

# Tensor-Compressed and Fully-Quantized Training of Neural PDE Solvers

Jinming Lu	Jiayi Tian	Yequan Zhao	Hai Li	Zheng Zhang
<i>UC Santa Barbara</i>	<i>UC Santa Barbara</i>	<i>UC Santa Barbara</i>	<i>Intel Corporation</i>	<i>UC Santa Barbara</i>
Santa Barbara, CA	Santa Barbara, CA	Santa Barbara, CA	Portland, OR	Santa Barbara, CA
jinminglu@ucsb.edu	jiayi_tian@ucsb.edu	yequan_zhao@ucsb.edu	hai.li@intel.com	zhengzhang@ece.ucsb.edu

**Abstract**—Physics-Informed Neural Networks (PINNs) have emerged as a promising paradigm for solving partial differential equations (PDEs) by embedding physical laws into neural network training objectives. However, their deployment on resource-constrained platforms is hindered by substantial computational and memory overhead, primarily stemming from higher-order automatic differentiation, intensive tensor operations, and reliance on full-precision arithmetic. To address these challenges, we present a framework that enables scalable and energy-efficient PINN training on edge devices. This framework integrates fully quantized training, Stein’s estimator (SE)-based residual loss computation, and tensor-train (TT) decomposition for weight compression. It contributes three key innovations: (1) a mixed-precision training method that use a square-block MX (SMX) format to eliminate data duplication during backpropagation; (2) a difference-based quantization scheme for the Stein’s estimator that mitigates underflow; and (3) a partial-reconstruction scheme (PRS) for TT-Layers that reduces quantization-error accumulation. We further design PINTA, a precision-scalable hardware accelerator, to fully exploit the performance of the framework. Experiments on the 2-D Poisson, 20-D Hamilton–Jacobi–Bellman (HJB), and 100-D Heat equations demonstrate that the proposed framework achieves accuracy comparable to or better than full-precision, uncompressed baselines while delivering  $5.5\times$  to  $83.5\times$  speedups and  $159.6\times$  to  $2324.1\times$  energy savings. This work enables real-time PDE solving on edge devices and paves the way for energy-efficient scientific computing at scale.

**Index Terms**—Physics-Informed Neural Networks, Quantization, Tensor-Train Decomposition, On-Device Training

## I. INTRODUCTION

Physics-Informed Neural Networks (PINNs) [1] have emerged as a promising approach to solving partial differential equations (PDE) by embedding physical laws directly into the training objective of neural networks. They have been successfully applied across diverse domains, including inverse-scattering problems in nano-optics [2], [3], thermomechanical modeling [4], and control of dynamical systems in robotics [5], [6]. Unlike traditional PDE solvers, PINNs leverage data-driven learning and automatic differentiation (AD) to enforce PDE constraints, yielding mesh-free solutions that generalize across problem settings.

Despite these advantages, training PINNs is computationally and memory intensive. This is primarily due to three factors: (1) reliance on higher-order AD to compute PDE residuals;

(2) the large model sizes required to capture complex physical behavior; and (3) pervasive use of high-precision floating-point arithmetic during training because of the higher sensitivity of PINN with respect to quantization errors. For example, second-order PDEs require computing and storing numerous Jacobian and Hessian terms at every collocation point [7], [8], often consuming  $10\times$  to  $100\times$  more memory than standard neural-network training. These costs are a major barrier to deploying PINNs on resource-contained edge platforms (e.g., autonomous robotics and embedded scientific instrumentation), where memory, latency, and power budgets are tightly limited.

To mitigate the computational and memory burdens, prior work has proposed algorithmic and hardware-focused solutions. For instance, methods such as [9], [10] leverage tensor decomposition to compress network weights and reducing model size. However, these methods remain software-only and still rely on full-precision AD, providing limited end-to-end acceleration. Other efforts, such as [11], [12] propose a backpropagation-free training framework that employs zeroth-order optimization to eliminate AD and gradient backpropagation. These methods are implemented on photonic hardware accelerators, achieving notable gains in efficiency. Nevertheless, their reliance on specialized photonic ASICs restricts portability and broader adoption on conventional digital platforms.

Motivated by these limitations, we propose a holistic framework that enables scalable and energy-efficient PINN training on edge devices. Our key contributions are as follows.

- We propose an efficient on-device PINN training framework by integrating three core techniques: fully quantized training, Stein’s estimator (SE)-based residual loss computation, and tensor-train (TT) decomposition. The framework features a novel mixed-precision strategy that leverages Square-block MX-INT (SMX) formats to avoid redundant data duplication while preserving representational fidelity.
- We introduce two accuracy-preserving techniques: (i) a difference-based quantization method (DiffQuant) that decouples quantization noise from perturbations within Stein’s estimator computation; and (ii) a partial-reconstruction scheme (PRS) for TT-Layers that mitigates quantization error accumulation across tensor contractions.
- We design and evaluate a precision-scalable hardware accelerator optimized for the proposed training pipeline.

This work is co-funded by Intel Strategic Research Sectors (SRS) - Systems Integration SRS & Devices SRS.

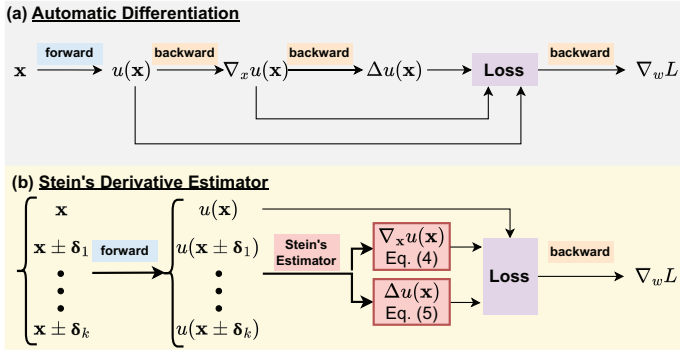


Fig. 1: Training flow of PINN with (a) Automatic Differentiation and (b) Stein's Estimator.

Implemented on 7-nm technology, the design provides up to  $83.5\times$  speedup and  $2324.1\times$  energy reduction compared to an AD-based baseline, and  $18.3\times$  speedup and  $16.0\times$  energy reduction compared to an SE-based baseline.

This work brings PINNs closer to practical deployment in constrained environments, opening the door to real-time PDE solving on edge devices and energy-efficient scientific modeling at scale.

## II. BACKGROUND

### A. Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) [1] are a class of deep learning models that integrate physical knowledge into the architecture of neural networks to solve forward and inverse problems of PDEs. Formally, we consider a generic PDE defined over a  $D$ -dimensional domain  $\Omega \subset \mathbb{R}^D$ :

$$\mathcal{F}[u(\mathbf{x})] = 0, \quad \mathbf{x} \in \Omega, \quad (1)$$

$$\mathcal{B}[u(\mathbf{x})] = 0, \quad \mathbf{x} \in \partial\Omega, \quad (2)$$

where  $\mathcal{F}$  denotes a differential operator and  $\mathcal{B}$  enforces boundary or initial conditions. A neural network  $u_\theta(\mathbf{x})$ , parameterized by weights  $\theta$ , is trained to approximate the solution  $u(\mathbf{x})$ . The training loss typically consists of physics-driven and optionally data-driven components:

$$\begin{aligned} \mathcal{L}(\theta) = & \frac{w_c}{N_c} \sum_{i=1}^{N_c} \|\mathcal{F}[u_\theta(\mathbf{x}_c^i)]\|^2 + \frac{w_b}{N_b} \sum_{i=1}^{N_b} \|\mathcal{B}[u_\theta(\mathbf{x}_b^i)]\|^2 \\ & + \frac{w_d}{N_d} \sum_{i=1}^{N_d} \|u_\theta(\mathbf{x}_d^i) - u(\mathbf{x}_d^i)\|^2, \end{aligned} \quad (3)$$

where  $w_c, w_b, w_d$  are loss weights, and  $N_c, N_b, N_d$  denote the number of data points respectively. The first two terms correspond to PDE residuals loss and boundary/initial condition loss, while the third is the regular data loss to fit the dataset. PINNs typically employ automatic differentiation (AD) to compute derivatives required for enforcing PDE constraints. However, as shown in Fig. 1(a), computing high-order derivatives via AD entails repeated backpropagation, making the process both memory- and compute-intensive, especially for high-dimensional or higher-order PDEs.

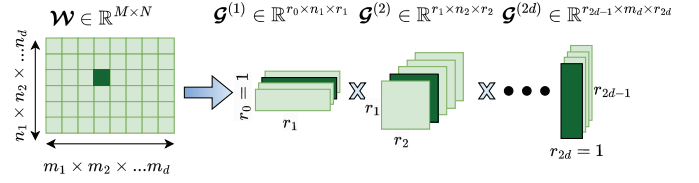


Fig. 2: Illustration of Tensor-Train Decomposition.

1) *Stein's Derivative Estimator for PDE Residuals*: To alleviate the computational burden of AD, He et al. [7] introduced Stein's derivative estimator (SE), a sampling-based forward-mode technique for derivative approximation. As shown in Fig. 1 (b), it enables derivative computation without explicit backpropagation.

For a differentiable function  $u : \mathbb{R}^d \rightarrow \mathbb{R}$ , the first-order derivative can be estimated via:

$$\begin{aligned} \nabla_{\mathbf{x}} u(\mathbf{x}) &= \mathbb{E}_{\delta} \left[ \frac{\delta}{2\sigma^2} (u(\mathbf{x} + \delta) - u(\mathbf{x} - \delta)) \right] \\ &\approx \frac{1}{K} \sum_{i=1}^K \frac{\delta_i}{2\sigma^2} (u(\mathbf{x} + \delta_i) - u(\mathbf{x} - \delta_i)), \end{aligned} \quad (4)$$

where  $\delta$  is a random perturbation sampled from  $\mathcal{N}(0, \sigma^2 \mathbf{I})$ . Higher-order derivatives such as the Laplacian can also be approximated through Eq. (5).

$$\begin{aligned} \Delta u(\mathbf{x}) &= \mathbb{E}_{\delta} \left[ \frac{\|\delta\|^2 - \sigma^2 D}{2\sigma^4} (u(\mathbf{x} + \delta) + u(\mathbf{x} - \delta) - 2u(\mathbf{x})) \right] \\ &\approx \frac{1}{K} \sum_{i=1}^K \frac{\|\delta_i\|^2 - \sigma^2 D}{2\sigma^4} \cdot (u(\mathbf{x} + \delta_i) + u(\mathbf{x} - \delta_i) - 2u(\mathbf{x})). \end{aligned} \quad (5)$$

### B. Tensor-Train Decomposition

Tensor-Train (TT) decomposition [13], [14] is a low-rank tensor factorization technique that efficiently represents high-dimensional tensors using a sequence of lower-dimensional core tensors. Originally introduced to address the curse of dimensionality in numerical computations, TT decomposition has found wide applications in machine learning for model compression and efficient inference. In the context of neural networks, TT decomposition is typically applied to fully connected layers by reshaping weight matrices into high-order tensors, followed by a low-rank decomposition in a chain structure. This approach drastically reduces the number of parameters and enables memory- and compute-efficient implementation on hardware accelerators.

Formally, consider a weight matrix  $\mathbf{W} \in \mathbb{R}^{M \times N}$ , which is first reshaped into a  $2d$ -dimensional tensor  $\mathcal{W} \in \mathbb{R}^{m_1 \times \dots \times m_d \times n_1 \times \dots \times n_d}$ , where  $N = \prod_{i=1}^d n_i$  and  $M = \prod_{i=1}^d m_i$ . As shown in Fig. 2, TT factorizes  $\mathcal{W}$  into the product of  $2d$  third-order core tensors  $\{\mathcal{G}^{(k)}\}_{k=1}^{2d}$  such that:

$$\begin{aligned} \mathcal{W}_{[i_1, \dots, i_d, j_1, \dots, j_d]} = & \sum_{r_1 \dots r_d} \mathcal{G}_{[r_0, i_1, r_1]}^{(1)} \dots \mathcal{G}_{[r_{d-1}, i_d, r_d]}^{(d)} \mathcal{G}_{[r_d, j_1, r_{d+1}]}^{(d+1)} \dots \mathcal{G}_{[r_{2d-1}, j_d, r_{2d}]}^{(2d)}. \end{aligned} \quad (6)$$

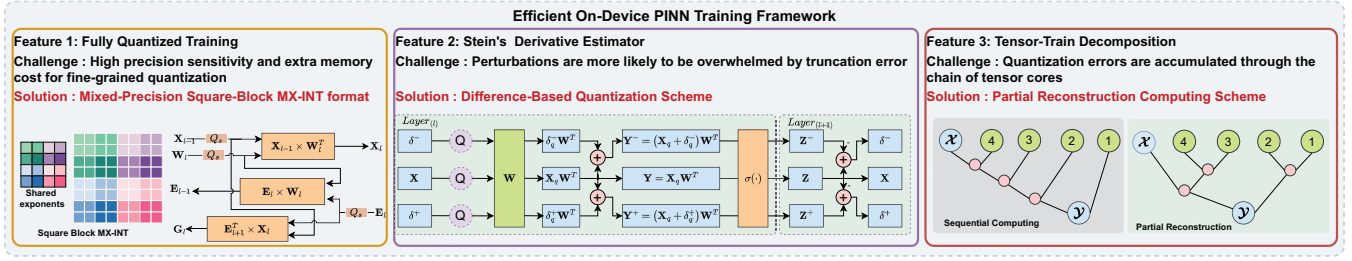


Fig. 3: Overview of the proposed efficient on-device PINN training framework.

where  $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times m_k \times r_k}$  for  $1 \leq k \leq d$ , and  $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_{k-d} \times r_k}$  for  $d < k \leq 2d$ .  $\{r_k\}_{k=1}^{2d}$  are known as TT-ranks, which determine the compression rate and expressiveness of the decomposition. By applying the TT decomposition, the standard linear layer  $\mathbf{Y} = \mathbf{X}\mathbf{W}^T$  is replaced with a TT-Layer. The TT-Layer can be expressed as:

$$\mathcal{Y}_{[b, i_1 \dots i_d]} = \sum_{j_1 \dots j_d} \mathcal{G}^{(1)}[i_1] \mathcal{G}^{(2)}[i_2] \dots \mathcal{G}^{(2d)}[j_d] \mathcal{X}_{[b, j_1 \dots j_d]}, \quad (7)$$

where  $b$  is the batch dimension,  $\mathcal{G}^{(i)}[i_k] \in \mathbb{R}^{r_{i-1} \times r_i}$  is the  $i_k$ -th slice of the TT-core  $\mathcal{G}^{(i)}$  by fixing its second index as  $i_k$ .

### III. METHOD

#### A. Overview

In this work, we develop an efficient framework for training PINNs on resource-constrained devices. To reduce the computational and memory overhead of PINN training, as illustrated in Fig. 3, the framework integrates three key components: **fully quantized training** to reduce memory footprint and arithmetic cost, **Stein's estimator** for low-cost derivative computation without backpropagation, and **TT decomposition** for compact weight representation.

However, naively combining these techniques into a cohesive, high-performance pipeline introduces several challenges. ❶ Compared with models in vision and natural language processing, PINNs are particularly sensitive to quantization-induced errors, especially due to their use in scientific domains [15]–[17]. ❷ The small perturbations used by SE-based training can be obscured by quantization noise, leading to inaccurate gradient estimates ❸ Tensorization reduces parameter count but increases the number of tensor contractions, which can exacerbate the accumulation of quantization errors.

We address these issues with **three key innovations**: (1) a mixed-precision training strategy that uses Square-block MX-INT (SMX) format to balance accuracy and efficiency; (2) a difference-based quantization scheme (DiffQuant) to preserve the sensitivity of Stein's estimator under low-bit arithmetic; and (3) a partial-reconstruction scheme (PRS) for TT-Layers that minimizes quantization error accumulation while maintaining computational efficiency. This design preserves the benefits of each component while mitigating adverse interactions among them.

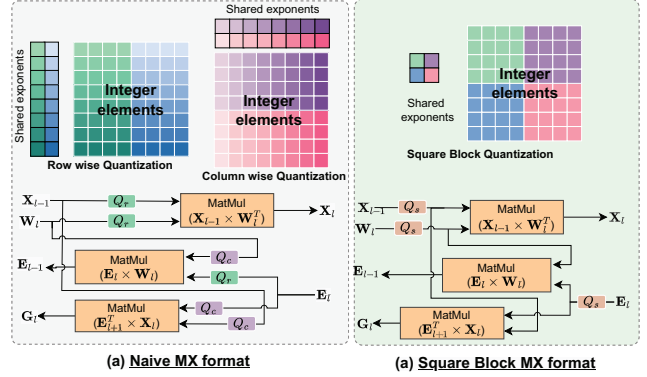


Fig. 4: Computing flow of fully quantized training with (a) MX format and (b) Square-block MX format.

#### B. Fully Quantized Training for PINNs

Microscaling (MX) data formats [18] are a class of quantization schemes that enable low-bit training and inference by assigning fine-grained per-block scaling factors. These formats have achieved excellent results across various vision and language models [19], [20]. However, conventional MX formats are directional. As shown in Fig. 4, this necessitates transposing and duplicating tensors during backpropagation, which increases both memory usage and runtime [21].

To address this limitation, we adopt the Square-block MX-INT (SMX) format, where each  $4 \times 4$  data block shares a common exponent [22]. This bidirectional design eliminates the need for redundant copies and supports efficient forward and backward passes. The quantization process is defined as

$$\begin{aligned} \text{shared\_exp}[i] &= \lfloor \log_2(\max(|\mathbf{X}[i]|)) - \text{emax} \rfloor, \\ s[i] &= 2^{\text{shared\_exp}[i]}, \\ \hat{\mathbf{X}}[i] &= s[i] \cdot \text{round}(\mathbf{X}[i]/s[i]), \end{aligned} \quad (8)$$

where  $\mathbf{X}[i]$  is  $i$ -th block of matrix  $\mathbf{X}$ , and  $\text{emax}$  is the maximum exponent value.

Given the numerical sensitivity of PINN training, we further explore mixed precision across tensor types. Our empirical analysis shows that INT8 is sufficient for activations and weights, whereas INT12 is necessary for gradients.

#### C. Difference-based Quantization for Stein's Estimator

The Stein's estimator avoids explicit computation of higher-order derivatives of  $u(\mathbf{x})$ , simplifying the quantization workflow

PDE Loss	Precision	MSE	$\ell_2$ Rel. Error
AD	FP32	1.15E-3	2.86E-3
	FP32	1.59E-3	3.93E-3
SE	W8-A8-E8	1.26E-1	3.73E-1
	W8-A32-E32	1.67E-3	2.65E-3
	W32-A8-E32	1.15E-1	3.15E-1

TABLE I: Comparison of PDE loss computation methods and numeric precision on PINN performance for Possion 2D.

to align with standard training processes. However, our preliminary experiments reveal that the performance of a fully quantized model is unsatisfactory. As shown in Table I, the quantized model exhibits significantly higher mean squared error (MSE) and  $\ell_2$  relative error compared to its full-precision counterpart. Through ablation studies, we identify that the primary source of performance degradation stems from the loss of activation precision.

This issue stems from the core mechanism of the Stein’s estimator. As illustrated in Fig. 5(a), derivative estimates are computed from the difference between  $u(\mathbf{X})$  and  $u(\mathbf{X} \pm \delta)$ , where  $\delta \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$  is a small random perturbation (typically,  $\sigma = 0.01$ ). In the quantized computation scheme shown in Fig. 5 (b), the injected noise is often smaller than the quantization step size and is consequently masked by quantization error. As a result, the quantized perturbed input  $\mathbf{X}_q^+ = Q(\mathbf{X} + \delta)$  frequently collapses to the same value as the quantized original input  $\mathbf{X}_q = Q(\mathbf{X})$ , thereby invalidating the gradient estimation via Stein’s method.

To formalize this “quantization masking” effect, we analyze the distinguishability of a scalar value  $x_i$  from its perturbed version  $x_i + \delta_i$ . Let  $Q(\cdot)$  denote a uniform quantizer with bit-width  $b$  and step size  $s = (x_{\max} - x_{\min})/(2^b - 1)$ . The quantized value is given by:

$$Q(x_i) = s \cdot \text{round}\left(\frac{x_i}{s}\right). \quad (9)$$

The difference between the quantized values,  $Q(x_i)$  and  $Q(x_i + \delta_i)$ , is non-zero only if the perturbation  $\delta_i$  forces  $x_i$  to cross the quantization threshold. This event depends on both the magnitude of  $\delta_i$  and the proximity of  $x_i$  to a threshold.

To quantify this, we calculate the probability of such a “quantization flip”. Assuming the position of  $x_i$  is uniformly distributed within its quantization bin, the distance  $l$  from  $x_i$  to the nearest quantization threshold follows a uniform distribution  $l \sim \mathcal{U}[0, s/2]$ . A flip occurs if  $|\delta_i| > l$ . Given that  $\delta_i \sim \mathcal{N}(0, \sigma^2)$ , the conditional probability of a flip is defined as:

$$\mathbb{P}(|\delta_i| > l) = 2 \cdot \Phi\left(-\frac{l}{\sigma}\right), \quad (10)$$

where  $\Phi(\cdot)$  is the cumulative distribution function (CDF) of the standard normal distribution. By marginalizing over all possible values of  $l$ , the overall flip probability is:

$$\begin{aligned} P_{\text{flip}} &= \mathbb{E}_{l \sim \mathcal{U}[0, s/2]} [\mathbb{P}(|\delta| > l)] \\ &= \frac{4}{s} \int_0^{s/2} \Phi\left(-\frac{l}{\sigma}\right) dl. \end{aligned} \quad (11)$$

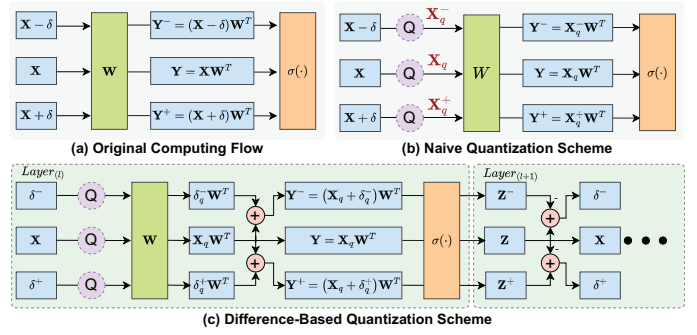


Fig. 5: Computing flow of difference-based quantization scheme.

For 8-bit quantization of data normalized to the range of  $[-1, 1]$ , this yields a flip probability of approximately 15.5%, implying that a significant portion of perturbed elements are indistinguishable after quantization. This quantization masking effect severely impairs the effectiveness of Stein’s estimators.

To resolve the quantization masking issue, we propose a difference-based quantization scheme, DiffQuant. Instead of quantizing the perturbed activation  $\mathbf{X} + \delta$  as a single term, our method quantizes the original activation  $\mathbf{X}$  and the perturbation  $\delta$  separately.

As illustrated in Fig. 5(c), the forward pass of a perturbed input through a linear layer can be decomposed:

$$\mathbf{Y}^+ = (\mathbf{X} + \delta)\mathbf{W}^T + \mathbf{b} = \mathbf{X}\mathbf{W}^T + \delta\mathbf{W}^T + \mathbf{b}. \quad (12)$$

This decomposition allows us to replace the naive quantization approach with our DiffQuant method (weight quantization is omitted for clarity):

$$\begin{aligned} \mathbf{Y}^+ &= Q(\mathbf{X} + \delta)\mathbf{W}^T + \mathbf{b} && \text{(NaiveQuant)} \\ \mathbf{Y}^+ &= Q(\mathbf{X})\mathbf{W}^T + Q(\delta)\mathbf{W}^T + \mathbf{b} && \text{(DiffQuant)} \end{aligned} \quad (13)$$

This formulation decouples the quantization of the perturbation  $\delta$  from the base activation  $\mathbf{X}$ . By doing so, it prevents the quantization error of  $\mathbf{X}$  from overwhelming the small perturbation signal, thereby preserving the accuracy of gradient estimates from the Stein’s method.

The perturbation must also be propagated through nonlinear activation functions,  $\sigma(\cdot)$ . For the subsequent layer ( $l+1$ ), the new perturbation is not  $\sigma(Q(\delta_l))$  but is instead recomputed as the difference between the activated outputs:

$$\begin{aligned} \delta_{l+1}^+ &= \sigma(\mathbf{Y}^+) - \sigma(\mathbf{Y}), \\ \delta_{l+1}^- &= \sigma(\mathbf{Y}) - \sigma(\mathbf{Y}^-). \end{aligned} \quad (14)$$

These newly computed perturbation terms,  $\delta_{l+1}^+$  and  $\delta_{l+1}^-$ , are then used to estimate the derivatives for the next layer.

#### D. Partial-Reconstruction Computing Scheme for TT Layers

Fig. 6 presents an example of a TT-Layer with  $B = 128$ ,  $m_i = [16, 16]$ ,  $n_i = [16, 16]$ , and  $r_i = [1, 8, 8, 1]$ . For clarity, Fig. 6 (a) visualizes the TT-Layer computation as a tensor network. Each node represents a multi-dimensional tensor, while the connecting edges denote contracted dimensions, often associated with tensor multiplication. The tensor network for

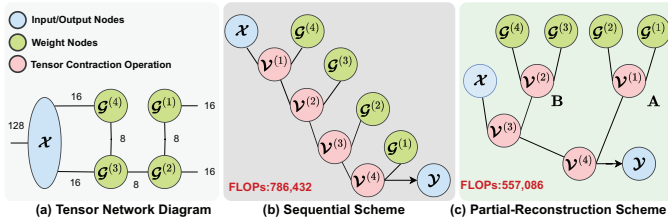


Fig. 6: Example computing schemes for a TT-Layer.

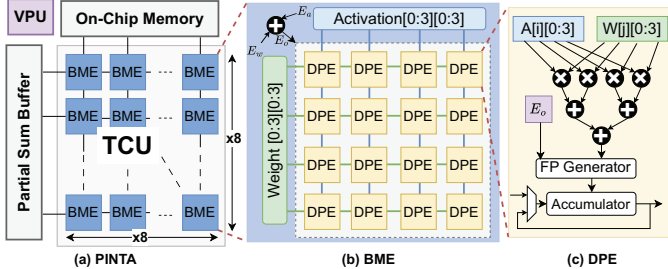


Fig. 7: Overview of PINTA architecture.

a TT-Layer consists of  $(2d + 1)$  nodes and requires  $2d$  tensor contraction operations to compute the final output.

Given a TT-Layer, there are many feasible computing orders. With perfect numerical precisions, different computing schemes are equivalent in terms of the final result. However, this equivalency is **not guaranteed** when quantization is applied. The placement of quantization operators within the TT-Layer critically affects the final output, as quantization errors accumulate along the contraction path. For example, a typical computing scheme is to perform the tensor contraction in descending order of the core tensor indices, referred to as the **sequential** scheme in Fig. 6 (b) [23]. This scheme imposes a contraction depth of  $2d$  on the activation path, leading to significant precision degradation.

To address the trade-off between accuracy and efficiency in quantized TT-Layers, we propose a novel computation strategy termed the Partial-Reconstruction Scheme (PRS). As illustrated in Fig. 6 (c), the computation involves three steps:

- 1 Output dimension reconstruction: core tensors associated with the output dimensions (i.e.,  $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(d)}$ ) are contracted to form a partial weight matrix  $\mathbf{A} \in \mathbb{R}^{r_a \times M}$ .
- 2 Input dimension reconstruction: core tensors associated with the input dimensions (i.e.,  $\mathcal{G}^{(d+1)}, \mathcal{G}^{(d+2)}, \dots, \mathcal{G}^{(2d)}$ ) are contracted to construct another partial matrix  $\mathbf{B} \in \mathbb{R}^{N \times r_a}$ .
- 3 Input contraction: the input node is sequentially contracted with  $\mathbf{A}$  and  $\mathbf{B}$  to obtain the final output:  $\mathbf{Y} = \mathbf{X} \times \mathbf{B} \times \mathbf{A}$ .

#### IV. HARDWARE ARCHITECTURE

To validate the performance of the proposed training framework, we designed and implemented an efficient hardware architecture, namely **Physics-Informed Neural Training Accelerator** (PINTA). As depicted in Fig. 7, PINTA comprises a Tensor Contraction Unit (TCU), a 32-way vector processing unit (VPU), a partial sum buffer, and on-chip memory banks.

TCU executes variable-precision tensor contractions using an  $8 \times 8$  array of Block Matrix computation Engines (BMEs). This array is organized as a transposable systolic array [24] to support the flexible dataflows required by TT training. Each BME performs a block of SMX matrix multiplication under a shared exponent, issuing  $4 \times 4 \times 4 = 64$  MACs per cycle. Internally, a BME is composed of a  $4 \times 4$  array of Dot-Product Engines (DPEs).

Each DPE performs four  $\text{INT4} \times \text{INT4}$  multiplications per cycle along the reduction dimension within each data blocks. The immediate result is converted to a floating-point format using a shared exponent ( $E_o = E_w + E_a$ ) and is then accumulated with previous partial sums. When operand precision is greater than 4 bits, the DPE accumulates partial sums bit-serially. This mechanism enables precision-scalable arithmetic while preserving block floating-point semantics. For instance, an  $\text{INT8} \times \text{INT8}$  operation requires four cycles to complete. Upon completion of each tensor contraction, the VPU processes subsequent operations, such as nonlinear activations or quantization.

## V. EXPERIMENTS

### A. Evaluation Setting

We evaluate our method on three representative PDE benchmarks of varying dimensionality: the 2D Poisson equation, the 20D Hamilton–Jacobi–Bellman (HJB) equation, and the 100D Heat equation. All models are trained on a single NVIDIA RTX A6000 Ada GPU using the Adam optimizer with a learning rate of  $1e-3$  for 1,000 iterations. The number of samples for Stein’s estimator is set to 512.

#### Poisson Equation:

$$\begin{cases} \Delta u(x) = g(x) & x \in \Omega \\ u(x) = h(x) & x \in \partial\Omega, \end{cases} \quad (15)$$

where  $\Omega = [0, 1] \times [0, 1]$  is the domain of the problem,  $g(x) = -\sin(x_1 + x_2)$  and  $h(x) = 1/2\sin(x_1 + x_2)$ . The base network is a 4-layer MLP with 256 neurons and  $\tanh$  activation function.

#### HJB Equation:

$$\begin{cases} \partial_t u(x, t) + \Delta u(x, t) - 0.5 \|\nabla_x u(x, t)\|_2^2 = -2, \\ u(x, 1) = \|x\|_1, & x \in [0, 1]^{20}, \quad t \in [0, 1]. \end{cases} \quad (16)$$

A 4-layer MLP with 512 neurons is utilized [11].

#### Heat Equation:

$$\begin{cases} u_t(x, t) = \Delta u(x, t) & x \in B(0, 1), \quad t \in (0, 1) \\ u(x, 0) = \|x\|^2/2N & x \in B(0, 1) \\ u(x, t) = t + 1/2N & x \in \partial B(0, 1), \quad t \in [0, 1] \end{cases} \quad (17)$$

The base network is a 4-layer MLP with 256 neurons.

### B. Accuracy Evaluation

We compare our proposed method against two primary baselines: a full-precision, full-rank model trained with Automatic Differentiation (AD-FP-FR) and its counterpart trained with the Stein’s Estimator (SE-FP-FR). The accuracy results are summarized in Table II, where we report mean squared

TABLE II: Experimental results on PDE solving.

		Method	MSE	$\ell_1$ Rel. Error	$\ell_2$ Rel. Error
Poisson	AD-FP-FR		1.15E-03	2.45E-03	2.86E-03
	SE-FP-FR		1.59E-03	3.10E-03	3.93E-03
	Ours	$R = 8$	3.28E-03	6.83E-03	8.16E-03
		$R = 16$	2.27E-03	4.42E-03	5.65E-03
		$R = 32$	1.96E-03	3.75E-03	4.88E-03
HJB	AD-Full		9.11E-02	6.51E-03	8.61E-03
	SE-Full		4.18E-02	3.37E-03	3.95E-03
	Ours	$R = 8$	4.65E-02	3.83E-03	4.38E-03
		$R = 16$	1.93E-02	1.58E-03	1.82E-03
		$R = 32$	2.47E-02	1.68E-03	2.33E-03
Heat	AD-Full		4.28E-03	7.33E-03	7.33E-03
	SE-Full		3.66E-03	6.28E-03	6.28E-03
	Ours	$R = 8$	4.37E-03	7.46E-03	7.46E-03
		$R = 16$	4.93E-03	8.48E-03	8.48E-03
		$R = 32$	4.60E-03	7.86E-03	7.86E-03

TABLE III:  $\ell_2$  relative error of different methods.

Problem	Poisson 2D	HJB 20D	HEAT 100D
AD-FP-FR	2.86E-03	8.61E-03	7.33E-03
SE-FP-FR	3.93E-03	3.95E-03	6.28E-03
SE-NaiveQuant	3.19E-01	3.82E-02	5.90E-02
SE-DiffQuant	2.21E-03	4.15E-03	6.45E-03
SE-FP-R16	5.93E-03	5.68E-3	7.39E-03
SE-TT-Seq-DiffQuant	1.23E-02	2.00E-02	1.78E-01
SE-TT-PRS-DiffQuant	5.65E-03	1.82E-03	8.48E-03

error (MSE),  $\ell_1$  relative error, and  $\ell_2$  relative error. The results demonstrate that our tensorized, fully quantized training strategy achieves accuracy comparable to, or in some cases exceeding, the uncompressed baselines.

We also analyze the effect of the TT-rank selection. A smaller rank yields a higher model compression ratio but can also lead to a larger approximation error. This highlights the crucial trade-off between model efficiency and predictive accuracy that must be considered when selecting the rank.

1) *Ablation Study*: The  $\ell_2$  relative error of different design choices, using a TT-rank of 16, are summarized in Table III. The SE-NaiveQuant method, which directly quantizes activations, exhibits significantly higher error compared to other configurations. In contrast, the SE-DiffQuant method successfully preserves the accuracy of the full-precision model.

The SE-TT-Seq-DiffQuant variant, which uses a sequential contraction order, suffers from substantial error accumulation due to the sequential contraction order in the TT-Layer, resulting in degraded performance. By adopting PRS, the SE-TT-PRS-DiffQuant method mitigates this issue and achieves a more favorable trade-off between accuracy and efficiency.

### C. Hardware Performance

We implemented PINTA in SystemVerilog RTL and synthesized it using the ASAP 7nm technology. The resulting prototype occupies 0.442 mm<sup>2</sup>, contains 384-KB on-chip memory, and operates at 1.0 GHz. The area and power of on-chip SRAM were modeled using PRACTI [25], while the

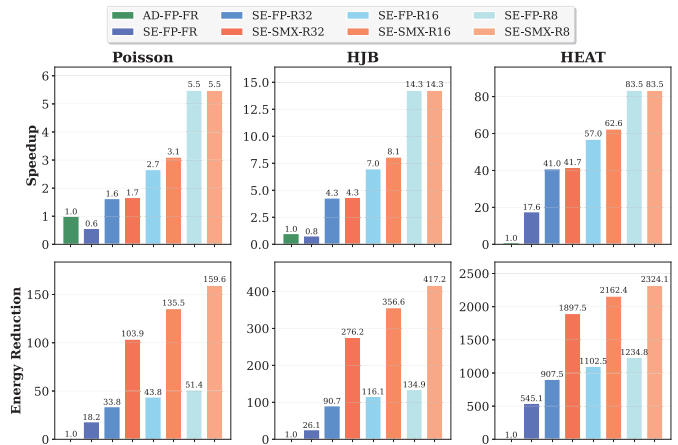


Fig. 8: Hardware performance comparison under different settings.

HBM2 memory was modeled with an access energy of 3.9 pJ/bit and a bandwidth of 256 GB/s.

The performance of AD-FP-FR baseline was evaluated on an NVIDIA RTX 4060 laptop GPU with 70W TDP. Training the model on the Poisson, HJB, and Heat equations required 25.96s, 134.30s, and 450.63s, respectively. To assess the full-precision SE models, We also implement a systolic array architecture with  $32 \times 32$  FP32 MACs units. This was used benchmark SE models with different TT-ranks, including SE-FP-FR and SE-FP-R\*. In contrast, models using our proposed framework (SE-SMX-R\*) were evaluated on the PINTA accelerator.

Fig. 8 shows a comparison of training speed and energy consumption across the different configurations. Lower TT-ranks (from  $R=32$  down to  $R=8$ ) and our SMX configuration (SE-SMX-R\*) yield monotonic performance improvements across all benchmarks. At a rank of  $R = 8$ , the proposed system achieves  $5.5\times$  to  $83.5\times$  speedups, together with  $159.6\times$  to  $2324.1\times$  reductions in energy consumption compared to AD-FP-FR baseline running on GPU. Compared to the SE-FP-FR configurations, SE-SMX-R8 improves speed by  $4.7\times$  to  $18.3\times$  and energy efficiency  $4.3\times$  to  $16.0\times$ , respectively. For any given rank, PINTA achieves  $1.88\times$  to  $3.10\times$  energy reductions over its full-precision SE counterpart (SE-FP-R\*).

## VI. CONCLUSION

This paper presents an efficient framework for solving high-dimensional PDEs with PINNs by integrating a Stein’s derivative estimator, tensor-train decomposition, and fully quantized training. Our framework utilizes a mixed-precision, square-block MX-INT format to ensure high representational fidelity and memory efficiency. We introduce two novel techniques to maintain accuracy in this low-precision setting: a difference-based quantization scheme that preserves the sensitivity of the Stein’s estimator, and a partial-reconstruction scheme that mitigates error accumulation during TT-Layer computations. Experimental results demonstrate that our proposed framework significantly improves training efficiency over full-precision baselines without compromising predictive accuracy.

## REFERENCES

- [1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.
- [2] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro, "Physics-informed neural networks or inverse problems in nano-optics and metamaterials," *Optics express*, vol. 28, no. 8, pp. 11618–11633, 2020.
- [3] Y. Chen and L. Dal Negro, "Physics-informed neural networks for imaging and parameter retrieval of photonic nanostructures from near-field data," *APL Photonics*, vol. 7, no. 1, 2022.
- [4] T. N. K. Nguyen, T. Dairay, R. Meunier, and M. Mougeot, "Physics-informed neural networks for non-newtonian fluid thermo-mechanical problems: An application to rubber calendaring process," *Engineering Applications of Artificial Intelligence*, vol. 114, p. 105176, 2022.
- [5] E. A. Antonelo, E. Camponogara, L. O. Seman, J. P. Jordanou, E. R. de Souza, and J. F. Hübner, "Physics-informed neural nets for control of dynamical systems," *Neurocomputing*, vol. 579, p. 127419, 2024.
- [6] M. Velioglu, S. Zhai, S. Rupprecht, A. Mitsos, A. Jupke, and M. Dahmen, "Physics-informed neural networks for dynamic process operations with limited physical knowledge and data," *Computers & Chemical Engineering*, vol. 192, p. 108899, 2025.
- [7] D. He, S. Li, W. Shi, X. Gao, J. Zhang, J. Bian, L. Wang, and T.-Y. Liu, "Learning physics-informed neural networks without stacked back-propagation," in *International conference on artificial intelligence and statistics*, pp. 3034–3047, PMLR, 2023.
- [8] Z. Hu, Z. Shi, G. E. Karniadakis, and K. Kawaguchi, "Hutchinson trace estimation for high-dimensional and high-order physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 424, p. 116883, 2024.
- [9] Z. Liu, X. Yu, and Z. Zhang, "TT-PINN a tensor-compressed neural PDE solver for edge computing," in *ICML Workshop on Hardware-aware Efficient Training*, 2022.
- [10] S. K. Vemuri, T. Büchner, J. Niebling, and J. Denzler, "Functional tensor decompositions for physics-informed neural networks," in *International Conference on Pattern Recognition*, pp. 32–46, Springer, 2025.
- [11] Y. Zhao, X. Yu, X. Xiao, Z. Chen, Z. Liu, G. Kurczveil, R. G. Beausoleil, S. Liu, and Z. Zhang, "Scalable back-propagation-free training of optical physics-informed neural networks," *arXiv preprint arXiv:2502.12384*, 2025.
- [12] Y. Zhao, X. Xiao, X. Yu, Z. Liu, Z. Chen, G. Kurczveil, R. G. Beausoleil, and Z. Zhang, "Real-time fj/mac pde solvers via tensorized, back-propagation-free optical pinn training," in *Machine Learning with New Compute Paradigms*.
- [13] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
- [14] J. Tian, J. Lu, H. Li, X. Wang, I. Young, Z. Zhang, *et al.*, "Ultra memory-efficient On-FPGA training of transformers via tensor-compressed optimization," *arXiv preprint arXiv:2501.06663*, 2025.
- [15] R. Tu, C. White, J. Kossaiji, B. Bonev, N. Kovachki, G. Pekhimenko, K. Azizzadenesheli, and A. Anandkumar, "Guaranteed approximation bounds for mixed-precision neural operators," *arXiv preprint arXiv:2307.15034*, 2023.
- [16] W. van den Dool, T. Blankevoort, M. Welling, and Y. M. Asano, "Efficient neural pde-solvers using quantization aware training," *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 1415–1424, 2023.
- [17] J. Hayford, J. Goldman-Wetzler, E. Wang, and L. Lu, "Speeding up and reducing memory usage for scientific machine learning via mixed precision," *Computer Methods in Applied Mechanics and Engineering*, vol. 428, p. 117093, 2024.
- [18] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, *et al.*, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023.
- [19] Y. Kim, C. Oh, J. Hwang, W. Kim, S. Oh, Y. Lee, H. Sharma, A. Yazdanbakhsh, and J. Park, "Dacapo: Accelerating continuous learning in autonomous systems for video analytics," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1246–1261, IEEE, 2024.
- [20] J. Zhang, J. Wei, P. Zhang, X. Xu, H. Huang, H. Wang, K. Jiang, J. Zhu, and J. Chen, "Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training," *arXiv preprint arXiv:2505.11594*, 2025.
- [21] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar, *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–13, 2023.
- [22] S. Cuyckens, X. Yi, N. S. Murthy, C. Fang, and M. Verhelst, "Efficient precision-scalable hardware for microscaling (MX) processing in robotics learning," *arXiv preprint arXiv:2505.22404*, 2025.
- [23] Y. Gong, M. Yin, L. Huang, J. Xiao, Y. Sui, C. Deng, and B. Yuan, "ETTE: efficient tensor-train-based computing engine for deep neural networks," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–13, 2023.
- [24] J. Lu, J. Tian, H. Li, I. Young, and Z. Zhang, "FETTA: flexible and efficient hardware accelerator for tensorized neural network training," *arXiv preprint arXiv:2504.06474*, 2025.
- [25] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI*, pp. 290–295, IEEE, 2014.