

High-performance in-situ ML Inference with dalotia: A Lightweight Tensor Loader API for Science Codes

AI is mostly BLAS so that's how we should treat it

Theresa Pollinger, Jens Domke

July 7, 2026



“*Turbulence* is the
most important unsolved problem of classical physics”
(Richard Feynman)



- ▶ Computational Fluid Dynamics:
Engineering and Design
- ▶ Climate
- ▶ Plasma Fusion
- ▶ ...



doi.org/10.1145/3773656.3773664

Direct Numerical
Simulation (DNS):
resolve all scales

Large Eddy Simulation (LES):
resolve large scales,
model small scales with *closures*

Reynolds-averaged Navier-Stokes
(RANS):
model all scales with *closures*,
average over ensembles

⇒ can still be costly!

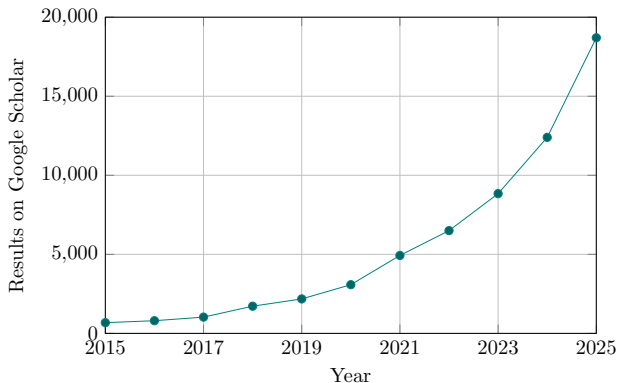


doi.org/10.1145/3773656.3773664

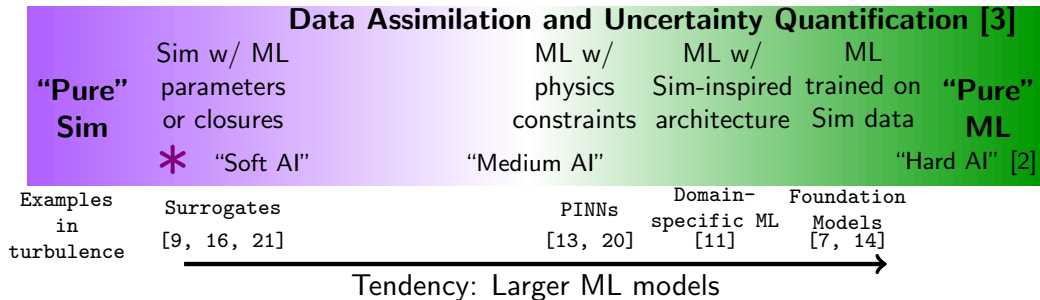
Google Scholar

Articles

About 55,300 results (0,11 sec)



[doi: 10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

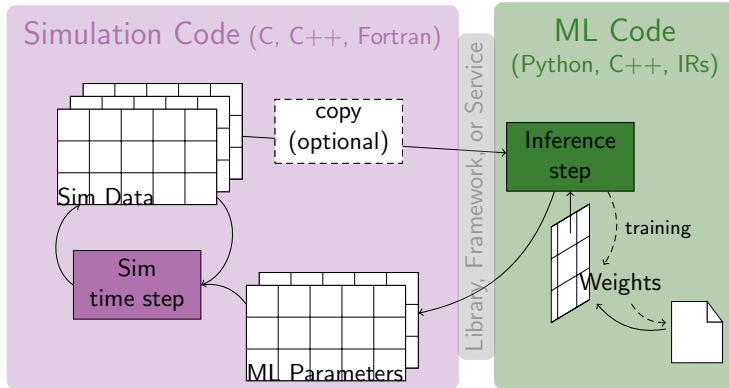


- ▶ range of methods to combine scientific computing with ML
- ▶ focus here: inference for Sim+ML (*)
- ▶ surrogate models tend to be small!



doi.org/10.1145/3773656.3773664

Current Approaches



- ▶ H3-Open-UTIL/MP [1]
- ▶ SmartSim [16]
- ▶ AIXeleratorService [15]

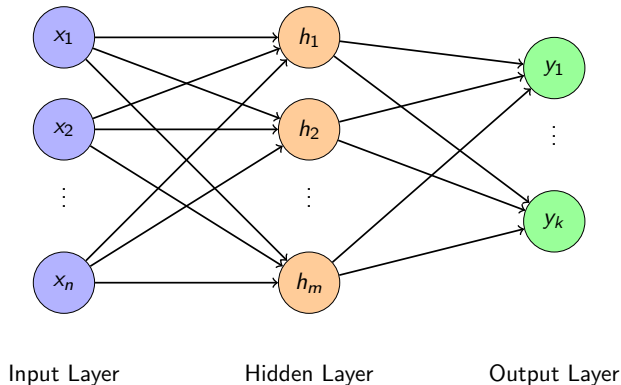
usually based on intermediate representations
(like TorchScript, Tensorflow Graphs, ONNX, ...)

Proposal: *in-situ* can be better!



doi.org/10.1145/3773656.3773664

Most ML Inference is BLAS

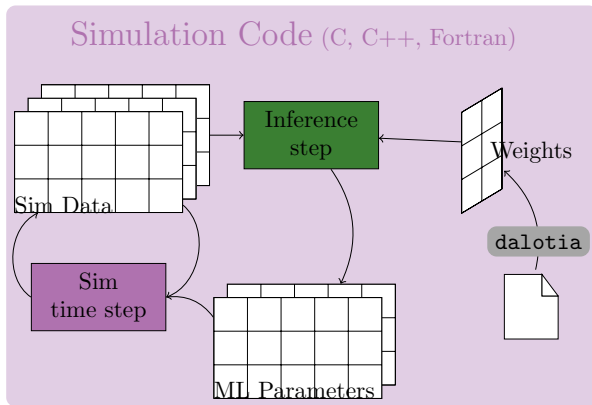


$$\mathbf{H} = f_1(\mathbf{W}_1 \cdot \mathbf{X} + \mathbf{1} \otimes \mathbf{b}_1^T) \quad \mathbf{Y} = f_2(\mathbf{W}_2 \cdot \mathbf{H} + \mathbf{1} \otimes \mathbf{b}_2^T)$$

$\Rightarrow$ GEMM + activation



Proposed Approach: dalotia + Matrix Multiplication



dalotia

lightweight API to load tensors from scientific codes

⇒ then treat inference like the GEMM calls it consists of!



dalotia: 10.1145/3773656.3773664

1. decoupling data loading from inference (no shared dependencies!)
2. avoiding copies of tensor data
3. support for multidimensional Fortran arrays (both fixed-size and allocatable)
4. extensibility towards C++ `std::mdspan`
5. optional interface for C++17 polymorphic memory resources (PMR)
6. simple setup via CMake and Spack ecosystem (incl. `safetensors-cpp`)
7. extensibility (file formats, language bindings, ...)



```
class SubgridLESNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 300)
        self.fc2 = nn.Linear(300, 6)
        # ...initialization...
        # e.g. load from safetensors file

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

```
// [...initialize inputs, allocate output arrays...]
auto dalotia_file = std::unique_ptr<dalotia::TensorFile>(
    dalotia::make_tensor_file("./model.st"));
#pragma omp parallel
{
    // [...allocate thread-local intermediate arrays...]
    for (int i = 0; i < this_thread_num_inputs; ++i)
        for (int j = 0; j < num_hidden_neurons; ++j)
            hidden[i*num_hidden_neurons + j] = bias_1[j];
    cblas_sgemv(CblasColMajor, CblasTrans, CblasNoTrans,
        num_hidden_neurons, this_thread_num_inputs,
        num_input_features, 1.0, weight_1.data(),
        num_input_features, &inputs[
            this_thread_inputs_start_index], num_input_features,
        1.0, hidden.data(), num_hidden_neurons);
    for (auto& value : hidden)
        value = value < 0. ? 0. : value; // ReLU
    // [...repeat for weight_2/bias_2...]
}
```



dalotia-1.0.0-1145/0773656-2773664

```
class SubgridLESNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 300)
        self.fc2 = nn.Linear(300, 6)
        # ...initialization...
        # e.g. load from safetensors file

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

```
type(C_ptr) :: dalotia_file_pointer
real(C_float), allocatable :: weight_1(:,,:), bias_1(:)
! [...other variables...]
dalotia_file = dalotia_open_file("./model.st")
! [...initialize inputs, allocate output arrays...]

!$OMP parallel
do o = 1, this_thread_num_inputs
    fc1_output(:,o) = bias_1(:)
end do
call sgemm('T', 'N', num_hidden_neurons, &
    this_thread_num_inputs, num_input_features, &
    1.0, weight_1, num_input_features, inputs, &
    num_input_features, 1.0, fc1_output, num_hidden_neurons)
fc1_output = max(0.0, fc1_output) ! reLU
! [...repeat for weight_2/bias_2...]
!$OMP end parallel
```



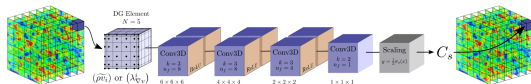
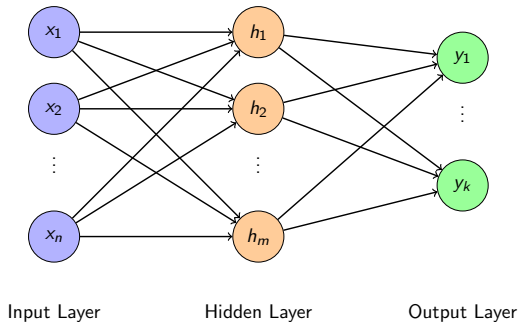
dalotia_repo/10_1145/1773656_2772664

Comparing PyTorch and libTorch
(= TorchScript intermediate representation called from C++)
vs.
BLAS and handcrafted implementations in C++ and Fortran

- ▶ Security
- ▶ Dependency complexity
- ▶ Memory footprint
- ▶ Time-to-solution
- ▶ Energy-to-solution

many more (less measurable) aspects can be important:
flexibility, maintainability, framework lock-in, ...

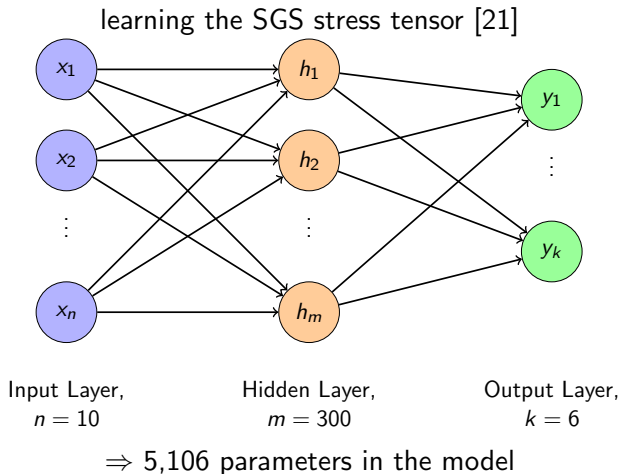




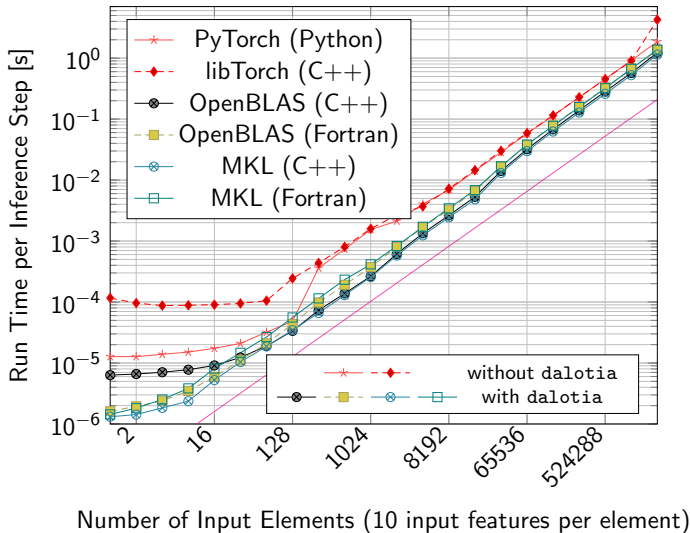
Hardware: Intel Xeon Max 9468 (Sapphire Rapids), 48 cores, HBM

Software: oneAPI toolchain, OpenBLAS, Intel PIN, likwid

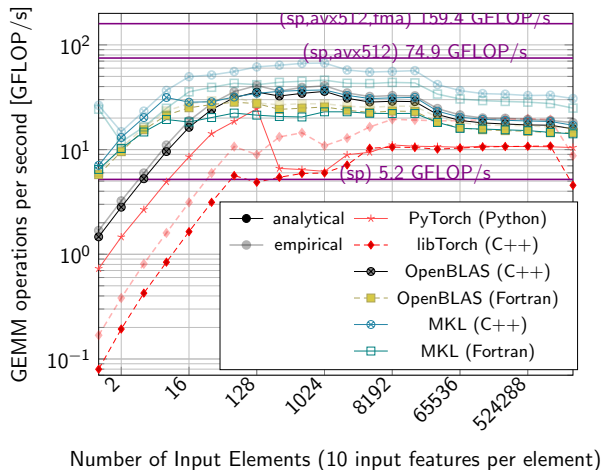




Single-Core Experiment: Fully-Connected Dense Network Inference



Caution: Don't trust performance counters blindly!



analytical FLOP

$$= \sum_{i=1,2} M \cdot N_i \cdot (2K_i - 1)$$

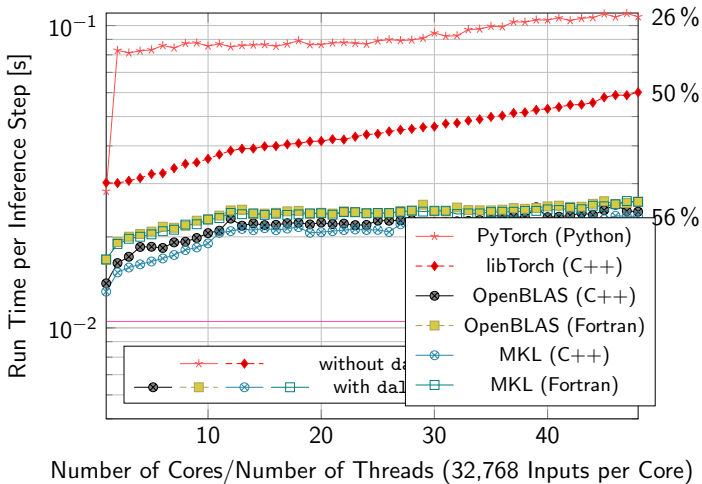
$$= \text{\#inputs} \cdot 9294$$

empirical FLOP/s as reported by
likwid

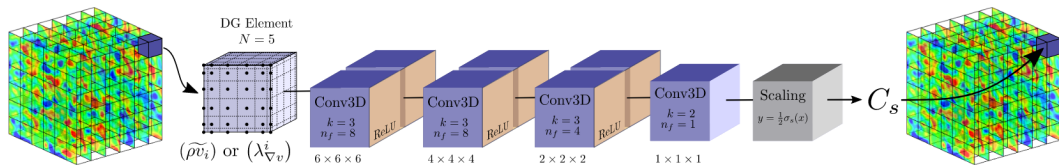


[doi: 10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

Many-Core Experiment: Fully-Connected Dense Network Inference



Reinforcement Learning for a Dynamic Smagorinsky Model [10]



source: [10] 

$\Rightarrow 3,293$ parameters in the model



doi.org/10.1145/3773656.3773664

```
class DeepRLEddyNet(nn.Module):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.conv1 = nn.Conv3d(3, 8, 3, 1)
```

```
        self.conv2 = nn.Conv3d(8, 8, 3, 0)
```

```
        self.conv3 = nn.Conv3d(8, 4, 3, 0)
```

```
        self.conv4 = nn.Conv3d(4, 1, 2, 0)
```

```
        # [...initialization...]
```

```
    def forward(self, x):
```

```
        x = self.conv1(x)
```

```
        x = F.relu(x)
```

```
        # [...layers 2 / 3 same...]
```

```
        x = self.conv4(x)
```

```
        x = torch.flatten(x)
```

```
        output = 0.5 * F.sigmoid(x)
```

```
        return output
```

```
auto conv1_prim_desc = dnnl::
```

```
    convolution_forward::primitive_desc(
```

```
        eng, dnnl::prop_kind::forward_inference,
```

```
        dnnl::algorithm::convolution_auto,
```

```
        adapted_input_md, conv1_adapted_weights_md,
```

```
        conv1_adapted_bias_md, conv1_adapted_output_md,
```

```
        conv_strides, conv_yes_to_padding,
```

```
        conv_yes_to_padding);
```

```
// [...]
```

```
net.push_back(dnnl::convolution_forward(conv1_prim_desc));
```

```
net_args.push_back({{DNNL_ARG_SRC, conv1_src_memory},
```

```
    {DNNL_ARG_WEIGHTS, conv1_weights_memory},
```

```
    {DNNL_ARG_BIAS, conv1_original_bias_memory},
```

```
    {DNNL_ARG_DST, conv1_dst_memory}});
```

```
// [...]
```

```
for (size_t i = 0; i < net.size(); ++i) {
```

```
    net.at(i).execute(s, net_args.at(i));
```

```
}
```

```
s.wait();
```



dal_app(10, 1145/1773656, 2772664)

```
class DeepRLEddyNet(nn.Module):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.conv1 = nn.Conv3d(3, 8, 3, 1)
```

```
        self.conv2 = nn.Conv3d(8, 8, 3, 0)
```

```
        self.conv3 = nn.Conv3d(8, 4, 3, 0)
```

```
        self.conv4 = nn.Conv3d(4, 1, 2, 0)
```

```
        # [...initialization...]
```

```
    def forward(self, x):
```

```
        x = self.conv1(x)
```

```
        x = F.relu(x)
```

```
        # [...layers 2 / 3 same...]
```

```
        x = self.conv4(x)
```

```
        x = torch.flatten(x)
```

```
        output = 0.5 * F.sigmoid(x)
```

```
        return output
```

```
    do o = 1, this_thread_num_inputs
```

```
        ! [... fill with bias...]
```

```
        do l = -1, 1
```

```
            do n = -1, 1
```

```
                do m = -1, 1
```

```
                    do i = 2, 7
```

```
                        do j = 1, 8
```

```
                    do k = 1, 8
```

```
                        do c = 1, size(weight_conv1, 2)
```

```
                            do f = 1, size(weight_conv1, 1)
```

```
                                ! apply 3*3*3 stencil
```

```
                                conv1_output(f, k, j, i, o) = &
```

```
                                    conv1_output(f, k, j, i, o) + &
```

```
                                        weight_conv1(f,c,m,n,l) * &
```

```
                                            conv1_input(c, k+m, j+n, i+l, o)
```

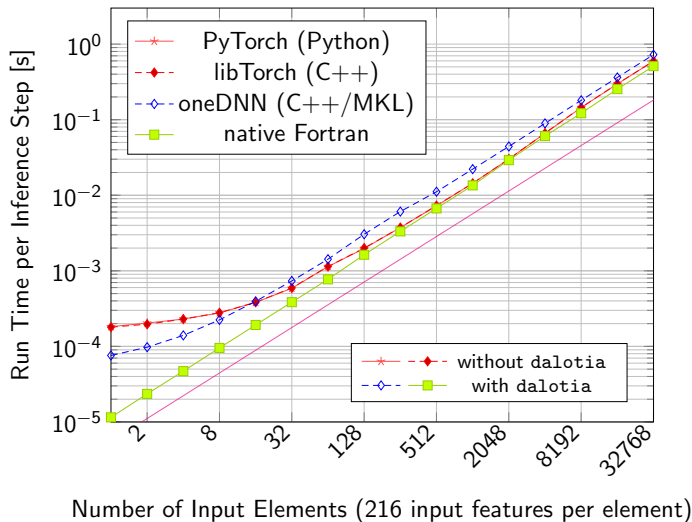
```
                            end do
```

```
                        ! [... end do ...]
```

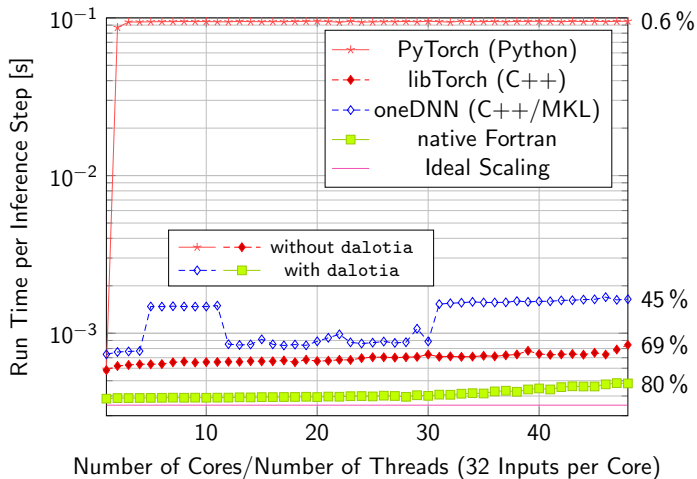


dalotia: app/10, 1145/1773656, 2772664

Single-Core Experiment: Convolutional Neural Network Inference

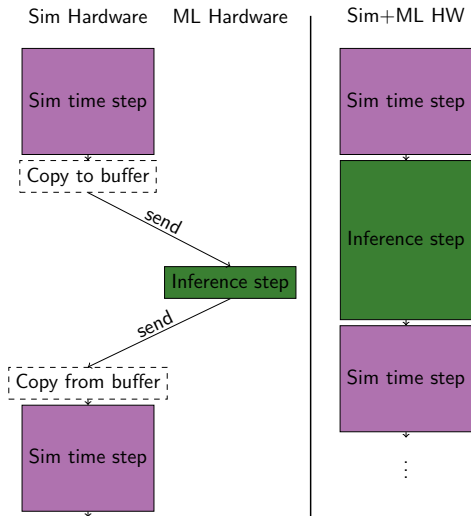


Many-Core Experiment: Convolutional Neural Network Inference



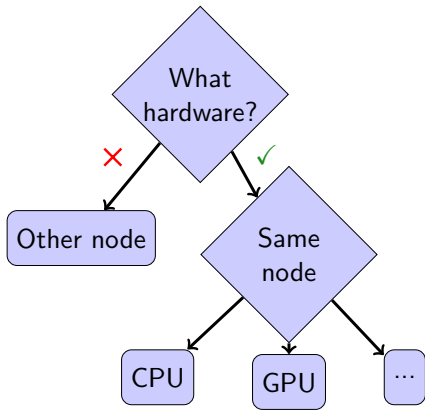
doi: [10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

Extrapolation to Large-Scale Sim+ML on Heterogeneous Architectures

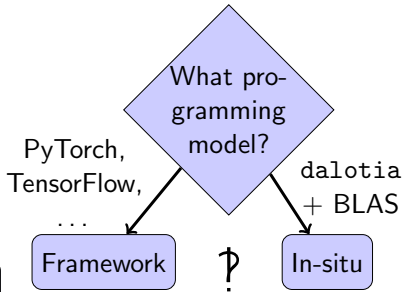


doi.org/10.1145/3773656.3773664

What to do for Sim+ML inference?



Where your data is ✓



If in doubt, abstract!

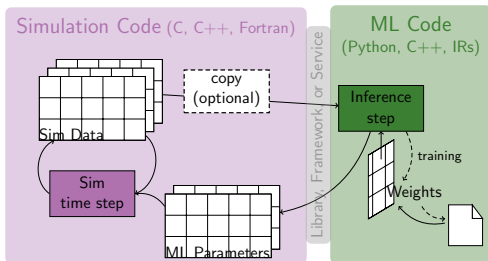


- ▶ `dalotia`: github.com/RIKEN-RCCS/dalotia (Apache 2.0)
- ▶ `experiments`: github.com/RIKEN-RCCS/dalotia_evaluation (Apache 2.0)
- ▶ `measurements`: zenodo.org/records/15129650

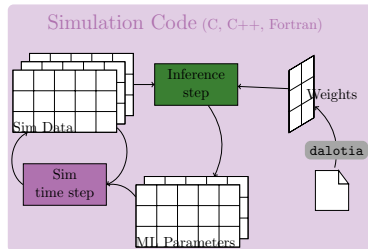


doi.org/10.1145/3773656.3773664

The bottom line



► flexibility ✓



- security ✓
- fewer dependencies ✓
- less memory ✓
- performance $\Leftrightarrow$ efficiency ✓

It's BLAS all the way down!

⇒ With *dalotia*, we have (most of) the tools we need for in-situ ML inference



dalotia: [doi:10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

- ▶ most expressive/standard/library/optimized ways to express tensor operations w/o BLAS?
- ▶ most expressive/standard/library/optimized autodiff / training?
- ▶ cost model: when is GPU offloading warranted? GPU direct loading?
- ▶ useful data formats beyond safetensors?
- ▶ show benefits in real-world applications



- [1] T. Arakawa, H. Yashiro, and K. Nakajima. “Development of a Coupler H3-Open-UTIL/MP”. In: *International Conference on High Performance Computing in Asia-Pacific Region*. HPCAsia '22. New York, NY, USA, Jan. 7, 2022.
- [2] M. Chantry, H. Christensen, P. Dueben, and T. Palmer. “Opportunities and Challenges for Machine Learning in Weather and Climate Modelling”. In: *Philos Trans A Math Phys Eng Sci* (Feb. 15, 2021).
- [3] S. Cheng, C. Quilodr  n-Casas, S. Ouala, A. Farchi, C. Liu, P. Tandeo, R. Fablet, D. Lucor, B. Iooss, J. Brajard, D. Xiao, T. Janjic, W. Ding, Y. Guo, A. Carrassi, M. Bocquet, and R. Arcucci. “Machine Learning With Data Assimilation and Uncertainty Quantification for Dynamical Systems: A Review”. In: *IEEE/CAA Journal of Automatica Sinica* 10.6 (June 2023).



- [4] V. Ferrari, R. Sousa, M. Pereira, J. P. L. De Carvalho, J. N. Amaral, J. Moreira, and G. Araujo. “Advancing Direct Convolution Using Convolution Slicing Optimization and ISA Extensions”. In: *ACM Trans. Archit. Code Optim.* 20.4 (Dec. 14, 2023).
- [5] S. Fujita. *Syoyo/Safetensors-Cpp*. Mar. 26, 2025. URL: <https://github.com/syoyo/safetensors-cpp> (visited on 03/27/2025).
- [6] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. “The Spack Package Manager: Bringing Order to HPC Software Chaos”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Austin, Texas). 2015.



doi: [10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

- [7] Z. Gao, X. Shi, H. Wang, Y. Zhu, Y. (Wang, M. Li, and D.-Y. Yeung. “Earthformer: Exploring Space-Time Transformers for Earth System Forecasting”. In: *Advances in Neural Information Processing Systems* 35 (Dec. 6, 2022). URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/a2affd71d15e8fedffe18d0219f4837a-Abstract-Conference.html (visited on 12/18/2024).
- [8] E. Georganas, K. Banerjee, D. Kalamkar, S. Avancha, A. Venkat, M. Anderson, G. Henry, H. Pabst, and A. Heinecke. “Harnessing Deep Learning via a Single Building Block”. In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). May 2020.



doi: [10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

References IV

- [9] M. Kurz, P. Offenhäuser, and A. Beck. “Deep Reinforcement Learning for Turbulence Modeling in Large Eddy Simulations”. In: *International journal of heat and fluid flow* 99 (2023). URL: <https://www.sciencedirect.com/science/article/pii/S0142727X2200162X> (visited on 09/25/2024).
- [10] M. Kurz, P. Offenhäuser, and A. Beck. “Deep Reinforcement Learning for Turbulence Modeling in Large Eddy Simulations”. In: *International Journal of Heat and Fluid Flow* 99 (Feb. 2023), p. 109094. URL: <http://arxiv.org/abs/2206.11038> (visited on 01/24/2026).
- [11] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia. *GraphCast: Learning Skillful Medium-Range Global Weather Forecasting*. Aug. 2023. Pre-published.



doi: [10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

- [12] J. Li, Z. Feng, Y. Gao, S. Tian, H. Zhang, H. Ye, and J. Zhang. “High-Performance 3D Convolution on the Latest Generation Sunway Processor”. In: *Proceedings of the 53rd International Conference on Parallel Processing*. ICPP '24. Aug. 12, 2024.
- [13] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. “Physics-Informed Neural Networks for High-Speed Flows”. In: *Computer Methods in Applied Mechanics and Engineering* 360 (Mar. 1, 2020).
- [14] T. Nguyen, J. Brandstetter, A. Kapoor, J. K. Gupta, and A. Grover. “ClimaX: A Foundation Model for Weather and Climate”. In: *Proceedings of the 40th International Conference on Machine Learning*. International Conference on Machine Learning. 2023. URL: <https://proceedings.mlr.press/v202/nguyen23a.html> (visited on 03/28/2025).



- [15] F. Orland, L. Nista, N. Kocher, J. Vanvinckenroye, H. Pitsch, and C. Terboven. “Efficient and Scalable Acceleration of Reactive CFD Solvers Coupled with Deep Learning Inference on Heterogeneous Architectures”. In: *Proceedings of the 2025 International Conference on High Performance Computing in Asia-Pacific Region Workshops*. HPC Asia '25 Workshops. 2025.
- [16] S. Partee, M. Ellis, A. Rigazzi, A. E. Shao, S. Bachman, G. Marques, and B. Robbins. “Using Machine Learning at Scale in Numerical Simulations with SmartSim: An Application to Ocean Climate Modeling”. In: *Journal of Computational Science* 62 (July 1, 2022). (Visited on 09/25/2024).
- [17] P. Wang, W. Yang, J. Fang, D. Dong, C. Huang, P. Zhang, T. Tang, and Z. Wang. “Optimizing Direct Convolutions on ARM Multi-Cores”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2023.



doi: [10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

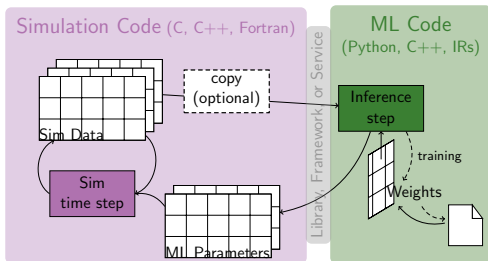
- [18] D. Wu, J. Meng, W. Zhu, M. Deng, X. Wang, T. Luo, M. Wahib, and Y. Wei. “autoGEMM: Pushing the Limits of Irregular Matrix Multiplication on Arm Architectures”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. Atlanta, GA, USA: IEEE Press, Nov. 17, 2024.
- [19] K. Zanki. *Malicious ML Models Discovered on Hugging Face Platform*. ReversingLabs. Feb. 6, 2025. URL: <https://www.reversinglabs.com/blog/rl-identifies-malware-ml-model-hosted-on-hugging-face> (visited on 02/25/2025).
- [20] S. Zhao, Z. Li, B. Fan, Y. Wang, H. Yang, and J. Wang. *LESnets (Large-Eddy Simulation Nets): Physics-informed Neural Operator for Large-Eddy Simulation of Turbulence*. Nov. 7, 2024. Pre-published.



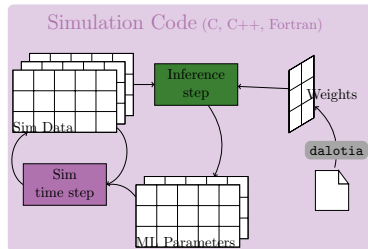
- [21] Z. Zhou, G. He, S. Wang, and G. Jin. “Subgrid-Scale Model for Large-Eddy Simulation of Isotropic Turbulent Flows Using an Artificial Neural Network”. In: *Computers & Fluids* 195 (Dec. 15, 2019).



The bottom line



► flexibility ✓



- security ✓
- fewer dependencies ✓
- less memory ✓
- performance $\Leftrightarrow$ efficiency ✓

It's BLAS all the way down!

⇒ With *dalotia*, we have (most of) the tools we need for in-situ ML inference



dalotia: [doi:10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

software	version / value
oneAPI compilers	icx, ifx: 2025.2.1 20250806
compiler flags	-ffast-math -mavx512f -march=native -mtune=native
PyTorch	2.5.1 (inc. Intel Extension for PyTorch)
oneDNN	3.8.1
Intel MKL	2025.2.1
OpenBLAS	0.3.29
Intel PIN	3.30
likwid	5.4.1
dalotia	1.1.0



[doi: 10.1145/3773656.3773664](https://doi.org/10.1145/3773656.3773664)

Intermediate representations (IRs) like TorchScript, Tensorflow Graphs, ONNX, ...
 are prone to attacks based on arbitrary code execution
 Example: Malicious .pt files on Huggingface platform [19]



doi.org/10.1145/3773656.3773664

Backup Slide: Dependency Complexity

solution	no. of spack dependencies	install volume
Torch	62	1.4 GB
OpenBLAS	2	97 MB
MKL	7	5.1 GB
oneDNN	2	58 MB



doi.org/10.1145/3773656.3773664

Backup Slide: Why the speedup?

My personal hypotheses:

- ▶ no unnecessary data copies
- ▶ no overhead from framework abstractions
- ▶ in Fortran: compiler optimizations
- ▶ first-touch memory allocation in parallel region



doi.org/10.1145/3773656.3773664

- ▶ tall/skinny MM: AutoGEMM [18]
- ▶ direct convolutions: [4, 12, 17]



doi.org/10.1145/3773656.3773664