MICA: Activation Checkpointing for Double-Backward Training of Machine Learning Interatomic Potentials

Hongyu Wang[1,2] **Mingzhen Li**[2*] **Hongtao Xu**[1,2] **Yuanchang Zhou**[2,3]
Weijian Liu[2,3] **Weile Jia**[2*] **Guangming Tan**[2,3*]
[1]School of Advanced Interdisciplinary Sciences,
University of Chinese Academy of Sciences, Beijing, China
[2]State Key Lab of Processors, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
[3]University of Chinese Academy of Sciences, Beijing, China
wanghongyu231@mails.ucas.ac.cn
{limingzhen,jiaweile,tgm}@ict.ac.cn

Abstract

Conservative machine learning interatomic potentials (MLIPs) predict potential energy and obtain atomic forces as the negative gradient of energy with respect to atomic positions. Computing forces requires a backward pass from energy to atomic positions, and backpropagating the force loss differentiates through this pass, creating double-backward execution. This execution creates two activation classes with different lifetimes: forward activations and first-backward activations. Heterogeneous atomic graph batches make their memory footprint input dependent, while standard checkpointing mainly targets forward activations and does not directly manage first-backward activations. Therefore, we present MICA, an activation checkpointing planner for double-backward execution in conservative MLIP training. MICA combines a checkpoint primitive for double-backward phases with an input-aware dynamic programming planner that selects checkpoint modes for dynamic atomic graph batches under a specified memory budget. Across EqV3, MatRIS, PET, and UMA, MICA's full-checkpoint mode reduces average peak memory by 39% to 71%, compared with only 11% to 18% from standard PyTorch checkpointing. On an 80GB GPU, training EquiformerV3 without checkpointing peaks at 73.29GB and exceeds 50GB in 74.8% of 1000 iterations, while MICA keeps all 1000 iterations within a 50GB memory budget with 25.38% step time overhead. Under the same GPU memory capacity, MICA increases the maximum feasible atoms per batch by $3.6\times$ to $8.5\times$. In distributed training, the reduced peak memory allows MICA to reduce reliance on heavier memory saving techniques, such as Fully Sharded Data Parallel (FSDP) and graph parallelism (GP). On a single H100 node with NVLink, MICA trains a MatRIS-MoE model with 2B parameters, achieving a $1.88\times$ to $2.03\times$ speedup over FSDP and GP baselines.

1 Introduction

Machine learning interatomic potentials (MLIPs) enable molecular dynamics (MD) simulations with near *ab initio* accuracy at much lower cost than Kohn-Sham density functional theory (KS-DFT) [2, 11, 13, 23]. Recent universal MLIPs scale model size and training datasets across diverse

*Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

chemical domains [4, 19, 21, 27, 28, 31, 32], making training under fixed GPU memory budgets harder. This makes MLIP training an important systems problem. As shown in Figure 1, conservative MLIP training does not follow the standard forward and single-backward pattern. A conservative MLIP first predicts a scalar energy E (Forward, FW). It then derives forces with a backward pass with respect to atomic positions, i.e., $F = -\nabla_x E$. We refer to this force computation as First-Backward (FBW). The standard objective includes both energy and force losses. The force loss backpropagates through First-Backward, forming Double-Backward (DBW). The energy loss backpropagates through the original Forward graph as Backward (BW). Thus, each iteration can be viewed as four conceptual phases: FW, FBW, DBW, and BW [14]. This four-phase execution changes the activation lifetimes that checkpointing should manage, and the peak memory now includes both FW activations and FBW activations later consumed by DBW.

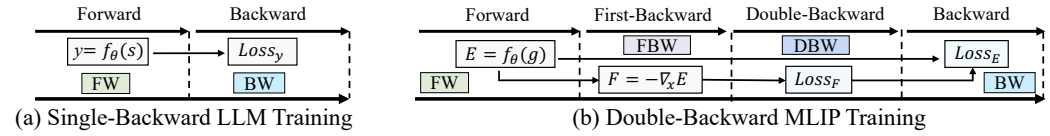
In large language model (LLM) training, activation checkpointing is commonly used to reduce the peak memory footprint [20], but applying it to double backward MLIP training faces two challenges. *1)* Changed activation lifetime. In ordinary single-backward training, FW activations are produced once and then consumed once by BW. Activation checkpointing exploits this pattern by discarding selected FW activations and recomputing them when BW needs them [5, 9, 10, 12]. Conservative MLIP training breaks this pattern. FW activations can be consumed by FBW for force computation and by BW for energy-loss backpropagation. Moreover, FBW produces additional activations along the force-computation path, later consumed during DBW. Thus, standard checkpointing reduces the FW activation footprint, but does not directly manage activations produced during FBW. *2)* Input heterogeneity. MLIP inputs are dynamic atomic graphs drawn from heterogeneous structures, including molecules, crystals, catalytic systems, and metal-organic frameworks (MOFs) [1, 8, 15, 25]. Even with a fixed atom budget or fixed sample count, graph batches can have substantially different edge and triplet counts. These graph features strongly affect activations produced during FW and FBW, as well as recomputation time. A single static checkpointing plan may over-checkpoint smaller-workload batches and waste recomputation, or under-checkpoint larger-workload batches and still cause memory budget violations.

We present **MICA**, a double-backward-aware activation checkpointing system for conservative MLIP training. MICA introduces a checkpoint primitive with three region modes: no checkpointing, checkpointing FBW activations, and checkpointing both FW and FBW activations. For each incoming graph batch, MICA predicts each mode's memory saving and recomputation cost, and then uses a region-level dynamic programming planner to minimize recomputation overhead under an effective memory budget. Together, the primitive and planner reduce the per-GPU memory footprint of graph batches while preserving training correctness. This work makes the following contributions:

- We identify the four-phase execution structure of double-backward MLIP training and show that FBW activations are a key memory bottleneck not directly managed by standard activation checkpointing.
- We design a phase-aware checkpoint primitive that controls activations produced during both FW and FBW while preserving the original training semantics.
- We develop an input-aware dynamic programming planner that selects checkpoint modes using graph features such as atom count, edge count, and triplet count, allowing the checkpointing plan to adapt to dynamic atomic graph batches under an effective memory budget.
- We evaluate MICA on EqV3, MatRIS, PET, and UMA. MICA-M2 reduces average peak memory by 39% to 71%. On EqV3, MICA keeps all 1000 iterations below a 50GB budget with 25.38% time overhead, while the baseline without checkpointing peaks at 73.29GB. MICA also increases the maximum feasible graph batch size by up to $8.5\times$.
- We show that MICA also improves distributed MLIP training throughput. On a single H100 node with NVLink, MICA trains a MatRIS model with 2B parameters at $1.88\times$ to $2.03\times$ the throughput of FSDP and graph parallel baselines.

2 Preliminaries

Conservative MLIP training. Modern MLIPs are commonly implemented as graph neural networks (GNNs) composed of stacked interaction blocks, where atoms are represented as nodes and local interatomic neighbor pairs are represented as edges [2, 6, 19, 27]. Some architectures also construct



(a) Single-Backward LLM Training (b) Double-Backward MLIP Training

Figure 1: (a) LLM workloads perform one forward pass and one backward pass. (b) Conservative MLIP training involves a four-phase double-backward execution pattern.

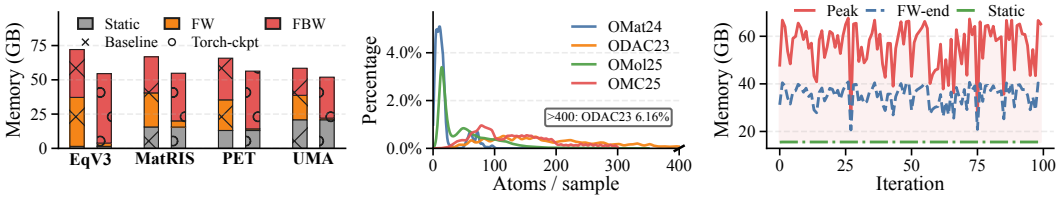


Figure 2: **Observations.** (a) Standard PyTorch checkpointing reduces FW activations but leaves FBW activations. (b) Source datasets have long-tailed atom-count distributions. (c) Peak memory varies across iterations and can exceed FW-end memory, motivating input-aware planning.

triplets from two edges sharing the same center atom to encode angular geometric information [2, 31, 32]. Given an atomic graph, an embedding layer first initializes node, edge and triplet features. A stack of interaction blocks then updates these features, and a readout layer predicts the total energy. A foundational physical principle of modern conservative MLIPs is energy conservation, which requires deriving forces as the negative gradient of potential energy with respect to atomic positions, i.e., $F = -\nabla_x E$ [3]. Although non-conservative MLIPs can directly predict forces via a readout layer [18], recent studies show that they may violate physical invariants in downstream simulations, such as energy or temperature drift [3].

Four-phase execution in conservative MLIP training. For memory analysis, we view each conservative MLIP training iteration as four conceptual phases. **(1) Forward (FW)** computes the predicted energy E and builds the original FW graph. **(2) First-Backward (FBW)** differentiates E with respect to atomic positions x to obtain forces $F = -\nabla_x E$. This phase is a backward pass from energy to positions, not a parameter update pass. The forward path of the force loss therefore consists of both FW and FBW. When Loss_F is backpropagated, gradients first traverse the derivative states created by FBW and then reach the original FW graph. The energy loss Loss_E also backpropagates through the original FW graph. Thus, in an actual implementation, the final traversal of the shared FW graph can accumulate gradient contributions from both Loss_F and Loss_E. We separate this computation into two conceptual phases for memory analysis. **(3) Double-Backward (DBW)** denotes the backward through the derivative states created by FBW. **(4) Backward (BW)** denotes the final backward over the shared FW graph.

Activation checkpointing. During training, GPU memory consists of static model states and activation memory. Static model states include parameters, gradients, and optimizer states, which activation checkpointing does not reduce. Activation memory refers to intermediate tensors that remain live until later gradient computation uses them. Activation checkpointing reduces activation memory by saving only the boundary state of a selected computation segment and discarding intermediate tensors inside the segment [5]. When backward later needs these intermediate tensors, the segment is recomputed from the saved boundary state. Thus, the peak memory of a training step is roughly static model memory plus the maximum live activation memory, together with runtime overheads such as allocator fragmentation and temporary workspaces.

3 Observations

Conservative MLIP training exposes two memory behaviors not addressed by standard activation checkpointing. Double-backward execution changes activation lifetimes, while dynamic atomic graph inputs make activation memory and recomputation cost batch dependent.

Observation 1: Double-backward training changes the activation production and consumption structure. Standard activation checkpointing targets the one producer, one consumer pattern of ordinary single-backward training: the FW pass produces activations, and the BW pass consumes them once. Conservative MLIP training breaks this simple pattern: activations produced during FW can be consumed by both FBW and BW, while activations produced during FBW are consumed by DBW. When a checkpointed region is recomputed during FBW, that recomputation is not a final consumption of FW activations. It also constructs derivative states along the FBW path, and these states may remain live until DBW differentiates through them. As a result, standard checkpointing can reduce the footprint of activations produced during FW, but it does not directly manage activations produced during FBW. Figure 2(a) shows this effect across representative MLIP models: PyTorch checkpointing reduces the FW activation, but a large FBW activation remains. This motivates a checkpoint primitive that distinguishes activation producer and consumer phases.

Observation 2: Input heterogeneity affects memory footprint. MLIP inputs are dynamic atomic graphs rather than regular dense tensors. Figure 2(b) shows that representative source datasets have atom-count distributions with long tails. However, atom count is only a coarse proxy for the actual workload of a graph batch. Even under the same atom budget, different batches can have very different edge counts and, when used, triplet counts. These graph features affect FW activations, FBW activations, and recomputation time. Figure 2(c) shows this effect during MatRIS-MoE 1B [32] training (model configuration and experiment details in Appendix B). Static model memory is nearly constant, but peak memory changes substantially across iterations. The memory at the end of FW alone does not capture these peaks. Rare batches with similar atom counts but larger edge or triplet counts can still exceed the memory budget. To avoid such OOM cases, users often lower the graph batch size. This conservative setting avoids OOM, but it leaves GPU memory underutilized on many ordinary batches. Static checkpointing faces the same mismatch under dynamic inputs. Worst-case plans over-checkpoint smaller batches, while typical-case plans may under-checkpoint larger batches and still exceed the memory budget. This motivates an input-aware planner that selects checkpoint modes from the current batch features and the memory budget.

4 Design

4.1 Overview

MICA has two components: a phase-aware checkpoint primitive and an input-aware planner. The primitive exposes three execution modes for each candidate region, ranging from normal execution to checkpointing both FW and FBW activations. For each incoming batch, the planner predicts the memory saving and recomputation cost of every region-mode choice from graph features such as atom count, edge count, and triplet count. It abstracts mode selection as a multiple-choice knapsack problem and solves it with a region-level dynamic program under the memory budget. The resulting assignment is used for the current training step, preserving double-backward correctness while minimizing predicted recomputation under the budget.

4.2 Phase-aware checkpointing

Candidate regions. MICA operates on a set of candidate regions $\mathcal{R} = \{R_i\}_{i=1}^{R}$. Each region is a deterministic and recomputable submodule, layer, or computation segment. For a new MLIP architecture, users can manually register these regions with a simple API, as shown in Appendix C. For the MLIPs evaluated in this paper, MICA provides adapters that instantiate these regions automatically. MICA defines regions around expensive components such as message passing modules, edge updates, node updates, attention blocks, and feed-forward sublayers. Appendix B.1.2 lists the regions used for all evaluated models. The three modes share the same runtime wrapper described next and differ in which FW/FBW activations are kept or recomputed.

Runtime wrapper and phase hooks. MICA wraps each candidate region with a runtime wrapper and inserts phase-aware autograd Functions only when checkpointing is selected. During FW, the wrapper records boundary inputs, output structure, gradient requirements, RNG state, and metadata for deterministic recomputation; this metadata is the boundary state. For differentiable region outputs in checkpointed modes, MICA inserts an FW boundary `Function` tagged with the region ID. This Function is transparent during FW, but PyTorch invokes its backward when a later FBW or BW

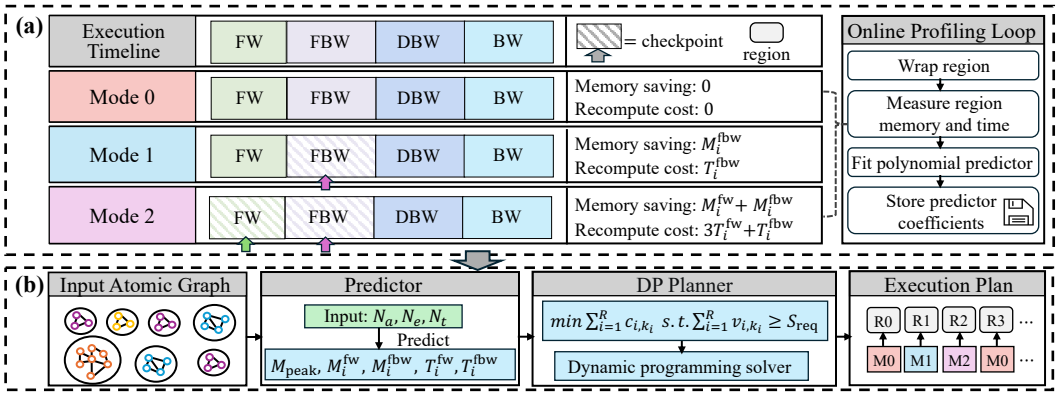


Figure 3: **Design overview of MICA.** (a) MICA supports three checkpoint modes for each candidate region, with different activation lifetimes and recomputation paths. (b) MICA first profiles memory and time online, then predicts mode costs from graph features (N_a, N_e, N_t), and finally solves a dynamic program under the memory budget.

traversal crosses the region output boundary. When FBW reaches this boundary, the FW boundary Function receives output gradients and dispatches the local gradient computation according to the current mode. If the first-order gradient tensors produced by the boundary require gradients, MICA attaches tensor hooks to them. These tensor hooks act as FBW hooks and are invoked when DBW differentiates the corresponding first-order gradients. MICA maintains a runtime phase tag over FW, FBW, DBW, and BW. The runtime uses this tag and the current mode to decide whether to keep internal activations, save only boundary state, or recompute missing local FW or FBW computations. Thus, MICA implements phase-aware checkpointing without modifying the PyTorch autograd engine. Figure 3(a) visualizes the three modes, which we define next. **Mode 0: normal execution.** Mode 0 executes the region normally. The runtime retains activations produced during FW and activations produced during FBW. These activations are kept until their later consumer phases. Mode 0 introduces no recomputation overhead and provides no memory saving. **Mode 1: checkpoint FBW activations.** Mode 1 keeps activations produced during FW live, but checkpoints activations produced during FBW. During FBW, the region computes the same local values as in normal execution. However, MICA does not retain the full internal activation state of this local FBW computation. Instead, MICA stores the boundary state needed to recompute the local FBW computation when DBW later needs these activations. Since the activations produced during FW remain live, Mode 1 avoids recomputing the region's FW computation. Therefore, for region i, the saved memory is the memory of activations produced during FBW, denoted by M_i^{fbw}. Its recomputation overhead is the FBW computation time, denoted as T_i^{fbw}. **Mode 2: checkpoint FW and FBW activations.** Mode 2 checkpoints both activations produced during FW and activations produced during FBW. During FW, MICA stores only the region boundary state instead of retaining internal FW activations. When FBW needs the region's FW activations, MICA recomputes the local FW computation. When DBW later needs the local FBW activations, MICA recomputes the local FW computation first and then recomputes the corresponding local FBW computation. BW also needs the local FW activations for the shared FW path, so Mode 2 recomputes the local FW computation once more. Thus, Mode 2 recomputes FW in three later phases and recomputes FBW only for DBW. Mode 2 saves activation memory from both FW and FBW, denoted by $M_i^{\text{fw}} + M_i^{\text{fbw}}$. Its recomputation cost is $3T_i^{\text{fw}} + T_i^{\text{fbw}}$. MICA enables joint backward by default, first summing the activation-gradient contributions from DBW and BW on the shared forward path. It then computes the corresponding weight gradients once, avoiding separate weight-gradient computations in DBW and BW, at the cost of additional buffering for the pending activation gradients.

4.3 Input-aware dynamic programming planner

MICA's planner chooses one execution mode for each candidate region under a memory budget. Because MLIP inputs are dynamic atomic graphs, both the activation memory and recomputation cost of each region vary across batches. MICA therefore predicts mode-level memory saving and

Table 1: Notation.

Symbol	Meaning	Symbol	Meaning
g, x, s	Atomic graph, atomic positions, sequence	E, F, θ	Energy, force, and model parameters
N_a, N_e, N_t	Atom count, edge count, triplet count	$\mathcal{R}$, i	Candidate regions and region index
M_i^{fw}	FW activation memory of region i	M_i^{fbw}	FBW activation memory of region i
T_i^{fw}	Local FW recomputation time	T_i^{fbw}	Local FBW recomputation time
$v_{i,k}$	Saved memory of region i in mode k	$c_{i,k}$	Recompute time of region i in mode k
M_{peak}	Predicted no-checkpoint peak memory	$M_{\mathrm{bud}}, M_{\mathrm{res}}$	Memory budget and adaptive reserve
$M_{\mathrm{prev_peak}}$	Observed peak memory from the previous step	S_{req}	Required memory saving

recomputation cost from the current batch graph features, and then solves a budgeted mode selection problem using dynamic programming.

Region-level predictor. MICA profiles whole training steps on observed graph batches to estimate region-level memory savings and recomputation costs. For each profiled batch, MICA also records graph features, including the number of atoms N_a, edges N_e, and triplets N_t. MICA then fits lightweight quadratic polynomial models that map these graph features to memory and time measurements. For a new batch, the fitted models predict the peak memory without checkpointing, denoted by M_{peak}, and the per region quantities M_i^{fw}, M_i^{fbw}, T_i^{fw}, and T_i^{fbw}. Here, M_i^{fw} and M_i^{fbw} denote the FW and FBW activation memory that can be saved, while T_i^{fw} and T_i^{fbw} denote the corresponding recomputation times. These predictions provide the memory saving and recomputation cost options used by the planner.

Mode-level predictor. For each candidate region i, MICA considers three mutually exclusive execution modes. Mode 0 uses the default execution and introduces no memory saving or recomputation cost. Mode 1 saves M_i^{fbw} and incurs recomputation cost T_i^{fbw}. Mode 2 saves both M_i^{fw} and M_i^{fbw} and incurs recomputation cost $3T_i^{\mathrm{fw}} + T_i^{\mathrm{fbw}}$. Equivalently, each region-mode choice is represented by a saved-memory and recomputation-cost pair $(v_{i,k}, c_{i,k})$:

$$(v_{i,0}, c_{i,0}) = (0, 0), \;\; (v_{i,1}, c_{i,1}) = \left(M_i^{\mathrm{fbw}}, T_i^{\mathrm{fbw}}\right), \;\; (v_{i,2}, c_{i,2}) = \left(M_i^{\mathrm{fw}} + M_i^{\mathrm{fbw}}, 3T_i^{\mathrm{fw}} + T_i^{\mathrm{fbw}}\right). \tag{1}$$

Dynamic programming planner for mode selection. Given the mode cost pairs above, the planner chooses one mode $k_i \in \{0, 1, 2\}$ for each candidate region i. The goal is to minimize recomputation overhead while satisfying the required memory saving:

$$\min_{\{k_i\}} \sum_{i=1}^{R} c_{i,k_i} \quad \text{s.t.} \quad \sum_{i=1}^{R} v_{i,k_i} \geq S_{\mathrm{req}}, \quad S_{\mathrm{req}} = \max\left(0, M_{\mathrm{peak}} + M_{\mathrm{res}} - M_{\mathrm{bud}}\right). \tag{2}$$

Here, R is the number of candidate regions, M_{bud} is the memory budget, and S_{req} is the amount of memory that must be saved. M_{peak} is the predicted peak memory of the current batch under default execution without checkpointing. This peak includes static model states and live activations, but checkpointing reduces only the activation component. M_{res} is an adaptive reserve for short-lived activation peaks during recomputation and allocator fragmentation. The reserve is added because M_{peak} measures the default live state, whereas checkpointed execution may create transient activation peaks when a region is recomputed. After each step, MICA increases the reserve for the next step when the observed memory headroom falls below the initial reserve:

$$M_{\mathrm{res}} \leftarrow M_{\mathrm{res}} + \max\left(0, \text{ initial reserve} + M_{\mathrm{prev_peak}} - M_{\mathrm{bud}}\right). \tag{3}$$

Here, $M_{\mathrm{prev_peak}}$ is the observed peak memory of the completed step. The first planner invocation uses the initial reserve. The reserve remains fixed during each step. After each step, MICA records the observed peak as $M_{\mathrm{prev_peak}}$ and updates M_{res} for the next planner invocation.

This is a multiple-choice knapsack problem. Each region forms one group, and each group contains three mutually exclusive modes; the planner selects the least costly assignment whose total saving covers S_{req}. MICA solves Eq. (2) with a discretized dynamic program (DP). Let $\Delta = 16$ MB be the memory bin size. To avoid optimistic plans, MICA rounds the required saving up and rounds each mode saving down: $\bar{S} = \left\lceil \frac{S_{\mathrm{req}}}{\Delta} \right\rceil$, $\bar{v}_{i,k} = \left\lfloor \frac{v_{i,k}}{\Delta} \right\rfloor$. We define $D[i][s]$ as the minimum recomputation overhead after assigning modes to the first i regions and achieving discretized saving s, with s clipped

to $\bar{S}$. The initialization is $D[0][0] = 0$ and $D[0][s > 0] = +\infty$. For each region i, each previous saving state s, and each mode k, the transition is

$$s' = \min(\bar{S},\, s + \bar{v}_{i,k}), \qquad D[i][s'] = \min\left(D[i][s'], D[i-1][s] + c_{i,k}\right). \tag{4}$$

After processing all regions, $D[R][\bar{S}]$ gives the minimum recomputation overhead for the conservative discretized memory target. MICA stores backpointers during the transition and recovers the mode assignment $\kappa = (k_1, \ldots, k_R)$ by backtracking. The recovered plan is conservative with respect to the rounded predicted savings, although the undiscretized optimum may be slightly cheaper. If $D[R][\bar{S}] = +\infty$, the requested saving is infeasible under the current candidate regions. In this case, MICA returns the reachable assignment with the largest saving and marks the plan as infeasible. Appendix A gives the full planner pseudocode.

The dynamic programming (DP) complexity is $O(R\bar{S}K)$, where R is the number of candidate regions and $K = 3$ is the number of modes. Because regions are coarse-grained checkpointing units and K is constant, MICA's measured online prediction and planning overhead is below 11 ms and below 0.8% of the baseline step time for all evaluated models, as reported in Appendix B.6.

5 Evaluation

5.1 Experimental setup

We evaluate MICA on four representative conservative **MLIP training workloads**: EqV3 [19], MatRIS [31, 32], PET [4], and UMA [27]. In our evaluation, MatRIS refers to the MoE variant of the MatRIS model [32]. Unless otherwise stated, MatRIS denotes the 1B MoE configuration; the distributed throughput experiment uses MatRIS-2B. **Datasets:** Unless otherwise stated, single-GPU experiments use a fixed atom budget of 400 atoms per batch, with samples drawn equally from ODAC23 [25], OMat24 [1], OMol25 [15], and OMC25 [8]. Detailed model configurations and dataset settings are in Appendix B.1.1. **Baselines:** For single GPU experiments, we compare MICA against *Base*, the original double backward training without activation checkpointing, and *Torch*, standard PyTorch checkpointing using `torch.utils.checkpoint`. For planner-level comparison, we also evaluate *Mi*, a Mimose-style greedy planner [17] adapted to MICA's three-mode primitive. Both Mi and MICA use the same double-backward checkpoint primitive. For distributed throughput, we compare MICA-DDP8 with GP8, FSDP8, and FSDP4-GP2. All experiments use the same model initialization and random seed. **Measurement:** MICA uses 200 profiling steps per calibration policy to fit the no-checkpoint peak memory predictor and the region-level memory and time predictors. MICA applies the checkpoint regions described in Appendix B.1.2. Each configuration is warmed up for 10 steps and then measured over 100 training steps to report mean peak memory and mean step time. Long-running training experiments are measured over 1000 iterations. **Hardware and software:** Single-GPU experiments run on NVIDIA A100 80GB GPUs. The distributed throughput experiment in Figure 6(a) is run on an 8-GPU H100 node with NVLink, with additional details in Appendix B.

5.2 Memory and time tradeoff across MLIPs

Figure 4 compares memory and time across four representative MLIPs. Hatched bars mark static memory unreduced by activation checkpointing. Torch reduces average peak memory by only 10.5% to 18.1%, because it mainly targets FW activations and leaves FBW activations unmanaged. Mode 1 (M1) removes FBW activations, while Mode 2 (M2) removes both FW and FBW activations. M2 gives the largest memory reduction but incurs the highest recomputation cost. For example, on EqV3, M2 reduces average peak memory from 56.9GB to 16.8GB but increases step time by 63%. Mi has higher average peak memory than MICA-DP at the same target budget, with shorter step times on EqV3, MatRIS, and PET but a slightly longer step time on UMA. On EqV3, replacing DP-40 with Mi-40 decreases memory reduction from 43% to 30% relative to Base and lowers step-time overhead from 37% to 29%. However, Mi-40 exceeds the budget in 65 of the 100 measured steps, whereas DP-40 has no violations. MICA-DP solves the three-mode multiple-choice knapsack problem to minimize predicted recomputation cost while satisfying the predicted memory-saving requirement. At tighter budgets, such as UMA DP-35, the selected plan approaches fixed M2 because most reducible activations must be removed. Appendix B.2 validates prediction accuracy, and Appendix B.4 reports DP mode distributions under different budgets.

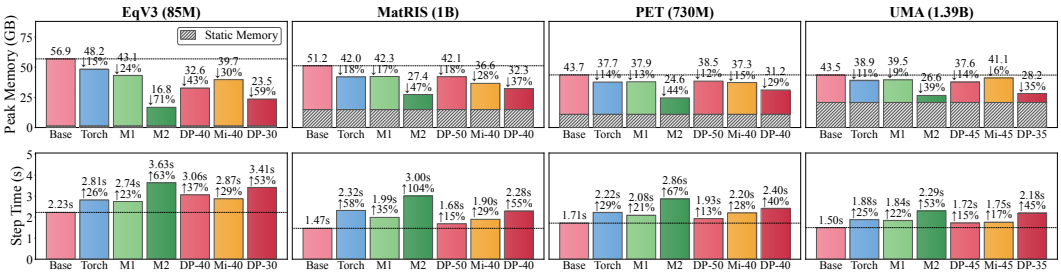


Figure 4: Memory and time tradeoff across MLIPs. Top and bottom rows report average peak memory and step time. Base is no checkpointing, Torch is PyTorch checkpointing, M1/M2 apply fixed MICA modes to all regions, DP-X is MICA-DP under an X GB budget, and Mi-X is a Mimose-style greedy planner adapted to MICA's three-mode primitive under the same budget. Hatched bars show static memory. Dashed lines mark Base; percentages are relative to Base.

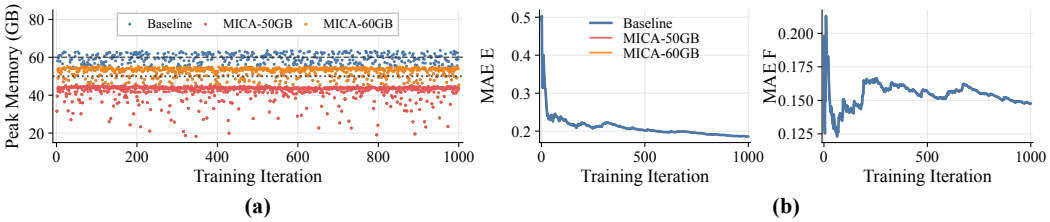


Figure 5: Budget compliance and training consistency. (a) Per step peak memory for 1000 MatRIS-1B iterations under 50GB and 60GB budgets. (b) Cumulative average Energy MAE and Force MAE.

5.3 Memory budget compliance and training consistency

Memory budget compliance. Figure 5(a) shows the memory footprint during long-running MatRIS-1B training. The baseline exhibits large memory spikes on dynamic graph batches. Across 1000 iterations, 64.9% of baseline steps exceed 50GB, and 21.8% exceed 60GB. MICA-50GB keeps the average peak memory at 42.1GB, with no overshoots above 50GB and a maximum peak of 45.24GB. MICA-60GB keeps all 1000 iterations below the 60GB budget. The step time overheads are 12.45% for MICA-50GB and 2.43% for MICA-60GB. Thus, tighter budgets require more recomputation, while a moderate budget can remove memory spikes with little overhead. For EqV3, baseline training exceeds 50GB in 74.8% of iterations and peaks at 73.29GB, while MICA keeps all 1000 iterations below 50GB with 25.38% step time overhead. Appendix B.6 reports MICA's online prediction and planning overhead, which is below 0.8% of the baseline step time.

Training consistency. Figure 5(b) evaluates training consistency. For MICA-50GB, the relative differences from Base in the 1000-step average energy MAE and force MAE are 1.25×10^{-6} and 2.26×10^{-7}. For MICA-60GB, they are 1.34×10^{-6} and 2.46×10^{-7}. This shows that MICA changes only activation lifetime management while preserving the DBW training trajectory within numerical tolerance. Appendix B.5 reports analogous 50GB long-run traces for EqV3, PET, and UMA. We further validate the custom autograd primitive with PyTorch `gradcheck` and `gradgradcheck` on representative wrapped regions under all MICA modes.

5.4 Leveraging the saved memory

Distributed training throughput. Figure 6(a) compares MICA-DDP8 with distributed memory-saving baselines on MatRIS-2B. MatRIS-2B uses about 32GB of static memory and has high activation memory, so running large batches on a single 8-GPU H100 node with NVLink requires memory-saving techniques. For each method, we report its highest atoms-per-second throughput under its largest feasible global atom budget. This is a max feasible throughput comparison rather than an equal-batch efficiency comparison, since GP and FSDP obtain memory capacity through graph partitioning or state sharding, which introduces overheads that MICA-DDP avoids. MICA-DDP8 combines MICA with PyTorch Distributed Data Parallel (DDP), reducing local activation memory while avoiding the graph partitioning used by GP and the parameter sharding used by FSDP.

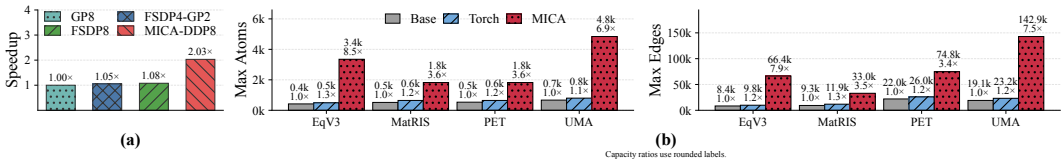


Figure 6: (a) Normalized MatRIS-2B throughput for GP8, FSDP4-GP2, FSDP8, and MICA-DDP8. (b) Maximum feasible workload under the same GPU memory capacity. We increase batch size until OOM and report the largest successful atom and edge counts. Labels show absolute workload and improvement over Base.

MICA-DDP8 delivers $2.03\times$, $1.93\times$, and $1.88\times$ the throughput of GP8, FSDP4-GP2, and FSDP8, respectively. The DP planner limits recomputation by checkpointing only the regions needed to meet the memory budget. GP reduces activation memory by partitioning graphs across GPUs but incurs partition imbalance and feature-exchange communication overheads. FSDP reduces static memory, but incurs parameter gathering, resharding, synchronization, and parameter-shard management overheads that are hard to hide for relatively small MLIP computation blocks. Appendix B.1.4 provides more details about the distributed baselines. In distributed training, by reducing peak activation memory, MICA can further reduce reliance on expensive memory-saving mechanisms such as FSDP and GP. MICA, GP, and FSDP reduce different memory components and should be viewed as complementary mechanisms.

Larger feasible batches. Figure 6(b) shows that MICA's memory savings enable larger trainable batches. For these capacity tests, we disable MICA's joint backward optimization to reduce gradient buffering. Under the same GPU memory capacity, MICA increases the maximum feasible atom count by $3.6\times$ to $8.5\times$ and edge count by $3.4\times$ to $7.9\times$ over Base. Compared with Torch, MICA increases the atom limit by $3.0\times$ to $6.8\times$ and the edge limit by $2.8\times$ to $6.8\times$. This capacity helps multi-task MLIP training use larger batches before resorting to additional devices or heavier distributed overhead.

6 Related work

Activation checkpointing and rematerialization. Prior checkpointing and rematerialization systems use static, dynamic, or input-aware policies to reduce activation memory [5, 9, 12, 17], but mainly target ordinary first-order training. Conservative MLIPs additionally create First-Backward (FBW) states that are later consumed by Double-Backward (DBW). Although a sufficiently general rematerialization system could express equivalent policies in principle, current implementations have practical abstraction gaps: PyTorch checkpointing baseline and Mimose's planner operate on Forward regions, DTR requires operator-specific kernel support, and HiRemate planners require a complete higher-order graph. MICA instead provides a PyTorch-compatible region-level primitive for rematerializing FBW activations and combines it with input-aware mode selection. Table 2 summarizes these differences,

Table 2: Comparison with prior checkpointing and rematerialization methods. MICA supports input-aware FBW rematerialization for double-backward MLIP training.

Method	Input-aware	Direct FBW remat.	Direct MLIP DBW
PyTorch checkpoint[†]	✗	✗	✓
Mimose [17][§]	✓	✗	✓
DTR [12][*]	✗	✗	✗
HiRemate [9][‡]	✗	✗	✗
MICA	✓	✓	✓

[†] Supports DBW but checkpoints only Forward regions. [*] DTR requires operator-specific replay implementations; the version evaluated here lacks the required replay support for some operators in our FW, FBW, and DBW paths. [‡] In our HiRemate evaluation, `torch.export` failed at `torch.autograd.grad` with `Unsupported: skip function grad`. Bypassing this check with `torch._dynamo.allow_in_graph` still did not preserve the required higher-order dependencies of parameters and saved activations. [§] Mimose uses PyTorch's `torch.utils.checkpoint` API.

Memory optimization for large-scale MLIP training. Large-scale MLIP training often relies on distributed parallelism to fit larger models or graph batches under GPU memory limits [24, 27]. Graph parallelism [24] partitions large atomic graphs and their activations across GPUs. State-sharding methods such as ZeRO [22] and FSDP [29] partition parameters, gradients, and optimizer states across devices. These methods can enable larger models or graph batches, but they also introduce communication, synchronization, load imbalance, and distributed state-management overheads. Kernel and compiler optimizations reduce memory and time at a finer granularity by fusing MLIP operators, eliminating temporary tensors, or improving data locality [14, 26]. Load-balancing algorithms can reorder samples or graph batches to improve balance across batches, but such reordering may affect convergence or accuracy [7, 30]. MICA is complementary to distributed training, load balancing, and kernel-level optimizations.

7 Conclusion

This paper presented MICA, a double-backward-aware activation checkpointing system for conservative MLIP training. We showed that conservative MLIPs follow a four-phase execution structure, in which activations produced during First-Backward are later consumed by Double-Backward and are not directly managed by standard activation checkpointing. MICA addresses this bottleneck with a phase-aware checkpoint primitive and an input-aware dynamic-programming planner that selects region-level checkpoint modes from graph-batch features under a target memory budget. Across EqV3, MatRIS, PET, and UMA, MICA-M2 reduces average peak memory by 39% to 71%; MICA keeps EqV3 training within a 50 GB target with 25.38% step time overhead and increases the maximum feasible atom and edge counts by up to $8.5\times$ and $7.9\times$, respectively. By reducing activation memory with controlled recomputation overhead, MICA lowers the memory barrier for conservative MLIP training, an important AI for Science workload. It offers MLIP practitioners a practical primitive for future scaling studies.

Acknowledgments

This work was supported by the following funding: National Natural Science Foundation of China (T2125013, 92270206, 62372435, 62502501), Beijing Natural Science Foundation (4254087), National Key Research and Development Program of China (2025YFB4507000), and the Innovation Funding of ICT, CAS. Part of the training is performed on the robotic AI-Scientist platform of the Chinese Academy of Sciences. The authors declare no competing interests.

References

[1] 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. *arXiv e-prints*, art. arXiv:2410.12771, October 2024. doi: 10.48550/arXiv.2410.12771.

[2] Ilyes Batatia, David Peter Kovacs, Gregor N. C. Simm, Christoph Ortner, and Gabor Csanyi. MACE: Higher order equivariant message passing neural networks for fast and accurate force fields. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=YPpSngE-ZU.

[3] Filippo Bigi, Marcel F. Langer, and Michele Ceriotti. The dark side of the forces: assessing non-conservative force models for atomistic machine learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=OEl3L8osas.

[4] Filippo Bigi, Paolo Pegolo, Arslan Mazitov, Jonathan Schmidt, and Michele Ceriotti. Pushing the limits of unconstrained machine-learned interatomic potentials, 2026. URL https://arxiv.org/abs/2601.16195.

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL https://arxiv.org/abs/1604.06174.

[6] 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.

[7] Jesun Sahariar Firoz, Franco Pellegrini, Mario Geiger, Darren Hsu, Jenna A. Bilbrey, Han-Yi Chou, Maximilian Stadler, Markus Höhnerbach, Tingyu Wang, Dejun Lin, Emine Kucukbenli, Henry W. Sprueill, Ilyes Batatia, Sotiris S. Xantheas, Mal-Soon Lee, Christopher Mundy, Gabor Csanyi, Justin S. Smith, Ponnuswamy Sadayappan, and Sutanay Choudhury. Optimizing data distribution and kernel performance for efficient training of chemistry foundation models: A case study with mace. In *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718694. doi: 10.1145/3731545.3731594. URL https://doi.org/10.1145/3731545.3731594.

[8] Vahe Gharakhanyan, Luis Barroso-Luque, Yi Yang, Muhammed Shuaibi, Kyle Michel, Daniel S. Levine, Misko Dzamba, Xiang Fu, Meng Gao, Xingyu Liu, Haoran Ni, Keian Noori, Brandon M. Wood, Matt Uyttendaele, Arman Boromand, C. Lawrence Zitnick, Noa Marom, Zachary W. Ulissi, and Anuroop Sriram. Open molecular crystals 2025 (omc25) dataset and models, 2025. URL https://arxiv.org/abs/2508.02651.

[9] Julia Gusak, Xunyi Zhao, Théotime Le Hellard, Zhe LI, Lionel Eyraud-Dubois, and Olivier Beaumont. Hiremate: Hierarchical approach for efficient re-materialization of neural networks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=rnx11J4hsg.

[10] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Kurt Keutzer, Ion Stoica, and Joseph E. Gonzalez. Checkmate: Breaking the memory wall with optimal tensor rematerialization, 2020. URL https://arxiv.org/abs/1910.02653.

[11] Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, Weinan E, and Linfeng Zhang. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

[12] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization, 2021. URL https://arxiv.org/abs/2006.09616.

[13] Walter Kohn and Lu Jeu Sham. Self-consistent equations including exchange and correlation effects. *Physical review*, 140(4A):A1133, 1965.

[14] 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 *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=wiQe95BPaB.

[15] 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, Nathan C. Frey, Xiang Fu, Vahe Gharakhanyan, Aditi S. Krishnapriyan, Joshua A. Rackers, Sanjeev Raja, Ammar Rizvi, Andrew S. Rosen, Zachary Ulissi, Santiago Vargas, C. Lawrence Zitnick, Samuel M. Blau, and Brandon M. Wood. The open molecules 2025 (omol25) dataset, evaluations, and models, 2026. URL https://arxiv.org/abs/2505.08762.

[16] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. URL https://arxiv.org/abs/2006.15704.

[17] Jianjin Liao, Mingzhen Li, Hailong Yang, Qingxiao Sun, Biao Sun, Jiwei Hao, Tianyu Feng, Fengwei Yu, Shengdong Chen, Ye Tao, Zicheng Zhang, Zhongzhi Luan, and Depei Qian. Exploiting input tensor dynamics in activation checkpointing for efficient training on gpu. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 156–166, 2023. doi: 10.1109/IPDPS54959.2023.00025.

[18] Yi-Lun Liao, Brandon Wood, Abhishek Das, and Tess Smidt. Equiformerv2: Improved equivariant transformer for scaling to higher-degree representations, 2024. URL https://arxiv.org/abs/2306.12059.

[19] Yi-Lun Liao, Alexander J. Hoffman, Sabrina C. Shen, Alexandre Duval, Sam Walton Norwood, and Tess Smidt. Equiformerv3: Scaling efficient, expressive, and general se(3)-equivariant graph attention transformers. *arXiv preprint arXiv:2604.09130*, 2026.

[20] Weijian Liu, Mingzhen Li, Guangming Tan, and Weile Jia. Mario: Near zero-cost activation checkpointing in pipeline parallelism. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '25, page 197–211, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714436. doi: 10.1145/3710848.3710878. URL https://doi.org/10.1145/3710848.3710878.

[21] Eric Qu and Aditi S. Krishnapriyan. The importance of being scalable: Improving the speed and accuracy of neural network interatomic potentials across chemical domains, 2024. URL https://arxiv.org/abs/2410.24169.

[22] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[23] Benjamin Rhodes, Sander Vandenhaute, Vaidotas Šimkus, James Gin, Jonathan Godwin, Tim Duignan, and Mark Neumann. Orb-v3: atomistic simulation at scale, 2025. URL https://arxiv.org/abs/2504.06231.

[24] Anuroop Sriram, Abhishek Das, Brandon M. Wood, Siddharth Goyal, and C. Lawrence Zitnick. Towards training billion parameter graph neural networks for atomic simulations, 2022. URL https://arxiv.org/abs/2203.09697.

[25] Anuroop Sriram, Sihoon Choi, Xiaohan Yu, Logan M. Brabson, Abhishek Das, Zachary Ulissi, Matt Uyttendaele, Andrew J. Medford, and David S. Sholl. The open dac 2023 dataset and challenges for sorbent discovery in direct air capture, 2023. URL https://arxiv.org/abs/2311.00341.

[26] Chuin Wei Tan, Marc L. Descoteaux, Mit Kotak, Gabriel de Miranda Nascimento, Seán R. Kavanagh, Laura Zichi, Menghang Wang, Aadit Saluja, Yizhong R. Hu, Tess Smidt, Anders Johansson, William C. Witt, Boris Kozinsky, and Albert Musaelian. High-performance training and inference for deep equivariant interatomic potentials, 2025. URL https://arxiv.org/abs/2504.16068.

[27] Brandon M Wood, Misko Dzamba, Xiang Fu, Meng Gao, Muhammed Shuaibi, Luis Barroso-Luque, Kareem Abdelmaqsoud, Vahe Gharakhanyan, John R. Kitchin, Daniel S. Levine, Kyle Michel, Anuroop Sriram, Taco Cohen, Abhishek Das, Sushree Jagriti Sahoo, Ammar Rizvi, Zachary Ward Ulissi, and C. Lawrence Zitnick. UMA: A family of universal models for atoms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=SvopaNxYWt.

[28] Duo Zhang, Anyang Peng, Chun Cai, Wentao Li, Yuanchang Zhou, Jinzhe Zeng, Mingyu Guo, Chengqian Zhang, Bowen Li, Hong Jiang, Tong Zhu, Weile Jia, Linfeng Zhang, and Han Wang. A graph neural network for the era of large atomistic models, 2026. URL https://arxiv.org/abs/2506.01686.

[29] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, August 2023. ISSN 2150-8097. doi: 10.14778/3611540.3611569. URL https://doi.org/10.14778/3611540.3611569.

[30] Yuanchang Zhou, Siyu Hu, Chen Wang, Lin-Wang Wang, Guangming Tan, and Weile Jia. Fastchgnet: Training one universal interatomic potential to 1.5 hours with 32 gpus. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 236–246, 2025. doi: 10.1109/IPDPS64566.2025.00029.

[31] Yuanchang Zhou, Siyu Hu, Xiangyu Zhang, Hongyu Wang, Guangming Tan, and Weile Jia. MatRIS: Toward reliable and efficient pretrained machine learning interatomic potentials. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5xBT5Ziute.

[32] Yuanchang Zhou, Hongyu Wang, Yiming Du, Yan Wang, Mingzhen Li, Siyu Hu, Xiangyu Zhang, Weijian Liu, Chen Wang, Zhuoqiang Guo, Long Wang, Jingde Bu, Yutong Lu, Guangming Tan, and Weile Jia. Breaking the training barrier of billion-parameter universal machine learning interatomic potentials, 2026. URL https://arxiv.org/abs/2604.15821.

A MICA dynamic programming planner

Algorithm 1 gives the full pseudocode of MICA's input aware dynamic programming planner. The algorithm first predicts the no-checkpoint peak memory and per region mode savings and overheads. Using the adaptive reserve, it computes the required saving, discretizes the saving values into bins of size Δ, and runs a multiple choice dynamic program.

Algorithm 1 MICA input-aware dynamic programming planner

Input: batch b, memory budget M_{bud}, checkpoint regions $\mathcal{R} = \{R_i\}_{i=1}^{R}$, predictors $\mathcal{P}_{\text{peak}}$ and $\mathcal{P}_{\text{reg}}$, adaptive reserve M_{res}, bin size Δ, recomputation coefficients α, β
Output: mode assignment $\kappa = (k_1, \ldots, k_R)$ and feasibility flag
$\phi \leftarrow \text{GraphFeatures}(b)$ {atoms, edges, triplets}
$M_{\text{peak}} \leftarrow \mathcal{P}_{\text{peak}}(\phi)$
for each region $i \in \{1, \ldots, R\}$ **do**
  Predict $M_i^{\text{fw}}, M_i^{\text{fbw}}, T_i^{\text{fw}}, T_i^{\text{fbw}}$ using $\mathcal{P}_{\text{reg}}(i, \phi)$
  $(v_{i,0}, c_{i,0}) \leftarrow (0, 0)$
  $(v_{i,1}, c_{i,1}) \leftarrow (M_i^{\text{fbw}}, T_i^{\text{fbw}})$
  $(v_{i,2}, c_{i,2}) \leftarrow (M_i^{\text{fw}} + M_i^{\text{fbw}}, \alpha T_i^{\text{fw}} + \beta T_i^{\text{fbw}})$
  for each mode $k \in \{0, 1, 2\}$ **do**
    $\bar{v}_{i,k} \leftarrow \lfloor \max(0, v_{i,k})/\Delta \rfloor$
  end for
end for
$S_{\text{req}} \leftarrow \max(0, M_{\text{peak}} + M_{\text{res}} - M_{\text{bud}})$
if $S_{\text{req}} = 0$ **then**
  return $(0, \ldots, 0)$, feasible
end if
$\bar{S} \leftarrow \lceil S_{\text{req}}/\Delta \rceil$
Initialize $D[0][0] \leftarrow 0$ and $D[0][s > 0] \leftarrow +\infty$
Initialize backpointer table Prev
for $i = 1$ to R **do**
  for $s = 0$ to $\bar{S}$ **do**
    $D[i][s] \leftarrow +\infty$
  end for
  for $s = 0$ to $\bar{S}$ **do**
    if $D[i-1][s] < +\infty$ **then**
      for each mode $k \in \{0, 1, 2\}$ **do**
        $s' \leftarrow \min(\bar{S}, s + \bar{v}_{i,k})$
        if $D[i-1][s] + c_{i,k} < D[i][s']$ **then**
          $D[i][s'] \leftarrow D[i-1][s] + c_{i,k}$
          $\text{Prev}[i][s'] \leftarrow (s, k)$
        end if
      end for
    end if
  end for
end for
if $D[R][\bar{S}] < +\infty$ **then**
  $\kappa \leftarrow \text{Backtrack}(\text{Prev}, R, \bar{S})$
  return κ, feasible
else
  $s^\star \leftarrow \arg\max_s \{s \mid D[R][s] < +\infty\}$
  $\kappa \leftarrow \text{Backtrack}(\text{Prev}, R, s^\star)$
  return κ, infeasible
end if

After each training step, MICA records the observed peak memory as $M_{\text{prev_peak}}$ and updates M_{res} according to Eq. (3). The next planner invocation uses the updated reserve without applying the update again. This update does not change checkpoint modes inside an active autograd step. By default, MICA uses $\alpha = 3$ and $\beta = 1$ for the force and energy objective used in our evaluation.

B Additional experiments

B.1 Experimental setup details

B.1.1 Model configurations

We evaluate MICA on four representative material graph neural network backbones: EqV3, MatRIS, PET, and UMA. Their parameter scales, depths, and key architectural hyperparameters are summarized in Table 3. Unless otherwise stated, MatRIS refers to the 1B configuration in Table 3; the distributed throughput experiment in Figure 6(a) uses a larger MatRIS 2B configuration. EqV3, MatRIS, and UMA use AdamW, while PET uses Adam. These backbones cover diverse architectural patterns, including equivariant message passing, attention based graph modeling, and mixture of experts designs, allowing us to evaluate MICA across models with different activation memory and recomputation characteristics.

Table 3: Model configurations used in the evaluation.

Model	Params	Depth	Configuration
EqV3	85M	8 interaction layers	`num_channels=128`; `ffn_hidden_channels=512`; `num_heads=8`; `max_neighbors=20`; `max_radius=12.0`
MatRIS	1.00B	8 interaction blocks	`num_experts=64`; `top_k=4`; `node=1536`; `edge=384`; `triplet=384`; `attention=384`; `max_neighbors=20`; `max_tri_neighbors=20`; `pair_cutoff=6.0`; `three_body_cutoff=4.0`
MatRIS	2.00B	16 interaction blocks	`num_experts=64`; `top_k=4`; `node=1536`; `edge=384`; `triplet=384`; `attention=384`; `max_neighbors=20`; `max_tri_neighbors=20`; `pair_cutoff=6.0`; `three_body_cutoff=4.0`
PET	730M	5 GNN layers (3 attention layers per layer)	`d_pet=640`; `d_node=2560`; `d_feedforward=1280`; `adaptive_neighbors=40`; `cutoff=10.0`
UMA	1.39B	10 layer blocks	`num_experts=32`; `lmax=4`; `mmax=2`; `sphere_channels=128`; `hidden_channels=128`; `edge_channels=128`; `num_distance_basis=128`; `max_neighbors=30`; `cutoff=6.0`

B.1.2 Checkpoint region design

MICA uses coarse grained checkpoint regions for all evaluated MLIPs. A region is a deterministic and recomputable computation unit with well defined boundary inputs and outputs. We do not wrap individual operators because this would create many small checkpoint units and add runtime overhead. Instead, we define regions around the main computations inside interaction blocks, which produce most activations in our evaluated MLIPs. The planner then selects Mode 0, Mode 1, or Mode 2 for each region.

EqV3 [19]. Each EqV3 interaction layer contains equivariant attention followed by feed forward and update computation. We split each layer into two regions: `attn_region` and `ffn_region`. The `attn_region` covers attention based message computation. The `ffn_region` covers the subsequent feed forward and update computation. Since EqV3 uses 8 interaction layers in our evaluation, this gives $8 \times 2 = 16$ planner selectable regions.

MatRIS [32]. Each MatRIS interaction block contains triplet geometry computation, attention based message passing, and FFN or MoE update computation. We define three region families: `triplet_region`, `attn_region`, and `ffn_region`. The `triplet_region` covers triplet feature construction and aggregation. The `attn_region` covers attention message computation. The `ffn_region` covers edge FFN updates. The evaluated MatRIS configuration has 8 triplet regions, 8 attention regions, and 7 FFN regions, giving 23 planner-selectable regions.

PET [4]. PET contains 5 GNN layers, each with neighbor message passing, internal attention, and combine or update computation. We use two region families: `gnn_region` and `combine_region`. The `gnn_region` covers the main neighbor message passing and internal attention computation.

The `combine_region` covers the layer level combine and update computation. We do not split the internal attention layers into separate regions because this would increase boundary overhead while adding limited scheduling flexibility. PET therefore has $5 \times 2 = 10$ planner selectable regions.

UMA [27]. UMA layers are dominated by edgewise equivariant computation and atomwise aggregation or update computation. We split each layer into two regions: `edgewise_region` and `atomwise_region`. The `edgewise_region` covers the main equivariant edge computation. The `atomwise_region` covers the subsequent per-node update on spherical node embeddings. Our UMA configuration uses 10 layer blocks, giving $10 \times 2 = 20$ planner selectable regions.

MICA does not assign planner modes to embedding, graph construction, basis construction, or readout. These components have relatively small activation memory footprints compared with the main interaction regions, so checkpointing them provides limited additional memory savings.

Table 4: Checkpoint region families used by MICA.

Model	Region families	Instances	Notes
EqV3	`attn_region`, `ffn_region`	16	Attention and message passing, followed by FFN and update computation.
MatRIS	`triplet_region`, `attn_region`, `ffn_region`	23	Triplet, attention, and FFN computation.
PET	`gnn_region`, `combine_region`	10	Neighbor message passing with internal attention, followed by layer level combine and update computation.
UMA	`edgewise_region`, `atomwise_region`	20	Edgewise equivariant computation and atomwise aggregation or update computation.

B.1.3 Datasets.

The mixed data pool is constructed from ODAC23 [25], OMat24 [1], OMol25 [15], and OMC25 [8] with an equal sampling ratio. We shuffle the pooled samples and construct each training batch using fixed atom budget batching. The default atom budget is 400 atoms per batch.

B.1.4 Baselines.

Single-GPU baselines. For single-GPU checkpointing experiments, we compare MICA with Base, Torch, and Mi.

Base and Torch. Base is the original Double-Backward training execution without activation checkpointing and keeps the default autograd intermediate states. Torch uses the standard PyTorch activation checkpointing API on the same candidate regions as MICA. For Torch, we use the checkpoint implementation executable for our double-backward force training loop in PyTorch 2.6.0 while preserving the default RNG behavior. Torch is therefore the directly executable primitive-level baseline.

Mi. We do not use Mimose [17] as an unmodified baseline because Mimose assumes ordinary first-order training and schedules standard checkpointing primitives. In our setting, those primitives can reduce Forward activations but cannot manage activations produced during First-Backward and later consumed by Double-Backward. To evaluate the scheduling idea separately from the primitive, we implement an adapted Mimose planner, denoted as Mi, on top of MICA's double-backward-safe primitive. Mi uses the same candidate regions, graph-feature predictors, and M0/M1/M2 execution modes as MICA. Mi replaces Eq. 2's dynamic-programming solver with a memory-driven greedy scheduler inspired by Mimose. The scheduler greedily selects region-mode candidates according to predicted memory saving and execution order until the required saving is covered, while enforcing that each region uses at most one mode. Because M0, M1, and M2 are mutually exclusive choices for each region and have different recomputation costs, this greedy scheduler is a heuristic rather than an optimal solver for MICA's multiple-choice knapsack objective.

Distributed throughput baselines. For the distributed throughput experiment in Figure 6(a), we use the larger MatRIS 2B configuration and compare MICA-DDP8 with FSDP8, FSDP4-GP2, and GP8 under a max-feasible-throughput protocol. MICA-DDP8 combines MICA with Distributed Data Parallel (DDP) [16] using data parallel degree 8. It reduces local activation memory through

MICA checkpointing while keeping standard data-parallel execution. FSDP8 uses Fully Sharded Data Parallel (FSDP) [29] with sharding degree 8, and FSDP4-GP2 combines FSDP degree 4 with GP degree 2. FSDP partitions model states, including parameters, gradients, and optimizer states, across GPUs, thereby reducing replicated static memory. It all-gathers parameters before computation and applies reduce-scatter to gradients during backward propagation. In addition to communication, FSDP introduces overhead from managing many small parameter shards. For each sharded FSDP unit, parameter shards must be gathered before computation and released or resharded after use, which adds runtime cost. This overhead is difficult to hide for MLIP workloads because each local computation block is relatively small and provides limited opportunity for communication overlap. In the FSDP baseline, we apply full sharding to the attention, triplet, and FFN components of MatRIS. We use an adapted PyTorch FSDP implementation that supports the Double-Backward MLIP training loop and enables parameter prefetching on the H100 SXM/NVSwitch node. GP8 uses Graph Parallel (GP) [24] with graph parallel degree 8. GP partitions an atomic graph across GPUs and exchanges node features across partitions when required by each interaction block, thereby reducing local activation memory. All distributed methods use the same MatRIS 2B model and graph batching policy. They differ in memory-saving and parallel execution strategies, and each method is evaluated at its largest feasible global atom budget.

B.1.5 Profiling and measurement.

MICA collects 200 profiling steps per calibration policy at the beginning of each evaluated configuration. These profiling steps are part of the online MICA startup phase and do not require a separate offline profiling run. During profiling, MICA estimates region-level memory savings and recomputation costs from whole-step memory and time measurements. MICA then fits lightweight second order polynomial predictors for the runtime planner. For each incoming batch, the planner predicts the memory saving and recomputation cost of candidate checkpoint modes from graph features such as atom count, edge count, and triplet count when available. Unless explicitly stated otherwise, the reported steady state step time is measured after this profiling phase. Before collecting timing and memory statistics, we run 10 warmup training steps so that persistent model states, gradients, and optimizer states are materialized. For the default measurements, we then report statistics over 100 measured training steps. For long training experiments, we report measurements over 1000 iterations.

Profiling scope. The 200 profiling steps per calibration policy are part of MICA startup and are used to fit the no-checkpoint peak predictor and the region-level memory and time predictors. In the reported target-budget experiments, profiling and measurement are run on 80 GB GPUs, while M_{bud} is used as the target for the steady-state controller. These experiments therefore evaluate steady-state control under 30–60 GB targets. A practical deployment path is to reuse a predictor bundle for the same model, precision, batching rule, and cutoff configuration, profile smaller batches first, or seed shape-identical stacked blocks from one profiled block before online refinement. This seeding only initializes region-level predictors; the full model still uses online peak observations and the adaptive reserve.

B.1.6 Hardware and software.

Unless otherwise stated, all experiments are conducted on an NVIDIA A100 80GB PCIe GPU.

The distributed throughput experiment in Figure 6(a) is conducted on an H100 SXM/NVSwitch node.

All experiments use a conda environment. The software stack includes Python 3.10.18, PyTorch 2.6.0, NCCL 2.21.5.

B.2 Prediction accuracy

MICA's planner relies on two types of predictors. The first predicts the no-checkpoint global peak memory $M_{\mathrm{peak}}(N_a, N_e, N_t)$ from the current batch graph. Here, N_a is the atom count, N_e is the edge count, and N_t is the triplet count when available. The second predicts the memory saving and recomputation cost of each candidate region under different checkpoint modes. For this validation experiment, we collect 200 measured steps for each model. We use 50 samples for calibration and keep the remaining 150 samples as the validation set.

Table 6: Cold-start peak-memory prediction accuracy on 100 held-out training steps per model.

Model	Peak-memory MAE (MB)	Peak-memory MAPE
EqV3	191.2	0.38%
PET	587.8	0.97%
UMA-10L	364.8	0.59%
MatRIS-1B	20.8	0.03%

Figure 7 compares predicted and measured global peak memory. The predictions closely follow the perfect prediction line across all four models. The largest global peak memory MAPE is 0.632%. This error is small compared with the tens of GB peak memory in these workloads and is sufficient for the planner to decide whether the current batch needs checkpointing.

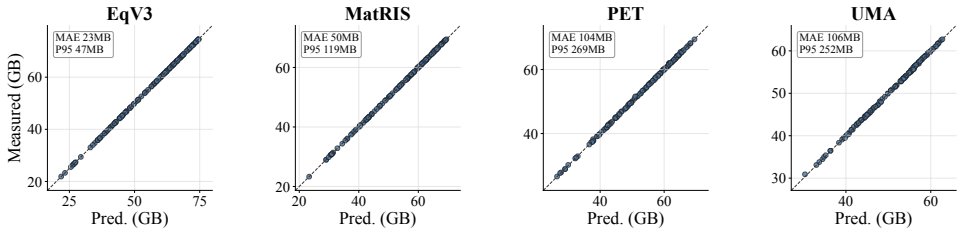


Figure 7: Prediction accuracy of the no-checkpoint global peak memory predictor. The dashed line denotes perfect prediction.

Table 5 further reports the prediction accuracy for mode saving $v_{i,k}$ and recomputation cost $c_{i,k}$. For mode saving, the highest MAPE is 5.65% for Mode 1 and 2.28% for Mode 2. For recomputation cost, the highest MAPE is 10.12%. The recomputation cost predictor is used to choose a lower cost plan among feasible mode assignments, so this accuracy is sufficient for practical mode selection. The table also reports *Max risk*, defined as the largest overestimation of memory saving. In execution, MICA uses conservative DP rounding and an adaptive reserve to mitigate such tail errors.

Table 5: Prediction accuracy of MICA's mode saving $v_{i,k}$ and recomputation cost $c_{i,k}$ models.

Model	Mode 1 memory $v_{i,1}$			Mode 2 memory $v_{i,2}$			Time MAPE $c_{i,k}$	
	MAE	MAPE	Max risk	MAE	MAPE	Max risk	M1	M2
EqV3	12.3 MB	1.05%	505.8 MB	3.1 MB	0.11%	19.2 MB	1.86%	3.69%
MatRIS	31.1 MB	5.65%	132.4 MB	6.9 MB	0.63%	61.7 MB	9.43%	8.14%
PET	52.4 MB	3.82%	242.0 MB	8.4 MB	0.63%	25.9 MB	1.72%	5.32%
UMA	12.8 MB	4.03%	277.1 MB	15.8 MB	2.28%	108.3 MB	10.12%	5.58%

B.3 Alternative calibration for cold start

As a supplementary study, we evaluate a cold-start calibration procedure that does not require a full uncheckpointed (M0) run during calibration.

We first run one complete training step with a feasible all-M2 configuration and use its measured peak memory as a reference. We then switch one region at a time to M1 or M0 while keeping all other regions in M2, and record the changes in memory and execution time. These measurements are used to fit predictors for the peak memory of mixed M0/M1/M2 plans.

For each model, we evaluate prediction accuracy on 100 mixed-mode training steps. Table 6 reports the results.

Peak-memory MAPE is below 1% for all four models. These results show that, in the evaluated workloads, this procedure can accurately predict the peak memory of mixed-mode plans without a full M0 calibration run.

B.4 Mode allocation under memory budgets

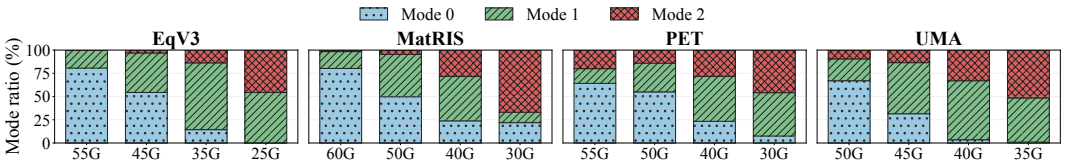


Figure 8: Mode distribution selected by MICA under different memory budgets. Each bar reports the ratio of planner assignments over candidate regions and measured steps for one model and one budget.

Figure 8 shows how MICA changes its checkpointing strength under different memory budgets. Mode 0 keeps the region uncheckpointed. Mode 1 checkpoints activations produced during First-Backward. Mode 2 checkpoints activations produced during both Forward and First-Backward. At the highest evaluated budget for each model, most regions remain in Mode 0, which avoids recomputation. As the budget decreases, the planner reduces the fraction of Mode 0 and moves regions to Mode 1 or Mode 2. This trend appears across all four models. The selected mode mix is model dependent. Some budgets can be satisfied mostly with Mode 1, while lower budgets require Mode 2 because Forward activations must also be removed. This shows that a single fixed checkpointing strategy is not enough for different architectures and memory targets. MICA uses the expanded Mode 0/1/2 execution space to adjust checkpointing at region granularity, giving a finer memory and time tradeoff than always using no checkpointing or always using full checkpointing.

B.5 Long run memory traces under a 50GB budget

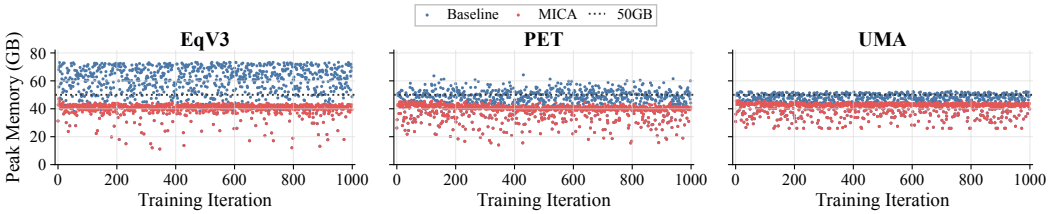


Figure 9: Long run peak memory traces under a fixed 50GB memory budget. Each point reports the peak GPU memory of one training iteration. Baseline uses the original Double-Backward execution without checkpointing. MICA uses a 50GB target budget. The dashed line marks the 50GB budget.

Figure 9 extends the long run budget compliance study to EqV3, PET, and UMA. The baseline shows large memory spikes under dynamic graph batches. Across 1000 iterations, 74.8%, 27.8%, and 16.7% of baseline steps exceed 50GB on EqV3, PET, and UMA, respectively. The maximum peak memory reaches 73.29GB, 64.28GB, and 52.64GB.

MICA controls these traces near the 50GB target. EqV3 and PET have no budget violations over all 1000 iterations, with maximum peak memory of 47.15GB and 46.03GB. UMA also has no budget violations, and its maximum peak memory is 45.81GB. The average peak memory is reduced from 57.31GB to 39.89GB on EqV3, from 43.83GB to 38.48GB on PET, and from 43.80GB to 40.75GB on UMA.

This memory control comes with moderate recomputation overhead. The average step time overheads are 25.38%, 11.20%, and 6.57% for EqV3, PET, and UMA. MICA also preserves the original training computation. Using the 1000 step average metrics as the reference, the largest relative errors across the three models are 1.83×10^{-7} for Energy MAE and 6.93×10^{-9} for Force MAE. These results show that MICA suppresses tail memory spikes while keeping the training computation numerically consistent with the baseline.

B.6 MICA online control overhead

Table 7: MICA online prediction and planning overhead. The overhead is below 0.8% of the baseline step time for all models.

Model	Step (s)	Pred. (ms)	Plan (ms)	Total (ms)	% step
EqV3	2.071	0.1643	5.4856	5.6500	0.2728%
MatRIS	1.484	0.2392	10.6848	10.9240	0.7359%
PET	1.599	0.1272	0.3843	0.5115	0.0320%
UMA	1.609	0.1897	3.0625	3.2522	0.2021%

Prediction includes memory and recomputation cost prediction. Plan includes the dynamic programming solver and mode assignment bookkeeping. The total online control overhead is below 0.8% of the baseline step time for all models.

B.7 Correctness analysis

1. Objective preservation. MICA changes only the lifetime of intermediate states. It does not change the training objective. It does not change the model function. It does not change the optimizer update rule. Within each training step, model parameters are updated only after all backward phases finish. Therefore, every recomputed region uses the same parameter values as the original execution in that step.

2. Recomputation state preservation. For each checkpointed region, MICA records the boundary state needed to recompute the local computation. This state includes boundary inputs, PyTree metadata, gradient requirements, and CPU/CUDA random number generator (RNG) states. Before recomputing a region, MICA restores the recorded RNG state. MICA then calls the region again with the same boundary state. After recomputation finishes, MICA restores the RNG state that was active before recomputation so that subsequent computation follows the original random stream. Thus, stochastic operations inside the region and subsequent computation observe the same random streams as in the original execution.

3. Mode level equivalence. The three checkpoint modes differ only in which intermediate states are kept live and which states are recomputed. Mode 0 executes the region normally. Mode 1 keeps Forward activations live. Mode 1 recomputes the local First-Backward computation when Double-Backward needs it. Mode 2 recomputes the local Forward computation when later phases need it. Mode 2 also rematerializes the local First-Backward computation when Double-Backward needs it. In all modes, the differentiated function is unchanged. MICA only replaces stored internal activations with an equivalent local recomputation.

4. Numerical tolerance. In exact arithmetic, recomputation gives the same gradients as the original Double-Backward execution. On GPUs, reductions, atomic operations, and backend kernel selection can introduce small order dependent differences, so bitwise equality is not guaranteed. These differences are normal floating point effects, not changes to the training objective or gradient path. Using the same initialization, random seed, batch order, and optimizer setting, MICA matches the baseline within numerical tolerance in Energy MAE, Force MAE, and loss statistics, as shown in Figure 5 and Appendix B.5.

C MICA API examples

This appendix shows a simplified user API for MICA. MICA provides a Double-Backward aware checkpointing API that can replace standard `torch.utils.checkpoint` in conservative MLIP training. The implementation wraps recomputable regions with MICA checkpoint primitives and uses a MICA gradient entry only where force computation requires `create_graph=True`.

Manual checkpointing. Listing 1 shows the minimal use case. A submodule is wrapped as a MICA checkpoint region, and force computation uses `mica.autograd.grad` with `create_graph=True`.

```python
from mica import double_backward_checkpoint_module
from mica.autograd import grad

# M2 checkpoints both Forward and First-Backward states for this region.
model.blocks[3] = double_backward_checkpoint_module(
    model.blocks[3],
    mode="M2",
)

graphs.atom_position.requires_grad_(True)

energy = model(graphs)
forces = -grad(
    energy.sum(),
    inputs=graphs.atom_position,
    create_graph=True,
)[0]

loss = loss_fn(targets, energy, forces)
loss.backward()
```

Listing 1: Manual MICA use. A submodule is wrapped as a Mode 2 checkpoint region for conservative force training.

Planner based checkpointing. Listing 2 shows simplified pseudocode for integrating MICA with an input-aware planner. At each step, the planner extracts graph features (atoms, edges, triplets) from the current batch and assigns one of M0, M1, and M2 to each concrete region.

```python
from mica.adapters.matris import apply_matris_region_wrappers
from mica.planners import (
    RegionPlannerRuntime,
    extract_input_stats_from_graph_batch,
    load_planner_predictor_bundle,
)
from mica.autograd import grad

predictor = load_planner_predictor_bundle("matris_predictor.json")

planner = RegionPlannerRuntime(
    predictor_bundle=predictor,
    memory_budget_mb=50 * 1024.0,
    safety_margin_mb=2048.0,
    adaptive_safety_enabled=True,
)

# Register triplet, attention, and FFN regions as selectable regions.
model = apply_matris_region_wrappers(model, planner)

for graphs, batch_metadata, targets in loader:
    stats = extract_input_stats_from_graph_batch(graphs, batch_metadata)
    planner.start_step(stats)

    graphs.atom_position.requires_grad_(True)

    energy = model(graphs)
    forces = -grad(
        energy.sum(),
        inputs=graphs.atom_position,
        create_graph=True,
    )[0]

    loss = loss_fn(targets, energy, forces)
    loss.backward()

    planner.finish_step(
        actual_peak_mb=observed_peak_mb,
        actual_step_time_s=step_time_s,
    )
```

Listing 2: Planner based MICA use. The planner predicts per region memory savings and recomputation costs, then selects M0, M1, or M2 under a memory budget.

D Limitations

MICA is a user-facing activation checkpointing library for conservative MLIP training. Its scope is specific: it targets the double backward execution pattern created by force matching, where First-Backward produces activations that are later consumed by Double-Backward. Thus, MICA is not intended as a general checkpointing optimizer for ordinary first-order neural network training or non-MLIP workloads. **We view this specialization as a deliberate design choice.** Conservative MLIPs are a focused but important AI for Science workload, because they enable fast and stable molecular dynamics simulation for materials science, chemistry, catalysis, and related domains. This makes scalable conservative MLIP training a concrete systems problem rather than a narrow implementation detail. MICA contributes to this subfield by identifying a double-backward activation lifetime bottleneck that standard checkpointing does not manage and by providing a PyTorch-compatible runtime and planner to address it. Applying MICA to a new MLIP architecture requires defining deterministic and recomputable checkpoint regions, which lets MICA preserve autograd correctness while integrating with existing PyTorch-based MLIP code. We hope MICA encourages more infrastructure research for MLIP training and other AI for Science workloads with non-standard execution patterns.

E Broader impacts

MICA improves the efficiency and accessibility of double-backward MLIP training by reducing activation memory requirements. This can lower the hardware barrier for scientific modeling workloads. A possible indirect negative impact is that efficiency gains may encourage larger training runs, increasing aggregate compute and energy use.

F Existing assets used in this paper

Table 8: Existing assets used in this paper.

Asset	Type	Source	License
EquiformerV3	Code	GitHub	MIT
MatRIS	Code	GitHub	BSD-3-Clause
PET	Code	GitHub	BSD-3-Clause
UMA / fairchem	Code	GitHub	MIT
OMat24	Dataset	HF	CC BY 4.0
OMol25	Dataset	HF	CC BY 4.0
ODAC23	Dataset	OpenDAC	CC BY 4.0
OMC25	Dataset	HF	CC BY 4.0

Note: HF denotes Hugging Face. License information was taken from official repositories or dataset pages when available.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state that MICA targets activation checkpointing for conservative MLIP training with double-backward execution and dynamic atomic graph inputs.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix D discusses MICA's scope and limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A].

Justification: The paper does not present formal theoretical results; it provides an efficient activation checkpoint library for double-backward MLIP training and validates it empirically.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We release the MICA source code. Sections 5 and Appendix B provide the experimental settings and implementation details needed to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We publicly release the MICA software library. Reproducing the experiments requires integrating MICA with third-party MLIP implementations and obtaining the corresponding datasets from their official sources.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Section 5 and Appendix B report the model configurations, datasets, and measurement method used in the experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: The main memory results are measured over 100 or 1000 training steps after warmup, and timing results are averaged over the measured steps. We do not report error bars, confidence intervals, or statistical significance tests across independent runs because repeating all experimental configurations is computationally expensive.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 and Appendix B report the GPU type, node configuration, software stack, and measurement protocol. Most single GPU experiment groups require approximately 10–20 A100 80GB GPU-hours, while the distributed throughput experiment requires approximately 24 H100 SXM/NVSwitch node-hours. In total, reproducing the reported experiments requires approximately 168 A100 80GB GPU-hours and 24 H100 SXM/NVSwitch node-hours.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and are not aware of any violations.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Section E discusses that MICA can improve the efficiency and accessibility of MLIP training by reducing memory requirements. It also notes the indirect risk that efficiency gains may encourage larger training runs and higher aggregate compute use.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk models or datasets, such as pretrained language models, image generators, or scraped datasets. The released artifact is a training-system implementation for activation checkpointing in MLIP training.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers and repositories for all datasets, models, and software assets used in the experiments. Appendix F lists their sources and licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: The paper releases a new software artifact implementing MICA.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing, human annotation, user studies, or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, human annotation, user studies, or research with human subjects, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A].

Justification: The core method development of this work does not involve LLMs as an important, original, or non-standard component.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.