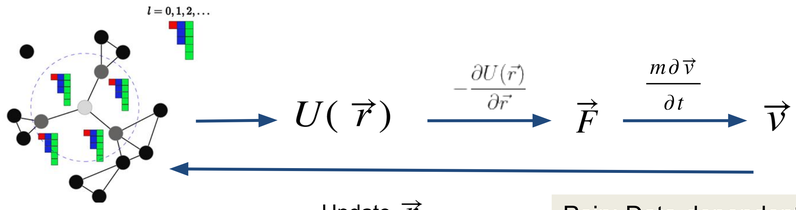
Neural Network Potentials with PyTorch 2

Mit Kotak
mkotak@mit.edu

Chuin-Wei Tan
chuinwei_tan@g.harvard.edu

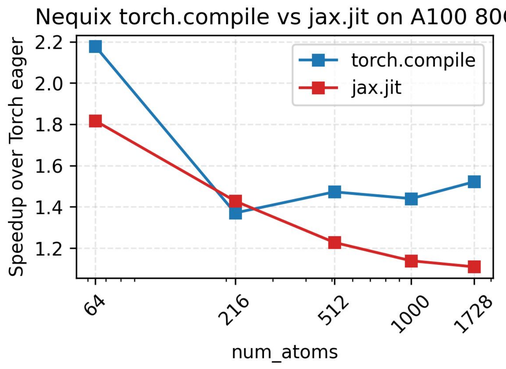
torch.fx.experimental.proxy_tensor.make_fx



Pain: Data-dependent control flow
Dynamic shapes

https://arxiv.org/abs/2504.16068 inspired by @ezyang
https://github.com/pytorch/pytorch/issues/91469#issuecomment-1873221656

torch.compile vs jax.jit



JAX is needed for higher order gradients

https://github.com/mitkotak/matbench-speed

torch._inductor.aoti_compile_and_package

train → .ckpt → torch.package → .zip → torch.export → .pt2

https://nequip.readthedocs.io/en/latest/guide/getting-started/workflow.html

Workflow: Train locally and run distributed inference

C++ distributed MD engine

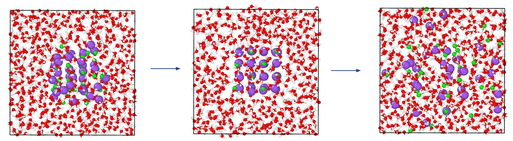
LAMMPS

kokkos

Vendor-agnostic, Direct GPU-GPU transfer

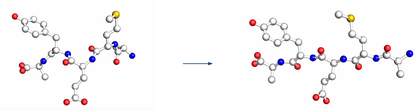
Simulation Showcase

Salt Nucleation



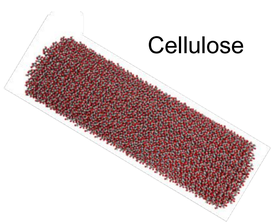
https://github.com/teddykoker/nequix-examples

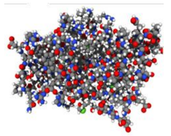
Peptides



https://github.com/prescient-design/jamun

Cellulose



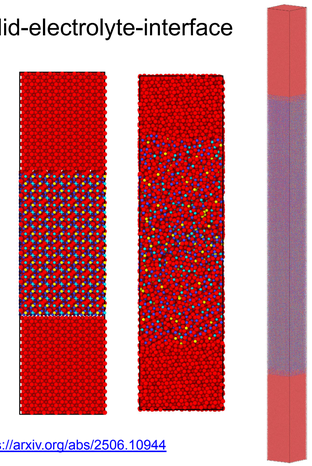


Enzymes

https://dl.acm.org/doi/abs/10.1145/3581784.3627041

Low latency needed for both small and large batch size

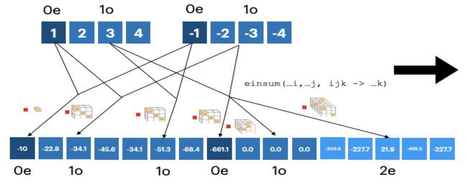
Solid-electrolyte-interface

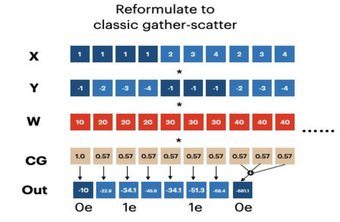


https://arxiv.org/abs/2506.10944

Helion

```python
@helion.kernel(config=helion.Config(block_sizes=[1, 128, 4],
 loop_orders=[[1, 2, 0]], num_warps=1, num_stages=6, indexing='block_ptr'))
def tensor_product_fused_graphconv(
    # Tensors
    node_embedding: torch.Tensor,           #  (Nnodes, channel, Ldim1)
    edge_sph:       torch.Tensor,           #  (Nedges, 1,       Ldim2)
    out_embedding: torch.Tensor,            #  (Nnodes, channel, LdimOut)
    # Sparse data structures
    input1_to_nnz_mapper: torch.Tensor,     #  (NNZ,)
    input2_to_nnz_mapper: torch.Tensor,     #  (NNZ,)
    nnz_to_output_mapper: torch.Tensor,     #  (Nnodes+1,)
    paths_to_nnz_mapper: torch.Tensor,      #  (NNZ,)
    cg_coeffs: torch.Tensor,                #  (NNZ,)
    src: torch.Tensor,                      #  (Nedges,)
    dst: torch.Tensor,                      #  (Nedges,)
    weight: torch.Tensor,                   #  (Nedges, channel, num_paths)
) -> torch.Tensor:
    _, channel, Ldim1 = node_embedding.shape
    Nedges = src.shape[0]
    _, _, LdimOut = out_embedding.shape
    # Tile across the Nedges, channel, and LdimOut dimensions
    for tile_e, tile_u, tile_k in hl.tile([Nedges, channel, LdimOut]):
        src_idx_chunk = src[tile_e]
        dst_idx_chunk = dst[tile_e]
        start_ptr_chunk = nnz_to_output_mapper[tile_u]
        end_ptr_chunk = nnz_to_output_mapper[tile_u.index+1]
        nnz = end_ptr_chunk - start_ptr_chunk
        max_nnz = nnz.amax()
        acc = hl.zeros((tile_e, tile_u, tile_k),dtype=node_embedding.dtype)
        for tile_p in hl.tile(max_nnz, block_size=1):
            l1_idx_chunk =    hl.load(input1_to_nnz_mapper, [start_ptr_chunk + tile_p.index], extra_mask=tile_p.index < nnz)
            l2_idx_chunk =    hl.load(input2_to_nnz_mapper, [start_ptr_chunk + tile_p.index], extra_mask=tile_p.index < nnz)
            w_idx_chunk =     hl.load(paths_to_nnz_mapper,  [start_ptr_chunk + tile_p.index], extra_mask=tile_p.index < nnz)
            cg_coeff_chunk = hl.load(cg_coeffs,             [start_ptr_chunk + tile_p.index], extra_mask=tile_p.index < nnz)
            acc += node_embedding[src_idx_chunk, tile_u, l1_idx_chunk] * edge_sph[tile_e, l2_idx_chunk][:, None, :] *
                   cg_coeff_chunk * weight[tile_e, tile_u, w_idx_chunk]
        hl.atomic_add(out_embedding, [dst_idx_chunk, tile_u, tile_k], acc)
    return out_embedding
```

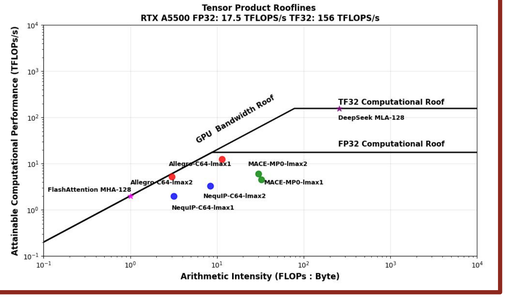




Upto 3x end to end speedups with < 100 lines !

Still lots of room to co-design kernels !

https://mitkotak.github.io/assets/pdf/sm_thesis_v2.pdf



H/Welp

MPS support
torch.export + autograd

Backward kernels

This work was supported by the National Science Foundation through the Harvard University Materials Research Science and Engineering Center Grant No. DMR-2011754, the Department of Navy award N00014-20-1-2418 issued by the Office of Naval Research, and the Department of Energy Office of Basic Energy Sciences Award No. DE-SC0022199.This work is also supported by Robert Bosch LLC and the National Science Foundation under Grant No. DMR-2119351. An award of computer time was provided by the INCITE program. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.This research also used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC awards BESERCAP0026720 and BESERCAP0032494.

The NSF Institute for Artificial Intelligence and Fundamental Interactions