

---

# GeoMaNO: Geometric Mamba Neural Operator for Partial Differential Equations

---

**Xi Han\***

Department of Computer Science  
Stony Brook University  
Stony Brook, NY 11794, USA  
xihan1@cs.stonybrook.edu

**Jingwei Zhang\***

Department of Computer Science  
Stony Brook University  
Stony Brook, NY 11794, USA  
jzhang@cs.stonybrook.edu

**Dimitris Samaras**

Department of Computer Science  
Stony Brook University  
Stony Brook, NY 11794, USA  
samaras@cs.stonybrook.edu

**Fei Hou**

Institute of Software  
Chinese Academy of Sciences  
Beijing, 100190, China  
houfei@ios.ac.cn

**Hong Qin**

Department of Computer Science  
Stony Brook University  
Stony Brook, NY 11794, USA  
qin@cs.stonybrook.edu

## Abstract

The neural operator (NO) framework has emerged as a powerful tool for solving partial differential equations (PDEs). Recent NOs are dominated by the Transformer architecture, which offers NOs the capability to capture long-range dependencies in PDE dynamics. However, existing Transformer-based NOs suffer from quadratic complexity, lack geometric rigor, and thus suffer from sub-optimal performance on regular grids. As a remedy, we propose the Geometric Mamba Neural Operator (GeoMaNO) framework, which empowers NOs with Mamba’s modeling capability, linear complexity, plus geometric rigor. We evaluate GeoMaNO’s performance on multiple standard and popularly employed PDE benchmarks, spanning from Darcy flow problems to Navier-Stokes problems. GeoMaNO improves existing baselines in solution operator approximation by as much as 58.9%.

## 1 Introduction

**Background and Major Challenges** Partial differential equations (PDEs) are quintessential to various computational problems in science, engineering, as well as relevant applications in simulation, modeling, and scientific computing. Neural operators (NOs) have emerged as a powerful paradigm for learning mappings between function spaces. By learning the solution operator to PDEs, NOs offer a novel data-driven alternative to classical numerical solvers.

Recent advances in the field of NOs have been propelled by the Transformer architecture [1] for its modeling capability in capturing long-range dependencies and physical relations. However, Transformer-based NOs suffer from quadratic computational complexity inherent to the self-attention

---

\*These authors contributed equally to this paper.

mechanism. Their architecture also lacks geometric rigor, and struggles when applied to large-scale, high-dimensional PDEs on regular grids. These shortcomings incur prohibitive computational costs, significantly degrading the inference accuracy and efficiency [2, 3].

On the other hand, the Mamba architecture [4, 5], benefiting from Transformer’s parallelism and modeling capability, yet with a linear time complexity, has emerged as a promising avenue for neural operator instantiation. Research work like [6, 7] tried to integrate Mamba with NOs, yet, these methods still lack geometric rigor on regular grids, and suffer from sub-optimal performance. Mamba’s potential in scientific computation remains largely unexplored.

**Motivation and Method Overview** As a remedy to the challenges detailed above, we propose our novel **Geometric Mamba Neural Operator (GeoMaNO)** framework, which is illustrated in Fig. 1. GeoMaNO manifests a geometric-aware architecture specifically tailored for regular grids, combined with our GeoMamba-SSM module with geometric rigor and mathematical insights. GeoMamba-SSM employs a geometric correction component to the SSM formulation, effectively dampening duplicate hidden states introduced by multi-way cross-scans [8, 9]. For 1D and 3D problems, GeoMamba-SSM employs a one-dimensional geometry-aware state-space representation; to further enhance its geometric rigor, for 2D problems, GeoMamba-SSM employs a two-dimensional state-space representation, as inspired by [10].

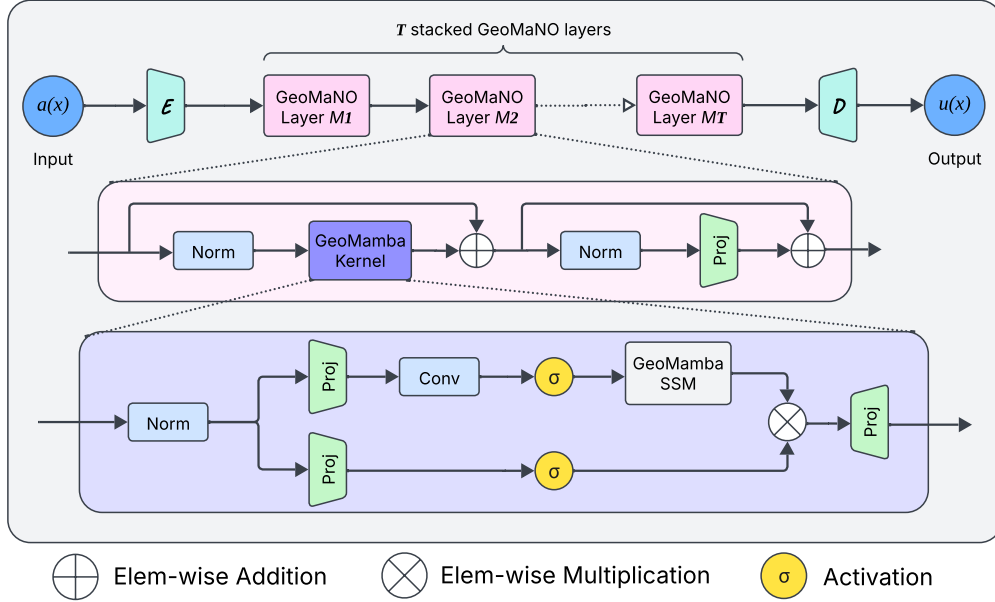


Figure 1: **Overview of the novel GeoMaNO architecture.** (1) The input function  $a(x)$ <sup>2</sup> is lifted and patchified to a higher-dimensional latent representation by the geometry-aware encoder layer  $\mathcal{E}$ ; (2)  $T$  stacked GeoMaNO layers  $M_1 \cdots M_T$  (middle) performs the kernel integral via GeoMamba kernels (bottom); (3) Each GeoMamba kernel performs geometry-aware Mamba scans, combined with skip-connections and non-linear activations; and (4) The last GeoMaNO layer’s output is transformed back to the physical domain with the decoder layer  $\mathcal{D}$ . This yields the output function  $u(x)$ .

**Key Contributions** The salient contributions of this paper comprise: (1) **Theoretical insights:** We propose a novel GeoMamba-SSM formulation, which manifests a geometrically rigorous state-space representation. It preserves the geometry of the input domain, and employs a learnable geometric correction component to dampen duplicate hidden states arising from multi-way cross-scans [8, 9]. (2) **The novel GeoMaNO architecture:** We propose a novel family of neural operators, GeoMaNO, which utilizes GeoMamba-SSM’s geometric rigor to approximate PDE solution operators, offering users high efficiency, high accuracy, high scalability, and high generalization power; and (3) **Superior performance:** We evaluate GeoMaNO’s performance on the standard PDE solving

<sup>2</sup>E.g., for the Darcy flow problem  $\nabla \cdot (f(x)\nabla u(x)) = g(x)$ ,  $a(x)$  encodes  $f$  and  $g$ .

benchmark [11], which is popularly employed by NO frameworks. GeoMaNO improves the SOTA by 58.9% (accuracy) and 29.2% (efficiency) on the Navier-Stokes benchmark, and 36.8% (accuracy) and 49.8% (efficiency) on the Darcy flow benchmark.

## 2 Related Work

**Neural Operators** The NO framework learns mappings between function spaces. The foundational work, DeepONet [12], is grounded in the universal approximation theorem for operators. Concurrently, FNO and its variants [13–18] approximate integration with linear projection in the Fourier domain. GNO and its variants [19–21] utilize graph neural networks (GNNs) as the base network, adapting neural operators to finite elements and complex mesh domains. The Transformer [1] architecture, as a crucial milestone of deep learning, is also employed in the NO framework [22, 23, 3, 24, 25]. To address the quadratic complexity of the attention mechanism, GNOT [2] and ONO [22] utilize the well-established linear Transformers [26–28]. As another alternative, work like [29, 6] attempted to employ the Mamba [4, 5] architecture as an alternative to Transformers in the NO framework.

**Neural PDE Solvers with Geometric Rigor** Many neural PDE solvers incorporate geometric rigor and deliver superior performance on regular grids. Greenfeld et al. [30] proposed to learn the prolongation operators of the geometric multigrid. UGrid [31] proposed a learnable multigrid solver with a convergence guarantee, and adapts to arbitrary Dirichlet boundaries. For NOs, however, although the foundational work FNO [13] was proposed for regular grids, its successors focused on extensions to other domains, leaving regular grids largely unexplored. Most existing NO frameworks, both Transformer-based and Mamba-based, lack geometric rigor and mathematical insights, and fail to deliver optimal performance on regular grids.

**State-Space Model (SSM) and Mamba** SSM [32–35] is a mathematical framework representing time-variant systems by defining hidden states and their transitions. Mamba [4] introduced a selective mechanism and a hardware-aware implementation to SSMs, delivering a modeling capability in par with Transformers yet only with a linear complexity. When applied to the vision domain, Mamba-based methods employ a multiway cross-scan pattern [36, 37, 8]. However, this pattern lacks mathematical rigor, involves duplicate components in the hidden state, and decreases performance. Moreover, existing Mamba-based methods model one-dimensional sequences, requiring two-dimensional input to be flattened before being processed. This results in a loss of geometric information and also leads to sub-optimal performance. As a remedy, 2DMamba [10] manifests a two-dimensional state space representation, which performs well on the 2D domain for vision tasks. However, its potential in scientific computation remains unexplored. In summary, geometric rigor is an unexplored topic for Mamba-based NOs.

## 3 Mathematical Preliminaries

**Neural Operators** The primary objective of a NO is to learn a mapping between two infinite-dimensional function spaces via a finite array of observed input-output pairs observed via numerical simulation, real-world applications, etc. This problem can be mathematically formulated as follows [13]:

We suppose that  $\Omega \subset \mathbb{R}^d$  is an open and bounded  $d$ -dimensional domain. We then let  $\mathcal{A} = \mathcal{A}(\Omega; \mathbb{R}^{d_a})$  denote a Banach space of vector-valued functions mapping  $\Omega \rightarrow \mathbb{R}^{d_a}$ , and  $\mathcal{U} = \mathcal{U}(\Omega; \mathbb{R}^{d_u})$  denote another Banach space of vector-valued functions  $\Omega \rightarrow \mathbb{R}^{d_u}$ . We further let  $\mathcal{U}^*$  be the dual space of  $\mathcal{U}$ . We then consider a generic family of PDEs in the form of:

$$\begin{cases} (\mathcal{D}_a u)(x) = f(x), & x \in \Omega \\ u(x) = 0, & x \in \partial\Omega \end{cases} \quad (1)$$

Here,  $a \in \mathcal{A}$  is a parameter;  $\mathcal{D}_a : \mathcal{A} \rightarrow \mathcal{D}(\mathcal{U}; \mathcal{U}^*)$  is a mapping from the parameter Banach space  $\mathcal{A}$  to the space of linear differential operators  $\mathcal{D}$  in the form of  $\mathcal{U} \rightarrow \mathcal{U}^*$ ; and  $f \in \mathcal{U}^*$  is the right-hand side. Eq. 1’s solution  $u : \Omega \rightarrow \mathbb{R}$  is a scalar-valued function living in the Banach space  $\mathcal{U}$ . We further let  $G^\dagger := \mathcal{D}_a^{-1} f : \mathcal{A} \rightarrow \mathcal{U}$  be an *solution operator* mapping the parameter  $a$  to the solution  $u: a \mapsto u$ . We assume that we have  $N$  i.i.d. observations pairs  $\{(a_j, u_j)\}_{j=1}^N$ , where  $a_j$  is drawn

from a probability measure  $\mu$  supported on  $\mathcal{A}$ , and  $u_j = G^\dagger(a_j) + \varepsilon_j$  is potentially corrupted with noise  $\varepsilon_j$ . In the training process, we aim to construct an approximation of  $G^\dagger$  by constructing a parametric mapping  $G : \mathcal{A} \times \mathbb{R}^p \rightarrow \mathcal{U}$ , or equivalently,  $G_\theta : \mathcal{A} \rightarrow \mathcal{U}$ ,  $\theta \in \mathbb{R}^p$ , by choosing the optimal parameter  $\theta^\dagger \in \mathbb{R}^p$  such that  $G(\cdot, \theta^\dagger) = G_{\theta^\dagger} \approx G^\dagger$ . This formulation is equivalent to an optimization problem in infinite-dimensional spaces, with the cost functional  $L : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$  defined as  $\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{a \sim \mu} [L(G_\theta(a), G^\dagger(a))]$ .

**State-Space Models and Mamba** An SSM manifests a time-evolving linear system that maps a one-dimensional input sequence  $\mathbf{x}(t)$  to an output response sequence  $\mathbf{y}(t)$  recurrently through  $N$  mutually-independent hidden states  $\mathbf{h}_s(t)$ ,  $s = 1, 2, \dots, N$ . For clarity, in this paper, we refer to the  $t$  dimension as “*state*” dimension, and  $s$  as “*dstate*” dimension. Mamba [4, 5] introduces a selective mechanism to allow the SSMs to dynamically adapt to the input context. Mathematically, the continuous-form of the Mamba-SSM is formulated as Eq. 2:

$$\begin{cases} \mathbf{h}'_s(t) = \mathbf{A}_s(t)\mathbf{h}_s(t) + \mathbf{B}_s(\mathbf{x}(t))\mathbf{x}(t), \\ \mathbf{y}(t) = \sum_{s=1}^N \mathbf{C}_s(\mathbf{x}(t))\mathbf{h}_s(t). \end{cases} \quad (2)$$

In Eq. 2,  $\mathbf{A}_s(t)$ ,  $\mathbf{B}_s(\mathbf{x}(t))$ , and  $\mathbf{C}_s(\mathbf{x}(t))$  are learnable model parameters, with  $\mathbf{B}_s$  and  $\mathbf{C}_s$  being input-dependent. Following the zero-order hold (ZOH) rule, Eq. 2 could be discretized into Eq. 3. In Eq. 3,  $\overline{\mathbf{A}}_s[t] = \exp(\Delta[t]\mathbf{A}_s(t))$ ,  $\overline{\mathbf{B}}_s[t] = \Delta_t \mathbf{B}_s(\mathbf{x}(t))$ ,  $\overline{\mathbf{C}}_s[t] = \mathbf{C}_s(\mathbf{x}(t))$ , and  $\Delta[t] = \text{softplus}(\Delta(\mathbf{x}(t))) = \ln(1 + \exp(\Delta(\mathbf{x}(t))))$ , where  $[t]$  denotes the discrete value at time step  $t$ ,  $\ln(\cdot)$  denotes element-wise natural logarithm, and  $\Delta(\mathbf{x}(t))$ ,  $\mathbf{B}_s(\mathbf{x}(t))$ , and  $\mathbf{C}_s(\mathbf{x}(t))$  are linear functions of the input  $\mathbf{x}(t)$ .  $\overline{\mathbf{A}}_s$ ,  $\overline{\mathbf{B}}_s$ , and  $\overline{\mathbf{C}}_s$  jointly constitute the learnable parameters of the Mamba-SSM, whereas  $\Delta[t]$  represents the time step granularity of the input-adaptive discretization process.

The correctness of the NO framework itself, as well as that of employing Mamba in NOs, is a well-studied problem with mathematically-rigorous proofs [6]. And thus, these are not our major concerns. For the theoretical completeness of this paper, we will discuss these mathematical details in Sec. B.

**2DMamba** In contrast to the vanilla Mamba, 2DMamba [10] manifests a two-dimensional hidden state, aggregating not only semantic, but also geometric information directly from a 2D domain. 2DMamba’s coefficients remain the same as 1D, with the subscript being  $[i, j]$  to index discretized 2D inputs instead of  $[t]$ , and is formulated as follows:

$$\begin{cases} \mathbf{g}_s[i, j] = \overline{\mathbf{A}}_s[i, j] \mathbf{g}[i, j-1] + \overline{\mathbf{B}}_s[i, j] \mathbf{x}[i, j], \\ \mathbf{h}_s[i, j] = \overline{\mathbf{A}}_s[i, j] \mathbf{h}[i-1, j] + \mathbf{g}_s[i, j], \\ \mathbf{y}[i, j] = \sum_{s=1}^N \overline{\mathbf{C}}_s[i, j] \mathbf{h}_s[i, j]. \end{cases} \quad (4)$$

On a 2D domain, 2DMamba first aggregates a horizontal hidden state  $\mathbf{g}$ , which is then aggregated vertically into the final hidden state  $\mathbf{h}$ . 2DMamba is highly efficient due to its hardware-aware design. Yet, it is only applied to the vision domain, and its potential in scientific computation remains unexplored.

## 4 Novel Method

To improve neural operators on their accuracy, efficiency, and enhance their geometric rigor, we present our novel GeoMaNO framework. GeoMaNO manifests a geometric-aware architecture tailored for regular grids. As illustrated in Fig. 1, GeoMaNO constructs the following iterative mapping from input  $a$  to solution  $u$ :

$$a \mapsto \underbrace{v_0 \mapsto v_1 \mapsto \dots \mapsto v_T}_{\text{encode } T \text{ GeoMaNO layers}} \mapsto \underbrace{u}_{\text{decode}}, \quad (5)$$

which is conducted by our geometry-aware encoder layer  $\mathcal{E}$  (Sec. 4.1), our GeoMaNO layers  $M_{1 \dots T}$  (Sec. 4.2), and our decoder layer  $\mathcal{D}$  (Sec. 4.1). Depending on the dimensionality of the underlying

physical domain<sup>3</sup>, our geometry-aware encoder and the GeoMaNO layers will behave differently, with the details to be discussed in the corresponding subsections.

#### 4.1 Geometry-Aware Encoder and Decoder

**Geometry-Aware Encoder Layer** The encoder layer seamlessly combines the legacy lifting operator [38] with the patch embedding operator [39]. The lifting operation is conducted by a feed-forward network [40], which lifts a lower-dimensional input to a higher-dimensional latent representation. This effectively maps a potentially non-linear physical-domain input into a linear representation that resides in a higher-dimensional potential space [41]. Because we are focusing on regular grids, we further adopt the patch embedding [39] technique, which is extensively employed in the vision domain [42]. Mathematically, the geometry-aware encoding layer  $\mathcal{E}$  could be formulated as:

$$\mathbf{Y} = \text{Linear}(\mathbf{X}), \quad \mathbf{W} = \text{Softmax}(\text{Linear}(\mathbf{Y})), \quad \mathbf{Z} = \text{LayerNorm}(\mathbf{W}^\top \mathbf{Y}), \quad (6)$$

where  $\text{Linear}(\cdot)$  denotes a standard biased feed-forward network with activation,  $\text{Softmax}(\cdot)$  denotes feature-wise softmax, and  $\text{LayerNorm}(\cdot)$  denotes the layer normalization [43]. Here,  $\mathbf{X} \in \mathbb{R}^{P \times D_i}$  denotes the physical-domain input with  $P$  grid points, and  $D_i$  is the depth of the input<sup>4</sup>.  $\mathbf{X}$  is first lifted to  $\mathbf{Y} \in \mathbb{R}^{P \times D}$ , where  $D \gg D_i$ , then embedded into  $L$  latent patches, with  $P \ll L$ . Inspired by Mamba’s [4, 5] input-adaptive design, our encoding process is conducted with an input-adaptive set of parameters  $\mathbf{W} \in \mathbb{R}^{P \times L}$ ,  $\mathbf{W} = \mathbf{W}(\mathbf{X})$ .

For PDEs on 1D or 3D physical domains, we adopt the regular patch embedding pipeline, which embeds positional encodings into the latent patches. However, for PDEs on 2D domains, we do *not* embed the positional encodings. In both cases, the input is *not* flattened, and the geometric structure is maintained for postprocessing. This geometry-aware patch embedding operation preserves the structured geometry and the spatial relationship between latent patches for the succeeding GeoMaNO layers.

**Decoder Layer** The decoder layer  $\mathcal{D}$  can be viewed as the inverse of the encoding layer  $\mathcal{E}$ . It converts the latent patches back to the physical domain. Mathematically,  $\mathcal{D}$  could be formulated as:

$$\mathbf{W} = \text{Softmax}(\text{Linear}(\mathbf{Z})), \quad \mathbf{Y} = \mathbf{W}^\top \mathbf{Z}, \quad \mathbf{X} = \text{LayerNorm}(\text{Linear}(\mathbf{Y})), \quad (7)$$

where the latent patches  $\mathbf{Z} \in \mathbb{R}^{L \times D}$  are projected back to the physical grid  $\mathbf{X} \in \mathbb{R}^P$ .

#### 4.2 GeoMaNO Layer

As illustrated in Fig. 1 (middle), our GeoMaNO layer parallels the legacy self-attention [1] block, but we use pre-norms instead of post-norms to preserve the skip-connection gradients:

$$\begin{aligned} \mathbf{S}_t &= \mathbf{Z}_{t-1} + \text{GeoMambaKernel}(\text{LayerNorm}(\mathbf{Z}_{t-1})), \\ \mathbf{Z}_t &= \mathbf{S}_t + \text{Linear}(\text{LayerNorm}(\mathbf{S}_t)). \end{aligned} \quad (8)$$

**GeoMamba Kernel** Following the common practice employed by the vision domain [8, 9], we adopt a *four-way cross-scan pattern* over the input, which has been proven effective on regular grids. Given a 2D input grid (for 3D input domains, we treat the time dimension as an additional feature dimension), we traverse the input in four orders: (1) top-left to bottom-right (row-major), (2) bottom-right to top-left (row-major), (3) top-left to bottom-right (column-major), and (4) bottom-right to top-left (column-major). The traversal results are stacked at a new dimension, as illustrated in Fig. 2 (left half).

The GeoMamba-SSM regresses each scan direction independently, and the results for the four scan directions are merged. Our novel GeoMamba kernel could be formulated as Algorithm 1, where  $B$  denotes the batch size,  $L$  denotes the size of the input patch grid,  $ED$  denotes the feature embedding dimension,  $N$  denotes the Mamba state dimension,  $\sigma$  denotes element-wise activation, and  $\circ$  denotes element-wise multiplication. In Algorithm 1, lines 3-6 correspond to the left half of Fig. 2, whereas lines 8-9 correspond to the right half of Fig. 2.

<sup>3</sup>E.g., the Navier-Stokes problem manifests a three-dimensional physical domain  $(x, y, t)$ , whereas the Darcy flow problem manifests a two-dimensional physical domain  $(x, y)$ .

<sup>4</sup>E.g., for the Darcy flow problem  $\nabla \cdot (f(x, y)\nabla u(x, y)) = g(x, y)$ ,  $\mathbf{X}$  encodes the coefficients  $f$  and the right-hand side  $g$ . For this example, we have  $D_i = 2$ .

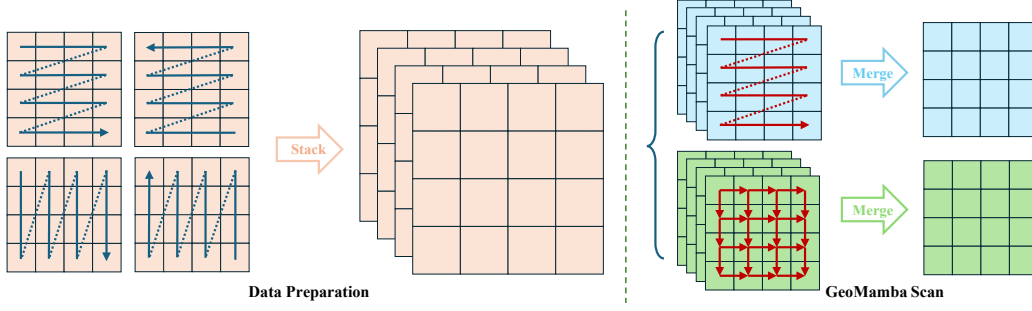


Figure 2: The four-way cross-scan pattern. **Left:** In the data preparation stage, the input grid is traversed in four orders, and the traversal results are stacked at an additional order dimension. **Right:** For each slice along the order dimension, we process it with our GeoMamba-SSM module (shown in blue color). As a special case, for 2D PDEs, in GeoMamba-SSM, we employ a 2D SSM representation with geometric rigor (shown in green color). The results for all four cross-scan directions are merged.

---

#### Algorithm 1 GeoMamba kernel

---

**Require:** Input patch grid  $\mathbf{Z} : (B, L, ED)$ ;  
**Require:** GeoMamba-SSM scanning coefficients  $\mathbf{As} : (N, ED)$ ,  $\mathbf{Rs} : (N, ED)$ ;  
**Ensure:** Output patch grid  $\mathbf{Y} : (B, L, ED)$ .

- 1:  $\mathbf{Z} = \text{Linear}(\mathbf{Z})$ ; ▷ Input projection.
- 2:  $\mathbf{X} = \sigma(\text{Conv}(\mathbf{Z}))$ ;
- 3: **for** Scan directions  $d = 1, 2, 3, 4$  **do** ▷ Cross-scan data preparation.
- 4:    $\mathbf{X}_d = \text{Traverse}_d(\mathbf{X})$ ;
- 5: **end for**
- 6:  $\mathbf{Xs} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4)$ ;
- 7:  $\Delta\mathbf{s}, \mathbf{Bs}, \mathbf{Cs} = \text{Linear}(\mathbf{Xs})$ ;
- 8:  $\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3, \mathbf{Y}_4 = \text{GeoMamba-SSM}(\mathbf{Xs}, \Delta\mathbf{s}, \mathbf{As}, \mathbf{Bs}, \mathbf{Cs}, \mathbf{Rs})$ ; ▷ Parallel cross-scan.
- 9:  $\mathbf{Y} = \text{CrossMerge}(\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3, \mathbf{Y}_4)$ ;
- 10:  $\mathbf{Y} = \sigma(\text{LayerNorm}(\mathbf{Y})) \circ \sigma(\mathbf{Z})$ ;
- 11:  $\mathbf{Y} = \text{Linear}(\mathbf{Y})$ . ▷ Output projection.

---

### 4.3 GeoMamba-SSM Module

**Theoretical Insights** Although proven effective on structured domains like regular grids, the four-way cross-scan pattern lacks geometric rigor: When the results from different scan orders are merged, it brings *duplicate latent information* into the hidden states. Consider a simplified form of Eq. 3:  $h_t = Ah_{t-1} + Bx_t$  (with  $A$  and  $B$  be constants for simplicity). When this recursion is applied to latent tokens on a regular grid (left), the hidden states (along the four scan directions) for  $x_4$  are listed as follows (right):

$$\begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \quad \begin{aligned} h_4^{(1)} &= A^4 Bx_0 + A^3 Bx_1 + A^2 Bx_2 + ABx_3 + \textcolor{red}{Bx_4}, \\ h_4^{(2)} &= A^4 Bx_8 + A^3 Bx_7 + A^2 Bx_6 + ABx_5 + \textcolor{red}{Bx_4}, \\ h_4^{(3)} &= A^4 Bx_0 + A^3 Bx_3 + A^2 Bx_6 + ABx_1 + \textcolor{red}{Bx_4}, \\ h_4^{(4)} &= A^4 Bx_8 + A^3 Bx_5 + A^2 Bx_2 + ABx_7 + \textcolor{red}{Bx_4}, \end{aligned}$$

where the superscripts  $(1) \sim (4)$  separately denote the four cross-scan directions. When the four scan directions are merged, the hidden state  $\textcolor{red}{Bx_4}$  is duplicated four times. In practice, although  $A$  and  $B$  vary for scan directions and are input-dependent, the SSM still regresses over duplicate information. These duplicates pose an excessive optimization burden to the training process, and as shown in Table 3, will degrade the performance.

**GeoMamba-SSM** As a remedy to this challenge, we propose our novel GeoMamba-SSM formulation, which could be formulated as:

$$\begin{cases} \mathbf{h}_s^{(d)}[t] = \overline{\mathbf{A}}_s^{(d)}[t] \mathbf{h}_s^{(d)}[t-1] + \overline{\mathbf{B}}_s^{(d)}[t] \mathbf{x}^{(d)}[t], \\ \mathbf{y}^{(d)}[t] = \sum_{s=1}^N \left( \mathbf{C}_s^{(d)}[t] \mathbf{h}_s^{(d)}[t] - \overline{\mathbf{R}}_s^{(d)}[t] \overline{\mathbf{B}}_s^{(d)}[t] \mathbf{x}^{(d)}[t] \right), \\ \mathbf{y}[t] = \text{CrossMerge}(\mathbf{y}^{(d)}[t]). \end{cases} \quad (9)$$

To dampen the duplicate hidden states, GeoMamba-SSM introduces a **geometric correction component**, which is highlighted in blue color. The correction component manifests a configurable coefficient  $\overline{\mathbf{R}}_s$ , which could be either fixed or learnable. With coefficients fixed, the correction component is enforced as a hard constraint; otherwise, it will be a soft constraint.

**Remark 4.1 (Mamba’s D Coefficients).** Some Mamba-SSM variants allow an extra coefficient  $\mathbf{D} = (D_1, \dots, D_N) \in \mathbb{R}^N$ , which will rewrite the second line of Eq. 3 into  $\mathbf{y}[t] = \sum_{s=1}^N (\mathbf{C}_s[t] \mathbf{h}_s[t] + D_s \mathbf{x}[t])$ . However, per *dstate* dimension  $s$ ,  $D_s$  is a scalar constant across the entire grid, and this residual offers no input-adaptive control over  $B$ . In contrast, GeoMamba-SSM introduces input-adaptive residuals explicitly on  $Bx$ : see Eq. 9 for the difference. The  $\mathbf{D}$  coefficients are **already integrated** into our GeoMamba-SSMs. As shown in Table 3, compared with  $\mathbf{D}$ -only versions, GeoMaNO performs better when it also has  $\overline{\mathbf{R}}_s$  enabled.

**Remark 4.2 (Design Choices for Fixed Geometric Correction Coefficients).** We employ a scan-order-wise constant implementation for fixed correction coefficients. For each scan direction, we set  $\overline{\mathbf{R}}_s$  to either 0 or 1 for the entire grid. Since  $Bx$  is duplicated four times, it would be beneficial to dampen one, two, or three copies of  $Bx$ . We ablate  $\overline{\mathbf{R}}_s$  with three configurations: 0001, 0011, or 0111, where the four digits correspond to the four scan directions. Results are available in Table 3.

**2DGeoMamba-SSM** The vanilla Mamba-SSM employs a general-purpose one-dimensional state-space representation, which, in principle, generalizes to arbitrary dimensions and domains. However, being generic indicates the vanilla Mamba-SSM may not perform optimally under domain-specific settings, e.g., 2D problems on regular grids. As illustrated in Fig. 3 (middle), 1DGeoMamba-SSM leads to **geometric inconsistency** in the hidden states. To further enhance GeoMamba-SSM’s geometric rigor, for 2D problems, we specialize it with a two-dimensional state-space representation, as inspired by [10]. Our 2DGeoMamba-SSM is formulated as:

$$\begin{cases} \mathbf{g}_s^{(d)}[i, j] = \overline{\mathbf{A}}_s^{(d)}[i, j] \mathbf{g}_s^{(d)}[i, j-1] + \overline{\mathbf{B}}_s^{(d)}[i, j] \mathbf{x}^{(d)}[i, j], \\ \mathbf{h}_s^{(d)}[i, j] = \overline{\mathbf{A}}_s^{(d)}[i, j] \mathbf{h}_s^{(d)}[i-1, j] + \mathbf{g}_s^{(d)}[i, j], \\ \mathbf{y}^{(d)}[i, j] = \sum_{s=1}^N \left( \overline{\mathbf{C}}_s^{(d)}[i, j] \mathbf{h}_s^{(d)}[i, j] - \overline{\mathbf{R}}_s^{(d)}[i, j] \overline{\mathbf{B}}_s^{(d)}[i, j] \mathbf{x}^{(d)}[i, j] \right), \\ \mathbf{y}[i, j] = \text{CrossMerge}(\mathbf{y}^{(d)}[i, j]), \end{cases} \quad (10)$$

where the notations remain the same as Eq. 9 and Eq. 4. 2DGeoMamba-SSM conducts a **two-dimensional cross-scan over the input grid**, as illustrated in Fig. 3 (right). 2DGeoMamba-SSM preserves the neighboring relations in the hidden states, and as shown in Table 3 (a), improves the performance. More mathematical insights will be discussed in Sec. C.

## 5 Experiments

### 5.1 System Setup

**Benchmarks and Baselines** We evaluate GeoMaNO’s performance on regular grids with the standard neural operator PDE solving benchmarks [11]. We use the Darcy flow benchmark, which is a two-dimensional linear elliptic problem, and the Navier-Stokes benchmark, which is a three-dimensional (including the time dimension) non-linear parabolic/hyperbolic problem. We compare GeoMaNO with 14 baselines, with more details available in Sec. A.

**Implementation Details** We follow the standard configuration employed by Transolver [3]. For the main results, we set the model depth  $T$  to 8 layers, and the feature embedding dimension  $ED$  to 64

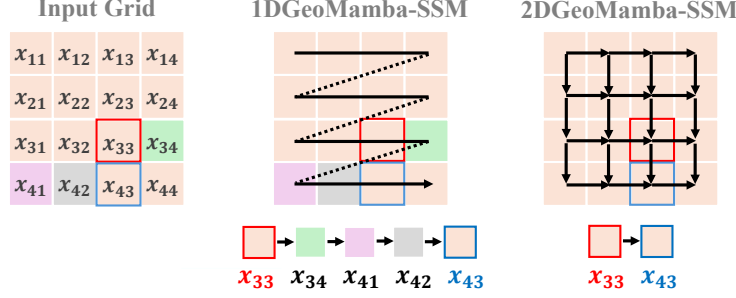


Figure 3: Illustration of our GeoMamba-SSM variants. **Left:** A sample  $4 \times 4$  input grid. **Middle:** 1DGeoMamba-SSM flattens the input into a 1D sequence and regresses the hidden states over it. However, after flattening, vertically-adjacent patches  $x_{33}$  (highlighted with red margins) and  $x_{43}$  (highlighted with blue margins) are no longer adjacent. This leads to *geometric inconsistency* in the hidden states. **Right:** 2DGeoMamba-SSM regresses over the input grid in a 2D manner. Adjacent patches remain adjacent in the hidden states, properly preserving geometric information.

for Darcy flow and 256 for Navier-Stokes. Regarding the GeoMaNO layers, we employ the vanilla GeoMamba-SSM for Navier-Stokes and 2DGeoMamba-SSM for Darcy flow. Our experiments are conducted on one NVIDIA Quadro RTX 8000 GPU, and repeated five times for consistency. More details will be discussed in Sec. A.

## 5.2 Main Results

**Standard Benchmarks** We report GeoMaNO’s performance on standard benchmarks [11]. As shown in Table 1, GeoMaNO achieves consistent SOTA performance across both benchmarks, covering both linear elliptic problems and non-linear parabolic/hyperbolic problems. GeoMaNO surpasses the current SOTA, Transolver [3], by as much as 58.9% on the Navier-Stokes benchmark and 36.8% on the Darcy flow benchmark. These results highlight GeoMaNO’s superior performance in PDE solution operator approximation.

Table 1: Performance comparison on standard benchmarks. We report the relative L2 error (the smaller, the better). The best result is highlighted in **bold**, whereas the second best is highlighted *underlined italic*. For clarity, we also report our relative performance gain  $(1 - \frac{\text{our error}}{\text{2nd-best error}})\%$ .

Model	Relative L2 ( $\downarrow$ )	
	N.-S.	Darcy Flow
FNO [13]	0.1556	0.0108
WMT [44]	0.1541	0.0082
U-FNO [14]	0.2231	0.0183
Geo-FNO [45]	0.1556	0.0108
U-NO [46]	0.1713	0.0113
F-FNO [18]	0.2322	0.0077
LSM [47]	0.1535	0.0065
-	-	-
-	-	-

Model	Relative L2 ( $\downarrow$ )	
	N.-S.	Darcy Flow
Galerkin [28]	0.1401	0.0084
HT-Net [25]	0.1847	0.0079
OFormer [48]	0.1705	0.0124
GNOT [2]	0.1380	0.0105
FactFormer [49]	0.1214	0.0109
ONO [22]	0.1195	0.0076
Transolver [3]	<i>0.0900</i>	<i>0.0057</i>
<b>GeoMaNO (Ours)</b>	<b>0.0370</b>	<b>0.0036</b>
Performance Gain	58.9%	36.8%

**Efficiency** We evaluate GeoMaNO’s efficiency against the current SOTA, Transolver [3]. In our evaluation, we employ model configurations that achieve the results in Table 1. We report the training time  $T_{\text{train}}$  (seconds per epoch), inference time  $T_{\text{infer}}$  (seconds per epoch), and the model’s GPU memory consumption (MB). For all metrics, the smaller, the better. The epoch size is kept consistent between both models. The results are available in Table 2. GeoMaNO consistently outperforms Transolver [3] over all metrics and on both benchmarks, notably by 49.8% for Darcy flow inference. The results highlight GeoMaNO’s superior computational efficiency.



Table 2: Efficiency comparison of GeoMaNO and the current SOTA, Transolver [3].

(a) Navier-Stokes				(b) Darcy Flow			
Model	$T_{\text{train}} (\downarrow)$	$T_{\text{infer}} (\downarrow)$	Mem. ( $\downarrow$ )	Model	$T_{\text{train}} (\downarrow)$	$T_{\text{infer}} (\downarrow)$	Mem. ( $\downarrow$ )
Transolver	305.33	23.47	91.85	Transolver	30.99	2.33	42.55
<b>GeoMaNO</b>	<b>268.68</b>	<b>16.61</b>	<b>70.44</b>	<b>GeoMaNO</b>	<b>23.98</b>	<b>1.17</b>	<b>5.21</b>
Perf. Gain	12%	29.2%	23.3%	Perf. Gain	22.6%	49.8%	87.8%

**Scalability** To evaluate GeoMaNO’s scalability, we ablate its model depth, the number of Mamba dstates, and the number of embedding dimensions on the Darcy flow benchmark. The results are illustrated in Fig. 4. It can be observed that GeoMaNO scales well to these three metrics when the hyperparameters are small. However, GeoMaNO won’t scale infinitely: When these parameters go too large, the performance gain becomes less significant.

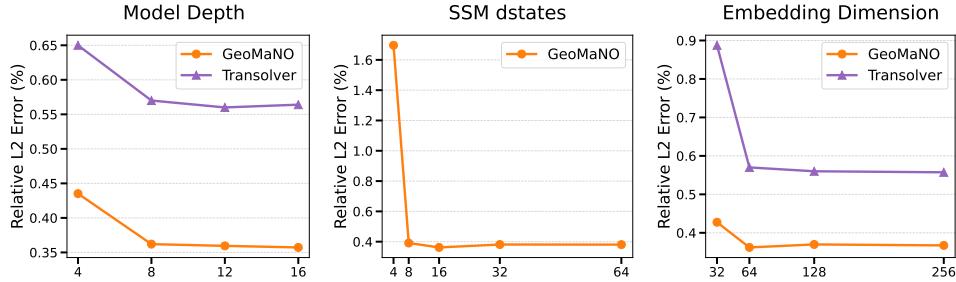


Figure 4: GeoMaNO’s scalability. We ablate model depth and embedding dimensions, and report the relative L2 errors (%).

### 5.3 Ablation Studies

We ablate GeoMaNO by the following factors: (1) Vanilla GeoMamba-SSM vs. 2DGeoMamba-SSM; (2) Positional encoding (none vs. yes); (3) The geometric correction coefficient  $\mathbf{R}_s$  (none vs. fixed vs. learnable), and the quantitative results are available in Table 3.

**Darcy Flow** The Darcy flow problem models fluid pressure over a 2D regular grid. Its 2D physical nature calls for a NO specialized for the 2D domain. As shown in Table 3 (a), 2DGeoMamba-SSM consistently outperforms the general-purpose vanilla GeoMamba-SSM. Among the 2D variants, the positional embedding decreases the performance. This is because positional embedding injects extra content into the grid and pollutes geometric information. The best performance is achieved with geometric corrections enabled, showcasing the necessity of our geometrically rigorous design.

**Navier-Stokes** The Navier-Stokes problem models fluid vorticity for a continuous period over a 2D regular grid. This problem is governed by a 3D physical domain  $(x, y, t)$ . The vanilla GeoMamba-SSM performs well, whereas 2DGeoMamba-SSM didn’t converge well because it’s specialized for 2D domains, and fails to capture the temporal dependencies. Again, the best performance is achieved with geometric correction enabled, highlighting the necessity of geometric rigor.

**Generalization to Other Domains** We further evaluate GeoMamba-SSM’s generalization to other domains. We integrate GeoMamba-SSM into the SOTA Mamba-based foundation framework, VMamba [8], and evaluate it with the ImageNet-100 classification benchmark [50]. GeoMamba-SSM outperforms vanilla Mamba-SSM by 0.22% on the top-1 accuracy score, which is significant in the natural image domain. More details are available in Sec. E.

## 6 Conclusion

In this paper, we propose **GeoMaNO**, a Mamba-based neural operator framework with geometric rigor and mathematical insights. GeoMaNO manifests a geometric-aware architecture specifically

Table 3: Ablation studies on the two benchmarks. We report the relative L2 error, and the rows for the best configurations are highlighted in **bold**. For the “Geo. Coeff.” columns, “None” denotes no geometric correction, “Learnable” denotes learnable geometric correction coefficients, and “0001” ~ “0111” denotes fixed coefficients as detailed in *Remark 4.2*.

(a) Darcy Flow				(b) Navier-Stokes			
SSM	Pos. Emb.	Geo. Coeff.	Rel. L2 ( $\downarrow$ )	SSM	Pos. Emb.	Geo. Coeff.	Rel. L2 ( $\downarrow$ )
Vanilla	✓	None	0.00389	Vanilla	None	None	0.04190
Vanilla	None	None	0.00381	Vanilla	None	0001	0.04212
Vanilla	None	0001	0.00385	Vanilla	None	0011	0.04062
Vanilla	None	0011	0.00376	Vanilla	None	0111	0.04153
Vanilla	None	0111	0.00387	Vanilla	None	Learnable	0.04190
Vanilla	None	Learnable	0.00386	Vanilla	✓	None	0.03938
2D	✓	None	0.00428	Vanilla	✓	0001	0.03795
2D	None	None	0.00380	Vanilla	✓	0011	0.03783
2D	None	0001	0.00387	Vanilla	✓	0111	0.03709
<b>2D</b>	<b>None</b>	<b>0011</b>	<b>0.00362</b>	<b>Vanilla</b>	✓	<b>Learnable</b>	<b>0.03701</b>
2D	None	0111	0.00383	2D	None	None	0.13554
2D	None	Learnable	0.00372	2D	✓	None	0.18287

tailored for regular grids, combined with our GeoMamba-SSM module, which employs a geometric correction component to dampen duplicate hidden states. GeoMaNO outperforms the SOTA in PDE solution operator approximation in both accuracy and efficiency. Extensive experiments validate all our claimed technical merits.

**Limitation and Future Research Directions** GeoMaNO currently works only on regular grids. However, in theory, our GeoMamba-SSM should also generalize to other unstructured domains like meshes and point clouds. Furthermore, the 2D version of our GeoMamba-SSM works well on 2D physical domains (Darcy benchmark) but fails to capture temporal dependencies on 3D domains (Navier-Stokes benchmark). It would be more beneficial to extend 2DMamba-SSM to 3D space. We would like to focus our future research efforts on these extensions and enhancements.

## References

- [1] A. Vaswani, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017.
- [2] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu, “Gnot: a general neural operator transformer for operator learning,” in *Proceedings of the 40th International Conference on Machine Learning*, 2023.
- [3] H. Wu, H. Luo, H. Wang, J. Wang, and M. Long, “Transolver: a fast transformer solver for pdes on general geometries,” in *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [4] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” in *Proceedings of the First Conference on Language Modeling*, 2024.
- [5] T. Dao and A. Gu, “Transformers are SSMS: Generalized models and efficient algorithms through structured state space duality,” in *Proceeding of the 41st International Conference on Machine Learning*, 2024.
- [6] J. Zheng, W. Li, N. Xu, J. Zhu, X. Lin, and X. Zhang, “Alias-free mamba neural operator,” in *Advances in Neural Information Processing Systems*, vol. 37, 2024.
- [7] Z. Hu, N. A. Daryakenari, Q. Shen, K. Kawaguchi, and G. E. Karniadakis, “State-space models are accurate and efficient neural operators for dynamical systems,” in *Computing Research Repository*, vol. abs/2409.03231, 2025.
- [8] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu, “Vmamba: Visual state space model,” in *Advances in Neural Information Processing Systems*, 2024.
- [9] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang, “Vision mamba: Efficient visual representation learning with bidirectional state space model,” in *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [10] J. Zhang, A. T. Nguyen, X. Han, V. Q.-H. Trinh, H. Qin, D. Samaras, and M. S. Hosseini, “2DMamba: Efficient state space model for image representation with applications on giga-pixel whole slide image classification,” in *Proceedings of 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [11] THUML, “PDE solving standard benchmark.” <https://github.com/thuml/Transolver/tree/main/PDE-Solving-StandardBenchmark>, 2025.
- [12] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, “Learning nonlinear operators via deeponet based on the universal approximation theorem of operators,” in *Nature Machine Intelligence*, pp. 218–229, 2021.
- [13] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier Neural Operator for Parametric Partial Differential Equations,” in *Proceedings of the 9th International Conference on Learning Representations*, 2021.
- [14] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, “U-fno – an enhanced fourier neural operator-based deep-learning model for multiphase flow,” *Advances in Water Resources*, vol. 163, pp. 104–180, 2022.
- [15] J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro, “Adaptive fourier neural operators: Efficient token mixers for transformers,” in *Proceedings of the 10th International Conference on Learning Representations*, 2022.
- [16] R. J. George, J. Zhao, J. Kossaifi, Z. Li, and A. Anandkumar, “Incremental spatial and spectral learning of neural operators for solving large-scale PDEs,” *Transactions on Machine Learning Research*, 2024.

- [17] J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, P. Hassanzadeh, K. Kashinath, and A. Anandkumar, “Four-castnet: A global data-driven high-resolution weather model using adaptive fourier neural operators,” in *PASC ’23: Proceedings of the Platform for Advanced Scientific Computing Conference*, 2022.
- [18] A. Tran, A. P. Mathews, L. Xie, and C. S. Ong, “Factorized fourier neural operators,” in *Proceedings of the 11th International Conference on Learning Representations*, 2023.
- [19] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Neural operator: Graph kernel network for partial differential equations,” in *Proceedings of ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.
- [20] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Multipole graph neural operator for parametric partial differential equations,” in *Advances in Neural Information Processing Systems*, 2020.
- [21] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Parallel prefix computation,” *Journal of Machine Learning Research*, vol. 89, pp. 1–97, 2023.
- [22] Z. Xiao, Z. Hao, B. Lin, Z. Deng, and H. Su, “Improved operator learning by orthogonal attention,” in *Proceedings of the 41st International Conference on Machine Learning*, vol. 235, 2024.
- [23] Z. Li, D. Shu, and A. B. Farimani, “Scalable transformer for PDE surrogate modeling,” in *Advances in Neural Information Processing Systems*, 2023.
- [24] C. Mao, R. Lupoiu, T. Dai, M. Chen, and J. A. Fan, “Towards general neural surrogate solvers with specialized neural accelerators,” in *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [25] X. Liu, B. Xu, and L. Zhang, “HT-net: Hierarchical transformer based operator learning model for multiscale PDEs,” 2023.
- [26] N. Kitaev, L. Kaiser, and A. Levskaya, “Reformer: The efficient transformer,” in *Proceedings of the 8th International Conference on Learning Representations*, 2020.
- [27] K. M. Choromanski, V. Likhoshesterov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, “Rethinking attention with performers,” in *Proceedings of the 9th International Conference on Learning Representations*, 2021.
- [28] S. Cao, “Choose a transformer: Fourier or galerkin,” in *Advances in Neural Information Processing Systems*, 2021.
- [29] C.-W. Cheng, J. Huang, Y. Zhang, G. Yang, C.-B. Schönlieb, and A. I. Aviles-Rivero, “Mamba neural operator: Who wins? transformers vs. state-space models for pdes,” in *Computing Research Repository*, vol. abs/2410.02113, 2024.
- [30] D. Greenfeld, M. Galun, R. Kimmel, I. Yavneh, and R. Basri, “Learning to Optimize Multigrid PDE Solvers,” in *Proceedings of the 36th International Conference on Machine Learning*, 2019.
- [31] X. Han, F. Hou, and H. Qin, “UGrid: An efficient-and-rigorous neural multigrid solver for linear PDEs,” in *Proceedings of the 41st International Conference on Machine Learning*, vol. 235, 2024.
- [32] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, “Combining recurrent, convolutional, and continuous-time models with linear state space layers,” *Advances in Neural Information Processing Systems*, 2021.
- [33] A. Gu, K. Goel, and C. Ré, “Efficiently modeling long sequences with structured state spaces,” in *Computing Research Repository*, 2021.
- [34] A. Gupta, A. Gu, and J. Berant, “Diagonal state spaces are as effective as structured state spaces,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 22982–22994, 2022.

- [35] Y. Li, T. Cai, Y. Zhang, D. Chen, and D. Dey, “What makes convolutional models great on long sequence modeling?,” in *Computing Research Repository*, vol. abs/2210.09298, 2022.
- [36] C. Yang, Z. Chen, M. Espinosa, L. Ericsson, Z. Wang, J. Liu, and E. J. Crowley, “Plainmamba: Improving non-hierarchical mamba in visual recognition,” in *Proceedings of the 35th British Machine Vision Conference*, 2024.
- [37] F. Xie, W. Zhang, Z. Wang, and C. Ma, “Quadmamba: Learning quadtree-based selective scan for visual state space model,” *Advances in Neural Information Processing Systems*, 2024.
- [38] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Neural operator: learning maps between function spaces with applications to pdes,” *The Journal of Machine Learning Research*, vol. 24, no. 1, 2023.
- [39] A. Trockman and J. Z. Kolter, “Patches are all you need?,” in *Proceedings of the 11th International Conference on Learning Representations*, 2023.
- [40] G. Bebis and M. Georgiopoulos, “Feed-forward neural networks,” *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.
- [41] P. Bevanda, S. Sosnowski, and S. Hirche, “Koopman operator dynamical models: Learning, analysis and control,” *Annual Reviews in Control*, vol. 52, pp. 197–212, 2021.
- [42] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, “Perceiver: General perception with iterative attention,” in *Proceedings of the 38th International Conference on Machine Learning*, 2021.
- [43] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” in *Proceedings of the 3rd International Conference for Learning Representations*, 2016.
- [44] G. Gupta, X. Xiao, and P. Bogdan, “Multiwavelet-based operator learning for differential equations,” in *Advances in Neural Information Processing Systems*, 2021.
- [45] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar, “Fourier neural operator with learned deformations for pdes on general geometries,” *Journal of Machine Learning Research*, vol. 24, no. 388, pp. 1–26, 2023.
- [46] M. A. Rahman, Z. E. Ross, and K. Azizzadenesheli, “U-NO: U-shaped neural operators,” *Transactions on Machine Learning Research*, 2023.
- [47] H. Wu, T. Hu, H. Luo, J. Wang, and M. Long, “Solving high-dimensional pdes with latent spectral models,” in *Proceedings of the 40th International Conference on Machine Learning*, 2023.
- [48] Z. Li, K. Meidani, and A. B. Farimani, “Transformer for partial differential equations’ operator learning,” *Transactions on Machine Learning Research*, 2023.
- [49] Z. Li, D. Shu, and A. B. Farimani, “Scalable transformer for PDE surrogate modeling,” in *Advances in Neural Information Processing Systems*, 2023.
- [50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [51] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proceedings of the 7th International Conference on Learning Representations*, 2019.
- [52] L. N. Smith and N. Topin, “Super-convergence: very fast training of neural networks using large learning rates,” in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006, p. 1100612, International Society for Optics and Photonics, 2019.
- [53] A. Kulikovskiy, “A more accurate scharrfetter-gummel algorithm of electron transport for semiconductor and gas discharge simulation,” *Journal of Computational Physics*, vol. 119, no. 1, pp. 149–155, 1995.

- [54] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and memory-efficient exact attention with IO-awareness,” in *Advances in Neural Information Processing Systems*, 2022.
- [55] T. Dao, “FlashAttention-2: Faster attention with better parallelism and work partitioning,” in *Proceedings of the 12th International Conference on Learning Representations*, 2024.
- [56] R. E. Ladner and M. J. Fischer, “Parallel prefix computation,” *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.
- [57] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, “Scan primitives for gpu computing,” in *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, 2007.
- [58] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2016.

# Supplementary Materials

## GeoMaNO: Geometric Mamba Neural Operator for Partial Differential Equations

### A Specifications for Experiments

#### A.1 Benchmarks

We evaluate GeoMaNO’s performance on regular grids with the standard neural operator PDE solving benchmarks [11], which is employed by most neural operator frameworks from FNO [13] to Transolver [3]. We use the Darcy flow benchmark, which is a two-dimensional linear elliptic problem, and the Navier-Stokes benchmark, which is a three-dimensional (including the time dimension) non-linear parabolic/hyperbolic problem.

**Darcy Flow [13]** The Darcy Flow equation represents the steady-state diffusion process over porous media. The 2D Darcy Flow equation over a unit square is formulated as:

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

where  $a(x, y)$  is the viscosity,  $f(x, y)$  is the forcing term, and  $u(x, y)$  is the solution. Further, Eq. 11 is modified in the form of a temporal evolution as follows:

In this benchmark, the input is represented by the parameter  $a(x, y)$ , and the corresponding output is the solution  $u(x, y)$ . For consistency with FNO [13]’s evaluation, the coefficients  $a(x, y)$  are generated according to a 2D Gaussian distribution:  $a \sim \mathcal{N}(0, (-\Delta + 9\mathbf{I})^{-2})$ , with zero Neumann boundary conditions on the Laplacian. The forcing term is enforced as a constant value  $f(x, y) \equiv f_0$ . Such constructions are prototypical models for many physical systems, such as permeability in subsurface flows and material microstructures in elasticity. The solution term  $u(x, y)$  is obtained by using a second-order finite difference scheme on a  $421 \times 421$  regular grid, and then downsampled to a resolution of  $85 \times 85$  for the main experiments. Following the standard practice, we use 1000 samples for training, 200 samples for testing, and different samples will contain different medium structures.

**Navier-Stokes [13]** The 2D Navier-Stokes equation models the flow of a viscous, incompressible fluid in vorticity form on the unit torus. It is mathematically formulated as:

$$\begin{cases} \frac{\partial}{\partial t} w(x, y, t) + u(x, y, t) \cdot \nabla w(x, y, t) = \nu \nabla^2 w(x, y, t) + f(x, y), & (x, y) \in (0, 1)^2, \quad t \in (0, T] \\ \nabla \cdot u(x, y, t) = 0, & (x, y) \in (0, 1)^2, \quad t \in (0, T] \\ w(x, y, 0) = w_0(x, y), & (x, y) \in (0, 1)^2 \end{cases} \quad (12)$$

where  $u(x, y, t)$  represents the velocity field,  $w(x, y, t) = \nabla \times u(x, y, t)$  is the vorticity,  $w_0(x, y)$  is the initial vorticity,  $\nu$  is the viscosity coefficient, and  $f(x, y)$  is the forcing function. In this benchmark, the viscosity  $\nu$  is fixed at  $10^{-5}$ , and the 2D field has a resolution of  $64 \times 64$ . Each sample within the dataset comprises 20 consecutive frames. The objective is to predict the subsequent ten frames based on the preceding ten. Following the common practice, we use 1000 fluid samples with different initial conditions for training, and 200 samples for testing.

#### A.2 Baselines

We compare GeoMaNO with 14 baselines, half of which are frequency-based NOs: FNO [13], WMT [44], U-FNO [14], Geo-FNO [45], U-NO [46], F-FNO [18], and LSM [47], while the re-

mainders are Transformer-based NOs: Galerkin [28], HT-Net [25], OFormer [48], GNOT [2], FactFormer [49], ONO [22], and Transolver [3]. The current SOTA is the Transolver [3] framework.

**Remark A.1 (GeoMaNO vs. Zheng et al. [6]).** We have noted Zheng et al.[6], the only peer-reviewed prior work on Mamba-based neural operators at the time of this submission. However, Zheng et al. have *not yet open-sourced their code*, making it impossible for us to conduct experiments against them. For the theoretical completeness of this paper, we will conduct a theoretical and architectural comparison between GeoMaNO and Zheng et al.[6].

Zheng et al. [6] highlight their contributions on theoretical derivations regarding Mamba-based NO’s mathematical correctness, as well as its alias-free property. Their architecture employs a U-shaped multi-resolution architecture, which is more commonly seen in vision-domain applications like segmentation and detection.

On the contrary, our novel GeoMaNO framework focuses on integrating geometric and mathematical rigor into the NO framework. We propose the GeoMamba-SSM formulation, which dampens duplicate hidden states due to the multiway cross-scan mechanism. GeoMaNO further enhances its geometric rigor by employing two-dimensional SSM recursions on 2D physical domains. We further integrate GeoMamba-SSM into the legacy NO framework.

### A.3 Evaluation Metric

We follow the common practice founded by FNO [13] and employ the relative L2 error metric:

$$\mathcal{L}(\text{pred}, \text{gt}) = \frac{\|\text{pred} - \text{gt}\|_2}{\|\text{gt}\|_2},$$

where **pred**, **gt** separately denotes the prediction and the ground truth, and  $\|\mathbf{x}\|_2$  denotes the vector L2 norm.

### A.4 Implementation Details

**Implementation of Baselines** All of these baselines have been widely examined in previous papers. For FNO [13] and Geo-FNO [45], we report their results following their official papers. Other spectral baselines, i.e., WMT [44], U-FNO [14], U-NO [46], F-FNO [18], and LSM [47], we follow the configurations and results reported in LSM [47]. For Transformer-based baselines, i.e., Galerkin [28], HT-Net [25], OFormer [48], GNOT [2], FactFormer [49], and ONO [22], we follow the configurations and results reported in Transolver [3]. For Transolver [3], in Table 1, we compare with results reported in its original paper; for other experiments, we run their official code in our local environment, following its original configurations.

**Implementation of GeoMaNO** The training configurations are listed in Table 4, and the model hyperparameters are listed in Table 5.

Table 4: Training configurations of GeoMaNO. Training configurations are directly from previous works without special tuning [3]. For the Darcy flow benchmark, following the practice of ONO [22], we employ an additional spatial gradient regularization term  $\mathcal{L}_g$ , in addition to the regular relative L2 loss  $\mathcal{L}_{rL2}$ .

Benchmarks	Loss	Epochs	Initial LR	Optimizer	Scheduler	Batch Size
Navier-Stokes	$\mathcal{L}_{rL2}$	500	$3 \times 10^{-4}$	AdamW [51]	OneCycleLR [52]	2
Darcy Flow	$\mathcal{L}_{rL2} + 0.1\mathcal{L}_g$					4

Table 5: Model configurations of GeoMaNO.

Benchmarks	Model Depth	SSM dstates	Embedding Dimensions
Navier-Stokes	8	16	256
Darcy Flow			64



## A.5 Error Bounds

For the mathematical completeness of this paper, we report the error bounds (standard deviations) for our numerical experiments. The results are available in Table 6 and Table 7.

Table 6: Error bounds (standard deviations) of Table 1.

Model	Relative L2 ( $\downarrow$ )	
	N.-S.	Darcy Flow
<b>GeoMaNO</b>	<b>0.0370</b> $\pm(3.40 \times 10^{-3})$	<b>0.0036</b> $\pm(1.17 \times 10^{-4})$

Table 7: Error bounds (standard deviations) of Table 2.

(a) Darcy Flow			
Model	$T_{\text{train}}$ ( $\downarrow$ )	$T_{\text{infer}}$ ( $\downarrow$ )	Mem. ( $\downarrow$ )
Transolver	30.99 $\pm(4.71 \times 10^{-3})$	2.33 $\pm(2.44 \times 10^{-4})$	42.55 $\pm 0.00$
<b>GeoMaNO</b>	<b>23.98</b> $\pm(1.09 \times 10^{-3})$	<b>1.17</b> $\pm(1.98 \times 10^{-4})$	<b>5.21</b> $\pm 0.00$
(b) Navier-Stokes			
Model	$T_{\text{train}}$ ( $\downarrow$ )	$T_{\text{infer}}$ ( $\downarrow$ )	Mem. ( $\downarrow$ )
Transolver	305.33 $\pm(4.41 \times 10^{-1})$	23.47 $\pm(7.72 \times 10^{-2})$	<b>5.21</b> $\pm 0.00$
<b>GeoMaNO</b>	<b>268.68</b> $\pm(4.71 \times 10^{-3})$	<b>16.61</b> $\pm(8.88 \times 10^{-1})$	<b>70.44</b> $\pm 0.00$

## B Mathematical Insights on Mamba-Based Neural Operators

For the theoretical completeness of this paper, we outline the mathematical derivations of Mamba-SSM’s equivalence to the neural operator (NO)’s critical component, the *kernel integral operator*. This section is credited to Zheng et al. [6].

For more mathematical details and insights, we refer the readers to Kovachki et al. [38] for the neural operator (NO) framework’s mathematical correctness and convergence guarantees, and to Zheng et al. [6] for Mamba-SSM’s equivalence to the kernel integral operators in NOs.

**Integral Kernels for Neural Operators** The general formulation of an NO’s kernel integral operator is defined as:

$$\mathcal{K}_t : \{v_t : D \rightarrow \mathbb{R}^{d_t}\} \rightarrow \{v_{t+1} : D \rightarrow \mathbb{R}^{d_{t+1}}\},$$

which could be parameterized by a factor  $\alpha$  such as:

$$(\mathcal{K}_t(v_t); \alpha)(x) = \int_D K_t(x, y, v_t(x), v_t(y)) v_t(y) dy, \quad \forall (x, y) \in D, \quad (13)$$

where the parameter of the *integral kernel*  $K_t$  is learned from the training data. For example, FNO [13] employs convolution as the integral kernel, while Transformer-based NOs manifest the attention blocks [1] as integral kernels.

**Mamba Integration** For simplicity, we omit the subscript  $t$  in the right-hand-side of Eq. 13, which is used to denote the number of iterations during the integration flow. Eq. 14 is reformulated as:

$$(\mathcal{K}_t(v_t); \alpha)(x) = \int_D K(x, y, v(x), v(y)) v(y) dy, \quad \forall (x, y) \in D. \quad (14)$$

Following the tradition of FNO [13], we first assume that  $K_t : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^{d_{t+1} \times d_t}$  concerns little on the spatial variables  $(v(x), v(y))$ , but only on the input pair  $(x, y)$ . We further let

$$K_t(x, y) = C e^{Ax} B e^{-Ay}, \quad (15)$$

where  $A$ ,  $B$  and  $C$  are treated as constants for simplicity. To further ensure the possible employment of Mamba’s scanning pattern [4, 5], we set the integration interval to  $y \in (-\infty, x)$  instead of the entire definition domain  $D$ . Eq. 14 hence becomes

$$(\mathcal{K}_t(v_t); \alpha)(x) = \int_{-\infty}^x C e^{Ax} B e^{-Ay} dy. \quad (16)$$

Since  $C e^{Ax}$  is independent of the integral variable  $y$ , we could further rewrite Eq. 16 as:

$$\begin{cases} (\mathcal{K}_t(v_t); \alpha)(x) = Ch(x), \\ h(x) = e^{Ax} \int_{-\infty}^x B(e^{-Ay})v(y) dy. \end{cases} \quad (17)$$

By differentiating both sides of Eq. 17 with respect to  $x$ , we get:

$$\begin{aligned} h'(x) &= A e^{Ax} \int_0^x B(e^{-Ay})v(y) dy + e^{Ax} B e^{-Ax} v(x) \\ &= A e^{Ax} \int_0^x B(e^{-Ay})v(y) dy + Bv(x) \\ &= Ah(x) + Bv(x). \end{aligned} \quad (18)$$

Combining Eq. 17 with Eq. 18 yields

$$\begin{cases} h'(x) = Ah(x) + Bv(x), \\ u(x) = Ch(x), \end{cases} \quad (19)$$

where  $u(x) = (\mathcal{K}_t(v_t); \alpha)(x)$ . Eq. 19 directly parallels Eq. 2 in the main paper. This seamlessly marries a NO’s kernel integral with a state space model (SSM) [33]. Drawing inspiration from the theory of continuous systems, the goal of Eq. 19 is to map a two-dimensional function, denoted as  $v(x)$ , to  $u(x)$  through the hidden space  $h(x)$ . Here,  $A$  serves as the evolution parameter, while  $B$  and  $C$  act as the projection parameters.

**Discretization of Mamba Integration** To integrate Eq. 19 into the deep learning paradigm, a discretization process is necessary. Following the Scharfetter-Gummel method [53], which approximates the matrix exponential using Bernoulli polynomials, the parameters could be discretized as follows:

$$\begin{cases} \bar{\mathbf{A}} = \exp(\Delta A), \\ \bar{\mathbf{B}} = (\Delta A)^{-1}(\exp(\Delta A) - \mathbf{I}) \Delta B, \end{cases} \quad (20)$$

where  $\Delta$  is a timescale parameter converting continuous parameters  $A$  and  $B$  into their discrete counterparts  $\bar{\mathbf{A}}$  and  $\bar{\mathbf{B}}$ . The discrete representation of Eq. 19 can be formulated as

$$\begin{cases} h(x_k) = \bar{\mathbf{A}}h(x_{k-1}) + \bar{\mathbf{B}}v(x_k), \\ u(x_k) = Ch(x_k). \end{cases} \quad (21)$$

Eq. 21 directly parallels Mamba’s selective state-space representation, i.e., Eq. 3 in the main paper. More details can be found in the supplementary materials of Zheng et al. [6].

## C Theoretical Insights on 2DMamba-SSM

This section discusses the 2DMamba [10] formulation and its mathematical and geometrical rigor. This illustrates our design choice to employ 2DMamba-SSM in our GeoMamba formulation for two-dimensional physical domains.

### C.1 2DMamba-SSM Formulation

In contrast to vanilla 1D Mamba-SSM, which aggregates information from a flattened 1D sequence, a 2DMamba-SSM aggregates both geometric and semantic information directly from a 2D feature map. For simplicity, we *omit* the independent state dimension superscript  $s$  in Eq. 4, and get:

$$\begin{cases} \mathbf{g}[i, j] = \overline{\mathbf{A}}[i, j] \mathbf{g}[i, j-1] + \overline{\mathbf{B}}[i, j] \mathbf{x}[i, j], \\ \mathbf{h}[i, j] = \overline{\mathbf{A}}[i, j] \mathbf{h}[i-1, j] + \mathbf{g}[i, j], \\ \mathbf{y}[i, j] = \sum \overline{\mathbf{C}}[i, j] \mathbf{h}[i, j]. \end{cases} \quad (22)$$

Note that, for the first line in Eq. 22, we define  $\mathbf{g}[i, 0] = 0$ , and thus  $\mathbf{g}[i, 1] = \overline{\mathbf{B}}[i, 1] \mathbf{x}[i, 1]$ . The two parameters  $\overline{\mathbf{A}}[i, j]$  and  $\overline{\mathbf{B}}[i, j]$  depend on the input  $\mathbf{x}[i, j]$ , regulating the information of previous hidden states  $\mathbf{h}[i, j-1]$ ,  $\mathbf{h}[i-1, j-1]$ , and  $\mathbf{h}[i-1, j]$ , as well as the current input  $\mathbf{x}[i, j]$ . Similarly, for the second line, we define  $\mathbf{h}[0, i] = 0$ , and thus  $\mathbf{h}[1, j] = \mathbf{g}[1, j]$ .

For simplicity, in the following reasoning, we replace the square brackets with subscripts, omit the subscripts of  $\overline{\mathbf{A}}$  and  $\overline{\mathbf{B}}$ , and expand Eq. 22. With these simplifications, the hidden state  $\mathbf{h}_{i,j}$  can be formulated as the following recurrence:

$$\mathbf{h}_{i,j} = \sum_{i' \leq i} \sum_{j' \leq j} \overline{\mathbf{A}}^{(i-i'+j-j')} \overline{\mathbf{B}} \mathbf{x}_{i',j'}, \quad (23)$$

where  $(i - i' + j - j')$  equals the *Manhattan distance*<sup>5</sup> between patches  $(i', j')$  and  $(i, j)$ . It represents a path from  $(i', j')$  to  $(i, j)$ , which goes right horizontally and then down vertically. After two scans, the output  $\mathbf{y}$  is aggregated from  $\mathbf{h}$  by the parameter  $\mathbf{C}$  similar to 1D-Mamba:  $\mathbf{y}_{i,j} = \mathbf{C} \mathbf{h}_{i,j}$ . For each location  $(i, j)$ , the aggregation information is obtained from its upper-left locations.

## C.2 Geometric Consistency

In this subsection, we will use a concrete example to illustrate our GeoMamba-SSM’s geometric rigor. Fig. 5 illustrates the 1D scanning path employed by the vanilla Mamba-SSM vs. the 2D scanning path employed by our 2DGeoMamba-SSM.

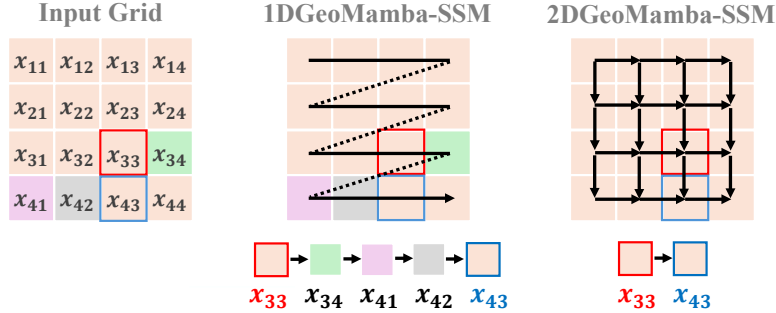


Figure 5: Illustration of 1D v.s. 2D scanning paths and spatial discrepancy. **Left:** The two-dimensional input grid. **Middle:** 1D Mamba-SSM flattens the input into a 1D sequence, but adjacent patches (the red and blue) are far away in this sequence, leading to the *geometric inconsistency* in the hidden states. **Right:** 2DMamba-SSM the input grid in a 2D manner, and maintains spatial continuity.

As illustrated in Fig. 5, the hidden state of the vanilla Mamba [4] on a flattened image is

$$\mathbf{h}_i^{1D} = \sum_{i' \leq i} \overline{\mathbf{A}}^{(i-i')} \overline{\mathbf{B}} \mathbf{x}_{i'}, \quad (24)$$

where  $i$  is an 1D index. The order  $i - i'$  is the distance between  $i$  and  $i'$  in a flattened sequence (a higher order results in forgetting). From Fig. 5, flattening in the 1D direction results in a loss of spatial structure in other directions, and thus is sub-optimal, regardless of various formulations for flattening. We denote this phenomenon as *spatial discrepancy*.

<sup>5</sup>In contrast, for the vanilla Mamba-SSM, the power of  $\overline{\mathbf{A}}$  will be the *row-major linear distance* between the two patches, as shown in Eq. 24. This reflects the issue of geometric inconsistency in a mathematical manner.

To solve this problem, 2DMamba-SSM formulates Eq. 23. We will then use a concrete example to compare these two formulations.

For an input feature map of size  $3 \times 3$ , the last hidden state of the vanilla Mamba-SSM is formulated as:

$$\begin{aligned} h_{3,3}^{1D} = & \overline{A}^8 x_{1,1} + \overline{A}^7 x_{1,2} + \overline{A}^6 x_{1,3} + \\ & \overline{A}^5 x_{2,1} + \overline{A}^4 x_{2,2} + \overline{A}^3 x_{2,3} + \\ & \overline{A}^2 x_{3,1} + \overline{A}^1 x_{3,2} + \overline{A}^0 x_{3,3}, \end{aligned} \quad (25)$$

whereas 2DMamba-SSM formulation yields:

$$\begin{aligned} h_{3,3}^{2D} = & \overline{A}^4 x_{1,1} + \overline{A}^3 x_{1,2} + \overline{A}^2 x_{1,3} + \\ & \overline{A}^3 x_{2,1} + \overline{A}^2 x_{2,2} + \overline{A}^1 x_{2,3} + \\ & \overline{A}^2 x_{3,1} + \overline{A}^1 x_{3,2} + \overline{A}^0 x_{3,3}. \end{aligned} \quad (26)$$

It can be observed that, for the vanilla Mamba-SSM, the coefficient of  $x_{1,1}$  is  $\overline{A}^8$ , where the power reflects the *row-major linear distance* between patches (1, 1) and (3, 3). On the contrary, 2DMamba-SSM's coefficient for  $x_{1,1}$  is  $\overline{A}^4$ , where the power reflects the *Manhattan distance* between patches (1, 1) and (3, 3). Mathematically speaking, 1D Mamba's higher order of  $\overline{A}$  leads to more forgetting and loss of 2D structure. Meanwhile, semantically speaking, given that these adjacent patches are spatially connected, our 2DMamba-SSM better reflects their neighboring relationship. On the contrary, the 1D Mamba creates spatial discrepancy and may dilute the information from vertically near patches.

## D Hardware-Aware Implementation of GeoMamba-SSM

Our GeoMamba-SSM formulation is a novel neural architecture, and it comes with no ready-to-use implementation<sup>6</sup>. As a remedy, we propose a novel GPU algorithm for our GeoMamba-SSM.

GPU algorithms must be designed with special care for the GPU memory hierarchy, minimizing memory transfers between on-chip registers/caches and off-chip GPU VRAM (a.k.a. HBM). In GPU algorithms, data is transferred from HBM to on-chip memory for computation, and the results are stored back to HBM to vacate on-chip memory for succeeding on-chip operations. Memory transfers are expensive. Therefore, instead of computation, many GPU algorithms, including the vanilla 1D Mamba-SSM [4, 5], are memory-bounded. Here, a naive implementation will ruin the GPU memory hierarchy and thus suffer from prohibitively low throughput and large GPU VRAM requirements.

In this section, we first revisit the GPU memory hierarchy and the hardware-aware 1D selective SSE solver proposed by Mamba [4]. Next, we analyze the major challenges for implementing our GeoMamba-SSM kernels. Then, we present our selective scanning algorithm and the GPU algorithm to implement GeoMamba-SSM in detail.

**GPU Memory Hierarchy** Fig. 6 illustrates the memory hierarchy of modern GPUs. The **green** area represents off-chip GPU memory. *Off-chip memory* is characterized by low speed and high capacity, and typically consists of High-Speed Memory (HBM). Here, for simplicity, it is referred to as HBM. On the other hand, the **orange** area denotes on-chip GPU memory. *On-chip memory* is characterized by high speed and low capacity, and typically consists of Static Random-Access Memory (SRAM, a.k.a. caches) and registers. Again, for simplicity, it is referred to as SRAM.

In GPU algorithms, data is transferred from HBM to SRAM for computation, and the results are stored back to HBM to vacate SRAM for succeeding computation, as illustrated in Fig. 7. Memory transfers are expensive. Therefore, instead of computation, many GPU algorithms [54, 55] are bounded by memory. Mamba-SSM [4] is also *memory-bounded*.

<sup>6</sup>The vanilla Mamba-SSM has an efficient GPU implementation, yet it does not support our geometric correction components.

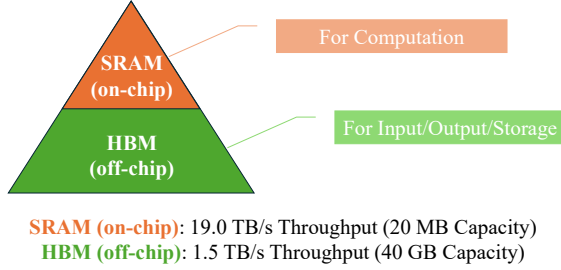


Figure 6: Illustration of the typical GPU Memory hierarchy. The typical GPU memory hierarchy. On-chip SRAMs are small but fast; off-chip HBMs have large capacities but are slow. Data of throughput and capacity reflect an NVIDIA Quadro RTX 8000 GPU.

**The Vanilla Mamba-SSM** The neural module of Mamba-SSM iterates through the input sequence  $\mathbf{x}[t]$  and regresses the hidden states  $\mathbf{h}_s[t]$  via a parallel prefix scan [56, 57] operation. In the literature of Computer Vision, this solving process is referred to as a *selective scan* due to the selective nature of the SSE coefficients.

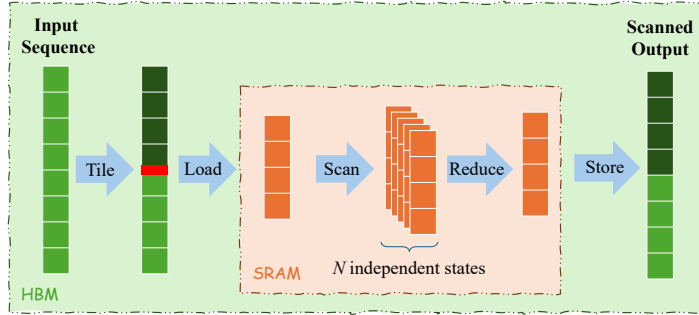


Figure 7: Illustration of Mamba’s 1D scanning operator. Here, **orange** color represents data and operations on SRAM; while **green** color represents those on HBM. Mamba’s 1D scanning operator intakes a flattened sequence on HBM. It tiles the input into sub-sequences. Each sub-sequence is loaded from HBM to SRAM, scanned and reduced across  $N$  intermediate mutually-independent *state dimensions*, and then stored back to HBM. The total memory access complexity is  $\mathcal{O}(L)$ .

Mamba-SSM is efficient because it respects the GPU memory hierarchy by 1D tiling and caching. As shown in Fig. 7, a long feature sequence in HBM is divided into smaller tiles. Each tile is loaded into SRAM, scanned across  $N$  independent state dimensions, aggregated into a single output, and finally stored back to HBM. The intermediate results of the  $N$  state dimensions are *not explicitly materialized on HBM* and will be recomputed during back-propagation. The overall memory access complexity is  $\mathcal{O}(L)$ , where  $L$  denotes the sequence length.

**GeoMamba-SSM** Algorithm 2 details our novel GeoMamba-SSM scanner module. In Algorithm 2,  $\circ$  denotes element-wise multiplication. Algorithm 2 is designed as a hardware-aware GPU kernel, which operates at the granularity of a single GPU thread. Our geometric correction mechanism is applied at line 24 of Algorithm 2.

The proposed hardware-aware GeoMamba-SSM module optimizes memory transactions by conducting the geometric corrections at the GPU thread level, instead of appending PyTorch-level successors in the computation graph. This eliminates unnecessary GPU kernel launches and avoids creating nodes for our geometric correction in the computation graph. Our hardware-aware implementation of GeoMamba-SSM maintains an overall  $\mathcal{O}(L)$  memory complexity, avoiding a multiplication with the

---

**Algorithm 2** Hardware-aware GeoMamba-SSM Module.

---

**Require:** The current two-dimensional patch grid  $\widehat{\mathbf{X}} : (B, ED, H, W)$ ;  
**Require:** Time step  $\Delta_t : (B, ED, H, W)$  and bias term  $\Delta_{\text{bias}} : (N, )$ ;  
**Require:** SSM coefficients  $\widehat{\mathbf{A}}_t : (N, ED)$  and  $\widehat{\mathbf{B}}_t : (B, H, W)$ ;  
**Require:** Aggregation weight  $\widehat{\mathbf{C}}_t : (B, N)$ ;  
**Require:** Skip-connection weight  $\widehat{\mathbf{D}}_t : (ED, )$ ;  
**Require:** Geometric Correction weight  $\widehat{\mathbf{R}}_t : (N, ED)$ ;  
**Require:** GPU tile size  $T$ .  
**Ensure:** Aggregated patch grid  $\mathbf{Y} : (B, ED, H, W)$ .  
1:  $K_H, K_W = \lceil H/T \rceil, \lceil W/T \rceil$ ;  
2: On-chip  $\Delta_{\text{bias}} = \text{Load from HBM}$ ;  
3: On-chip  $D = \text{Load from HBM}$ ;  
4: **for**  $k_h = 1$  to  $K_H$  and  $k_w = 1$  to  $K_W$  **do** ▷ Loop for  $K_H \times K_W$  tiles.  
5:   On-chip  $\mathbf{x}[k_h, k_w] : (T, T) = \text{Load from HBM}$ ;  
6:   On-chip  $\Delta[k_h, k_w] : (T, T) = \text{Load from HBM}$ ;  
7:    $\Delta_{\mathbf{x}} : (T, T) = \text{softplus}(\Delta \circ \mathbf{x}[k_h, k_w] + \Delta_{\text{bias}})$ ;  
8:    $\mathbf{y}[k_h, k_w] : (T, T) = \mathbf{0}$ ;  
9:   **for** Independent SSM dstates  $s = 1$  to  $N$  **do**  
10:     On-chip  $A_s = \text{Load from HBM}$  ▷ Scalar.  
11:     On-chip  $\mathbf{B}_s[k_h, k_w] : (T, T) = \text{Load from HBM}$  ▷ Matrix.  
12:     On-chip  $C_s = \text{Load from HBM}$  ▷ Scalar.  
13:     On-chip  $R_s = \text{Load from HBM}$  ▷ Scalar.  
14:      $\mathbf{B}_{\Delta}^s \mathbf{x} : (T, T) = \mathbf{B}_s \circ \Delta \circ \mathbf{x}[k_h, k_w]$ ;  
15:      $\mathbf{A}_{\Delta}^s : (T, T) = \mathbf{A}_s[k_h, k_w] \circ \Delta_{\mathbf{x}}$ ;  
16:     # Scan each row in parallel based on  $\mathbf{A}_{\Delta}^s$ .  
17:     On-chip  $\mathbf{P}^{\text{hor}} = \text{Load } \mathbf{P}^{\text{hor}}[k_h, k_w - 1]$  from HBM;  
18:      $\mathbf{g}_s : (T, T) = \text{parallel\_horizontal\_scan}(\exp(\mathbf{A}_{\Delta}^s), \mathbf{B}_{\Delta}^s \mathbf{x}, \mathbf{P}^{\text{hor}})$ ;  
19:     Store last column of  $\mathbf{g}_s$  as  $\mathbf{P}^{\text{hor}}[k_h, k_w]$  to HBM;  
20:     # Scan each column in parallel based on  $\mathbf{g}_s$ .  
21:     On-chip  $\mathbf{P}^{\text{ver}} = \text{Load } \mathbf{P}^{\text{hor}}[k_h - 1, k_w]$  from HBM;  
22:      $\mathbf{h}_s : (T, T) = \text{parallel\_vertical\_scan}(\exp(\mathbf{A}_{\Delta}^s), \mathbf{g}_s, \mathbf{P}^{\text{ver}})$ ;  
23:     Store last row of  $\mathbf{h}_s$  as  $\mathbf{P}^{\text{ver}}[k_h, k_w]$  to HBM;  
24:      $\mathbf{y}[k_h, k_w] = \mathbf{y}[k_h, k_w] + C_s \circ (\mathbf{h}_s - R_s \mathbf{x}[k_h, k_w])$  ▷ Geometric correction and aggregation.  
25:   **end for** ▷ End state dimensions.  
26:    $\mathbf{y}[k_h, k_w] = \mathbf{y}[k_h, k_w] + D \mathbf{x}[k_h, k_w]$  ▷ Learnable skip-connection.  
27:   Store  $\mathbf{y}[k_h, k_w]$  to HBM.  
28: **end for** ▷ End tiles.

---

Mamba dstate dimension  $N$ . This enhances the overall computational efficiency of our GeoMaNO framework.

## E Generalization to Other Domains

To further evaluate the geometric rigor and generalization power of our GeoMamba-SSM formulation, we integrate GeoMamba-SSM in the SOTA Mamba-based foundation model, the VMamba [8] framework, for a standard downstream vision-domain benchmark: ImageNet-100 classification benchmark [50]. As shown in Table 8, GeoMamba-SSM outperforms vanilla Mamba-SSM [4] by 0.22%, which is a significant performance gain in the field of natural images. This showcases the necessity of our geometric correction mechanism.

Table 8: Results on ImageNet-100 classification benchmark. The best configuration and result is highlighted in **bold**.

SSM	Geo. Coeff.	Top-1 Acc. (%)
Mamba-SSM	None	90.90
<b>GeoMamba-SSM</b>	<b>0011</b>	<b>91.12</b>
GeoMamba-SSM	0111	90.90

**Visualization of Effective Receptive Field (ERF)** ERF refers to the region in the input space that contributes to the activation of a certain output unit [58]. As illustrated in Fig. 8, the latent space of the central pixel is less concentrated along the central cross<sup>7</sup>, and distributes more across the whole domain. This highlights GeoMamba-SSM’s geometric rigor.

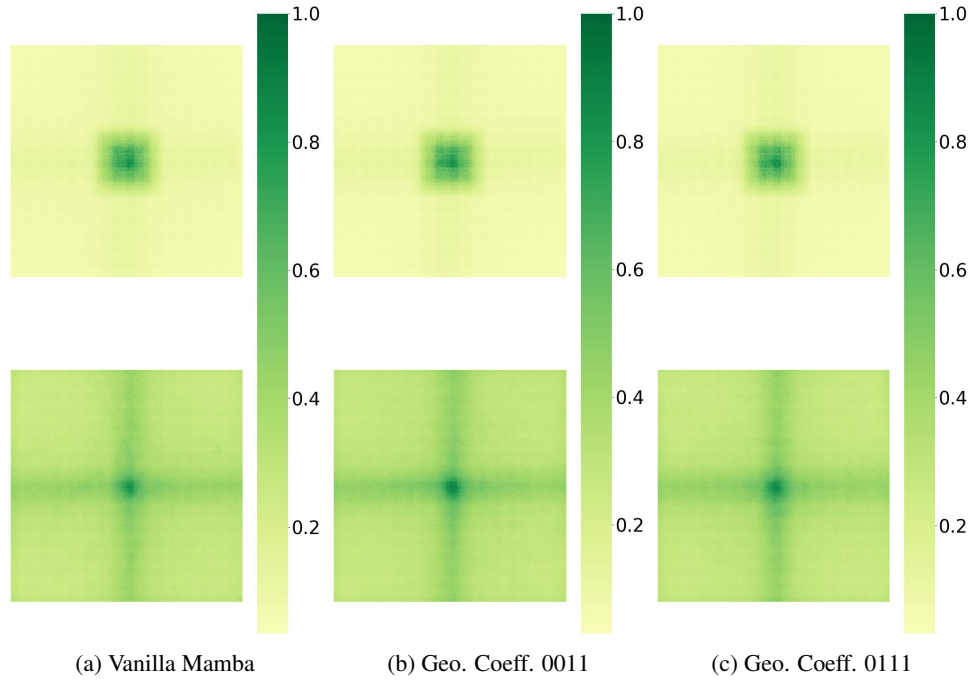


Figure 8: Comparasion of ERFs on ImageNet100 classification tasks. Pixels with higher intensity indicate larger responses regarding the central pixel. With our geometric coefficients, the potential space focuses less along the central cross region, showcasing GeoMamba-SSM’s geometric rigor.

<sup>7</sup>This cross-like pattern is caused by the cross-scan pattern.