Winograd Neural Operators

Yash Patel
Department of Statistics
University of Michigan
Ann Arbor, MI 48104
yppatel@umich.edu

Aniket Jivani
Department of Mechanical Engineering
University of Michigan
Ann Arbor, MI 48104
ajivani@umich.edu

Abstract

Numerical simulation of PDEs is notoriously computationally challenging, often requiring significant expert hand-tuning to achieve high-fidelity results in reasonable time frames. Towards this end, recent emphasis has been placed on leveraging machine-learned neural operators to amortize the simulation cost, thereby circumventing such hand-crafted setup. Recent interest has honed in on leveraging such neural operators for robust control in nonlinear dynamical systems, where hand-crafted models of dynamics tend to lack insufficient fidelity to produce high-quality control. However, the deployment of such neural operator-based control to edge devices is infeasible due to the significant computational burden arising from computing the kernel integral operator, both in older graph-based operator methods and more recent Fourier Neural Operators. For this reason, we propose a new method building upon Winograd convolutions to efficiently compute the kernel integral transform on edge devices in settings where regular mesh discretizations are present. In such cases, we demonstrate that our proposed Winograd Neural Operator achieves a speedup of roughly 33% over FFT-based alternatives in PDE benchmark tasks.

1 Introduction

The use of partial differential equations (PDEs) to describe aspects of nature has a long-standing history, extending back to the study of fluid mechanics, heat flow, and plasma physics [1, 2]. Traditionally, PDEs are posited as an appropriate description of a system, a famous example being the Navier-Stokes equations. For instance, in a linear 1D second-order PDE, we assume the evolution of a state $u(x,t) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ can be prescribed by some equation of the form $a_1 u_{xx} + a_2 u_{xt} + a_3 u_{tt} + a_4 u_x + a_5 u_t + a_6 u = f(x,t)$, where we denote $\partial u / \partial x$ by u_x and collect instance-dependent parameters into θ.

"Solving" such PDEs is then defined to mean providing a strategy by which u can be queried for any (x,t,θ) pair of interest from some initial condition (x_0, t_0). The solution of such systems, however, quickly becomes analytically intractable, thus having led to extensive study in the numerical simulation of such systems [3, 4, 5, 6, 7]. Despite significant progress in computing hardware and their extensive use in engineering for high fidelity reconstructions of various phenomena, large scale simulations remain a challenging task. For instance, ensemble weather forecasting can only accomodate a few tens of simulations from different initial conditions to issue probabilistic forecasts of short-term weather events.

Therefore, there remains much interest in accelerating such simulation, increasingly leveraging machine learning systems [8]. In particular, such approaches generally seek to learn a parametric flow map $\mathcal{F}_\varphi(x,t,\theta)$, where simulations are amortized over instance conditions, such that, after training, *any* θ and initial condition (x_0, t_0) can simulated forward arbitrarily [9, 10, 11, 12, 13, 14, 15, 16,

Preprint. Under review.

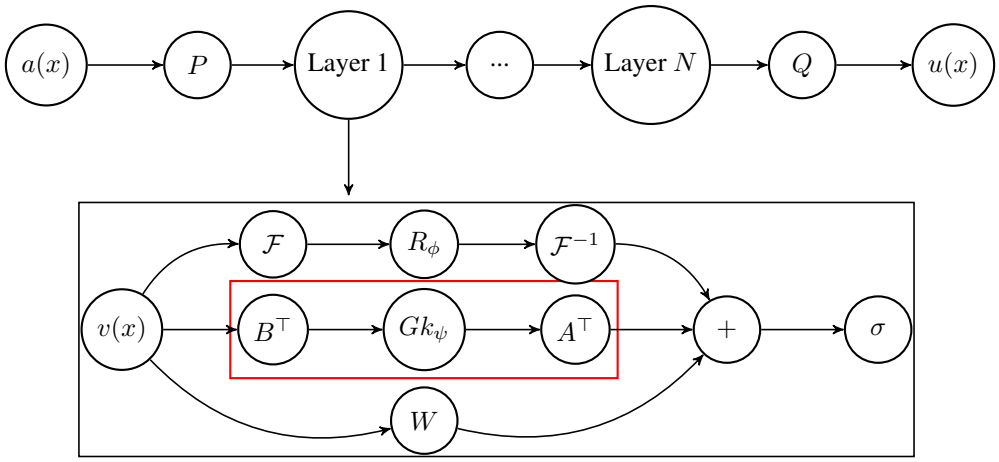

Figure 1: WNO introduces a novel Winograd Neural Operator to enable significant speedup in inference without deterioration in inference accuracy by leveraging the Winograd transform to compute the kernel integral transform of neural operators.

17, 18, 19]. This in turn enables rapid inference for multi-query tasks such as optimization, Bayesian inference and uncertainty quantification.

Given the emergence of such accelerated simulation schemes, there has been recent emerging interest in their application to online control in difficult-to-model nonlinear dynamical systems [20], such as for nuclear reactors [21] and traffic control [22]. Such online dynamical control is especially of interest in edge systems, since the deployment of such robust control oftentimes cannot handle the additional latency encountered in communicating from the edge device to a stationed server. Work on inference acceleration on edge devices has recently become of interest due to the pervasive use of machine learning in robotics and Internet of Things devices [23, 24, 25, 26].

Towards this end, much of the efforts of the edge-ML community are focused on reducing the FLOPs (floating-point operations) required for inference, typically through post-training pruning, student-teacher training, or quantization [27, 28, 29, 30, 31]. Such efforts, however, are maximally effective when crafted into bespoke methods for particular architectures, apparent from the profundity of quantization works aimed at different models [32, 29, 33]. Towards this end, we propose extending the neural operator line of work by introducing a Winograd Neural Operator, which reduces the FLOPs required for inference compared to the common neural operators used in practice.

In particular, neural operators all center their choices surrounding the key kernel integral operator. Prior works focused on exploiting the generality offered by graph neural networks (GNNs) for estimating this operator over arbitrarily discretized spatial domains [34, 35]. To remain computationally tractable, however, such networks impose further sparsification, thereby reducing their resulting accuracies. For this reason, a parallel approach [36] based on the Fast Fourier Transform (FFT) has grown in popularity in the case of having a regular spatial discretization. Such networks, however, typically have a CNN backbone in addition to their components operating in the Fourier domain, limiting their deployability to edge devices. We, therefore, demonstrate Winograd Neural Operator achieves state-of-the-art accuracy on PDEs with regular spatial discretization while greatly improving upon the computational cost of existing techniques. In particular, our contributions are as follows:

- Introducing a novel operator architecture that retains inference accuracy of current operator methods for PDEs with regular spatial gridding while improving inference speeds by $\approx$ 2x.

- Demonstrating the practical utility of the WNO framework across a suite of PDEs.

2 Background

2.1 Neural Operators

For surrogate models, of particular interest is the solution of PDEs with spatial coordinates $x \in \mathbb{R}^d$ and defined over a system with instance-specific parameters $a \in \mathbb{R}^d \to \mathbb{R}$. These may be of the form:

$$(L_a u)(x) = f(x), \quad x \in D \tag{1}$$
$$u(x) = u_0, \quad x \in \partial D \tag{2}$$

Concretely, an example of such a system may be the modeling of the distribution of charge density u over a sheet with spatially varying conductivity a. An ML surrogate model would be trained on various instances of a and u.

Most abstractly, the operator learning problem can be then formulated as seeking to learn a map $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ between two function spaces $\mathcal{A}$ and $\mathcal{U}$, where observations $\mathcal{D} := \{(a_i, u_i)\}$ have been made i.e. $\mathcal{G}$ inverts the action of L_a in Equation 1. We assume that there exists some true operator $\mathcal{G}^\dagger$ such that $u = \mathcal{G}^\dagger(a)$. While many different learning-based approaches have been proposed to solve this learning problem, they all can be abstractly framed as seeking to recover this true map, formally

$$\min_{\mathcal{G}} ||\mathcal{G} - \mathcal{G}^\dagger||^2_{\mathcal{L}^2(\mathcal{A},\mathcal{U})} = \int_{\mathcal{A}} ||\mathcal{G}(a) - \mathcal{G}^\dagger(a)||^2_{\mathcal{U}} da. \tag{3}$$

In practice, however, full observation of functions is not possible and are instead characterized by their evaluation on a discretized spatial grid. In particular, dataset observations u_i are typically obtained by running finite-element and finite-difference (FEM) solvers over input conditions a_i. These methods discretize the spatial dimension into a finite set of points $\{x_i\}$, called "meshing," with which spatial derivatives are approximated. High-fidelity PDE simulation often requires significant insight to appropriately perform such discretization, with special care required for spatial meshing, evidenced by the increasing interest in producing grid-free methods for PDE simulation [37, 38, 39, 40]. We, therefore, seek to learn the operator by minimizing Equation (3) over its discretization:

$$\varphi^* = \arg\min_{\phi} \sum_{i=1}^{N} \sum_{k=1}^{K} ||\mathcal{G}_\varphi(a_i)(x_k) - u_i(x_k)||_2^2. \tag{4}$$

As mentioned, many approaches to amortized ODE/PDE solution have been proposed, from leveraging CNNs to more recent discretization-free DeepONets and neural operators [41, 42]. The fully convolutional architecture used in [43] requires train and test solution instances to be restricted to a fixed mesh size, which means their error grows when tested on higher-fidelity simulations downsampled to the same grid, without retraining to accomodate different fidelities. By contrast, neural operators aim to act on any discretization of the input fucntion, can be evaluated on any point of the output domain, and converge to a continuum operator as the discretization is refined.

Neural operators seek to learn a parameterized representation of a kernel integral operator. That is, the initial state is lifted into a higher dimensional representation $v : \mathcal{X} \to \mathbb{R}^{d_v}$, which is then evolved over the time discretization of interest, namely from $t = 0, ..., T-1$. Such an evolution consists of

$$v_{t+1} = \sigma(W v_t(x) + (\mathcal{K}_\phi(\theta) v_t)(x)) \text{ where } (\mathcal{K}_\phi(\theta) v_t)(x) := \int_D k_\phi(x, y, \theta(x), \theta(y)) v_t(y) dy, \tag{5}$$

where k_ϕ is the learned kernel of interest. The learning of such a k_ϕ is universal amongst operator methods, as it serves as the natural analog of a linear layer for functional data analysis. Methods in this space, therefore, divide in how this kernel integral operator is computed. Certain approaches, for example, approximate this integral directly with a numerical discretization, namely assuming the kernel has a small bounded support $\mathcal{B}_r(x)$ around each x:

$$\int_D k_\phi(x, y, \theta(x), \theta(y)) v_t(y) dy \approx \frac{1}{|\mathcal{N}(x)|} \sum_{y_i \in \mathcal{N}(x)} k_\phi(x, y_i, \theta(x), \theta(y_i)) v_t(y_i) dy. \tag{6}$$

In the case of regular gridding over the spatial domain, this sum can be efficiently estimated using a standard convolution, as done in the FCN network of [44]. If spatial domains are irregular, extensions

such as the Graph Neural Operator (GNO) [34] proposed leveraging graph neural networks, where the discretization is naturally taken to define the GNN topology, from which Equation (6) can be computed via standard aggregation operations. Such a method was similarly applied in an extension to Multipole Graph Neural Operators (MGNO) in [35]. While such methods offer expressivity and are able to handle nonhomogeneous spatial discretizations, they suffer from the high computational cost in directly approximating the aforementioned integral.

An alternative to the aforementioned approaches that attempts to mitigate this computational burden is to make structural assumptions to simplify the evaluation of Equation (5). For instance, under the assumption the kernel is translation invariant and has no dependence on θ, the kernel integral operator simplifies to a convolution with the unknown kernel $k_\phi(x - y)$. To avoid directly estimating this convolution, one line of work reparameterized the kernel under a Fourier transform [36, 45]. That is, they proposed learning k_ϕ instead in Fourier space, as follows

$$(\mathcal{K}_\phi(\theta)v_t)(x) = \mathcal{F}^{-1}\left(R_\phi \cdot (\mathcal{F}v_t)\right)(x), \tag{7}$$

where $\mathcal{F}$ is the Fourier transform and R_ϕ the learned Fourier-transformed kernel. Alternative methods have also been proposed to learn the kernel directly in real space, such as in [46], where nonlinear activations are specifically constructed to avoid aliasing artifacts.

2.2 Winograd Convolutions

Given the prevalence of computer vision in robotics, there is little surprise in the research interest placed in the optimization of CNNs to fit in restricted compute and energy budgets. Towards this end, efforts have demonstrated that convolutions are the most demanding component of CNNs [47, 48]. Such a finding, in turn, has compelled significant work towards their optimization. One such line of work proposed performing convolutions in the Fourier domain rather than in the measured space, paralleling the optimization exploited by the Fourier Neural Operator.

In addition to this line of work focusing on optimizing the computation of convolutions with mathematical reformulation, there is an orthogonal line of work concentrating on specifically reframing the computation to lend itself better to particular aspects of the hardware. In particular, a well-known property of hardware is the greater computational resources required to compute a multiply than an addition operation. Such a property is especially important in settings of limited compute or energy budget, where any savings can make the difference of whether or not a model can be deployed. This, in turn, has led to adjustments of architectures to replace multiply operations with addition operations, occasionally with the seeming increase in total operation count, specifically due to this near-universal property. One particular instance of this is in the optimization of the convolution operations in CNNs, where a method known as Winograd convolutions have been demonstrated to achieve greater computational performance than standard alternatives [49, 50, 51].

The Winograd algorithm is a general strategy for accelerating the computation of the product of two polynomials. That is, for two polynomials $p(x)$ and $q(x)$, certain problems can be reframed to seeking the evaluation at some point of interest x_0, i.e. $p(x_0)q(x_0)$. Naive implementation of this operation would likely proceed through the separate computation of each factor followed by a product or computation of the coefficients of the resulting polynomial product $p(x)q(x)$ and subsequently evaluating the result at x_0. The Winograd approach follows the latter path but additionally reframes the problem of seeking the coefficients as solving a system of linear equations, where the equations are formed by evaluating $p(x)q(x)$ at a series of points $\{x_i\}$ that can be efficiently computed, often taken to be values such as $0, \pm 1$, and ∞.

This framing as a product of polynomials has been leveraged for a diverse array of problems, including the multiplication of large integers. Of interest herein is its application to the computation of discrete convolution, where it was demonstrated to be minimal in the number of multiplies required to perform this computation [52]. In particular, if we assume some f is being convolved with a discretized kernel $k \in \mathbb{R}^r$ with the result in $\mathbb{R}^m$, the Winograd convolution takes on a form resembling the structure of the FFT convolution approach, although the transform space differs:

$$f * k = A^\top[(Gk) \cdot (B^\top f)], \tag{8}$$

where A, G, and B are matrices that are known for particular tuples of (m, r). Such matrices are often denoted by the general Winograd parameterization $F(m, r)$. In practice, however, such matrices are only defined for sufficiently small m, r. As a result, Winograd convolutions in practice act on

subregions of the matrices encountered. That is, if $f * g \in \mathbb{R}^d$ for $d \gg 3$, the computation can be decomposed into $\lceil d/m \rceil$ separate convolutions, with each computed in the aforementioned manner and subsequently concatenated together. For this reason, Winograd convolutions tend to be leveraged only in cases where kernel sizes are small, generally for $k = 3$ or $k = 5$.

3 Method

We now discuss the modification of current neural operator architectures required to enable using Winograd convolutions, specifically focusing on modifications to the base architecture in Section 3.1 and corresponding implementation details in Section 3.2. Subsequent demonstration of empirical results of the proposed methods is provided in Section 4.

3.1 WNO: Model Architecture

At its core, the Winograd Neural Operator simply computes the kernel convolutions during inference in the alternate approach offered by Winograd. As mentioned, practical implementations of FNOs, such as in [10], generally augment the core Fourier domain operation by one in the real domain, resulting in a layer

$$v_{t+1} = \sigma(W v_t + k_\psi * v_t + \mathcal{F}^{-1}\left(R_\phi \cdot (\mathcal{F} v_t)\right)), \tag{9}$$

where k_ψ corresponds to the real-space kernel and R_ϕ the Fourier-domain kernel. In the WNO, we propose computing this real-space convolution via a Winograd operation, resulting in this final form:

$$v_{t+1} = \sigma(W v_t + A^\top[(G k_\psi) \cdot (B^\top v_t)] + \mathcal{F}^{-1}\left(R_\phi \cdot (\mathcal{F} v_t)\right)), \tag{10}$$

where A, G, and B are again pre-known matrices chosen by the spatial discretization of v_t and kernel dimension. Notably, while our primary point of interest lies in edge deployment and hence primarily in accelerating *inference*, this computational reframing can also be leveraged for accelerating training passes. The full architectural structure is shown in Figure 1.

Importantly, while we focus on the Fourier Neural Operator in this exposition due to its popularity compared to alternative approaches in the case of homogeneous gridding, the approach discussed herein can be leveraged in any case where convolutions form the backbone of a neural operator, such as in Convolutional Neural Operators [46]. In such cases, the speedup demonstrated in Section 4 would likely be even more pronounced due to the increased presence of convolutions in the architecture.

3.2 WNO: Implementation Details

Implementation of the WNO was specifically performed for PyTorch compiled with a CUDA backend [53]. In this case, PyTorch implementations of `nn.Conv2d` dispatch to Nvidia's CUDA Deep Neural Network (cuDNN) library, which provides accelerated implementations of low-level primitives, such as pooling, softmax, normalization, matrix multiplies, and convolutions. We, therefore, are specifically interested in the implementation of convolutions, provided by `cudnnConvolutionForward`.

This method takes as input pointers to buffers of the input and filter data and a selection of what low-level convolution should be executed, where the former buffers are implicitly allocated by PyTorch as in invoking the typical call `tensor.to("cuda")`. For the latter, there are a number of methods available, specifically `GEMM,FFT,WINOGRAD`. Each of these methods has several optimized variants that leverage tiling and fusion, which we additionally consider in Section 4, yet the core part of the algorithm remains unchanged in such cases. Importantly, unlike other forms of optimization, such as pruning or quantization, swapping out the convolution method retains the same numerical values in the resulting computation, so there is no loss of accuracy, as we verify in Section 4.

The `GEMM` algorithm transforms the computation of a convolution into a corresponding matrix multiply problem. `FFT` and `WINOGRAD` are implemented as previously described. Most libraries relying on cuDNN, including PyTorch, generally dispatch using the `FASTEST` option due to its generality and convenience, which dynamically determines the appropriate algorithm to use at runtime based on the matrix size and structure. Despite the offered convenience, however, such dynamic dispatch inevitably results in runtime overhead, as we empirically demonstrate in Section 4.

4 Experiments

We now study the efficiency of WNO empirically across several PDE benchmark tasks from [10]. All implementations build off of the baseline of the FNO provided in their package. As discussed in the preceding section, we specifically consider the following variations of convolution dispatch, with full descriptions of each available in the CUDA documentation:

- `CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM`
- `CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM`
- `CUDNN_CONVOLUTION_FWD_ALGO_GEMM`
- `CUDNN_CONVOLUTION_FWD_ALGO_FFT`
- `CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING`
- `CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD`
- `CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED`
- `FASTEST`

Over the following sections, we specifically consider the following tasks: 2D Darcy Flow, 2D Shallow Water, and 2D Diffusion Reaction. Brief descriptions of the tasks are provided in the following subsections and code is available at https://github.com/yashpatel5400/mcufno. All experiments were conducted on a single Nvidia RTX 2080 Ti GPU.

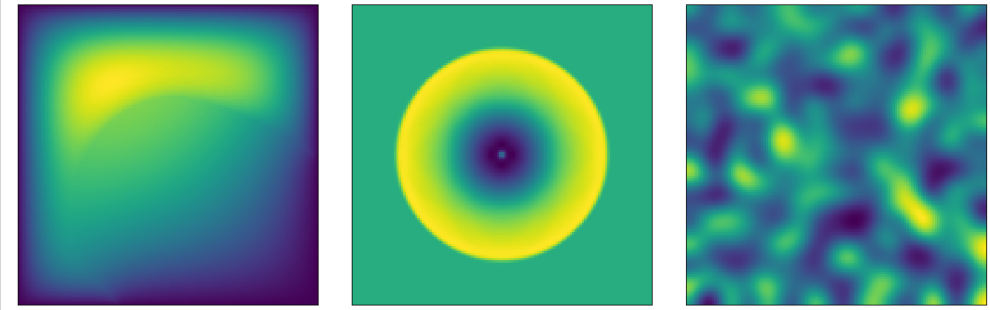

Figure 2: Examples of 2D Darcy Flow, Shallow Water and Diffusion-Reaction problems from benchmark datasets generated using [10]

4.1 2D Darcy Flow

We consider the 2D Darcy Flow problem over the unit square, governed by the following equations:

$$\begin{aligned} -\nabla \cdot (a(x)u(x)) &= f(x) \quad x \in (0,1)^2 \\ u(x) &= 0 \quad x \in \partial(0,1)^2, \end{aligned} \tag{11}$$

where $a : (0,1)^2 \to \mathbb{R}_+$ is the diffusion coefficient, $u : (0,1)^2 \to \mathbb{R}^2$ is the flow, and $f : (0,1)^2 \to \mathbb{R}^2$ is the forcing function. We, therefore, are interested in learning a map $\mathcal{G} : a \to u$.

4.2 2D Shallow Water

We consider the 2D Shallow Water problem, which is commonly employed to model free-surface flow problems. We specifically consider the spatial domain $\Omega = [-2.5, 2.5]^2$, governed by:

$$\begin{aligned} -\partial_t h + \partial_x hu + \partial_y hv &= 0 \\ \partial_t hu + \partial_x \left(u^2 h + \frac{1}{2} g_r h^2 \right) &= -g_r h \partial_x b \\ \partial_t hv + \partial_y \left(v^2 h + \frac{1}{2} g_r h^2 \right) &= -g_r h \partial_y b, \end{aligned} \tag{12}$$

where $u, v \in \mathbb{R}$ are respectively the velocities in the x and y directions, $h : \mathbb{R}^2 \to \mathbb{R}$ is the water depth, and $b : \mathbb{R}^2 \to \mathbb{R}$ is the bathymetry. We are interested here in learning a map $\mathcal{G} : b \to h$.

4.3 2D Diffusion Reaction

We also consider the 2D Diffusion-Reaction problem over the unit square, governed by:

$$\partial_t u = D_u \partial_{xx} u + D_u \partial_{yy} u + R_u \quad \partial_t v = D_v \partial_{xx} v + D_v \partial_{yy} v + R_v, \tag{13}$$

where $D_u, D_v \in \mathbb{R}$ are respectively the diffusion coefficients for the activator and inhibitor, $R_u, R_v : \mathbb{R}^2 \to \mathbb{R}$ respectively the activator and inhibitor reaction functions, and $u : \mathbb{R}^2 \to \mathbb{R}^2$ the spatially varying flow. We are interested in learning a map $\mathcal{G} : R \to u$, where $R = (R_u, R_v) : \mathbb{R}^2 \to \mathbb{R}^2$. The specific activator and inhibitor reaction functions are as follows:

$$R_u(u, v) = u - u^3 - k - v \tag{14}$$

$$R_v(u, v) = u - v. \tag{15}$$

4.4 PDE Benchmark Results

We now present the results over the aforementioned tasks. Notably, the `WINOGRAD` and `WINOGRAD_NONFUSED` algorithms are only available for small kernels. In practice, FNOs generally use kernels of such sizes, lending them naturally to this implementation. We report the results of the timings for kernels of size 3×3 in Table 1. In all cases, differences from the baseline models were confirmed to be 0 to within machine precision.

Table 1: Inference times across tasks are shown below, where times were assessed over minibatches of 50 i.i.d. test samples. Average time of inference for a single sample in the minibatch is reported, with standard deviations in parentheses.

	Darcy Flow	Shallow Water	Diffusion Reaction
`IMPLICIT_GEMM`	0.0063 (0.00184)	0.0079 (0.0019)	0.00962 (0.0019)
`IMPLICIT_PRECOMP_GEMM`	0.00644 (0.00068)	0.00774 (0.00066)	0.01033 (0.00076)
`GEMM`	0.00778 (0.00014)	0.00901 (0.00017)	0.01088 (0.0004)
`FFT`	0.02164 (0.00051)	0.02261 (0.00106)	0.0238 (0.00053)
`FFT_TILING`	0.00904 (0.00012)	0.00987 (0.00028)	0.01155 (0.00028)
`WINOGRAD`	**0.00611 (0.00011)**	**0.00698 (0.0001)**	**0.00898 (0.00019)**
`WINOGRAD_NONFUSED`	0.0128 (0.00018)	0.01385 (0.00016)	0.01574 (0.00047)
`FASTEST`	0.04992 (0.00042)	0.05196 (0.00083)	0.05513 (0.00252)

We, therefore, see that the `WINOGRAD` convolution has consistently lower execution times across tasks compared to alternate implementations. Notably, the `FASTEST` is also consistently the slowest algorithm. While seemingly surprising, this is largely due to the experimental setup, since each trial is run from scratch. As a result, the dominant cost is in the logic of ascertaining the optimal algorithm. In the standard use case, such a determination is typically made once and reused across calls to amortize the cost. Nonetheless, this demonstrates that, if the hardware of deployment is fixed and its architecture specifics known, significant improvements in efficiency can be made through the preselection of a convolution algorithm.

5 Discussion

We have presented WNO, a framework for accelerating inference of neural operators towards the end of streamlining their deployment to edge devices. This work suggests many directions for extension. In particular, much work in the application of Winograd convolutions has focused on its performance post quantization, since the Winograd algorithm is known to exhibit numerical instability [54, 55, 56]. Towards this end, work to demonstrate the numerical stability of quantized Winograd operators would be of great interest. Similarly, as discussed in the main text, the focus for this study was on inference; however, a similar extension to `cudnnConvolutionBackwardData` should enable the acceleration of backward passes and, therefore, perhaps lead to similar performance gains in the training of neural operators. Further, empirical testing herein was limited to synthetic dynamical systems: application of this framework to large-scale engineering systems is of great practical interest. Finally, while not the focus of this work, neural operators can also potentially be adapted to construct efficient methods for computer vision tasks, as has been done for convolutional architectures [57].

References

[1] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

[2] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

[3] Leighton Wilson, Robert Krasny, and Tyler Luchko. Accelerating the 3d reference interaction site model theory of molecular solvation with treecode summation and cut-offs. *Journal of Computational Chemistry*, 43(18):1251–1270, 2022.

[4] Mathias Winkel, Robert Speck, Helge Hübner, Lukas Arnold, Rolf Krause, and Paul Gibbon. A massively parallel, multi-disciplinary barnes–hut tree code for extreme-scale n-body simulations. *Computer physics communications*, 183(4):880–889, 2012.

[5] TD Arber, Keith Bennett, CS Brady, A Lawrence-Douglas, MG Ramsay, Nathan John Sircombe, Paddy Gillies, RG Evans, Holger Schmitz, AR Bell, et al. Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Physics and Controlled Fusion*, 57(11):113001, 2015.

[6] David Tskhakaya, Konstantin Matyash, Ralf Schneider, and Francesco Taccogna. The particle-in-cell method. *Contributions to Plasma Physics*, 47(8-9):563–594, 2007.

[7] Jianguo Ning, Ziyan Jin, and Xiangzhao Xu. A multigrid partition coupled eulerian–lagrangian method for fluid–solid interaction problems. *Physics of Fluids*, 35(9), 2023.

[8] Michael Brenner. Machine learning for partial differential equations. In *APS March Meeting Abstracts*, volume 2021, pages P61–003, 2021.

[9] Rishikesh Ranade, Chris Hill, Lalit Ghule, and Jay Pathak. A composable machine-learning approach for steady-state simulations on high-resolution grids. *Advances in Neural Information Processing Systems*, 35:17386–17401, 2022.

[10] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[11] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS workshop*, volume 2, 2020.

[12] Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations. *arXiv preprint arXiv:2010.00072*, 2020.

[13] Rishikesh Ranade, Chris Hill, and Jay Pathak. Discretizationnet: A machine-learning based solver for navier–stokes equations using finite volume discretization. *Computer Methods in Applied Mechanics and Engineering*, 378:113722, 2021.

[14] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[15] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[16] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. Climax: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

[17] Tung Nguyen, Jason Jewik, Hritik Bansal, Prakhar Sharma, and Aditya Grover. Climate-learn: Benchmarking machine learning for weather and climate modeling. *arXiv preprint arXiv:2307.01909*, 2023.

[18] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[19] Chaopeng Shen, Alison P Appling, Pierre Gentine, Toshiyuki Bandai, Hoshin Gupta, Alexandre Tartakovsky, Marco Baity-Jesi, Fabrizio Fenicia, Daniel Kifer, Li Li, et al. Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment*, 4(8):552–567, 2023.

[20] Luke Bhan, Yuanyuan Shi, and Miroslav Krstic. Operator learning for nonlinear adaptive control. In *Learning for Dynamics and Control Conference*, pages 346–357. PMLR, 2023.

[21] Kazuma Kobayashi and Syed Bahauddin Alam. Deep neural operator-driven real-time inference to enable digital twin solutions for nuclear energy systems. *Scientific reports*, 14(1):2101, 2024.

[22] Yihuai Zhang, Ruiguo Zhong, and Huan Yu. Neural operators for boundary stabilization of stop-and-go traffic. *arXiv preprint arXiv:2312.10374*, 2023.

[23] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1595–1623, 2022.

[24] Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. A comprehensive survey on tinyml. *IEEE Access*, 2023.

[25] Visal Rajapakse, Ishan Karunanayake, and Nadeem Ahmed. Intelligence at the extreme edge: A survey on reformable tinyml. *ACM Computing Surveys*, 55(13s):1–30, 2023.

[26] Ahmed I Awad, Mostafa M Fouda, Marwa M Khashaba, Ehab R Mohamed, and Khalid M Hosny. Utilization of mobile edge computing on the internet of medical things: A survey. *ICT Express*, 2022.

[27] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[28] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

[29] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[30] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International conference on machine learning*, pages 2498–2507. PMLR, 2017.

[31] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[32] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[33] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[34] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[35] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[36] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[37] Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. Grid-free monte carlo for pdes with spatially varying coefficients. *ACM Transactions on Graphics (TOG)*, 41(4):1–17, 2022.

[38] Rohan Sawhney, Bailey Miller, Ioannis Gkioulekas, and Keenan Crane. Walk on stars: A grid-free monte carlo method for pdes with neumann boundary conditions. *arXiv preprint arXiv:2302.11815*, 2023.

[39] Ekrem Fatih Yılmazer, Delio Vicini, and Wenzel Jakob. Solving inverse pde problems using grid-free monte carlo estimators. *arXiv preprint arXiv:2208.02114*, 2022.

[40] Auguste De Lambilly, Gabriel Benedetti, Nour Rizk, Chen Hanqi, Siyuan Huang, Junnan Qiu, David Louapre, Raphael Granier De Cassagnac, and Damien Rohmer. Heat simulation on meshless crafted-made shapes. In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games*, pages 1–7, 2023.

[41] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[42] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[43] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, August 2018.

[44] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[45] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. *arXiv preprint arXiv:2306.03838*, 2023.

[46] Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, and Emmanuel de Bezenac. Convolutional neural operators. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[47] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[48] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[49] Athanasios Xygkis, Lazaros Papadopoulos, David Moloney, Dimitrios Soudris, and Sofiane Yous. Efficient winograd-based convolution kernel implementation on edge devices. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[50] Lingchuan Meng and John Brothers. Efficient winograd convolution via integer arithmetic. *arXiv preprint arXiv:1901.01965*, 2019.

[51] Syed Asad Alam, Andrew Anderson, Barbara Barabasz, and David Gregg. Winograd convolution for deep neural networks: Efficient point selection. *ACM Transactions on Embedded Computing Systems*, 21(6):1–28, 2022.

[52] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[54] Vladimir Chikin and Vladimir Kryzhanovskiy. Channel balancing for accurate quantization of winograd convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12507–12516, 2022.

[55] Chen Tianqi, Weixiang Xu, Weihan Chen, Peisong Wang, and Jian Cheng. Towards efficient and accurate winograd convolution via full quantization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[56] Javier Fernandez-Marques, Paul Whatmough, Andrew Mundy, and Matthew Mattina. Searching for winograd-aware quantized networks. *Proceedings of Machine Learning and Systems*, 2:14–29, 2020.

[57] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, April 2020.