**0.518**	**0.7500**	**0.443**	**0.8173**	**0.460**	**0.8165**	**0.328**
	16	**0.277**	**1.248**	**0.272**	**1.498**	**0.236**	**1.539**	**0.243**	**1.786**	**0.203**
	32	**0.150**	**2.580**	**0.156**	**2.965**	**0.137**	**3.209**	**0.140**	**3.182**	**0.169**
LAMMPS	4	1.668	0.1697	1.646	0.1739	1.390	0.2119	1.426	0.2016	0.872
	8	0.909	0.3411	0.904	0.3400	0.765	0.4234	0.786	0.4080	0.508
	16	0.490	0.6653	0.488	0.6749	0.425	0.8454	0.423	0.8165	0.391
	32	0.288	1.325	0.285	1.373	0.261	1.692	0.272	1.632	0.293
ALCHEMI	4	1.257	0.3704	1.230	0.3707	1.091	0.4083	1.106	0.3888	0.742
	8	0.689	**0.6327**	0.675	0.6577	0.589	**0.8173**	0.604	0.7862	0.457
	16	0.454	1.100	0.447	1.156	0.473	1.483	0.379	1.364	0.495
	32	0.360	2.097	0.346	2.203	0.299	2.747	0.280	2.736	0.506
DistMLIP	4	13.458	0.1048	13.119	0.1094	11.633	0.1224	11.306	0.1201	OOM
	8	10.169	0.1915	9.691	0.1958	8.912	0.2283	8.519	0.2304	5.960
	16	—	—	—	—	—	—	—	—	—
	32	—	—	—	—	—	—	—	—	—
NequIP										
DistMLIP2	4	**0.668**	**0.9114**	**0.657**	**0.9293**	**0.554**	0.7504	**0.571**	**1.086**	0.445
	8	**0.350**	**1.806**	**0.340**	**1.848**	**0.291**	**2.218**	**0.299**	**2.154**	**0.247**
	16	**0.191**	**3.595**	**0.190**	**3.681**	**0.165**	**4.424**	**0.166**	**4.272**	**0.163**
	32	**0.116**	**7.130**	**0.115**	**7.290**	**0.101**	**8.779**	**0.104**	**8.467**	**0.120**
LAMMPS	4	0.869	0.6273	0.856	0.6310	0.737	**0.8067**	0.762	0.7800	**0.420**
	8	0.596	1.315	0.576	1.281	0.530	1.611	0.540	1.562	0.290
	16	0.350	2.528	0.351	2.501	0.302	3.209	0.299	3.110	0.221
	32	0.214	4.972	0.206	5.060	0.186	6.408	0.192	6.240	0.199
UMA										
DistMLIP2	4	**13.809**	**0.02304**	**13.560**	**0.02323**	**11.090**	**0.02870**	**11.506**	**0.02688**	OOM
	8	**6.907**	**0.04320**	**6.843**	**0.04570**	**5.601**	**0.05658**	**5.786**	**0.05400**	OOM
	16	**3.671**	**0.08736**	**3.544**	**0.09139**	**2.909**	**0.1117**	**3.002**	**0.1037**	OOM
	32	**1.910**	**0.1750**	**1.804**	**0.1760**	**1.475**	**0.2177**	**1.557**	**0.2160**	**0.869**
UMA-Distributed	4	15.121	0.01843	14.864	0.01760	12.181	0.02288	12.678	0.02016	OOM
	8	7.662	0.03600	7.536	0.03696	6.212	0.04234	6.434	0.04320	OOM
	16	3.990	0.07114	3.898	0.07296	3.256	0.08976	3.371	0.07920	OOM
	32	2.116	0.1348	2.086	0.1376	1.731	0.1797	1.799	0.1622	OOM

In Figure 3, we benchmark the MACE and NequIP MLIPs using DistMLIP2 partition cycling. Overall simulation throughput ($\mu s/$atom-timestep) shows little absolute decline as the number of partition cycles increases. With partition cycling over the MACE and NequIP models, DistMLIP2 is able to increase overall simulation capacity by 4-5x compared to the standard setup. The 1 cycle per node data point is calculated with 2-hop hierarchical partitioning for fair comparison. Although the benchmark in the figure used 2 nodes of 8 GPUs each, partition cycling is not necessarily restricted to multi-node GPU setups and works similarly in the single node/multi-GPU and even the single GPU regimes.

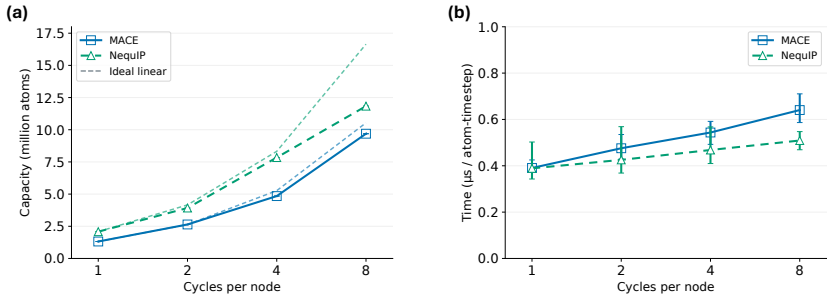

Figure 3: (a) The simulation capacity of the MACE and NequIP models under varying number of partition cycles per node. These experiments were run on 2 nodes of 8 GPUs each. Ideal linear scaling against the number of partition cycles is plotted. (b) Simulation throughput with respect to the number of cycles per node in μs/atom-timestep. The simulation sees little absolute degradation on overall throughput but is capable of scaling to 4-5x the overall simulation capacity.

4.4 Timing breakdown

In Table 3, we ablate inference time after the inclusion of each major DistMLIP2 component. We show the non-trivial impact each component has on reducing overall end-to-end inference latency on a 1.6 million atom system running a 3.8 million parameter MACE model with 16 GPUs.

Table 3: An ablation over the inference time when each major DistMLIP2 component is added. Full graph construction refers to DistMLIP2's standard graph construction. The simulated system was a 1.6 million atom cubic SiO_2 cell running on 16 GPUs. The model used was the 3.8 million parameter MACE-MP-0-small. The time column measures end-to-end inference time.

Configuration	Time (s)	Baseline speedup
Full graph construction	0.493	1.000×
+ Partial graph reconstruction	0.405	1.217×
+ Runtime skin recomputation	0.380	1.297×
+ Graph computation pruning	0.328	1.503×

5 Conclusion

In this paper, we introduce DistMLIP2, a novel distributed multi-node, multi-GPU molecular inference framework that achieves state-of-the-art performance in both runtime and system capacity. We accomplish this through introducing a collection of new components, including **partial graph reconstruction, runtime skin recomputation, hierarchical partitioning,** and **graph computation pruning.** Our experiments demonstrate that these system improvements result in up to 1.92x and 2.4x faster runtime compared to LAMMPS and ALCHEMI, respectively. DistMLIP2 also increases the maximum atom count by up to 2.16x compared to LAMMPS and 1.35x compared to ALCHEMI. Finally, we introduced **partition cycling,** which demonstrably enables users to scale to arbitrary system sizes at little cost in overall throughput. We believe that our advances in fast distributed inference for MLIPs will unlock scientists to now study previously un-studiable molecular systems at incredible time scales and system sizes – ultimately accelerating research in biology, chemistry, materials science, and drug discovery.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in implementing a small portion of the methods and providing feedback on research methodology or experiments. We have not used generative AI tools for developing theoretical models or conceptual frameworks, proposing or refining hypotheses, or supporting qualitative and thematic analysis or interpreting results, and generating synthetic datasets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools for editing software code and identifying relevant literature. We have reviewed all AI-assisted work. For example, all LLM-generated code was verified and tested for correctness by at least 2 authors as well as an extensive set of test cases, long-term stability tests, and multi-GPU discrepancy tests with single-GPU baselines. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ishan Amin, Sanjeev Raja, and Aditi Krishnapriyan. Towards fast, specialized machine learning force fields: Distilling foundation models via energy hessians. In *International Conference on Learning Representations*, volume 2025, pp. 71153–71175, 2025.

Luis Barroso-Luque, Muhammed Shuaibi, Xiang Fu, Brandon M. Wood, Misko Dzamba, Meng Gao, Ammar Rizvi, C. Lawrence Zitnick, and Zachary W. Ulissi. Open materials 2024 (omat24) inorganic materials dataset and models, 2024. URL https://arxiv.org/abs/2410.12771.

Ilyes Batatia, David P Kovacs, Gregor Simm, Christoph Ortner, and Gábor Csányi. Mace: Higher order equivariant message passing neural networks for fast and accurate force fields. *Advances in neural information processing systems*, 35:11423–11436, 2022.

Ilyes Batatia, Philipp Benner, Yuan Chiang, Alin M. Elena, Dávid P. Kovács, Janosh Riebesell, Xavier R. Advincula, Mark Asta, William J. Baldwin, Noam Bernstein, Arghya Bhowmik, Samuel M. Blau, Vlad Cărare, James P. Darby, Sandip De, Flaviano Della Pia, Volker L. Deringer, Rokas Elijošius, Zakariya El-Machachi, Edvin Fako, Andrea C. Ferrari, Annalena Genreith-Schriever, Janine George, Rhys E. A. Goodall, Clare P. Grey, Shuang Han, Will Handley, Hendrik H. Heenen, Kersti Hermansson, Christian Holm, Jad Jaafar, Stephan Hofmann, Konstantin S. Jakob, Hyunwook Jung, Venkat Kapil, Aaron D. Kaplan, Nima Karimitari, Namu Kroupa, Jolla Kullgren, Matthew C. Kuner, Domantas Kuryla, Guoda Liepuoniute, Johannes T. Margraf, Ioan-Bogdan Magdău, Angelos Michaelides, J. Harry Moore, Aakash A. Naik, Samuel P. Niblett, Sam Walton Norwood, Niamh O'Neill, Christoph Ortner, Kristin A. Persson, Karsten Reuter, Andrew S. Rosen, Lars L. Schaaf, Christoph Schran, Eric Sivonxay, Tamás K. Stenczel, Viktor Svahn, Christopher Sutton, Cas van der Oord, Eszter Varga-Umbrich, Tejs Vegge, Martin Vondrák, Yangshuai Wang, William C. Witt, Fabian Zills, and Gábor Csányi. A foundation model for atomistic materials chemistry. 2023.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13(1):2453, 2022. doi: 10.1038/s41467-022-29939-5.

Vivek Bharadwaj, Austin Glover, Aydın Buluç, and James Demmel. An efficient sparse kernel generator for o (3)-equivariant deep networks. In *2025 Proceedings of the Conference on Applied and Computational Discrete Algorithms (ACDA)*, pp. 32–46. SIAM, 2025.

Peter H. Colberg and Felix Höfling. Highly accelerated simulations of glassy dynamics using gpus: Caveats on limited floating-point precision. *Computer Physics Communications*, 182(5):1120–1129, 2011. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2011.01.009. URL https://www.sciencedirect.com/science/article/pii/S0010465511000294.

Murray S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, 29:6443–6453, Jun 1984. doi: 10.1103/PhysRevB.29.6443. URL https://link.aps.org/doi/10.1103/PhysRevB.29.6443.

Bowen Deng. Materials project trajectory (mptrj) dataset. *Figshare. https://doi. org/10.6084/m9. figshare*, 23713842:v2, 2023.

Bowen Deng, Peichen Zhong, KyuJung Jun, Janosh Riebesell, Kevin Han, Christopher J Bartel, and Gerbrand Ceder. Chgnet as a pretrained universal neural network potential for charge-informed atomistic modelling. *Nature Machine Intelligence*, 5(9):1031–1041, 2023.

Volker L Deringer, Miguel A Caro, and Gábor Csányi. Machine learning interatomic potentials as emerging tools for materials science. *Advanced Materials*, 31(46):1902765, 2019.

Paul Fuchs, Weilong Chen, Stephan Thaler, and Julija Zavadlav. chemtrain-deploy: A parallel and scalable framework for machine learning potentials in million-atom md simulations. *Journal of Chemical Theory and Computation*, 21(15):7550, 2025.

John LA Gardner, Daniel F Toit, Chiheb Ben Mahmoud, Zoé Faure Beaulieu, Veronika Juraskova, Laura-Bianca Paşca, Louise AM Rosset, Fernanda Duarte, Fausto Martelli, Chris J Pickard, et al. Distillation of atomistic foundation models across architectures and chemical domains. *arXiv preprint arXiv:2506.10956*, 2025.

Mario Geiger, Emine Kucukbenli, Becca Zandstein, and Kyle Tretina. Accelerate drug and material discovery with new math library nvidia cuequivariance, November 2024. URL `https://developer.nvidia.com/blog/accelerate-drug-and-material-discovery-with-new-math-library-nvidia-cuequivariance/`. Technical blog post.

Kevin Han, Bowen Deng, Amir Barati Farimani, and Gerbrand Ceder. Distmlip: A distributed inference platform for machine learning interatomic potentials. *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2025.

Kevin Han, Haolin Cong, Bowen Deng, and Amir Barati Farimani. Smooth dynamic cutoffs for machine learning interatomic potentials. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=BCVB3TFpk5`.

Katherine A Henzler-Wildman, Ming Lei, Vu Thai, S Jordan Kerns, Martin Karplus, and Dorothee Kern. A hierarchy of timescales in protein dynamics is linked to enzyme catalysis. *Nature*, 450 (7171):913–916, 2007.

Ask Hjorth Larsen, Jens Jørgen Mortensen, Jakob Blomqvist, Ivano E Castelli, Rune Christensen, Marcin Dułak, Jesper Friis, Michael N Groves, Bjørk Hammer, Cory Hargus, et al. The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017.

Lin Huang, Chengxiang Huang, Ziang Wang, Yiyue Du, Chu Wang, Haocheng Lu, Yunyang Li, Xiaoli Liu, Arthur Jiang, and Jia Zhang. E2former-v2: On-the-fly equivariant attention with linear activation memory. *Forty-third International Conference on Machine Learning*, 2026.

Anders Johansson, Evan Weinberg, Christian Trott, Megan McCarthy, and Stan Moore. Lammps-kokkos: Performance portable molecular dynamics across exascale architectures. In *Proceedings of the SC'25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1217–1232, 2025.

Dávid Péter Kovács, J Harry Moore, Nicholas J Browning, Ilyes Batatia, Joshua T Horton, Yixuan Pu, Venkat Kapil, William C Witt, Ioan-Bogdan Magdau, Daniel J Cole, et al. Mace-off: Short-range transferable machine learning force fields for organic molecules. *Journal of the American Chemical Society*, 147(21):17598–17611, 2025.

Seung Yul Lee, Hojoon Kim, Yutack Park, Dawoon Jeong, Seungwu Han, Yeonhong Park, and Jae W Lee. Flashtp: Fused, sparsity-aware tensor product for machine learning interatomic potentials. In *ICML*, 2025.

Niklas Leimeroth, Linus C Erhard, Karsten Albe, and Jochen Rohrer. Machine-learning interatomic potentials from a users perspective: a comparison of accuracy, speed and data efficiency. *Modelling and Simulation in Materials Science and Engineering*, 33(6):065012, 2025.

Daniel S Levine, Muhammed Shuaibi, Evan Walter Clark Spotte-Smith, Michael G Taylor, Muhammad R Hasyim, Kyle Michel, Ilyes Batatia, Gábor Csányi, Misko Dzamba, Peter Eastman, et al. The open molecules 2025 (omol25) dataset, evaluations, and models. *arXiv preprint arXiv:2505.08762*, 2025.

Albert Musaelian, Simon Batzner, Anders Johansson, Lixin Sun, Cameron J Owen, Mordechai Kornbluth, and Boris Kozinsky. Learning local equivariant representations for large-scale atomistic dynamics. *Nature Communications*, 14(1):579, 2023.

Yutack Park, Jaesun Kim, Seungwoo Hwang, and Seungwu Han. Scalable parallel algorithm for graph neural network interatomic potentials in molecular dynamics simulations. *Journal of chemical theory and computation*, 20(11):4857–4868, 2024.

John P. Perdew, Kieron Burke, and Matthias Ernzerhof. Generalized gradient approximation made simple. 77:3865–3868, 1996. ISSN 0031-9007. doi: 10.1103/physrevlett.77.3865.

Krishnan Raghavachari, Gary W. Trucks, John A. Pople, and Martin Head-Gordon. A fifth-order perturbation comparison of electron correlation theories. *Chemical Physics Letters*, 157(6):479–483, 1989. ISSN 0009-2614. doi: 10.1016/s0009-2614(89)87395-6.

Benjamin Rhodes, Sander Vandenhaute, Vaidotas Šimkus, James Gin, Jonathan Godwin, Tim Duignan, and Mark Neumann. Orb-v3: atomistic simulation at scale. *arXiv preprint arXiv:2504.06231*, 2025.

Guillem Simeon and Gianni De Fabritiis. Tensornet: Cartesian tensor representations for efficient learning of molecular potentials. *Advances in Neural Information Processing Systems*, 36:37334–37353, 2023.

Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8(4):3192–3203, 2017.

Erica Tsai, Justin S. Smith, Wen Jie Ong, Dallas Foster, and Kelvin Lee. Building custom atomistic simulation workflows for chemistry and materials science with NVIDIA ALCHEMI toolkit, April 2026. URL `https://developer.nvidia.com/blog/building-custom-atomistic-simulation-workflows-for-chemistry-and-materials-science-with-nvidia-alchemi-toolkit/`.

K. Vanommeslaeghe, E. Hatcher, C. Acharya, S. Kundu, S. Zhong, J. Shim, E. Darian, O. Guvench, P. Lopes, I. Vorobyov, and A. D. Mackerell. Charmm general force field: A force field for drug-like molecules compatible with the charmm all-atom additive biological force fields. *Journal of Computational Chemistry*, 31(4):671–690, 2010. ISSN 0192-8651. doi: 10.1002/jcc.21367.

Arthur F Voter, Francesco Montalenti, and Timothy C Germann. Extending the time scale in atomistic simulation of materials. *Annual review of materials research*, 32(1):321–346, 2002.

Junmei Wang, Romain M. Wolf, James W. Caldwell, Peter A. Kollman, and David A. Case. Development and testing of a general amber force field. *Journal of Computational Chemistry*, 25(9): 1157–1174, 2004. ISSN 0192-8651. doi: 10.1002/jcc.20035.

Brandon M Wood, Misko Dzamba, Xiang Fu, Meng Gao, Muhammed Shuaibi, Luis Barroso-Luque, Kareem Abdelmaqsoud, Vahe Gharakhanyan, John R Kitchin, Daniel S Levine, et al. Uma: A family of universal models for atoms. *arXiv preprint arXiv:2506.23971*, 2025.

Stephen R Xie, Matthias Rupp, and Richard G Hennig. Ultra-fast interpretable machine-learning potentials. *npj Computational Materials*, 9(1):162, 2023.

Dewi Yokelson, Nikolay V Tkachenko, Robert Robey, Ying Wai Li, and Pavel A Dub. Performance analysis of cp2k code for ab initio molecular dynamics on cpus and gpus. *Journal of chemical information and modeling*, 62(10):2378–2386, 2022.

Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Physical review letters*, 120(14):143001, 2018.

Table 4: DistMLIP2 results on Orbv3 model inference. Throughput is reported as T in μs/(atom·timestep) (lower is better); C is demonstrated capacity in millions of atoms, rounded to four significant figures (higher is better). Capsid throughput uses the fixed 249,360-atom system.

		H_2O ice		Li_3PO_4		GaN		MOF		Capsid
Implementation	GPUs	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$	$C\uparrow$	$T\downarrow$
Orbv3										
DistMLIP2	4	35.221	0.03802	34.846	0.03786	27.406	0.04872	28.840	0.04860	OOM
	8	17.764	0.07392	17.611	0.07616	14.034	0.09374	14.724	0.08712	OOM
	16	9.124	0.1440	8.858	0.1478	7.005	0.1873	7.431	0.1764	3.670
	32	4.647	0.2873	4.449	0.2960	3.578	0.3727	3.742	0.3499	2.008

A Appendix

A.1 Using DistMLIP2

One primary goal for DistMLIP2 is to develop a state of the art distributed inference library that is simple enough for real-world practitioners to easily use. In Listing 1, we show how straightforward the API for using DistMLIP2 is using the ASE library (Hjorth Larsen et al., 2017). Users simply need to add one line of code beyond the boilerplate ASE logic in order to run multi-GPU distributed simulations.

Listing 1: Running molecular dynamics with DistMLIP2 through ASE.

```
from ase import units
from ase.io import read
from ase.md.verlet import VelocityVerlet
from ase.md.velocitydistribution import MaxwellBoltzmannDistribution
from DistMLIP2 import DistributedMACE, DistributedMACECalculator

atoms = ...
MaxwellBoltzmannDistribution(atoms, temperature_K=300)

with DistributedMACE(
    "model.model", atoms.numbers, atoms.cell.array,
    num_gpus=2, dtype="float32", skin=3.0,
) as runtime:
    atoms.calc = DistributedMACECalculator(runtime)
    md = VelocityVerlet(atoms, timestep=1.0 * units.fs,
                        trajectory="md.traj", logfile="md.log")
    md.run(1000)
```

A.2 Simulation stability

DistMLIP2 maintains the exact forward and backward computation graphs such that distributed inference outputs remain identical, up to numerical precision, to single inference predictions. However, as a sanity check, we run a 100 ps NVE simulation of crystalline SiO_2 supercell on 16 GPUs after 10 ps of NVT at 300K. We used the 3.8 million parameter pretrained MACE model on a 51k atom system. The total energy delta and overall temperature throughout the simulation can be found in Figure 4. The simulation maintains its stability through a large number of neighbor switches on a real-world system, validating the overall stability and correctness of the DistMLIP2 platform.

A.3 Orbv3 Real World Simulation Results

Table 4 shows real world simulation results for the 25 million parameter Orbv3 model.

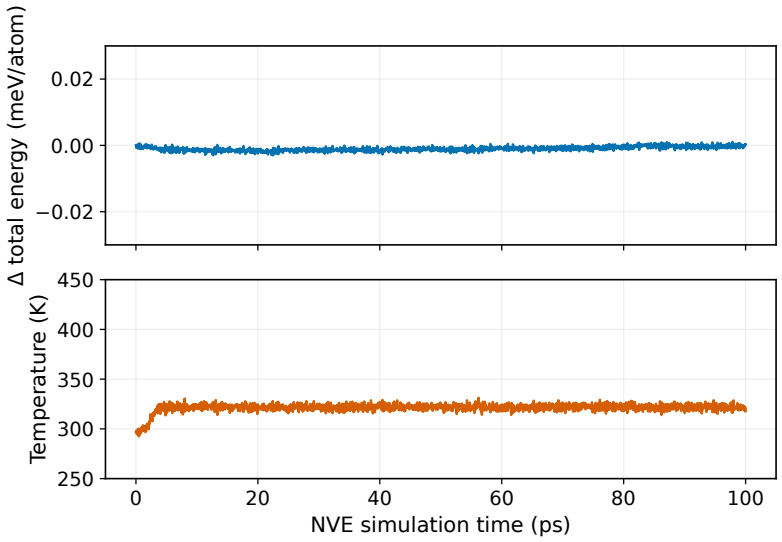

Figure 4: The total energy difference and temperature throughout a 100 ps NVE simulation of 51k atoms of SiO_2 using a pretrained 3.8 million parameter MACE model. The flat curves show no systemic drift whatsoever, highlighting and verifying the stability and correctness of the DistMLIP2 platform.

A.4 Main Algorithms

For simplicity, the implementations of many subroutines are not specified in this section. In addition, some function signatures have been simplified for clarity and brevity, so they do not contain all inputs and outputs. For more details, please refer to our code repository.

Notation:

- m: atom bin-sorted-to-global map
- m_p: local-to-global atom mapping
- $m_{b,p\to s}$: physical-to-dense-Morton bin map
- ρ_p: bin category flags
- $\Pi_w = (P_w, Q_w, c_w)$: node-level partition: pure node bins, expanded node bins, and cuts in the global bin list
- $\Pi_s = (P_s, c_s)$: subworker-level partition: subworker bins and cuts in each node's expanded bin list
- $Q_{s,p}$: one-bin search scope of subworker partition

Algorithm 1 Partial graph reconstruction. Detects dirty atoms, removes all edges with dirty endpoints, and replaces them with fresh edges. Updates coordinates of dirty atoms. Falls back to full graph construction for new bin grid and load rebalancing.

Require: Current positions x, state T_p, augmented radius R, skin distance r_{skin}, Boolean *checkLoad*

function PARTIALRECONSTRUCT($x, T_p, R, r_{\text{skin}}$, *checkLoad*)
  $E_p, x^{\text{ref}}, \Pi_w, \Pi_s, Q_{s,p}, m_p, m_{b,p\to s}, m^{-1}_{b,p\to s}, n_w, n_s, h, \ldots \leftarrow T_p$
  $D \leftarrow \{i \mid \|x_i - x_i^{\text{ref}}\|_2 \geq r_{\text{skin}}/2\}$ ▷ Global dirty atom IDs
  if $D = \varnothing$ **then**
    return T_p, NOCHANGE
  end if
  Update x^{ref}
  $x^f, \bar{o}, B, c, m_{b,p\to s}, m^{-1}_{b,p\to s}, \zeta \leftarrow$ BINANDSORT(x, D, T_p)
  rebuild $\leftarrow (\zeta =$ CALLFPIS$)$
  if $\neg$*rebuild* **and** *checkLoad* **then**

$rebuild \leftarrow$ CHECKREBALANCE$(B, c, \Pi_w, \Pi_s, n_w, n_s, h)$
end if
if *rebuild* **then**
$T'_p \leftarrow$ PARTITIONANDBUILDLIST$(x^f, \bar{o}, R, B, n_w, n_s, h, p)$
return T'_p, FULL
end if
$E^g_p \leftarrow [(m_p[u], m_p[v], o, d) \mid (u, v, o, d) \in E_p]$
$E^g_{p,\text{sub}} \leftarrow \{(j, i, o, d) \in E^g_p \mid j \notin D \ \wedge \ i \notin D\}$
$D^p \leftarrow \{i \in D \mid m^{-1}_{b,p\to s}[B[i]] \in Q_{s,p}\}$
$E'^g_p \leftarrow E^g_{p,\text{sub}} +$ CONSNEWLIST$(D^p, D, B, R, \Pi_s.P_s[p], m)$
Construct m'_p and m'^{-1}_p
Construct ownership flags ρ'_p for all atoms
$E'_p \leftarrow [(m'^{-1}_p[j], m'^{-1}_p[i], o, d) \mid (j, i, o, d) \in E'^g_p]$
Update contents of T_p
return T_p, PARTIAL
end function

Algorithm 2 Hierarchical partitioning. Performs a worker-level, then subworker-level partition. Subworker-level partitioning is performed over an expanded worker-level partition, with partitioning of the expanded region done layer by layer, radiating from the central pure region.

Require: Bin assignments B, populations c, bin maps $m_{b,p\to s}, m^{-1}_{b,p\to s}$, hierarchy configs (n_w, n_s, h)
Ensure: Node-level partition Π_w and subworker-level partition Π_s
function HIERARCHICALPARTITION$(B, c, m_{b,p\to s}, m^{-1}_{b,p\to s}, n_w, n_s, h)$
$t_w \leftarrow$ EXCLUSIVESCAN(c)
$M_w \leftarrow t_w[|\mathcal{B}|]$; $q_w \leftarrow \lceil M_w / n_w \rceil$
for $j = 0, \ldots, n_w - 1$ **do**
$c_w[j] \leftarrow \text{lower_bound}(t_w, \min(jq_w, M_w))$
end for
$c_w[n_w] \leftarrow |\mathcal{B}|$
for $w = 0, \ldots, n_w - 1$ **do**
$P_w[w] \leftarrow \mathcal{B}[c_w[w] : c_w[w+1]]$
$Q_w[w] \leftarrow$ EXPANDBINS$(P_w[w], h, B, m_{b,p\to s}, m^{-1}_{b,p\to s})$
$O_w, c_s[w] \leftarrow$ SOLVE$(B, c, P_w[w], Q_w[w], n_s)$
for $a = 0, \ldots, n_s - 1$ **do**
$P_s[w, a] \leftarrow O_w[c_s[w, a] : c_s[w, a+1]]$
end for
end for
return $\Pi_w = (P_w, Q_w, c_w)$, $\Pi_s = (P_s, c_s)$
end function

Algorithm 3 Partition cycling. Constructs n_p partitions to distribute to worker partitions. Performs a D-hop expansion on each partition, ensuring that energy and force calculations are independent of other partitions. After every iteration, aggregate results on rank 0 worker.

Require: Current positions x; actor partition count n_p, worker count n_w, model depth D
Ensure: Root energy E and global forces F
function CYCLEPARTITIONS$(x, e, \mathit{profile}, \mathit{forceRebuild})$
if *forceRebuild* **then**
Clear partition caches and saved cores
end if
if rank $= 0$ **then**
$C, u, H \leftarrow$ PREPAREPARTITIONS$(x, L, n_p, n_{\text{savedCores}})$
$E, F \leftarrow 0, 0$
$n_{\text{savedCores}} \leftarrow C$

end if
for $t = 0, n_w, 2n_w, \ldots < n_p$ **do**
  $S_w \leftarrow \textsc{GetScope}(x, L, C[t+w], u, H, DR)$ for valid tiles on rank 0
  $r_w \leftarrow [(I_w, c_w) \neq \text{cachedKey}_w]$; reuse cached position frames when $r_w = 0$
  $packet, n, r \leftarrow \textsc{Distribute}(S, r)$
  if $n > 0$ **then**
    $R, _ \leftarrow \textsc{VisitPartition}(packet, n, e, profile, t + node, r)$
    Node rank 0 for each node sends R to global rank 0
  end if
  if rank $= 0$ **then**
    Decode (E_w, F_w) from each nonempty partition's R_w
    $E \leftarrow E + \sum_w E_w$
    $F[I_w] \leftarrow F[I_w] + F_w$ for each nonempty partition
  end if
end for
return (E, F) after all visits completed
end function

A.5 Helper Algorithms

Below are selected major helper algorithms that are important in the implementation of the main components above.

Algorithm 4 Layer-aware subworker-level partition. Divides the partition into a series of subregions based on the depth, starting at 0 for the pure region and increasing with the number of hops in the expansion region. Each layer uses a prefix sum to produce bands of equal cost, and a Hungarian matching algorithm is used to match each band to each partition.

Require: Bin assignments B, populations c, pure bins P, expanded bin order O, subworker count n_s
Ensure: GPU-grouped bin order O' and cutoffs C

function $\textsc{Solve}(B, c, P, O, n_s)$
  if $O = \varnothing$ **then**
    return $(\varnothing, [0]^{n_s+1})$
  end if
  $d[b] \leftarrow \min_{p \in P} \text{dist}_O(p, b)$
  $q[b] \leftarrow c_b(1 + \sum_{b' \in \mathcal{N}_1(b)} c_{b'})$
  Require $d[b] < \infty$ for $b \in O$
  $A[a] \leftarrow [\,]$ for $a = 0, \ldots, n_s - 1$
  for $r = 0, \ldots, \max_{b \in O} d[b]$ **do**
    $G \leftarrow \text{sort}([b \in O : d[b] = r])$
    $Q \leftarrow \textsc{ExclusiveScan}([q[b] : b \in G])$
    $k_0 \leftarrow 0$, $k_{n_s} \leftarrow |G|$
    for all $s \in [n_s - 1]$ **do**
      $k_s \leftarrow \lfloor |G| s / n_s \rfloor$ if $Q_{|G|} = 0$ else choose k_s nearest to $sQ_{|G|}/n_s$ in Q
    end for
    $H[a] \leftarrow G[k_a : k_{a+1}]$; $\pi \leftarrow \text{id}$ if $r = 0$
    Otherwise $C_{as} \leftarrow -\sum_{b \in H[a]} \sum_{b' \in \mathcal{N}_1(b) : b' \in A[s]} c_b c_{b'}$
    $\pi \leftarrow \arg\min_{\text{permutations}} \sum_a C_{a,\pi[a]}$
    $A[\pi[a]] \leftarrow A[\pi[a]] + H[a]$ for each a
  end for
  $O' \leftarrow \text{sort}(A[0]) + \cdots + \text{sort}(A[n_s - 1])$
  $C \leftarrow \textsc{ExclusiveScan}([|A[a]| : a = 0, \ldots, n_s - 1])$
  return O', C
end function

Algorithm 5 Load-balance check against a fresh partition of the same bins. Compares the current partitions with an ideal partition given the current distribution of atoms. Layer-aware check uses the same layer-by-layer partitioning strategy described in Algorithm 4 for constructing the ideal partition. Sends signal for rebalancing and full graph construction if difference threshold reached.

function CHECKIMBALANCED(c, C, k)
  $t \leftarrow$ EXCLUSIVESCAN(c); $M \leftarrow t_{|c|}$
  $q \leftarrow \lceil M/k \rceil$
  for $j = 0, \ldots, k-1$ **do**
    $C'_j \leftarrow \text{lower_bound}(t, \min(jq, M))$
  end for
  $C'_k \leftarrow |c|$
  $a_j \leftarrow t_{C_{j+1}} - t_{C_j}$; $a'_j \leftarrow t_{C'_{j+1}} - t_{C'_j}$ for all j
  Sort a and a' independently
  return $\sum_j |a_j - a'_j| > 0.2\sqrt{k}\, M$
end function
function LAYERLOADCHANGED($B, c, \Pi_w, \Pi_s, w, n_w, n_s, h$)
  $O \leftarrow \Pi_s.P_s[w, 0] + \cdots + \Pi_s.P_s[w, n_s - 1]$ ▷ Frozen bin order
  $C \leftarrow \Pi_s.c_s[w]$
  if $C[n_s] = 0$ **then return** false
  $O', C' \leftarrow$ SOLVE($B, c, \Pi_w.P_w[w], O, n_s$)
  $d[b] \leftarrow \min_{p \in \Pi_w.P_w[w]} \text{dist}_O(p, b)$; $q[b] \leftarrow c_b(1 + \sum_{b' \in \mathcal{N}_1(b)} c_{b'})$
  for all $r \in \{h-1, \infty\}$ if $n_w > 1$, otherwise $r \in \{0, \infty\}$ **do**
    $L_{\text{old}} \leftarrow \max_a \sum_{j=C[a]}^{C[a+1]-1} q[O[j]]\, \mathbf{1}_{d[O[j]] \le r}$
    $L_{\text{new}} \leftarrow \max_a \sum_{j=C'[a]}^{C'[a+1]-1} q[O'[j]]\, \mathbf{1}_{d[O'[j]] \le r}$
    if $L_{\text{old}} > 1.20 L_{\text{new}}$ **then**
      return true
    end if
  end for
  return false
end function
function CHECKREBALANCE($B, c, \Pi_w, \Pi_s, n_w, n_s, h$)
  if CHECKIMBALANCED($c, \Pi_w.c_w, n_w$) **then**
    return true
  end if
  for $w = 0, \ldots, n_w - 1$ **do**
    if LAYERLOADCHANGED($B, c, \Pi_w, \Pi_s, w, n_w, n_s, h$) **then**
      return true
    end if
  end for
  return false
end function

B MLIP VERSIONS AND INFERENCE SETTINGS

Table 5 outlines the specific versions of each model we used during benchmarking. Table 6 outlines the specific inference settings used for each of the models. In order to test the frontier capabilities of MLIP inference, we used the fastest inference settings and CUDA kernels available. For fair comparison, each of the inference settings were held fixed while we benchmarked different inference platforms. For DistMLIP2 implementations, we used a skin of 3 angstroms. For other distributed inference platforms, we disabled skins entirely due to enabling skins resulting in slower inference and higher memory consumption.

Table 5: Pretrained MLIPs Model Specifications

Model	Version	Size	Interaction Range	Reference
MACE	MACE-MP-0 small	3.8M	12Å	(Batatia et al., 2023)
NequIP	untrained 2-layer NequIP	1.6M	12Å	(Batzner et al., 2022)
Orb	orb-v3-conservative-inf-omat	25.5M	30Å	(Rhodes et al., 2025)
UMA	UMA-S 1.2.1	291.5M (6.3M active)	30Å	(Wood et al., 2025)

Table 6: Frozen DistMLIP2 inference settings used in the ablation and scaling studies. Model-specific settings are fixed within each study.

Setting	MACE	NequIP	UMA	Orbv3
Convolutions	Fused cuEquivariance 0.9.1	OpenEquivariance 0.7.0	PyTorch / Triton	PyTorch
Compute execution	Eager	Eager	Compiled turbo	Eager
Model precision	FP32	FP32	FP32	FP32
TF32	Off	Off	On	Off