FTD-GNO: MEMORY-EFFICIENT GRAPH NEURAL OPERATORS THROUGH FUNCTIONAL TENSOR DECOMPOSITION OF THE KERNEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Operators (GNOs) provide flexible surrogate models for learning solution operators of partial differential equations (PDEs). However, standard GNOs typically parameterize the integral kernel with a monolithic neural network and evaluate kernel interactions over graph edges, leading to substantial computational and memory overhead at high resolutions or with large neighborhoods. To address these limitations, we propose Functional Tensor Decomposition Graph Neural Operator (FTD-GNO), a memory-efficient GNO framework that decouples the high-dimensional continuous integral kernel into low-dimensional mode-wise functions. By instantiating the kernel with classical tensor decomposition formats, including CP, Tensor-Train, and Tucker decompositions, FTD-GNO enables algebraic reconstruction of the integral operator without explicitly materializing full edge-wise kernel tensors. This factorized formulation reduces the memory footprint of kernel evaluation and aggregation while retaining the continuous operator-learning structure of GNOs. Theoretical complexity analysis shows that FTD-GNO substantially lowers parameter and activation-memory costs associated with high-dimensional kernel construction. Experiments show lower peak memory than the corresponding unfactorized graph-integral baselines, with shorter recorded training times. Fourier–graph experiments further demonstrate that FTD can improve the efficiency of a graph-integral layer within a hybrid operator and has good scalability.

1 INTRODUCTION

Partial differential equations (PDEs) are the core mathematical tools for characterizing the spatiotemporal evolution of complex physical systems (Zachmanoglou & Thoe, 1986; Jones et al., 2009). In practical scientific and engineering applications, such as uncertainty quantification, optimal control, and inverse problem solving, it is often necessary to perform massive, repeated evaluations of PDEs driven by varying physical parameters. Traditional numerical solvers (e.g., finite element (Zienkiewicz & Taylor, 2005; Hildebrand & Klinge, 2024) or finite difference methods (Godunov & Bohachevsky, 1959; Duffy, 2022) rely heavily on high-resolution grids, and their prohibitive iterative computational overhead struggles to meet the demands of real-time inference in complex systems (Huang et al., 2026; Parvizi et al., 2023). Recently, operator learning has emerged as a novel deep learning paradigm. Unlike conventional neural networks that only learn mappings between finite-dimensional vectors, neural operators (NOs) (Kovachki et al., 2023; Kossaifi et al., 2024) aim to directly construct nonlinear mappings between infinite-dimensional continuous function spaces. This discretization-independent architecture endows models with the remarkable capability of zero-shot cross-resolution generalization.

Graph Neural Operators (GNOs) (Li et al., 2020b) discretize continuous spaces into graph topologies and approximate integral transforms via message-passing mechanisms, demonstrating unparalleled flexibility in handling unstructured grids and complex geometric domains. However, when confronted with high-dimensional systems and high-resolution grids, the underlying architecture of standard GNOs exposes intertwined flaws. Specifically, during the integral kernel evaluation and spatial aggregation phases, information interaction relies heavily on edge-based message-passing mechanisms. This implies that the grid must explicitly construct, instantiate and store the complete

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

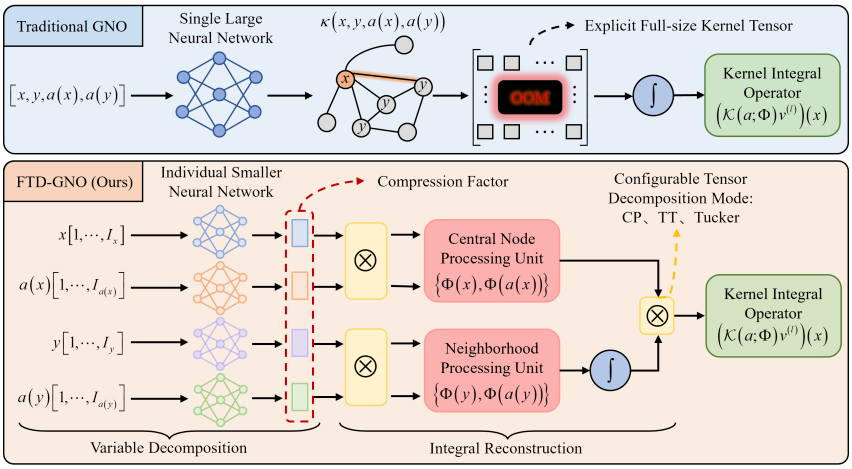


Figure 1: Efficient GNO via Function Tensor Decomposition: GNO calculates the integral kernel through a single large neural network and explicitly constructs the complete integral kernel tensor on each edge. The proposed methodology involves the decomposition of the integral kernel into multiple single-variable neural networks, in conjunction with the utilisation of configurable tensor decomposition modes. This approach facilitates the reconstruction of the integral operator, thereby circumventing the occurrence of memory overflow.

integral kernel tensor on each edge within the computational graph. As spatial dimensionality increases or grid nodes become denser, the memory requirement to construct the full input exhibits explosive exponential growth, manifesting as the "curse of dimensionality". Such edge-level storage dependency severely limits network depth and renders GNOs highly susceptible to Out-Of-Memory (OOM) (Zhou et al., 2021a) errors, profoundly restricting their practical utility. Furthermore, to adequately capture the complex feature couplings in multi-dimensional spaces, the hidden layer dimensions of the network are forced to expand drastically, triggering a nonlinear surge in model parameters.

To address the aforementioned challenges, this paper proposes a novel Graph Neural Operator framework based on Functional Tensor Decomposition (FTD-GNO). The essence of this framework lies in the deep decoupling of the high-dimensional tensor construction process within the continuous domain. The overall schematic of this setup is illustrated in Figure 1. Specifically, we elegantly strip the originally massive full-scale network inputs into low-dimensional variable sets corresponding to independent physical dimensions. Under the multi-branch architecture of FTD-GNO, lightweight subnetworks tailored for specific dimensions only need to focus on the 1D feature objects within their respective dimensions. Additionally, we leverage the associativity of tensor decomposition to perform an algebraic reconstruction of the spatial integral operators, with source-dependent factors aggregated before their combination with query-dependent factors.

This decomposition and reconstruction strategy thoroughly circumvents the explicit computation of full-size tensors. Without sacrificing the continuous spatial expressive capacity of the neural operator, it successfully compresses the memory complexity during the kernel evaluation and spatial integral aggregation stages from exponential to linear growth. Concurrently, benefiting from the low-rank factorized decoupling of multi-dimensional variables by the integral kernel, this architecture successfully addresses the bottleneck of exponential parameter explosion with respect to dimensionality. This has resulted in a substantial streamlining of the parameter space of the model.

Under the proposed framework, we focus on combining Graph Neural Operators with three prominent tensor decomposition techniques, including *Canonic-Polyadic (CP)* (Hitchcock, 1927), *Tensor-Train (TT)* (Oseledets, 2011) and *Tucker* decomposition (Tucker, 1966). The main contributions of our work are summarized as follows:

- **Parameter Compression via Tensor Decoupling:** We integrate functional tensor decomposition into GNOs to decouple high-dimensional kernels into low-dimensional variables, drastically slashing the parameter scale.

- **Memory Optimization via Algebraic Reconstruction:** We algebraically reconstruct the integral operator to shift from edge-based message passing to node-based computation, greatly reducing memory costs.
- **Comprehensive Empirical Validation:** We demonstrate the highly competitive performance of the proposed method compared to GNOs in PDE benchmarks providing compelling evidence that firmly corroborates our theoretical analysis.

2 RELATED WORKS

Neural Operators Distinguished from traditional surrogate models for PDEs, the core advantage of neural operators lies in their construction of a universal approximator defined between function spaces. According to their architectures, they can be broadly categorized into three families (Liu et al., 2025). The first family comprises operator architectures based on the universal approximation theorem. DeepONet (Lu et al., 2021) pioneered a dual-path network architecture, encoding discretized sensor data of input functions via a branch network and processing target query coordinates via a trunk network, thereby laying the foundational theory for operator learning. Subsequently, architectures based on integral kernel transforms further expanded the boundaries of this field. Among them, Fourier Neural Operator (FNO) (Li et al., 2020a) innovatively maps the integral kernel into the Fourier frequency domain, leveraging the Fast Fourier Transform (FFT) to extract low-frequency modes, thus achieving highly computationally efficient global convolutions. Meanwhile, Graph Neural Operator (GNO) (Li et al., 2020b) employs graph neural networks to perform Monte-Carlo integral approximations within the spatial domain, effectively modeling nonlocal operators on complex geometric domains. In recent years, models represented by GK-Transformer (Cao, 2021) and General Neural Operator (GNOT) (Hao et al., 2023) have further integrated attention mechanisms into operator architectures, significantly enhancing the capability to capture features in multi-scale physical fields.

Functional Tensor Decomposition in Deep Learning Classical tensor decomposition methods have long been widely utilized to compress high-dimensional data matrices and reduce model ranks (Zhou et al., 2021b; Zhang et al., 2021), including CP (Hitchcock, 1927), TT (Oseledets, 2011), and Tucker (Tucker, 1966) decomposition. In recent years, these algebraic tools have been introduced into deep learning models within the continuous function domain, spawning the emerging research direction of functional tensor decomposition (FTD). In the field of Implicit Neural Representations (INRs), frameworks such as F-INR (Vemuri et al., 2026) dismantle the massive Multi-Layer Perceptrons (MLPs) used for processing high-dimensional coordinates into a series of axially independent, lightweight subnetworks. This greatly alleviates the curse of dimensionality encountered when fitting complex videos or high-resolution scenes, and significantly accelerates convergence. Similarly, in Physics-Informed Neural Networks (PINNs) (Vemuri et al., 2025) for solving PDEs, employing tensor decomposition formats to separate spatial and temporal variables not only improves the network's ability to fit gradients within complex loss landscapes but also effectively circumvents the computational dissipation caused by high-dimensional dense sampling. These advancements thoroughly validate the immense potential of tensor decomposition in handling high-dimensional mappings in continuous domains.

Tensor Decomposition for Operator Learning As operator learning is applied to increasingly complex scientific computing scenarios, controlling its parameter scale and memory footprint has become a recent research hotspot. To this end, researchers have begun exploring the integration of various tensor dimensionality reduction and compression techniques into advanced neural operator variants. In terms of frequency-domain transformation dimensionality reduction, the Factorized FNO (F-FNO) (Tran et al., 2021) introduces fourier factorization to process each spatial dimension orthogonally and independently in the frequency domain. This successfully reduces model complexity by an order of magnitude while enabling stable scaling to deeper network structures. Following this dimensionality reduction paradigm, D-FNO (Li & Ye, 2025) decomposes high-dimensional latent representations into a series of rank-1 tensor products, converting 3D FFTs into 1D FFTs, thereby further reducing computational complexity. To address the parameter redundancy of global convolution kernels, architectures like Tucker-FNO (Zhou et al., 2026) and MG-TFNO (Kossaifi et al., 2023) impose strict low-rank tensor constraints directly on the weight matrices, aiming to capture richer frequency-domain features while simultaneously reducing parameter counts. In the dimension of spatial basis extraction, POD-DeepONet (Lu et al., 2022) leverages proper orthogonal

decomposition to pre-extract the dominant modes of the data as basis functions, significantly elevating the representational efficiency of the operator through effective spatial dimensionality reduction. Furthermore, the advantages of tensor decomposition have even extended to the optimization and solving stages of operator models. TensorGRaD (Loeschcke et al., 2025) decomposes gradients into the sum of low-rank and sparse tensors, efficiently capturing both global trends and high-frequency outliers in the evolutionary states of optimizers.

3 Proposed Methods

3.1 The Setting of GNOs

Neural Operators learn solution operators associated with PDEs, mapping input functions to output functions in function spaces. This formulation supports mesh-independent, high-fidelity, and instantaneous inference after training. The detailed PDE configurations are deferred to Appendix A.

Formally, let $\mathcal{A}$ and $\mathcal{U}$ be Banach spaces of functions defined on bounded domains $D \subset \mathbb{R}^{d_x}$, where $\mathcal{A}$ represents the input function space and $\mathcal{U}$ represents the objective function space. The purpose of operator learning is to approximate the ground-truth mapping $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ by learning a neural operator $\mathcal{G}_\theta$ from a set of observations $\{a_i, u_i\}_{i=1}^N$. The mapping $\mathcal{G}_\theta$ is defined as $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}, \theta \in \mathbb{R}^p$. We consider the loss function L and learn neural operators by solving the problem

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} L\left(\mathcal{G}_\theta\left(a_i\right), u_i\right). \tag{1}$$

A standard Neural Operator (NO) framework consists of a lifting operator, iterative NO layers, and a projection operator. For $l = 1, 2, \ldots, L$, each NO layer updates the node feature $v^{(l)}$ via:

$$v^{(l+1)}\left(x\right) := \sigma\left(W^{(l)} v^{(l)}\left(x\right) + \left(\mathcal{K}\left(a; \phi\right) v^{(l)}\right)\left(x\right)\right), \forall x = \left(x_1, ..., x_{d_x}\right) \in D, \tag{2}$$

where σ is a nonlinear activation function, $W^{(l)} v^{(l)}\left(x\right)$ is the residual connection, and $\mathcal{K}$ represents the kernel integration operator, which is parameterized with ϕ by a neural network as

$$\left(\mathcal{K}\left(a; \phi\right) v^{(l)}\right)\left(x\right) := \int_D \kappa\left(x, y, a\left(x\right), a\left(y\right); \phi\right) v^{(l)}\left(y\right) dy, \forall x = \left(x_1, ..., x_{d_x}\right) \in D, \tag{3}$$

where κ is an integral kernel. We assume that dy defaults to the Lebesgue measure on $\mathbb{R}^{d_x}$.

Consequently, NO can be expressed as

$$\mathcal{G}_\theta := \mathcal{Q} \circ \mathcal{L}^{(L)} \circ \mathcal{L}^{(L-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{P}. \tag{4}$$

Standard GNOs realize this continuous integration through message passing mechanism of the graph neural network (Bruna et al., 2013). To handle unstructured grids and manage computational overhead, GNOs employ domain truncation (restricting integration to a local radius r) and the Nyström approximation (Nyström, 1930) (subsampling nodes for Monte-Carlo integration). These approximations fundamentally convert the continuous integral into discrete message-passing aggregations (Gilmer et al., 2017) over a constructed graph topology. The extended mathematical formulations of these standard GNO approximation techniques are detailed in Appendix B.

However, although GNOs effectively reduce the computational cost of global integration by introducing domain truncation and Nyström approximation, thereby enabling it to handle unstructured grid data, it still faces severe challenges when faced with high-dimensional input spaces or complex integration kernel modeling. The primary impediment pertains to the parameterization approach of its integral kernel κ. Traditional GNO relies on a monolithic architecture neural network such as MLP to approximate high-dimensional kernel functions. As the input dimension increases, the number of parameters in this network grows exponentially, which is known as the "curse of dimensionality". Furthermore, during the spatial aggregation phase, the grid must explicitly construct and store the complete high-dimensional integral kernel tensor on every single edge. This edge-level dependency triggers an exponential explosion in parameter count and memory footprint, thereby exposing the fatal memory flaws of standard GNOs.

3.2 FUNCTIONAL TENSOR DECOMPOSITION FOR GNO

As discussed above, the kernel integral operator in the standard GNO relies on a single, large neural network to parameterize the integral kernel. This design leads to substantial computational complexity. To address this issue, we incorporate functional tensor decomposition (FTD) into GNO and propose the FTD-GNO model. The core idea is to represent the high-dimensional, parameter-intensive kernel as a composition of low-dimensional, low-rank component functions. This method significantly reduces the number of parameters while preserving the dominant components of the tensor structure, thereby enabling effective extraction of key features from the physical field.

3.2.1 FUNCTIONAL TENSOR DECOMPOSITION OF THE KERNEL

Let the generalized input to the kernel be denoted by the tuple $z = (z_1, \ldots, z_d)$, where d represents the number of physical input variable blocks. For our general integral kernel formulation, the input variables correspond to vector blocks of query point, the neighboring point, and their associated field values, i.e., $z = (x, y, a(x), a(y))$, hence we have $d = 4$. Assuming each variable block z_i is located/evaluated across I_i discrete elements, the fully instantiated integral kernel will essentially form a massive d-dimensional tensor of size $I_1 \times I_2 \times \cdots \times I_d$. Instead of using a single monolithic network to explicitly construct this dense tensor, we decouple the integral kernel $\kappa(z; \Phi)$ into a composition of d smaller neural networks:

$$\kappa(z; \Phi) = \text{Decomp}\left(\Phi^{(1)}(z_1; \theta_1), \ldots, \Phi^{(d)}(z_d; \theta_d)\right), \tag{5}$$

where $\Phi^{(i)}(\cdot)$ represents a univariate neural network processing the i-th variable block z_i with learnable parameter θ_i, and $\text{Decomp}(\cdot)$ denotes a classical tensor decomposition structure. Taking the full state $z = (x, y, a(x), a(y))$ as the standard example:

$$\kappa(z; \Phi) = \text{Decomp}\left(\Phi^{(1)}(x; \theta_1), \Phi^{(2)}(y; \theta_2), \Phi^{(3)}(a(x); \theta_3), \Phi^{(4)}(a(y); \theta_4)\right). \tag{6}$$

According to different PDEs, i.e. the dependence of u on function a, it is necessary to select different types of integral kernel. The complete mathematical formulations for functional tensor decomposition of these general kernels are detailed in Appendix C. In this work, we consider three established decomposition types in tensor theory, including CP (Hitchcock, 1927), TT (Oseledets, 2011) and Tucker decomposition (Tucker, 1966).

CP-GNO: The CP decomposition models the integral kernel as the sum of outer products of multiple rank-one functional tensors. The d factor networks generate functional vectors, and the integral kernel is explicitly reconstructed as

$$\kappa(z_1, \ldots, z_d; \Phi) = \sum_{r=1}^{R} A_r^{(1)}(z_1) \circ A_r^{(2)}(z_2) \circ \cdots \circ A_r^{(d)}(z_d), \tag{7}$$

where $A_r^{(i)}(z_i)$ explicitly denotes the continuous function of the r-th component for the i-th variable block. Here, R is the rank of the CP decomposition and $\circ$ denotes the outer product.

TT-GNO: The TT decomposition represents a tensor as a series of kernel tensors that are linked by chains. For $i = 1, 2, \cdots, d$, d factor networks are used to generate core tensors, and then the integral kernel is reconstructed as

$$\kappa(z_1, \ldots, z_d; \Phi) = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{d-1}=1}^{R_{d-1}} A^{(1)}_{i_1, r_0, r_1}(z_1) A^{(2)}_{i_2, r_1, r_2}(z_2) \cdots A^{(d)}_{i_d, r_{d-1}, r_d}(z_d), \tag{8}$$

where R_i represents the TT rank between modes i and $i+1$, and $R_0 = R_d = 1$ is satisfied.

Tucker-GNO: The Tucker decomposition represents a tensor as the product of a core tensor and multiple factor matrices. Given Tucker rank $\{R_i\}_{i=1}^d$, the Tucker-GNO uses d factor networks to generate the factor matrix for each mode. Then, the integral kernel is reconstructed as

$$\kappa(z_1, \ldots, z_d; \Phi) = \mathcal{C} \times_1 A^{(1)}(z_1) \times_2 A^{(2)}(z_2) \cdots \times_d A^{(d)}(z_d), \tag{9}$$

where $\mathcal{C} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_d}$ is the learnable core tensor and $\times_i$ represents the mode-i tensor-matrix product.

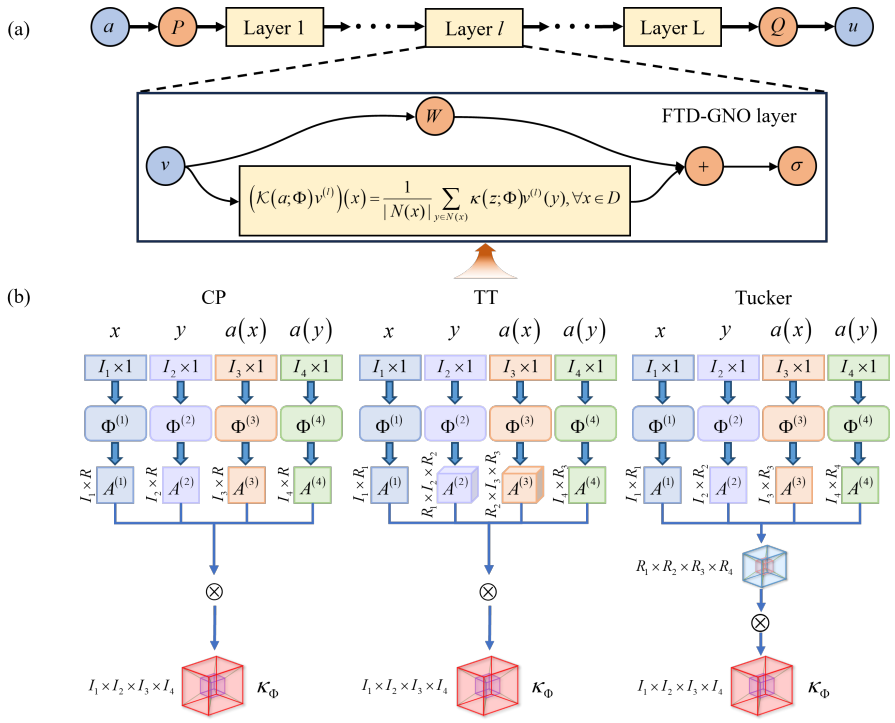


Figure 2: (a) The full architecture of proposed neural operator; (b) example of integral kernel parameterization with Functional Tensor Decompositions on a integral type of four variables.

For the above GNO based on function tensor decomposition, these factor matrices/core tensors are learned and parameterized through d small neural networks over their block variables, respectively. Subsequently, the reconstructed tensor, i.e. the integration kernel $\kappa(z;\Phi)$, is used in the integration operation of the kernel integration operator.

3.2.2 Model Architecture

Our FTD-GNO adheres to the standard GNO architecture, which is

$$\mathcal{G}_\theta := \mathcal{Q} \circ \mathcal{L}^{(L)} \circ \mathcal{L}^{(L-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{P}, \tag{10}$$

where $\mathcal{P}$ is the lifting layer, $\mathcal{Q}$ is the projection layer, and each $\mathcal{L}^{(l)}$ is a neural operator layer. Here, we parameterize our integral kernel by $\kappa(z;\Phi)$. To explicitly distinguish from the standard GNOs formulation Section 3.1, which uses ϕ to denote the parameter space of a single monolithic neural network, we employ Φ to represent the comprehensive collection of parameters from the d decoupled factor networks and the learnable core tensor. And then we learn the mapping of neural operator layer via kernel integration operator parameterization

$$\left(\mathcal{K}(a;\Phi)v^{(l)}\right)(x) := \int_D \kappa(z;\Phi)v^{(l)}(y)\,dy \approx \frac{1}{|N(x)|}\sum_{y\in N(x)} \kappa(z;\Phi)v^{(l)}(y), \forall x \in D \tag{11}$$

and node feature propagation

$$v^{(l+1)}(x) := \sigma\left(W^{(l)}v^{(l)}(x) + \left(\mathcal{K}(a;\Phi)v^{(l)}\right)(x)\right), \forall x \in D. \tag{12}$$

It is worth noting that we borrow the same calculation method as GNO, completing the calculation of the above equation through domain truncation and Nyström approximation. As shown in Figure 2, we provide the full architecture of proposed neural operator as well as an example of integral kernel parameterization for (6) in three tensor decomposition modes.

3.2.3 COMPLEXITY ANALYSIS

This section aims to provide a comprehensive analysis of the computational efficiency by comparing standard GNO with the proposed FTD-GNO. The focus is on elucidating how FTD effectively mitigates the so-called "curse of dimensionality" during the learning process of high-dimensional integral kernels. Assume that the single kernel network in the standard GNO has a width of W and a depth of L. For FTD-GNO, its d univariate subnetworks $\Phi^{(i)}$ have a width of w and a depth of l.

Under the standard network architecture, both the computational complexity and the parameter count for a single integral kernel evaluation in GNO are $O(W^2L)$. As the input dimension d increases, the network width W often needs to expand linearly or even exponentially to adequately capture high-dimensional feature interactions, thereby triggering a severe curse of dimensionality and significant computational redundancy. In contrast, FTD-GNO innovatively decouples the high-dimensional integral kernel into a composition of d factor networks. The comprehensive theoretical complexity comparison is summarized in Table 1. Detailed mathematical derivations for each tensor format (CP, TT, Tucker) and memory scaling proofs are provided in Appendix D.

Table 1: Complexity analysis of different methods.

Method	Single Kernel Evaluation	Network Parameters	Memory Occupancy
GNO	$O(W^2L)$	$O(W^2L)$	$O(I^d)$
CP-GNO	$O(dw^2lR)$	$O(dw^2lR)$	$O(Id)$
TT-GNO	$O(d(w^2l + R^2))$	$O(dw^2lR^2)$	$O(Id)$
Tucker-GNO	$O(dw^2l + R^d)$	$O(R^d + dw^2lR)$	$O(Id)$

As demonstrated in Table 1, FTD-GNO achieves an effective separation of variables. The divide-and-conquer approach is demonstrably advantageous in terms of model efficiency, as listed below:

1. Optimizing Memory Occupancy: Leveraging independent feature decomposition and integral operator reconstruction, the model reduces the memory footprint of input construction to a linear scale, unlocking scalable training for high-resolution physical fields.

2. Reducing Computational Complexity: The model has been demonstrated to reduce the computational cost of a single kernel evaluation from a heavy polynomial scale to a linear scale, thereby substantially enhancing the forward inference efficiency of the operator.

3. Slashing Parameter Scale: By avoiding redundant network expansion, the model preserves equivalent expressive capacity while inherently mitigating overfitting risks, thereby facilitating the lightweight deployment of large-scale operators.

4 EXPERIMENTS

We evaluate our proposed models on the Darcy flow, Navier-Stokes (NS) sourced from the NeuralOperator library and elasticity on sample-dependent irregular meshes. Appendix E.1 reports the detailed experimental settings and baseline comparisons.

4.1 MAIN COMPARISONS

Extensibility: Whether FTD benefits different operator architectures.

We consider two groups of controlled comparisons: GNO against CP-, TT-, and Tucker-GNO, and FNOGNO against CP-, TT-, and Tucker-FNOGNO. FNOGNO replaces the preceding graph operator layers with Fourier layers while retaining the final graph integral. Its factorized variants modify only that final kernel. As shown in Table 2, FTD reduces peak memory in both groups, extending its benefit from graph-only to Fourier-graph architectures. At higher NS resolution, the factorized variants remain trainable when FNOGNO encounter OOM. Because FTD modifies the kernel parameterization and contraction locally, it can be incorporated into other operator frameworks containing graph-integral layers that admit the proposed factorization.

Effectiveness: Whether FTD reduces graph-kernel costs beyond weight compression.

Table 2: Comprehensive benchmark of relative L_2 errors. Peak memory is reported in MB.

Model	Darcy 16×16		NS 16×16		NS 128×128	
	Error	Peak memory	Error	Peak memory	Error	Peak memory
GNO	0.1928	435.43	0.5482	420.61	OOM	OOM
CP-GNO	0.2114	197.02	0.5538	181.38	0.5510	6002.06
TT-GNO	0.2114	197.12	0.5538	181.38	0.5510	6009.94
Tucker-GNO	0.2104	201.68	0.5493	183.32	0.5489	8963.39
FNO	0.1042	93.58	0.0789	92.92	0.0783	1934.74
FNOGNO	0.1024	195.93	0.1152	276.77	OOM	OOM
CP-FNOGNO	0.1026	122.80	0.1156	123.37	0.1203	6104.55
TT-FNOGNO	0.1026	122.55	0.1156	123.00	0.1203	6207.68
Tucker-FNOGNO	0.1067	124.80	0.1163	128.49	0.1146	8469.61

Table 3: Experiment results of relative L_2 errors, training time, and parameters on Darcy flow.

Model	16×16			32×32			Parameters	Parameters
	Error	Time (s)	Speedup	Error	Time (s)	Speedup		Reduction
GNO	0.1872	6542.28	–	0.1723	22602.04	–	95265	–
CP-GNO	0.1975	647.00	**10.11**×	0.1815	545.27	**41.45**×	74725	**21.56%**
TT-GNO	0.1975	534.03	**12.25**×	0.1815	654.29	**34.54**×	74725	**21.56%**
Tucker-GNO	0.2157	665.17	**9.84**×	0.2214	635.18	**35.58**×	72709	**23.68%**

As shown in Table 9 (Appendix E.2.1), LR-GNO and FTD-GNO exhibit substantially different memory costs despite closely matched parameter counts. LR-GNO compresses the kernel MLP weights but preserves its evaluation on neighboring node pairs, whereas FTD reorganizes kernel evaluation and aggregation through reusable node-wise factors. This comparison indicates that parameter reduction alone does not explain the memory advantage and supports the role of the factorized contraction. The slightly lower error of LR-GNO also shows that matching parameter counts does not imply identical approximation or optimization behavior.

Benchmarks: Whether FTD remains effective in different PDE tasks.

For regular grid benchmarks, FTD reduces peak memory relative to the corresponding unfactorized baselines in both architecture groups, shown in Table 2. Factorization also enables higher-resolution NS training when the unfactorized baselines encounter OOM. Table 3 reports that the proposed models achieve substantial training speedups and significant parameter reductions. However, graph-only configurations exhibit relatively high errors in NS, where FNO provides more accurate predictions. Incorporating Fourier operator layers improves the predictive performance of operator models. Accordingly, we view FTD as a memory-efficient component for graph integral architectures, rather than establishing FTD-GNO as a uniformly superior operator across PDE tasks.

For the irregular benchmark, under the same training setting, all three FTD-GNO variants reduce peak memory and training time relative to GNO, shown in Table 10 (Appendix E.2.1). This demonstrates that the efficiency benefit extends to irregular meshes, where the graph connectivity varies across samples. Moreover, the additional 300-epoch runs improve the factorized models beyond the error of the 100-epoch GNO while retaining lower memory and recorded total runtime. These results show that the reduced training cost can be used to support additional optimization.

In summary, these results consistently support FTD as an effective approach for reducing graph-integral costs. Further analyses are provided in Appendix E.2.1. Moreover, Appendix E.2.2 reports the effects of rank and model capacity (Tables 12 and 13), as well as the effects of kernel input variable blocks on peak memory (Table 14).

4.2 Memory Footprint

Tables 4 and 5 compare peak memory across resolutions and neighborhood radii. GNO memory increases sharply as the graph becomes denser and reaches OOM at the 64×64 resolutions. Con-

Table 4: Peak GPU memory footprint (MB) of different models under varying resolutions and radii.

	16×16			32×32			64×64		
Model	r=0.08	r=0.12	r=0.16	r=0.08	r=0.12	r=0.16	r=0.08	r=0.12	r=0.16
GNO	420.6	696.3	1511.0	6426.6	13208.9	19733.3	OOM	OOM	OOM
CP-GNO (R=8)	182.7	182.7	182.9	680.3	682.0	683.6	2688.8	2715.9	2753.5
CP-GNO (R=16)	274.7	274.8	275.0	1048.4	1050.0	1051.6	4160.8	4187.4	4225.5
CP-GNO (R=32)	458.8	458.9	459.1	1784.5	1786.1	1787.7	7104.9	7132.0	7168.9
TT-GNO (R=8)	182.7	182.7	182.9	680.3	682.0	683.6	2688.9	2715.9	2753.5
TT-GNO (R=16)	274.7	274.8	275.0	1048.4	1050.0	1051.6	4160.8	4187.4	4225.5
TT-GNO (R=32)	458.8	458.9	459.1	1784.5	1786.1	1787.7	7104.9	7132.0	7168.9
Tucker-GNO (R=8)	183.7	183.7	183.9	684.4	686.0	687.6	2706.8	2731.9	2769.9
Tucker-GNO (R=16)	276.7	276.8	277.0	1056.4	1058.0	1059.6	4192.9	4218.8	4257.5
Tucker-GNO (R=32)	462.8	462.9	463.1	1800.5	1802.2	1803.7	7168.9	7196.1	7233.0

Table 5: Derived memory scaling and efficiency analysis from Table 4.

Metric	GNO	CP-8	CP-16	CP-32	TT-8	TT-16	TT-32	Tcr-8	Tcr-16	Tcr-32
$16\times16 \to 32\times32$, r=0.12	19.0×	3.7×	3.8×	3.9×	3.7×	3.8×	3.9×	3.7×	3.8×	3.9×
r: $0.08 \to 0.16$, 32×32	3.1×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×
Reduction vs. GNO (32×32, r=0.16)	1.0×	**28.9**×	**18.8**×	**11.0**×	**28.9**×	**18.8**×	**11.0**×	**28.7**×	**18.6**×	**10.9**×

versely, all FTD-GNO variants adhere strictly to a stable, linear growth trajectory. By algebraically decoupling computation from the edge dimension, FTD-GNO demonstrates remarkable memory invariance to radius expansion and ensures memory proportionality solely to the total node count, thereby completely shattering the hardware barriers for high-resolution physical field modeling.

4.3 Zero-Shot Super-Resolution

Resolution invariance enables neural operators trained on low-resolution grids to perform zero-shot inference on unseen higher-resolution grids without retraining. To assess zero-shot super-resolution, we train FTD-GNO at 16×16 and directly evaluate it at 16×16 and 32×32, shown in Table 6. The results reveal that the relative test errors for all models remained within the same order of magnitude, which suggests the robust zero-shot super-resolution generalization capability of the proposed architecture.

Table 6: l_2 relative test errors.

Model	16×16	32×32
GNO	0.1872	0.1838
CP-GNO	0.1975	0.1956
TT-GNO	0.1975	0.1956
Tucker-GNO	0.2157	0.2214

5 Conclusion

In this paper, we propose FTD-GNO, a novel Graph Neural Operator architecture that reconstructs the integral kernel via functional tensor decomposition. Through the implementation of variable decoupling and algebraic integral reconstruction methodologies, we have successfully overcome the computational and memory limitations inherent to conventional GNOs when processing high-resolution physical fields. As a result, the memory requirement scales linearly and smoothly with the node size. Moreover, a systematic evaluation of three core variants is conducted, utilising CP, TT, and Tucker decompositions as the underlying methodologies. Across the regular-grid and irregular-mesh tasks, FTD reduces peak memory relative to corresponding unfactorized graph-integral baselines and also benefits a Fourier–graph hybrid. This breakthrough paves a highly efficient path for the engineering deployment of GNOs in large-scale, high-fidelity scientific computing. Appendix F further discusses the limitations of FTD-GNO and outlines directions for future research.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing and text editing. We have not used generative AI tools for theoretical development, mathematical claims or proofs, and tasks involving synthetic dataset generation or qualitative and thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work. Specifically, we checked all AI-assisted revisions to ensure that they accurately reflect our research and preserve its intended scientific meaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Code and data are provided in the supplementary material, and a public GitHub repository will be released in the final version. The operator definitions and factorized formulations are specified in Section 3 and Appendices B-D. Appendix E provides the training configurations, relative error expression, timing procedure, additional architectural comparisons, sensitivity studies and repeated run memory results. The distinction between equal-epoch comparisons, additional optimization budgets,and zero-shot resolution transfer is stated in the corresponding captions and discussion.

ACKNOWLEDGMENTS

Acknowledgments are omitted for double-blind review and will be included in the final version.

REFERENCES

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 30:24924–24940, 2021.

Daniel J. Duffy. *Numerical methods in computational finance: a partial differential equation (PDE/FDM) approach*. John Wiley & Sons, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 1263–1272, 2017.

Sergei K. Godunov and Ihor Bohachevsky. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematicheskii sbornik*, 47(3): 271–306, 1959.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *Proceedings of Machine Learning Research*, volume 202, pp. 12556–12569, 2023.

Stefan Hildebrand and Sandra Klinge. Comparison of neural fem and neural operator methods for applications in solid mechanics. *Neural Computing and Applications*, 36(27):16657–16682, 2024.

Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

Yipeng Huang, Dejun Xu, Zexin Lin, Zhenzhong Wang, and Min Jiang. Neuro-symbolic multitasking: A unified framework for discovering generalizable solutions to pde families. *arXiv preprint arXiv:2602.11630*, 2026.

Douglas Samuel Jones, Michael Plank, and Brian D. Sleeman. *Differential equations and mathematical biology*. Chapman and Hall/CRC, 2009.

Jean Kossaifi, Nikola Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Multi-grid tensorized fourier neural operator for high-resolution pdes. *arXiv preprint arXiv:2310.00120*, 2023.

Jean Kossaifi, Nikola Kovachki, Zongyi Li, David Pitt, Miguel Liu-Schiaffini, Robert Joseph George, Boris Bonev, Kamyar Azizzadenesheli, Julius Berner, Valentin Duruisseaux, et al. A library for learning neural operators. *arXiv preprint arXiv:2412.10354*, 2024.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Kangjie Li and Wenjing Ye. D-fno: A decomposed fourier neural operator for large-scale parametric partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 436: 117732, 2025.

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

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

Shengjun Liu, Yu Yu, Ting Zhang, Hanchao Liu, Xinru Liu, and Deyu Meng. Architectures, variants, and performance of neural operators: A comparative review. *Neurocomputing*, 648:130518, 2025.

Sebastian Loeschcke, David Pitt, Robert Joseph George, Jiawei Zhao, Cheng Luo, Yuandong Tian, Jean Kossaifi, and Anima Anandkumar. Tensorgrad: Tensor gradient robust decomposition for memory-efficient neural operator training. *arXiv preprint arXiv:2501.02379*, 2025.

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

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Evert J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54:185–204, 1930.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Maryam Parvizi, Amirreza Khodadadian, and Thomas Wick. Analysis of a fully discretized fdm-fem scheme for solving thermo-elastic-damage coupled nonlinear pde systems. *arXiv preprint arXiv:2302.01994*, 2023.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021.

Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, 1966.

Sai Karthikeya Vemuri, Tim Büchner, Julia Niebling, and Joachim Denzler. Functional tensor decompositions for physics-informed neural networks. In *International Conference on Pattern Recognition*, pp. 32–46, 2025.

Sai Karthikeya Vemuri, Tim Büchner, and Joachim Denzler. F-inr: Functional tensor decomposition for implicit neural representations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 6557–6568, 2026.

Eleftherios C. Zachmanoglou and Dale W. Thoe. *Introduction to partial differential equations with applications*. Courier Corporation, 1986.

Miaohua Zhang, Yongsheng Gao, Changming Sun, and Michael Blumenstein. Robust tensor decomposition for image representation based on generalized correntropy. *IEEE Transactions on Image Processing*, 30:150–162, 2021.

Ao Zhou, Jianlei Yang, Yeqi Gao, Tong Qiao, Yingjie Qi, Xiaoyi Wang, Yunli Chen, Pengcheng Dai, Weisheng Zhao, and Chunming Hu. Brief industry paper: optimizing memory efficiency of graph neural networks on edge computing platforms. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 445–448. IEEE, 2021a.

Guancheng Zhou, Zelin Zeng, Yisi Luo, Qi Xie, and Deyu Meng. Tucker-fno: Tensor tucker-fourier neural operator and its universal approximation theory. In *The Fourteenth International Conference on Learning Representations*, 2026.

Pan Zhou, Canyi Lu, Jiashi Feng, Zhouchen Lin, and Shuicheng Yan. Tensor low-rank representation for data recovery and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1718–1732, 2021b.

Olgierd C. Zienkiewicz and Robert Leroy Taylor. *The finite element method set*. Elsevier, 2005.

A General Framework of Parametric PDEs

This section delineates the general mathematical framework of parametric PDEs to establish the physical context for the operator learning discussed in Section 3.1.

In scientific computing and engineering, the evolution of complex physical systems is typically governed by PDEs. When variable parameters are introduced to account for uncertainty quantification or physical inverse problems, these parameter-driven equations are collectively termed parametric PDEs. Mathematically, let $D \subset \mathbb{R}^{d_x}$ denote a bounded spatial domain with boundary ∂D. We consider a general class of parametric PDEs formulated as:

$$\begin{aligned} (\mathcal{L}_a u)(x) &= f(x), \quad x \in D \\ u(x) &= 0, \qquad x \in \partial D \end{aligned} \tag{13}$$

where $a(x)$ denotes the input parameter field of the system (e.g., physical properties of heterogeneous media, conductivity coefficients, or fluid viscosity), and $u(x)$ is the unknown target physical response field. The operator $\mathcal{L}_a$, which explicitly depends on the parameter field a, mathematically characterizes the underlying conservation laws and spatial differential rules governing the physical system. Meanwhile, $f(x)$ represents the source term or forcing function, corresponding to external driving forces (e.g., external energy sources, mass sinks, or body forces) acting within the interior of domain D. The aforementioned formulation is also subject to homogeneous Dirichlet boundary conditions.

To be more specific, taking the steady-state Darcy flow, a prototypical fluid-in-porous-media problem widely benchmarked in the operator learning, can be governed by the following second-order elliptic PDE:

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad x \in D \\ u(x) &= 0, \qquad x \in \partial D. \end{aligned} \tag{14}$$

Within this physical context, the input parameter $a(x)$ generally represents a highly heterogeneous permeability field within the medium, with the target solution $u(x)$ corresponding to the resulting fluid pressure field. Assuming the well-posedness of the PDE, for a fixed source term $f(x)$ and given boundary conditions, each specific parameter instance a in the input parameter space is associated with to a unique physical response solution u. This strict dependence, inherently dictated by the underlying physical conservation laws, essentially establishes a nonlinear mapping defined between infinite-dimensional function spaces, i.e., $a \mapsto u$.

B Extended Background on Standard GNOs

This section details the domain truncation and Nyström approximation techniques utilized by standard Graph Neural Operators (GNOs) to discretize the continuous integral kernel operator described in Section 3.1.

Domain truncation. The spatial domain D is discretised by J points. Each discrete point, designated $x \in D$, is regarded as a node of the directed graph, and the node feature $v^{(l)}(x)$ is assigned to it. The edge connectivity is determined by the Lebesgue measure. Let $s : D \to \mathcal{B}(D)$ be the mapping from the point of D to the Lebesgue measurable subsets of D, denoted as $\mathcal{B}(D)$. In general, we consider $s(x) = B(x,r) \cap D$, where $B(x,r) = \left\{ y \in \mathbb{R}^{d_x} : \|y - x\|_{\mathbb{R}^{d_x}} < r, r > 0 \right\}$. Each node x is connected to the node set located in $B(x,r)$ and is defined as the neighbourhood set $N(x)$. And for each neighbor $y \in N(x)$, we assign the edge weight $(x, y, a(x), a(y))$. Therefore, the integral equation becomes

$$\left(\mathcal{K}(a;\phi)\, v^{(l)}\right)(x) = \int_{B(x,r)} \kappa(x, y, a(x), a(y); \phi)\, v^{(l)}(y)\, dy, \forall x \in D. \tag{15}$$

This truncation is interpreted as constructing a local structure in the integral kernel κ. This local structure facilitates enhanced computational efficiency while remaining invariant to mesh-refinement. Indeed, given that the neighbourhood radius r is selected in physical space, this is a pivotal characteristic that renders GNO mesh-independent. In addition, if the cost of computing original integral equation is $O(J^2)$, then utilising the selection of $s(x) = B(x,r) \cap D$ and assuming $D = [0,1]^{d_x}$, the complexity of computing (15) is approximately $O(r^{d_x} J^2)$. In fact, for the general

case where the size of the set $s(x)$ is less than D, we always have the cost of calculating (15) as $O(c_s J^2)$, where c_s is a constant that depends on s and $c_s < 1$. This may significantly impact the actual calculation, so domain truncation is an effective means of reducing the cost of calculation.

Nyström approximation theorem. Nyström approximation is a simple and effective method commonly used to calculate the kernel integral operator, which is mainly used to solve the problem of large computational overhead of integrating over the entire domain D. Its main idea is to randomly sample the points over which we compute the output function u. Specifically, if we uniformly sample $J' \ll J$ nodes from the original graph, the computational cost decreases from $O(J^2)$ to $O(J'^2)$, which is only related to the number of subsampled points J'. It is evident that the number of subsampled points selected is typically much smaller than the original discretisation points. Consequently, this results in a substantial reduction in the final computational cost.

In accordance with the aforementioned definition, the message passing algorithm proposed by Gilmer et al. (2017) is utilised to construct a graph for the spatial domain D of partial differential equations. The kernel integral is regarded as the aggregation of messages. Consequently, the message passing kernel integral operator with average aggregation is designated as

$$\left(\mathcal{K}\left(a;\phi\right)v^{(l)}\right)\left(x\right) = \frac{1}{\left|N\left(x\right)\right|}\sum_{y\in N(x)}\kappa\left(x,y,a\left(x\right),a\left(y\right);\phi\right)v^{(l)}\left(y\right), \forall x \in D. \tag{16}$$

Here, (16) can be regarded as the Monte-Carlo approximation of the integral (15).

C Mathematical Foundations of FTD-GNO: Decomposition

Table 7 details the explicit mathematical formulations of the Functional Tensor Decomposition applied to the integral kernel within FTD-GNO. Unlike the standard GNO, we explicitly expand the continuous integral kernel into a product of multiple low-rank tensor factors. This factorization is a fundamental prerequisite for achieving high computational efficiency. The table systematically illustrates the decomposition process as the input variables expand from basic spatial coordinates to a full physical state. It demonstrates how three mainstream tensor decomposition mechanisms (CP, TT, and Tucker) decouple the multi-dimensional, coupled kernel function into independent factor functions, which are parameterized by neural networks. Through this explicit functional expansion, the complex interactions among multiple variables are mapped into an implicit feature space of rank R. Specifically for the Tucker decomposition, a learnable core tensor is introduced to capture the non-linear coupling relationships across different rank dimensions.

Proposition C.1 (increasing-capacity kernel approximation). Let $\mathcal{Z}_i$ be compact subsets of Euclidean spaces and let $\kappa \in C(\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_d; \mathbb{R})$. For every $\varepsilon > 0$, there are a finite rank R and continuous functions $A_r^{(i)} \in C(\mathcal{Z}_i; \mathbb{R})$ such that

$$\left\| \kappa(z_1, \ldots, z_d) - \sum_{r=1}^{R}\prod_{i=1}^{d} A_r^{(i)}(z_i) \right\|_\infty < \varepsilon. \tag{17}$$

If the factor-network family is uniformly dense in continuous functions on each $\mathcal{Z}_i$, the factors can be replaced by sufficiently capable networks with the same conclusion after increasing their capacities as necessary. The result extends component-wise to a finite-dimensional channel-wise kernel.

Proof. Consider the set $\mathcal{A}$ of finite sums of products $\sum_{r=1}^{R}\prod_i f_{ir}(z_i)$, with continuous block functions f_{ir}. This set contains constants, is closed under addition and multiplication, and therefore forms a unital subalgebra of the continuous functions on the compact product domain. It separates points: if two tuples differ in a block, a continuous function on that block separates them, while constants on the remaining blocks complete a product function. The real Stone–Weierstrass theorem implies that $\mathcal{A}$ is uniformly dense. Choose a finite expansion with d many bounded factors, the product perturbation inequality

$$\left\| \prod_{i=1}^{d} f_i - \prod_{i=1}^{d} g_i \right\| \leq \sum_{j=1}^{d} |f_j - g_j| \prod_{i<j} |g_i| \prod_{i>j} |f_i| \tag{18}$$

allows each factor to be approximated by a network with a sufficiently small uniform error so that the total finite-sum perturbation is below $\varepsilon/2$. The triangle inequality proves the result. TT can

Table 7: Functional tensor decomposition formulas of the integral kernel under CP, TT, and Tucker formats for multi-variable inputs.

Type	Standard GNO Formulation	FTD-GNO Decomposition
CP	$\kappa(x,y;\phi)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y)$
	$\kappa(x,y;\phi)\, v^{(l)}(y)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) v^{(l)}(y)$
	$\kappa(x,y,a(y);\phi)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) A_r^{(3)}(a(y))$
	$\kappa(x,y,a(y);\phi)\, v^{(l)}(y)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) A_r^{(3)}(a(y)) v^{(l)}(y)$
	$\kappa(x,y,a(x),a(y);\phi)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) A_r^{(3)}(a(x)) A_r^{(4)}(a(y))$
	$\kappa(x,y,a(x),a(y);\phi)\, v^{(l)}(y)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) A_r^{(3)}(a(x)) A_r^{(4)}(a(y)) v^{(l)}(y)$
TT	$\kappa(x,y;\phi)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y)$
	$\kappa(x,y;\phi)\, v^{(l)}(y)$	$\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) v^{(l)}(y)$
	$\kappa(x,y,a(y);\phi)$	$\sum_{r_1,r_2} A_{r_1}^{(1)}(x) A_{r_1,r_2}^{(2)}(y) A_{r_2}^{(3)}(a(y))$
	$\kappa(x,y,a(y);\phi)\, v^{(l)}(y)$	$\sum_{r_1,r_2} A_{r_1}^{(1)}(x) A_{r_1,r_2}^{(2)}(y) A_{r_2}^{(3)}(a(y)) v^{(l)}(y)$
	$\kappa(x,y,a(x),a(y);\phi)$	$\sum_{r_1,r_2,r_3} A_{r_1}^{(1)}(x) A_{r_1,r_2}^{(2)}(y) A_{r_2,r_3}^{(3)}(a(x)) A_{r_3}^{(4)}(a(y))$
	$\kappa(x,y,a(x),a(y);\phi)\, v^{(l)}(y)$	$\sum_{r_1,r_2,r_3} A_{r_1}^{(1)}(x) A_{r_1,r_2}^{(2)}(y) A_{r_2,r_3}^{(3)}(a(x)) A_{r_3}^{(4)}(a(y)) v^{(l)}(y)$
Tucker	$\kappa(x,y;\phi)$	$\sum_{r_1,r_2} \mathcal{C}_{r_1,r_2} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y)$
	$\kappa(x,y;\phi)\, v^{(l)}(y)$	$\sum_{r_1,r_2} \mathcal{C}_{r_1,r_2} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y) v^{(l)}(y)$
	$\kappa(x,y,a(y);\phi)$	$\sum_{r_1,r_2,r_3} \mathcal{C}_{r_1,r_2,r_3} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y) A_{r_3}^{(3)}(a(y))$
	$\kappa(x,y,a(y);\phi)\, v^{(l)}(y)$	$\sum_{r_1,r_2,r_3} \mathcal{C}_{r_1,r_2,r_3} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y) A_{r_3}^{(3)}(a(y)) v^{(l)}(y)$
	$\kappa(x,y,a(x),a(y);\phi)$	$\sum_{r_1,r_2,r_3,r_4} \mathcal{C}_{r_1,r_2,r_3,r_4} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y) A_{r_3}^{(3)}(a(x)) A_{r_4}^{(4)}(a(y))$
	$\kappa(x,y,a(x),a(y);\phi)\, v^{(l)}(y)$	$\sum_{r_1,r_2,r_3,r_4} \mathcal{C}_{r_1,r_2,r_3,r_4} A_{r_1}^{(1)}(x) A_{r_2}^{(2)}(y) A_{r_3}^{(3)}(a(x)) A_{r_4}^{(4)}(a(y)) v^{(l)}(y)$

realize a CP sum using diagonal intermediate cores, and Tucker using a superdiagonal core, so both inherit this increasing-rank approximation property. For finitely many channels, apply the argument to each component and pad ranks as needed. This proposition is consistent with functional tensor approximation results used by Vemuri et al. (2026; 2025).

D Detailed Complexity and Memory Analysis

The appendix provides the rigorous mathematical derivations for the complexity analysis summarized in Section 3.2.3 (Table 1).

D.1 Computational and Parameter Complexity

Specifically, the standard GNO integral kernel requires $\mathcal{O}(W^2L)$ for both computation and parameter count. FTD-GNO decomposes this into d factor networks. The specific complexities for the three variants are derived as follows:

CP-GNO models the integral kernel as a composition of factors with rank R. Its d factor networks have a forward propagation complexity of $O(dw^2l)$, and the outer product composition requires a computational complexity of $O(dR)$. Thus, the overall computational cost for a single kernel evaluation is $O\left(dw^2lR\right)$, with the total network parameter count identically being $O(dw^2lR)$.

TT-GNO leverages the chain-like contraction property of the tensor train structure to elegantly circumvent the explicit construction of full high-dimensional tensors. The boundary TT-ranks are set to $R_0 = R_d = 1$, and for computational simplicity, the intermediate ranks are assumed to be $R_i = R$. The forward propagation complexity of this architecture is $O(dw^2l)$, and the tensor contraction complexity is $O(dR^2)$. Consequently, its total computational cost for a single kernel evaluation is $O(d(w^2l + R^2))$. With regard to the parameter count, the first and last networks have an output dimension of R, contributing $O(w^2lR)$ parameters; the intermediate $d-2$ networks have an output dimension of $R \times R$, contributing $O(w^2lR^2)$ parameters. Therefore, the total network parameter count is $O(w^2l(2R + (d-2)R^2)) = O(dw^2lR^2)$.

Tucker-GNO introduces a learnable core tensor $\mathcal{C}$ to capture complex interactive correlations among variables. Assuming the Tucker ranks across all modes are $R_i = R$, the complexity of computing all factor vectors is $O(dw^2l)$, and the multilinear product is realized via step-by-step mode-n contractions, requiring a computational complexity of $O(|\mathcal{C}|) = O(R^d)$. Hence, the single kernel evaluation complexity for Tucker-GNO is $O(dw^2l + R^d)$. Since the output dimensions of the d factor networks are all R, the corresponding parameter count is $O(dw^2lR)$. In addition, the parameter count of the core tensor is $O(R^d)$. So the total parameter count for Tucker-GNO is $O(R^d + dw^2lR)$.

D.2 Memory Occupancy

Memory occupancy is a critical consideration in the practical engineering deployment of neural operators. In standard GNO, constructing the complete input during integral kernel evaluation consumes $O(\prod_{i=1}^{d} I_i)$ memory space. In the design of FTD-GNO, however, the network input is strictly decoupled into independent variables—x, y, $a(x)$, and $a(y)$—for decomposed training, and each branch network $\Phi^{(i)}$ only processes the I_i objects within its corresponding dimension. This means the model no longer needs to explicitly construct an exponential full-scale tensor of size $\prod_{i=1}^{d} I_i$; instead, it only needs to independently store d vector representations of length I_i. Therefore, assuming $I_1 = \cdots = I_d = I$, the memory footprint during input construction is successfully reduced from the exponential $O(I^d)$ to the linear $O(Id)$.

Beyond the integral kernel evaluation phase, the spatial integral aggregation phase also faces severe memory consumption challenges. Traditional spatial domain integration methods typically rely on edge-based message passing mechanisms, which require explicitly computing and storing the complete integral kernel values on every single edge during execution. As grid resolution increases, this edge-level storage requirement easily triggers GPU memory bottlenecks. To address this flaw, FTD-GNO ingeniously utilizes the associative property of tensor decomposition to achieve a fundamental reconstruction of the integral formulation. The core of this operation lies in isolating all tensor factors that depend exclusively on the central node x (e.g., $A^{(1)}(x)$ and $A^{(3)}(a(x))$). These terms are subsequently factored out of the neighborhood summation $\sum_{y \in N(x)}$. This algebraic manipulation fundamentally transforms the complex, high-dimensional integrals traditionally computed on graph edges. Taking the full-state CP-GNO as an example, its original integral kernel operator takes the following form:

$$\left(\mathcal{K}\left(a;\Phi\right) v^{(l)}\right)(x) \tag{19}$$
$$= \frac{1}{|N(x)|} \sum_{y \in N(x)} \left(\sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(2)}(y) A_r^{(3)}(a(x)) A_r^{(4)}(a(y)) \right) v^{(l)}(y), \forall x \in D.$$

Through algebraic reconstruction, it is equivalently transformed into

$$\left(\mathcal{K}\left(a;\Phi\right) v^{(l)}\right)(x) \tag{20}$$
$$= \frac{1}{|N(x)|} \sum_{r=1}^{R} A_r^{(1)}(x) A_r^{(3)}(a(x)) \left(\sum_{y_i \in N(x)} A_r^{(2)}(y) A_r^{(4)}(a(y)) v^{(l)}(y) \right), \forall x \in D.$$

Table 8 systematically summarizes these algebraic reconstruction formulas across all decomposition formats and input variables. This algebraic-level rewriting achieves an effective decoupling of the neighborhood aggregation operation, fundamentally evading the cumbersome process of constructing and storing high-dimensional concatenated tensors along the edge dimension. It converts the

computation into a highly efficient two-stage process: first, a local aggregation over neighboring nodes y, followed by a feature combination at the central node x. Thanks to this, GPU memory occupancy during network execution transitions from an uncontrollable state highly dependent on the number of edges, smoothly scaling down to a controlled state dependent solely on the number of nodes. Notably, we assume a uniform rank R for all tensor factors involved in these decompositions.

Table 8: Algebraic reconstruction formulas of the integral algorithm under CP, TT, and Tucker decomposition formats. $N(x)$ means the neighhourhood of x.

Type	Algebraic Reconstruction Formulation of FTD-GNO
CP	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, v^{(l)}(y)\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, A_r^{(3)}(a(y))\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, A_r^{(3)}(a(y))\, v^{(l)}(y)\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\, A_r^{(3)}(a(x))\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, A_r^{(4)}(a(y))\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\, A_r^{(3)}(a(x))\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, A_r^{(4)}(a(y))\, v^{(l)}(y)\right)$
TT	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\right)$
	$\sum_{r=1}^{R} A_r^{(1)}(x)\left(\sum_{y\in N(x)} A_r^{(2)}(y)\, v^{(l)}(y)\right)$
	$\sum_{r_1,r_2} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_1,r_2}^{(2)}(y)\, A_{r_2}^{(3)}(a(y))\right)$
	$\sum_{r_1,r_2} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_1,r_2}^{(2)}(y)\, A_{r_2}^{(3)}(a(y))\, v^{(l)}(y)\right)$
	$\sum_{r_1,r_2,r_3} A_{r_1}^{(1)}(x)\, A_{r_2,r_3}^{(3)}(a(x))\left(\sum_{y\in N(x)} A_{r_1,r_2}^{(2)}(y)\, A_{r_3}^{(4)}(a(y))\right)$
	$\sum_{r_1,r_2,r_3} A_{r_1}^{(1)}(x)\, A_{r_2,r_3}^{(3)}(a(x))\left(\sum_{y\in N(x)} A_{r_1,r_2}^{(2)}(y)\, A_{r_3}^{(4)}(a(y))\, v^{(l)}(y)\right)$
Tucker	$\sum_{r_1,r_2} \mathcal{C}_{r_1,r_2} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\right)$
	$\sum_{r_1,r_2} \mathcal{C}_{r_1,r_2} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\, v^{(l)}(y)\right)$
	$\sum_{r_1,r_2,r_3} \mathcal{C}_{r_1,r_2,r_3} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\, A_{r_3}^{(3)}(a(y))\right)$
	$\sum_{r_1,r_2,r_3} \mathcal{C}_{r_1,r_2,r_3} A_{r_1}^{(1)}(x)\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\, A_{r_3}^{(3)}(a(y))\, v^{(l)}(y)\right)$
	$\sum_{r_1,r_2,r_3,r_4} \mathcal{C}_{r_1,r_2,r_3,r_4} A_{r_1}^{(1)}(x)\, A_{r_3}^{(3)}(a(x))\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\, A_{r_4}^{(4)}(a(y))\, v^{(l)}(y)\right)$
	$\sum_{r_1,r_2,r_3,r_4} \mathcal{C}_{r_1,r_2,r_3,r_4} A_{r_1}^{(1)}(x)\, A_{r_3}^{(3)}(a(x))\left(\sum_{y\in N(x)} A_{r_2}^{(2)}(y)\, A_{r_4}^{(4)}(a(y))\, v^{(l)}(y)\right)$

E Extended Experiments

E.1 Experimental Settings

This section provides the dataset configurations, model settings, and evaluation protocols supporting the experiments in Section 4.

Datasets. We consider Darcy flow and two-dimensional Navier–Stokes (NS) on regular grids, together with elasticity on sample-dependent irregular meshes. The Darcy flow and NS benchmark contain 1,000 training samples and 200 test samples. The primary Darcy experiments use resolutions of 16×16 and 32×32, while the additional architectural comparison is conducted at 16×16. The NS comparison uses resolutions of 16×16 and 128×128. For elasticity, each sample is represented by an irregular mesh containing 972 nodes. We use a reduced-data protocol with 200 training samples, 50 validation samples, and the original 200 test samples. Within each controlled comparison, the models share the data split, normalization, and evaluation metric.

Architectures. The baseline graph operators use four operator layers with 32 hidden channels. For GNO, the internal kernel network κ is configured as a three-layer feedforward neural network with hidden layer widths of $[64, 64]$. For the evaluation of FTD-GNO, we utilize CP, TT, and Tucker decompositions for the kernel function. The decomposition rank is fixed at $R = 8$ for CP and TT, $R = 4$ for Tucker. The hidden layer dimension is set to 64. The rank, width, depth, and kernel-input studies explicitly vary these settings, as specified in the corresponding tables.

To evaluate extensibility, we construct FNOGNO by replacing the preceding graph operator layers with Fourier layers while retaining the final graph integral. CP-, TT-, and Tucker-FNOGNO replace only the kernel of this final layer. Each FNOGNO/FTD-FNOGNO comparison therefore shares the Fourier backbone, lifting and projection settings, and final graph neighborhood. FNO is included as an architectural reference.

To distinguish functional factorization from weight compression, we additionally consider LR-GNO. Each dense weight matrix $W_l \in \mathbb{R}^{m_l \times n_l}$ in its kernel MLP is parameterized as

$$W_l = A_l B_l, \quad A_l \in \mathbb{R}^{m_l \times r_l}, \quad B_l \in \mathbb{R}^{r_l \times n_l}. \tag{21}$$

The integer ranks r_l are selected to approximately match the parameter count of FTD-GNO. LR-GNO retains the same graph-integral architecture and edge-wise kernel evaluation as GNO.

Optimization and training budgets. In all experiments, we train the models using the AdamW optimizer. The batch size is set to 32, with an initial learning rate of $1e-3$ and a weight decay of $1e-4$. We also employ a cosine annealing schedule for learning rate decay. The primary Darcy baseline and resolution-transfer studies, together with the parameter matched comparison in Table 9, use 300 training epochs. The additional Darcy/NS architectural comparisons in Table 2 and the capacity studies in Tables 12 and 13 use 100 epochs. The elasticity study reports all models at 100 epochs and additional runs of the factorized variants at 300 epochs. The extended Darcy study varies the CP-GNO training budget while keeping its architecture fixed. These experiments are analyzed separately to distinguish equal-epoch comparisons from additional optimization.

Error metric. The l_2 relative error is utilised in both the training optimization and evaluation. Given a dataset of input-output pairs $\{a_i, u_i\}_{i=1}^N$ and a model $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$, the relative error is calculated as outlined below:

$$L\left(\mathcal{G}_\theta\left(a\right), u\right) = \frac{1}{N} \sum_{i=1}^{N} \frac{\left\|\mathcal{G}_\theta\left(a_i\right) - u_i\right\|^2}{\left\|u_i\right\|^2}. \tag{22}$$

Resource measurements. Experiments are conducted on a single NVIDIA GeForce RTX 5090 GPU using PyTorch 2.1.0. Total training time is measured around the complete training call, with CUDA synchronization before and after timing. It includes minibatch loading, preprocessing performed during training, forward and backward computation, in-model neighborhood operations, optimization, and scheduled validation. Initial dataset download and model construction are excluded. Peak GPU memory is reported in MB, and OOM denotes a configuration that exceeds the available memory under the tested setup. Point estimates and repeated run mean ± standard deviation measurements are presented separately.

E.2 Additional Experimental Results

E.2.1 Detailed analysis in Section 4.1

Weight compression. Table 9 compares GNO, parameter matched LR-GNO, and the three FTD-GNO variants on Darcy flow. LR-GNO and CP/TT-GNO have almost identical parameter counts,

but their peak-memory costs differ substantially. LR-GNO compresses the kernel MLP weights while continuing to evaluate the network on neighboring node pairs. In contrast, FTD reorganizes kernel evaluation and aggregation through reusable node-wise factors. This comparison indicates that reducing the number of parameters alone does not explain the memory advantage.

The prediction errors of LR-GNO and CP/TT-GNO are close, although LR-GNO performs slightly better. Thus, functional factorization introduces an accuracy–resource trade-off even at matched parameter counts. The result supports the contribution of the factorized contraction without implying identical approximation or optimization behavior between the two parameterizations.

Table 9: Parameter matched comparison on Darcy flow at 16×16 and 300 epochs. Peak memory is reported in MB.

Model	Parameters	Error	Peak memory
GNO	95265	0.1872	420.6
LR-GNO	74785	0.1946	479.8
CP-GNO	74725	0.1975	182.7
TT-GNO	74725	0.1975	182.7
Tucker-GNO	72709	0.2157	183.7

Regular grid benchmarks. Table 2 reports the complete architectural comparison on Darcy flow and NS under the 100 epochs protocol. Across the two datasets, FTD reduces peak memory relative to the corresponding unfactorized baselines in both the graph-only and Fourier–graph groups. The three FTD-FNOGNO variants maintain accuracy close to FNOGNO in the low-resolution comparisons. These results show that FTD remains effective when graph integration is combined with Fourier operator layers.

At the higher NS resolution, GNO and FNOGNO encounter OOM, whereas their factorized variants remain trainable. This extends the feasible resolution of the tested graph-integral architectures. Nevertheless, the graph-only configurations exhibit relatively high NS errors, and FNO provides more accurate predictions with lower memory consumption than the tested hybrids. Incorporating Fourier layers improves the predictive performance of the graph-based configurations, but does not eliminate their gap relative to FNO. The experiments therefore support FTD as an efficient graph-integral component whose final accuracy depends on the surrounding architecture and PDE task.

Irregular meshes. Table 10 evaluates FTD-GNO on the elasticity benchmark with sample-dependent irregular meshes. In accordance with the 100-epoch protocol, all three decomposition variants substantially reduce peak memory and recorded total training time relative to GNO. The additional 300-epoch runs improve the factorized models beyond the error of the 100-epoch GNO while retaining lower memory and recorded total runtime. The findings indicate that the reduced training cost can be used to support additional optimization. Together, these results suggest that the efficiency benefit applies to irregular discretizations as well as regular grids.

Table 10: Elasticity on sample-dependent irregular meshes with 972 nodes.

Model	Discretization	Epochs	Error	Peak memory	Time (s)
GNO	972 irregular nodes	100	0.2800	263.79	17640.04
CP-GNO	972 irregular nodes	100	0.3021	61.74	517.41
TT-GNO	972 irregular nodes	100	0.3021	61.74	521.04
Tucker-GNO	972 irregular nodes	100	0.3154	63.23	502.32
CP-GNO	972 irregular nodes	300	0.2623	61.74	1542.86
TT-GNO	972 irregular nodes	300	0.2623	61.74	1513.48
Tucker-GNO	972 irregular nodes	300	0.2709	63.23	1557.15

E.2.2 Supplementary experiments

Capacity: Whether longer training or greater model capacity improves accuracy.

For Darcy flow, longer training progressively narrows the accuracy gap between CP-GNO and GNO while leaving the CP-GNO architecture unchanged. At the longest tested budget, CP-GNO ap-

Table 11: Extended training study on Darcy flow at 16×16.

Model	Epochs	Error	Time (s)
GNO	300	0.1872	6542.28
CP-GNO	300	0.1975	647.00
CP-GNO	500	0.1971	1081.49
CP-GNO	700	0.1931	1603.67
CP-GNO	1000	0.1926	2120.37
CP-GNO	2000	0.1902	4521.40
CP-GNO	3000	0.1877	6423.93

Table 12: CP-GNO sensitivity to factor-network width, depth and rank on Darcy flow.

Ablation	w	l	R	Error	Parameters	Peak memory	Time (s)
w	16	2	8	0.2269	24293	152.82	205.09
w	32	2	8	0.2250	48101	189.09	214.35
w	64	2	8	0.2259	108005	269.38	259.47
l	64	1	8	0.2114	74725	197.02	232.84
l	64	2	8	0.2259	108005	269.38	259.47
l	64	3	8	0.2308	141285	334.16	283.20
R	64	1	4	0.2171	72645	201.77	233.54
R	64	1	8	0.2114	74725	197.02	232.84
R	64	1	16	0.2276	78885	221.16	239.54

proaches the baseline accuracy, but a small error gap remains and their recorded total training times become comparable, shown in Table 11. The elasticity results in Table 10 likewise demonstrate the value of additional optimization.

Table 12 examines factor network width w, depth l, and decomposition rank R under a fixed 100-epoch budget. Increasing width provides little accuracy improvement in the tested configurations while increasing parameter count and memory. At the tested width and rank, the shallowest configuration achieves both lower error and lower measured resource consumption than the deeper alternatives. Increasing rank also produces a non-monotonic error response. The study in Table 13 further shows that the preferred rank depends on the decomposition. CP and TT achieve their lowest tested error at $R = 8$, whereas Tucker performs best at $R = 4$.

These observations indicate that additional optimization can improve the accuracy of a fixed-capacity factorized model, while increasing capacity alone does not guarantee better finite-budget performance. In practice, rank, width, depth, and training budget should be selected jointly using validation performance and the available memory and runtime budget.

Kernel inputs: Whether different kernel inputs affect memory costs.

Beyond spatial grid topology, the chosen integration type, which determines the input variable dimensions, fundamentally dictates the complexity of multivariate coupling and the overall memory envelope. Table 14 examines memory consumption of different kernel inputs, with resolution, radius, and rank held fixed. When evaluated on basic two spatial inputs $\kappa(x, y)$, a rank-R TT decomposition with boundary ranks $R_0 = R_2 = 1$ satisfies

$$\kappa_{\mathrm{TT}}(x, y) = \sum_{r=1}^{R} A_{1,r}^{(1)}(x) A_{r,1}^{(2)}(y), \tag{23}$$

which is exactly the rank-R CP form

$$\kappa_{\mathrm{CP}}(x, y) = \sum_{r=1}^{R} A_{r}^{(1)}(x) A_{r}^{(2)}(y). \tag{24}$$

Therefore, when the factor subnetworks are configured identically, CP and TT represent the same function class and exhibit identical memory footprints. As the number of nuclear inputs increases, their costs begin to differ. When the input expands to encompass the full state $(x, y, a(x), a(y))$,

Table 13: Rank sensitivity of different decomposition methods.

R	CP L_2 error	TT L_2 error	Tucker L_2 error
2	0.2161	0.2161	0.2177
4	0.2171	0.2171	0.2104
8	0.2114	0.2114	0.2257
16	0.2276	0.2276	0.2134

Table 14: Peak GPU memory footprint (MB) versus the number of input variables, tested at grid resolution 16×16, radius $r = 0.08$, and decomposition rank $R = 8$.

Model	(x, y)	$(x, y, a(y))$	$(x, y, a(x), a(y))$
GNO	420.6	432.1	426.7
CP-GNO	182.7	203.1	223.2
TT-GNO	182.7	217.3	251.6
Tucker-GNO	183.7	211.2	861.5

CP-GNO and TT-GNO demonstrate exceptional robustness, maintaining highly compressed memory footprints with only marginal increases. In contrast, the memory consumption of Tucker-GNO surges significantly to 861.5 MB, an inflation intrinsically linked to the combinatorial storage requirements of its requisite core tensor. Consequently, for operator learning tasks involving complex multivariable inputs, CP and TT decompositions emerge as the superior architectural paradigms, delivering an optimal balance of parameter efficiency and memory compression.

Statistical Significance and Stability.

To rigorously validate the hardware stability of our proposed algebraic reconstruction, we report detailed peak GPU memory footprints across varying grid resolutions and radii in Table 15. To ensure statistical significance, all measurements are reported as mean $\pm$ standard deviation derived from multiple independent runs with different random seeds.The results reveal that FTD-GNO variants not only achieve substantial memory reductions compared to the standard GNO but also exhibit exceptionally low variance, with standard deviations generally remaining below 1.0 MB.

Table 15: Peak GPU memory footprint (MB) of different models under varying resolutions and radii. Results are consistently reported as mean $\pm$ standard deviation across independent runs to ensure statistical significance.

	16×16			32×32			64×64		
Model	r=0.08	r=0.12	r=0.16	r=0.08	r=0.12	r=0.16	r=0.08	r=0.12	r=0.16
GNO	421.13 $\pm$0.68	696.80 $\pm$0.62	1511.53 $\pm$0.68	6427.10 $\pm$0.62	13209.40 $\pm$0.62	19733.80 $\pm$0.62	OOM	OOM	OOM
CP-GNO (R=8)	183.20 $\pm$0.62	183.23 $\pm$0.68	183.43 $\pm$0.68	680.87 $\pm$0.67	682.50 $\pm$0.62	684.10 $\pm$0.62	2689.77 $\pm$0.95	2715.33 $\pm$0.67	2753.67 $\pm$0.47
CP-GNO (R=16)	275.20 $\pm$0.62	275.30 $\pm$0.62	275.50 $\pm$0.62	1048.90 $\pm$0.62	1050.53 $\pm$0.68	1052.13 $\pm$0.68	4161.77 $\pm$0.95	4187.80 $\pm$0.61	4225.70 $\pm$0.53
CP-GNO (R=32)	459.30 $\pm$0.62	459.40 $\pm$0.62	459.60 $\pm$0.62	1785.00 $\pm$0.62	1786.63 $\pm$0.68	1788.20 $\pm$0.62	7104.93 $\pm$0.45	7132.07 $\pm$0.50	7169.57 $\pm$0.76
TT-GNO (R=8)	183.20 $\pm$0.62	183.23 $\pm$0.68	183.43 $\pm$0.68	680.87 $\pm$0.67	682.50 $\pm$0.62	684.10 $\pm$0.62	2689.77 $\pm$0.95	2715.33 $\pm$0.67	2753.67 $\pm$0.47
TT-GNO (R=16)	275.20 $\pm$0.62	275.30 $\pm$0.62	275.50 $\pm$0.62	1048.90 $\pm$0.62	1050.53 $\pm$0.68	1052.13 $\pm$0.68	4161.77 $\pm$0.95	4187.80 $\pm$0.61	4225.70 $\pm$0.53
TT-GNO (R=32)	459.30 $\pm$0.62	459.40 $\pm$0.62	459.60 $\pm$0.62	1785.00 $\pm$0.62	1786.63 $\pm$0.68	1788.20 $\pm$0.62	7104.93 $\pm$0.45	7132.07 $\pm$0.50	7169.57 $\pm$0.76
Tucker-GNO (R=8)	184.20 $\pm$0.62	184.23 $\pm$0.68	184.43 $\pm$0.68	684.90 $\pm$0.62	686.50 $\pm$0.62	688.10 $\pm$0.62	2706.43 $\pm$0.55	2731.03 $\pm$0.90	2769.80 $\pm$0.46
Tucker-GNO (R=16)	277.23 $\pm$0.68	277.30 $\pm$0.62	277.50 $\pm$0.62	1056.93 $\pm$0.68	1058.57 $\pm$0.67	1060.13 $\pm$0.68	4193.80 $\pm$0.90	4219.30 $\pm$0.62	4257.87 $\pm$0.55
Tucker-GNO (R=32)	463.37 $\pm$0.67	463.43 $\pm$0.68	463.63 $\pm$0.68	1801.07 $\pm$0.67	1802.70 $\pm$0.62	1804.27 $\pm$0.67	7168.97 $\pm$0.50	7196.13 $\pm$0.45	7233.63 $\pm$0.71

F LIMITATIONS

Our limitations and future work are summarized as follows:

(1) **High-dimensional scalability of the Tucker architecture:** When handling inputs with more than four variables, the core tensor in Tucker-GNO remains susceptible to a combinatorial explosion in memory footprint. While CP and TT decompositions successfully avoid this issue, optimizing the Tucker architecture to fully harness its powerful cross-dimensional coupling capabilities requires further investigation.

(2) **Representation bottlenecks of static ranks:** Currently, the tensor decomposition rank R must be pre-defined as a hyperparameter. Empirical evidence shows that $R = 8$ is sufficient for most tasks. However, when modeling highly dynamic and complex physical processes (e.g., strong turbulent fields), a fixed rank may constrain the expressive capacity of the integral kernel.

(3) **Residual memory risks under dense topologies:** Despite deep decomposition and decoupling at the input variable level, FTD-GNO may still face potential OOM risks if the number of neighboring nodes increases drastically.

To address these limitations, future research will explore Dynamic Rank Allocation mechanisms, with a view to enabling the model to adaptively accommodate the complexities of varying physical fields. Additionally, we will investigate spatial decomposition models based directly on grid coordinates. The ultimate goal is to fundamentally eradicate the OOM risks inherent to GNOs when processing fully connected graphs or exceptionally large receptive fields.