Clad as a First-Class Citizen in LibTorch

Midterm presentation

Kacent Huang

2026-08-19

GSoC 2026

fogsong233@mail.com

Roadmap

1. Making Clad Tensor-ready
2. Mapping LibTorch semantics
3. Benchmark
4. Next steps

1. Making Clad Tensor-ready

What does LibTorch require?

Native Tensor in, native gradients out.

- shape, layout, dtype
- copies can alias storage
- opaque kernel implementations
- multiple call spellings (`+`/`tensor.add()`/ `torch::add()`)

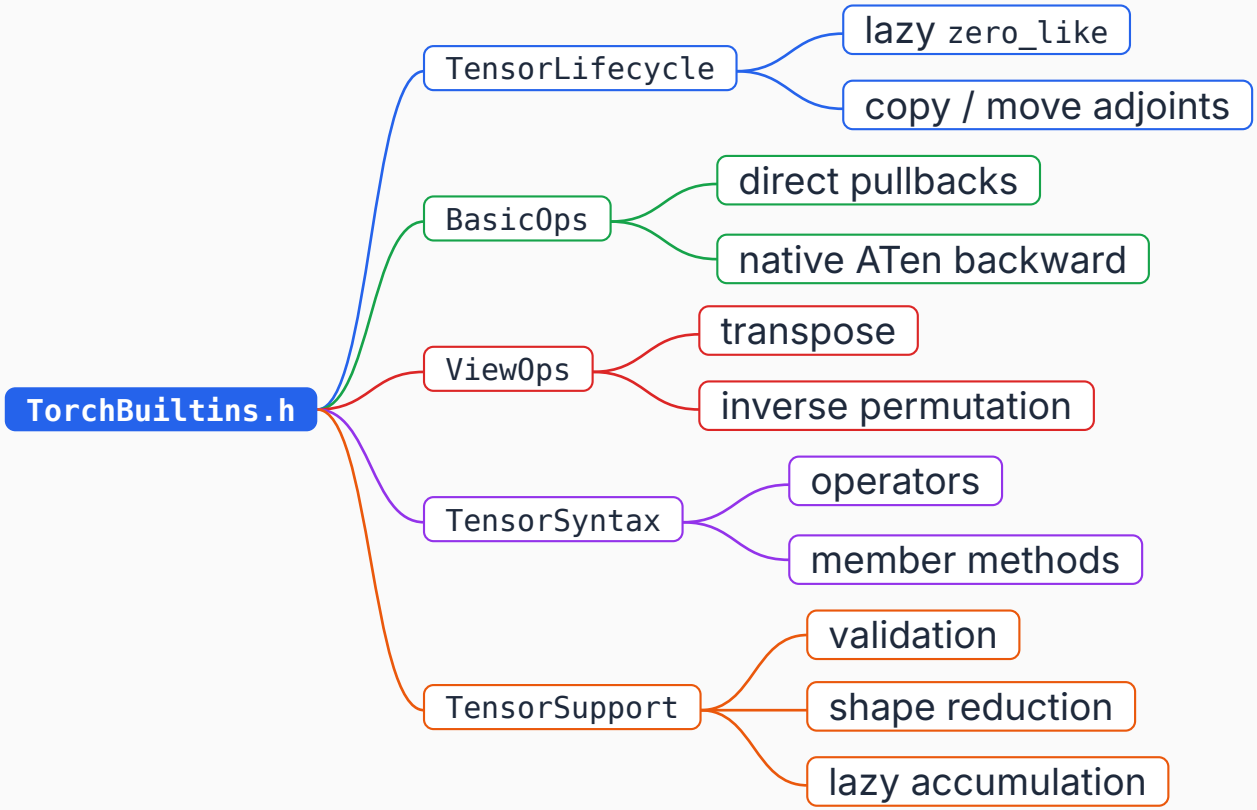
```
float loss(const at::Tensor& x,
           const at::Tensor& bias) {
  return x.mul(x)
          .add(bias)
          .relu()
          .sum()
          .item<float>();
}

auto grad = clad::gradient(loss, "x, bias");
grad.execute(x, bias, &d_x, &d_bias);
```

The missing compiler foundations

Blocker	Failure	Resolution
Nested reverse calls	Generated pullbacks could omit nested calls.	Add lookup fallback · #1832
Non-trivial temporaries	Temporary member calls crashed AST cloning.	Clone `CXXBindTemporaryExpr` · #1926
Dynamic adjoints	`zero_init` cannot preserve Tensor storage.	Add `zero_like` · #1932
Tensor ReturnStmt	Direct calls in `ReturnStmt` lost their adjoints.	Fix reverse return handling · #1922

2. Mapping LibTorch semantics

Layered backend



How to add an operator?

```
inline void relu_pullback(
    const Tensor& input, Tensor d_output,
    Tensor* d_input) {
  if (!d_input || !should_propagate(d_output))
    return;
  NoGradGuard guard;
  validate_tensor(input);
  accumulate(d_input,
    at::threshold_backward(d_output, input, 0));
}
```

Native backward

`derivatives.yaml` documents PyTorch's native backward formulas.

Multiple spellings, one pullback

Methods and operators delegate to one pullback.

Lazy gradients, fewer allocations

`zero_like` returns an undefined Tensor; the first contribution materializes it.

3. Benchmark

Benchmark design

Three execution modes

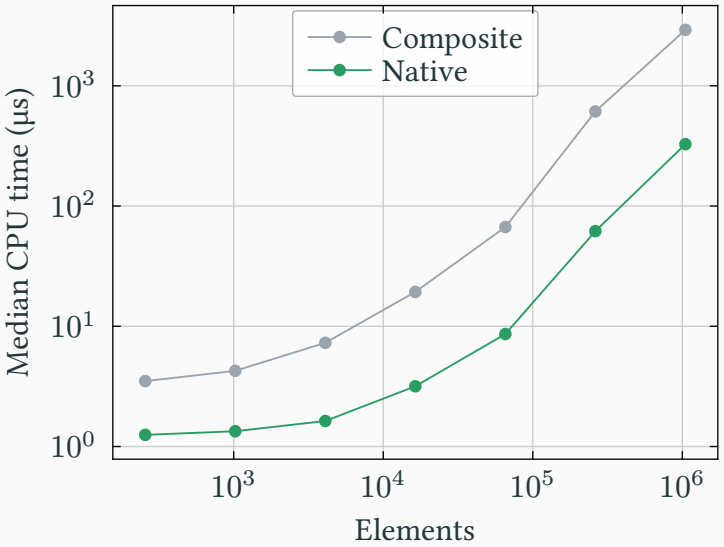
- **Forward**: lower-bound reference under `NoGradGuard`
- **Clad**: source-transformed reverse pass
- **Autograd**: native graph construction and `autograd::grad`

Three workload families

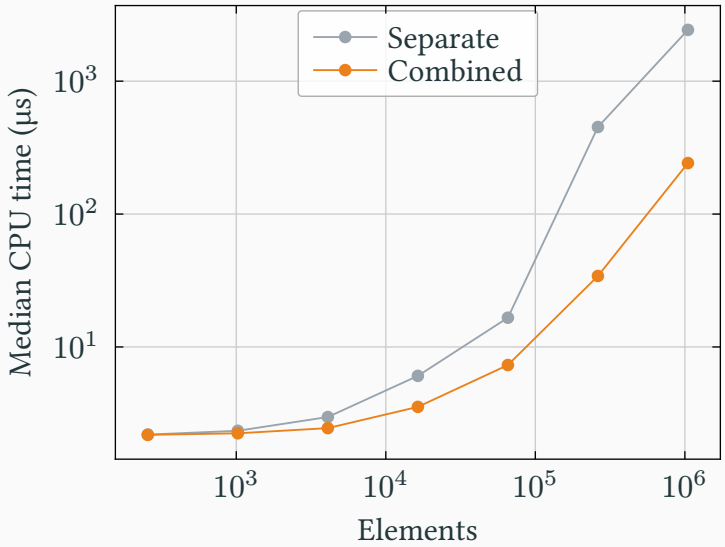
- vector squared error
  $L_{\mathrm{SE}} = \sum_i (p_i - t_i)^2$
- broadcast multiply + ReLU + sum
  $L_{\mathrm{bcast}} = \sum_{b,c,w} \mathrm{ReLU}\left(A_{b,1,w} \times B_{1,c,1}\right)$
- multi-dimension sum + broadcast weights
  $L_{\mathrm{sum}} = \sum_c W_c \sum_{b,w} X_{b,c,w}$

Local backward choices across Tensor sizes

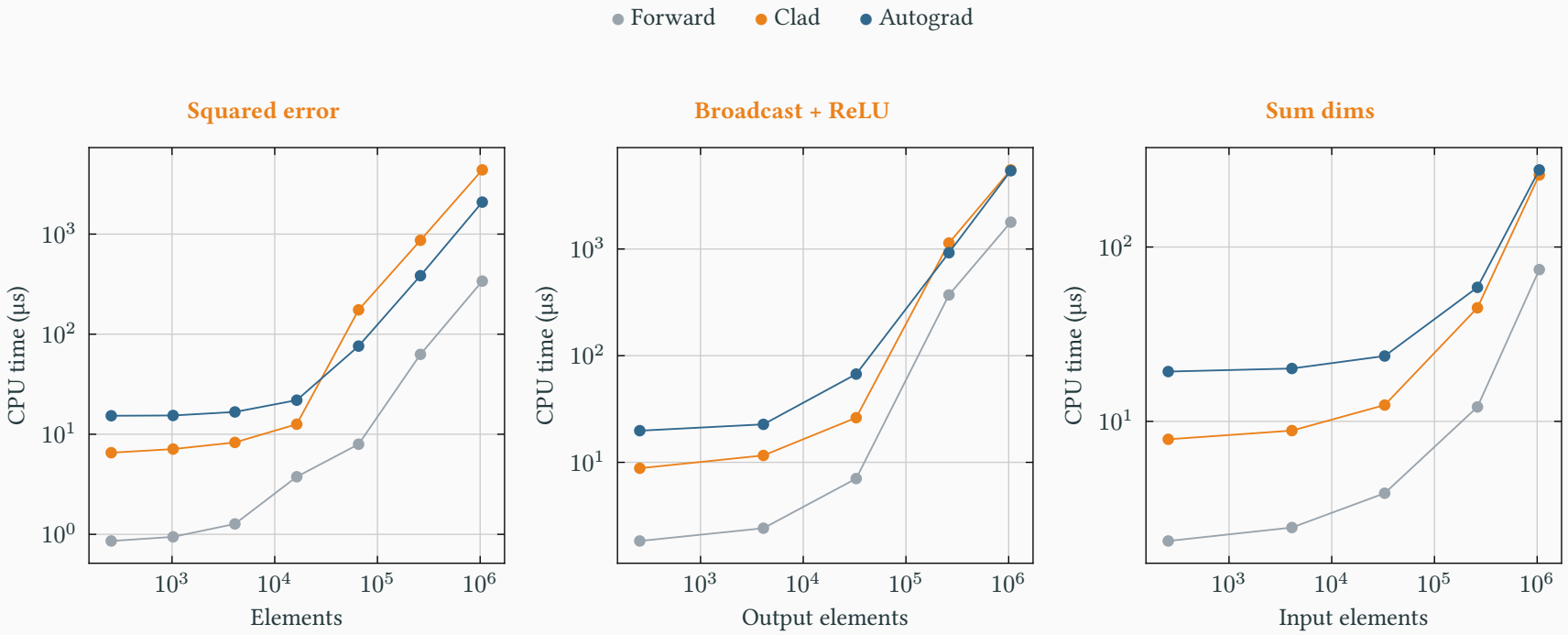
ReLU backward



Repeated-input dot backward



End-to-end scaling across workloads and shapes



Clad wins at small/medium broadcast and reduction sizes; memory traffic dominates the largest cases.

4. Next steps

Next steps

- float64 first; then half / bfloat16
- Define promotion and output-dtype rules for mixed inputs
- Typed scalar extraction and tolerances

Roadmap

Timeline	Proposal focus	Concrete next outcome
Week 16 · now	Stabilize	Direct pullbacks, lazy adjoints, strided views, tests
Weeks 17–18	LibTorch integration	`torch::autograd::Function` or ATen op; graph composition
Weeks 19–20	C++ model prototype	MLP + Clad-generated backward training loop
Weeks 21–22	Performance validation	CPU benchmarks on MNIST/GW-style workloads
Weeks 23–25	ROOT integration	ROOT macro/plugin, RDataFrame/TTrees, ONNX/TorchScript boundary
Weeks 26–27	Polish and handoff	Partial views, float64, docs, review, handoff

Thanks for listening


Questions?