CloudNFMM: A Hybrid Hierarchical and Local Neural Operator Inspired by the Fast Multipole Method

Emilio McAllister Fognini[1], **Marta M. Betcke**[1*], **Ben T. Cox**[2]

[1]Department of Computer Science, University College London, United Kingdom
[2]Department of Medical Physics and Biomedical Engineering, University College London, United Kingdom
emilio.fognini.17@ucl.ac.uk, m.betcke@ucl.ac.uk, b.cox@ucl.ac.uk

Abstract

The Fast Multipole Method (FMM) is an efficient numerical algorithm used to calculate long-range forces in many-body problems, leveraging hierarchical data structures and series expansions. In this work, we present the Cloud Neural FMM (CloudNFMM), a new neural operator architecture that integrates the hierarchical structure of the FMM to learn the Green's operator of elliptic PDEs on point cloud data. The architecture efficiently learns representations for both local and far-field interactions. The core innovation is the local attention, a specialised local attention mechanism which models complex dependencies within a small neighbourhood of points. We demonstrate the effectiveness of this approach, and discuss possible extensions and modifications to the CloudNFMM architecture.

Introduction

Solving partial differential equations (PDEs) is fundamental to countless fields in science and engineering. While traditional numerical solvers are highly refined, they can be either brittle – needing to be tuned for each new problem – or computationally expensive for large computational domains. This has spurred the development of deep learning-based methods, particularly neural operators, which aim to learn the underlying solution operator mapping from input parameters to the solution function. Among these, the **F**ourier **N**eural **O**perator (**FNO**) (Kovachki et al. 2021a) has emerged as a state-of-the-art architecture, demonstrating remarkable success by performing convolutions in the frequency domain. However, a significant limitation of the FNO is its reliance on the Fast Fourier Transform (FFT), which constrains it to data structured on uniform, regular grids.

Many real-world problems are defined on irregular domains or are naturally represented by unstructured data, such as point clouds or meshes. To address this, architectures based on Graph Neural Networks (GNNs) and Transformers (Vaswani et al. 2017) have been proposed. However, neural operators based on these methods introduce significant bottlenecks for large-scale simulations; either requiring many message-passing steps or by relying on global attention mechanisms, incurring a $O(N^2)$ computational cost.

To overcome these challenges, we draw inspiration from a very successful class of direct solver algorithms: FMM. The FMM is designed to compute long-range interactions in N-Body problems with near-linear complexity, typically $O(N)$ or $O\left(N \log\left(N\right)\right)$. This is achieved by hierarchically decomposing the computational domain and using low-rank approximations of the interaction kernel for far-field interactions. This principle of separating local and global computations provides a blueprint for a more scalable and geometrically robust neural operator architecture.

Contribution We introduce the CloudNFMM, a neural operator which uses the hierarchical structure of the FMM for global interactions with a local attention mechanism for local interactions. The proposed architecture operates directly on unstructured points (point clouds), making it resolution-invariant and freeing it from the grid-based constraints of methods like the FNO. We demonstrate the efficacy of the CloudNFMM on a variety of time-harmonic PDE problems, showing that our model achieves performance that is superior to, or on par with, other established neural operator architectures at a fraction of the parameter count.

Background

Fast Multipole Method

We will outline a high-level discussion of the FMM's information flow. For a more detailed discussion of the FMM, see Appendix . The FMM (Rokhlin 1985) designed to efficiently compute long-range forces in N-Body problems utilising both a *low-rank approximation* of the kernel, $G(x, y)$, and a *hierarchical decomposition* of the domain D via a *quad-tree* in 2D (or an *oct-tree* in 3D).

The FMM decomposes the domain D into 4^L disjoint boxes (in 2D, it is 8^L in 3D), β_i, where L is the depth of the FMM's quad-tree. For all β_τ, we partition the domain into its far-field, $\mathcal{F}_\tau$, and near-field, $\mathcal{N}_\tau$, neighbours. A source-box β_σ belongs to the near-field $\mathcal{N}_\tau$ of a target-box β_τ if we define the far-field, $\mathcal{F}_\tau$, and near-field, $\mathcal{N}_\tau$, of β_τ respectively,

[*]Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies. Funding acknowledgements go at the end of the paper.

furthermore, we denote the sub-scripts τ and σ to indicate a given target box and source box respectively. Boxes belong to $\mathcal{F}_\tau$ if β_σ – with c_τ and c_σ being the centre of each respective box – satisfies $2b_l \leq |c_\tau - c_\sigma|$ with b_l being the length of a box at a given level l.

Upwards Pass: The upwards pass aggregates information from the source points up the tree using the following operators:
$\mathbf{T}^{\text{ofs}}_\sigma$ – this computes the compact representation vector $\mathbf{q}_\sigma$ for each box on the leaf level.
$\mathbf{T}^{\text{ofo}}_{\Sigma,\sigma}$ – shifts the $\mathbf{q}_\sigma$ between the tree's levels towards level 2 of the tree[1], creating the outgoing potential tensors[2] for each level l; $\mathbf{Q}_l$. This is done to the parent box $\mathbf{q}_\Sigma$ from the children boxes, $\mathbf{q}_\sigma \in \mathcal{C}_\Sigma$. This operation is described mathematically in (7) and (8).

Downward Pass: The downward pass gathers the vectors outgoing potentials tensors $\mathbf{q}_\sigma \in \mathbf{Q}_l$ corresponding to boxes $\beta_\sigma \in \mathcal{F}_\tau$ for each $\tau \in \mathbf{Q}_l$ using the following operators:
$\mathbf{T}^{\text{ifo}}_{\tau,\sigma}$ – this computes the incoming vector $\mathbf{h}_\sigma$ from the outgoing vector $\mathbf{q}_\sigma$.
$\mathbf{T}^{\text{ifi}}_{\tau,T}$ – shifts $\mathbf{h}_\tau$ between the tree's levels from level 2 to the leaves, going from a parent box β_T to $\beta_\tau \in \mathcal{C}_T$. This operation is described mathematically in (9).

Leaf Pass: The leaf pass computes the potential at a point x from the contributions from both $\mathcal{N}_\tau$ and $\mathcal{F}_\tau$ using the following operators:
$\mathbf{T}^{\text{tfi}}_\tau$ – evaluates the potential $\mathbf{h}_\tau$ at all the points $x_i \in \beta_\tau$, this is the contribution to the potential from the points $x_j \in \mathcal{F}_\tau$.
$G(x, y_j)$ – is the kernel, which directly evaluates the potential between x and all the points $y_j \in \mathcal{N}_\tau$. This operation is described mathematically in (1):

$$v(x) = \overbrace{\sum_{y_j \in \mathcal{N}_\tau \setminus \{x\}} G(x, y_j) f(y_j)}^{\text{Near Field Contribution}} + \overbrace{\mathbf{T}^{\text{tfi}}_\tau (x; \mathbf{h}_\tau)}^{\text{Far Field Contribution}} \quad (1)$$

Collectively, the upward and downward passes constitute the *tree-level* operations – they are responsible for communicating information across the domain by aggregating sources into compact representations (up the tree) and propagating their far-field influence back down (down the tree). The *leaf-level* operations – described in the leaf pass – involve direct kernel evaluation in a local neighbourhood. This separation of long-range (tree) and short-range (leaf) computations is the core principle we adapt in our architecture.

Requirements: To achieve resolution independence for the NFMM, we need to:
Firstly remove the dependence of the NFMM on requiring that data is on a uniform grid[3].
Secondly, reformulate the NFMM's local interactions to learn a local interaction kernel[4], and have the information flow as the near-field contribution in (1).

Neural Operator

Neural operators (Kovachki et al. 2021a; Kovachki, Lanthaler, and Stuart 2024), aim to learn operators between different function spaces with some light conditions on their domains. Originally outlined by Kovachki et al. (2021a), and further formalised by Berner et al. (2025), neural operators have two key properties. One, neural operators should be discretisation-agnostic[5]. Two, they should have a fixed number of parameters for every discretisation.

Neural operators were inspired by the DeepONet (Lu et al. 2021), neural operators are fashioned after a traditional deep learning architecture, where for each layer t contains a linear operation with a bias followed by a non-linearity. In the neural operators framework, there are 3 major components: the lifting operator, $\mathcal{P}$, the blocks, $\{\mathcal{B}_t\}_{t=1}^T$, and the projection operator $\mathcal{Q}$. Note that $\mathcal{P}$ and $\mathcal{Q}$ are channel wise and only $\mathcal{B}_t$ operate along the spatial domain. There are T blocks, with each block containing: a channel-wise linear layer, W_t, and spatial kernel operator, $\mathcal{K}_t$, followed by a non-linearity, σ_t,

$$\tilde{\mathcal{L}}_\theta = \mathcal{Q} \circ \underbrace{\sigma_T(W_T + \mathcal{K}_T + b_T)}_{\text{Block } T} \circ \cdots \circ \underbrace{\sigma_1(W_1 + \mathcal{K}_1 + b_1)}_{\text{Block } 1} \circ \mathcal{P}. \quad (2)$$

Let v_t be our solution at our current step and $\kappa^{(t)}$ be our learnt integral kernel at a layer t, which may depend on $(x, y, a(x), a(y), v_t(x), v_t(y))$. To mathematically define our spatial kernel operator – dubbed the 'non-local' operator – we represent $\mathcal{K}_t$ as an integral. For some measure, $d\nu_t(dy)$, on the domain of integration D, we define $\mathcal{K}_t$ in terms of $\kappa^{(t)}$ as follows:

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y, a, v_t) v_t(y)\, d\nu_t(dy). \quad (3)$$

Depending on the class of problems and type of kernel we aim to learn, the structure of the kernel, $\kappa^{(t)}$, and the computation of the integral transform in (3) can be simplified, giving rise to different architectures. Boullé and Townsend (2023), viewing operator learning through the lens of linear algebra, outline four main approaches: these are the 'Graph neural operator' (GNO), 'Low-rank neural operator', 'Multipole Graph neural operator', and the FNO.

Related work

Neural Fast Multipole Method The **N**eural **F**ast **M**ultipole **M**ethod (**NFMM**) (Fognini, Betcke, and Cox 2025), a novel architecture that integrates the hierarchical information flow of the classical FMM into a neural operator framework for learning the Green's operator of elliptic

[1] As the spatial resolution in the higher levels, levels 0 and 1, is too coarse to allow for separation of the near-field and far-field.

[2] Note that each $\mathbf{q}_\sigma \in \mathbf{Q}_l$ corresponds to a box β_σ on level l of the quad-tree.

[3] This would require that the $\mathbf{T}^{\text{ofs}}$, $\mathbf{T}^{\text{tfi}}$, and $\mathbf{G}_\theta$ operators are can handle variable sized inputs.

[4] With the following requirements: One, no self interactions for sources/targets in β_τ. Two, that we only compute the contribution to the points in β_τ, from the points in $\mathcal{N}_\tau$, not the other way around.

[5] This is typically loosened to being resolution-agnostic for architectures such as the FNO.

PDEs. The core idea is to replace the FMM's traditional, handcrafted translation operators, which depend on an analytically available Green's kernel, with MLPs. This approach preserves the FMM's efficient partitioning of near and far-field interactions, including its characteristic upward and downward passes through a hierarchical tree structure, while circumventing the need for a-priori knowledge of the interaction kernel. For a detailed breakdown of the NFMM architecture, we refer the reader to Appendix . Our present work is expanding the NFMM to be discretisation-agnostic – much like the original FMM.

$\mathcal{H}$-matrix neural networks Fan et al. (2019b) propose an approach to learning the solution operator for PDEs through adapting hierarchical matrices, $\mathcal{H}$-matrices, into a neural network architecture. This allowed these $\mathcal{H}$-matrices to solve non-linear problems, and to learn these matrices from data. This was done by introducing a local deep neural network at each hierarchical scale, allowing them to approximate nonlinear maps like those found within the Schrödinger equation. However, this method required storing separate basis functions at each level, leading to large memory requirements. Expanding upon this, Fan et al. (2019a) developed an improved architecture incorporating nested bases inspired by $\mathcal{H}^2$-matrices and the FMM. These operators are very parameter efficient, despite some implementation complexity, and could perhaps replace some components of the long-range interaction part of this work. However, this core of this work is focused on local interactions and ensuring that our FMM-based Neural Operators parameters are not dependent on discretisation; which this work doesn't address.

Multipole Graph Neural Operator: The **M**ultipole **G**raph **N**eural **O**perator (**MgNO**) (Li et al. 2020) is a model that merges concepts from graph neural operators and low-rank neural operators to efficiently learn PDE solution operators. Inspired by the FMM and $\mathcal{H}^2$-matrices, it uses a message-passing algorithm called a V-Cycle on hierarchical graphs to enforce a low-rank structure on the interaction kernel, particularly for long-range components. This architecture functions as an iterative solver, with the final learned kernel resembling a hierarchical $\mathcal{H}$-matrix (Martinsson 2019). The MgNO differs significantly from the NFMM; while both draw inspiration from the FMM, the NFMM is a more direct adaptation that replaces the FMM's handcrafted operators with learnable MLPs, while explicitly preserving the upward and downward pass structure. In contrast, the MgNO employs a more generalised graph-based V-Cycle for its message passing, focusing on kernel decomposition rather than adapting the FMM's information flow.

Graph Neural Operators: Work on learning physics simulations directly on meshes has been significantly advanced by modern deep learning architectures. The MeshGraphNets (Pfaff et al. 2020) architecture is a Graph Neural Network (GNN) using an Encode-Process-Decode structure. Its key innovation is a dual message-passing scheme that operates in two distinct spaces: mesh-space, using the mesh's connectivity to model internal dynamics like material properties, and world-space, using proximity-based edges to capture external interactions such as collisions. Building on this, **EAGLE** (Janny et al. 2023) addresses the challenge of modelling more complex, unsteady turbulent flows and the inefficiency of iterative message passing for capturing long-range dependencies using a novel mesh transformer architecture. To overcome the quadratic complexity of attention on large meshes, the model first performs geometric clustering and learned graph pooling to create a coarser representation of the mesh, then applies multi-head self-attention on the expressive cluster embeddings. This allows the model to integrate global information and capture long-range interactions, such as airflow patterns, in a single step, outperforming iterative GNNs like MeshGraphNet on complex benchmarks. Similarly, the CloudNFMM also avoids the quadratic-complexity of transformers by solving the global solve on a coarse-grid, however, EAGLE only uses attention on the coarse graph and uses a decoder to update the fine global mesh.

Transformer Neural Operators: Recent advancements in neural operators have focused on overcoming the geometric and discretisation limitations of earlier models for solving PDEs, leveraging the improvements in transformer implementation and theory. The **Geo**metry-aware **F**ourier **N**eural **O**perator (Geo-FNO) (Li et al. 2022) addresses a key constraint of the popular FNO, which is its reliance on uniform rectangular grids due to its use of the Fast Fourier Transform (FFT). The Geo-FNO introduces a framework that learns a diffeomorphic deformation to map from an irregular domain into a regular domain where the FNO can be efficiently applied, before the result is mapped back to the irregular domain. The **O**perator Trans**former** (OFormer) (Li, Meidani, and Farimani 2022) proposes an attention-based architecture that makes few assumptions about the input grid structure. It leverages self and cross-attention to function as a learnable integral operator, with the cross-attention mechanism decoupling the input and output domains to allow for queries at arbitrary locations. For time-dependent problems, the OFormer employs a recurrent MLP to propagate the system's dynamics efficiently in the latent space. The grap**H** transfor**M**er neur**A**l op**E**ra**T**or (**HAMLET**) (Bryutkin et al. 2024) is the first neural operator framework to employ a graph transformer for solving PDEs. HAMLET constructs a graph from the input data and uses graph transformer blocks for encoding, a cross-attention operator for integrating query locations, and a similar recurrent MLP as OFormer for time-dependent PDEs. These neural operators are similar to the CloudNFMM, using a transformer-based architecture to learn an integral operator. However, the CloudNFMM differs from these approaches by splitting the computational domain into long-range and local interactions, as opposed to using an attention-mechanism between all points in the domain.

Method

Cloud NFMM

In the CloudNFMM, we expand upon the original NFMM – outlined in Appendix – by reworking the $\mathbf{T}^{\text{ofs}}$, $\mathbf{T}^{\text{tfi}}$, and $\mathbf{A}$

operators; although we will denote $\mathbf{A}$ as $\mathbf{G}_\theta$ in this work. The driving motivation behind this work is to replace the $\mathcal{K}_t$ from (2) with an operator which models the information flow of the FMM. As the FMM is a hierarchical algorithm, our FMM-inspired neural operator is also a hierarchal algorithm; having both a *tree pass* and a *leaf pass*. The tree pass is handled by the original version of the NFMM and was constructed by simply replacing each FMM operator with either a linear layer, or a $2-$layer MLP. The leaf pass is the focus of this work, and is implemented via the use of a spatially local attention mechanism. In order to exchange information between the leaf level and the tree level, we also need to rework the $\mathbf{T}^{\text{ofs}}$ and $\mathbf{T}^{\text{tfi}}$ operators from the original NFMM to meet the requirements outlined above.

Data Preprocessing To enable the hierarchical structure of the NFMM for point-based data, the input domain is first partitioned into a uniform grid of square cells, referred to as boxes, similar to the patches used by a Vision Transformer (Dosovitskiy et al. 2020). Our input data, consisting of $\{\mathbf{x_i}\}_{i=1}^{N}$ points represented as a tensor of feature vectors $\mathbf{f}_i \in \mathbb{R}^{d_L}$, with the pre-processing partitioning assigning each point $\mathbf{x}_i$ to a specific patch. As this architecture should be resolution invariant, in general there will not be a constant number of points within each patch. To create a tensor with full dimensions, each patch's point list is processed to have a fixed size, N_b, which is the maximum number of points in a given patch[6]. The output of this preprocessing step is a structured tensor of shape $[B, M, N_b, d_L]$, where B is the batch size, M is the number of boxes, N_b is the (maximum) number of points in each box, and d_L is the dimension of the leaf feature space. This format is crucial for the hybrid local and hierarchical structure of the NFMM, as the hierarchical FMM operator works on the box grid while the local operator works on the points in the spatial boxes.

Cloud NFMM Components

OFS Operator The $\mathbf{T}^{\text{ofs}}$ operator is responsible for the crucial aggregation step in the upward pass of the NFMM. Its primary purpose is to compress the rich information contained within a set of points in a single box into a compact, representative feature vector for the box. This is achieved by first lifting the leaf-level features – $\mathbf{x}_i \in \mathbb{R}^{d_L}$ – into a higher-dimensional – $\mathbf{q}_i \in \mathbb{R}^{d_T}$ – space using a MLP. The representative vector – $\mathbf{q}_\sigma$ – for each box is then computed as a magnitude-weighted centre of mass of these lifted features. The magnitude of each lifted feature vector is used to determine the relative contribution of each point to the final aggregated vector. This approach – outlined in (4) – is similar to a soft aggregation, allows points with more significant or prominent features to have a greater influence on the final aggregated vector.

TFI Operator The $\mathbf{T}^{\text{tfi}}$ operator is the final stage of the downward pass, translating the coarse-level FMM approximations into point-level updates. It takes the aggregated feature vector from a parent box and uses it to update the individual point representations within each of its child boxes. This is achieved by an inverse-weighted update mechanism, which is outlined in (5). First, the operator calculates the distance between each individual point's feature vector and the incoming representative vector in the high-dimensional space – $d_i = \|\mathbf{q}_i - \mathbf{h}_\tau\|_2$. This distance is then used to compute an interaction weight, using an inverse weighting scheme, which is then normalised. The point's features are then updated by 'pulling' them toward the incoming vector, with the strength of this pull determined by the calculated weight. The updated features, $\tilde{\mathbf{q}}_i$, now incorporating information from the parent box, are projected back to the original dimension using a MLP,

$$\mathbf{q}_\sigma = \sum_{i \in \beta_\sigma} \frac{\|\mathbf{q}_i\|_2 \cdot \mathbf{q}_i}{\sum_{i \in \beta_\sigma} \|\mathbf{q}_i\|_2}, \tag{4}$$

$$\tilde{\mathbf{q}}_i = \mathbf{q}_i + \frac{\|\mathbf{q}_i - \mathbf{h}_\tau\|_2^{-1} \cdot (\mathbf{h}_\tau - \mathbf{q}_i)}{\sum_{i \in \beta_\tau} \|\mathbf{q}_i - \mathbf{h}_\tau\|_2^{-1}}. \tag{5}$$

Local Attention Operator In order to satisfy the requirements outlined above, the only architectures available were either GNNs, transformer-based architectures, and state-space architectures. Latent-space models were not considered for A due to the fixed state dimension found within these models[7], leaving GNNs and transformers as possible architectures to build the $\mathbf{G}_\theta$ operator. We note that transformers are a special class of message passing neural networks – this is outlined in Appendix , with a similar discussion seen within (Bryutkin et al. 2024) – thus, we have focused on transformers due to existing efficient implementations and as we do not need to construct graphs due to the patched nature of the data.

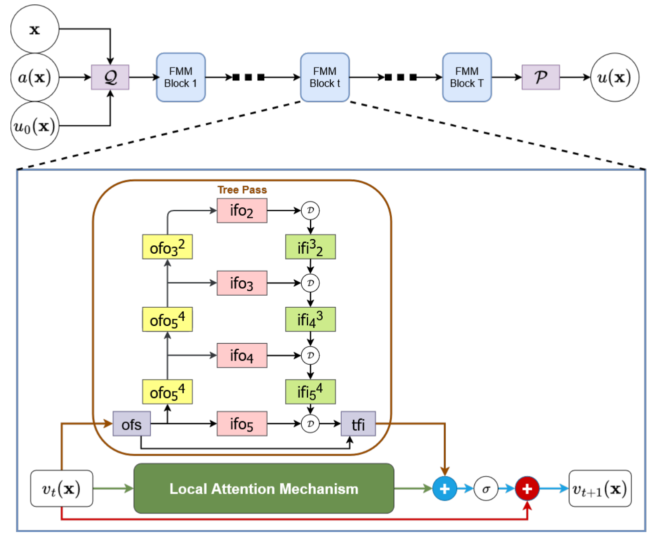

Figure 1: The architecture of the CloudNFMM neural operator.

[6]The value of N_b only needs to be done per example or batch, however to speed up training we process all examples in a dataset to have n_b points.

[7]As learning a fixed rank approximation to the Greens function – $G(x, y)$ – may cause an issue for oscillatory PDE problems.

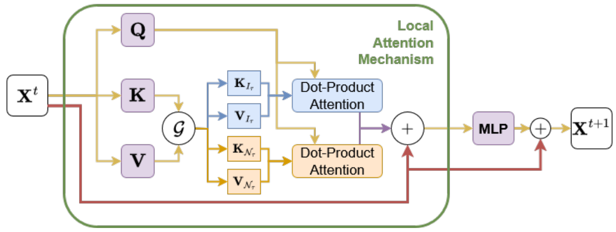

Figure 2: The architecture of the Local Attention Operator, where $\mathcal{G}$ is a gathering operation for the local boxes.

The algorithm represented in Figure 2 is a shared attention mechanism, using the same learnable parameters for both cross and self-attention but constructing the sequences from both the near field – $\mathcal{N}_\tau$ – and the points in β_τ respectively[8]. The new near-field operator, $\mathbf{G}_\theta(x_i, x_j)$ – represented in Figure 2 – is a local attention-based operator modelling the local contribution from (1). This is achieved by implementing a shared multi-head attention mechanism to compute interactions within the spatially local 3×3 neighbourhood. For each box, it gathers the features of the central box and their eight neighbours, performing two parallel attention passes: self-attention for within each box, and a cross-attention pass that incorporates the influence from neighbouring boxes. These outputs are added, and passed through an MLP and residual connections are applied – approximating the direct pairwise interaction kernel. We satisfy most of the requirements, as we mask-out self-interactions via a masked self-attention mechanism and cross-attention only computes the contribution from $\mathcal{N}_\sigma$ to β_τ. However – as we do not currently use a relative position encoding scheme – we currently use RoPE (Su et al. 2021) in the local attention operator, however, using a relative position encoding scheme is a current focus of future work.

Numerical Experiments

To evaluate the CloudNFMM against other neural operators, we utilised two time-harmonic datasets; PDEBench (KHOO, LU, and YING 2020), and WaveBench (Liu et al. 2024a). These datasets are both for time-harmonic PDE problems, these were used as the CloudNFMM is a direct solver and is in its present form not designed to solve time-dependent PDE problems.

The same hyperparameters were used for all the following numerical experiments on the CloudNFMM architecture. All scores in the tables below are the average relative L_2 error[9], $\mathcal{E}_2^{\text{rel}}$, values over the validation set. We trained the CloudNFMM using the average relative L_2 loss, $\mathcal{L}_2^{\text{rel}}$, for more information on training and implementation details see Appendix .

[8]We split up the attention between the self and local contribution, this is done to prevent saturation of the Softmax within the attention scores.

[9]This is also occasionally called the normalised root mean squared error (nRMSE).

Results

WaveBench Table 1 shows the results of the CloudNFMM against the WaveBench benchmarks, which contain different Helmholtz problems. These results for the baseline architectures are referenced from the WaveBench paper, they have all been trained with the protocol outlined in Appendix .

Table 1: Results of neural operators on the 2D acoustic Helmholtz datasets

GRF Type	Freq.	CloudNFMM	FNO-depth-4	FNO-depth-8
Isotropic	10 Hz	**0.037**	0.063	0.040
	15 Hz	0.059	0.093	**0.057**
	20 Hz	**0.064**	0.122	0.070
	40 Hz	**0.111**	0.283	0.165
Anisotropic	10 Hz	0.034	0.059	**0.025**
	15 Hz	0.050	0.098	**0.039**
	20 Hz	0.064	0.135	**0.060**
	40 Hz	**0.160**	0.315	0.172
Total Parameters		**1.8M**	4.2M	8.4M

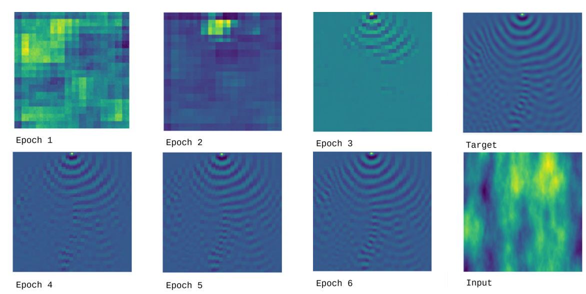

Figure 3: The first 6 epochs of the CloudNFMM model training on the ‘Anisotropic 40Hz‘ dataset.

We present Figure 3, where we have visualised the first 6 epochs of the CloudNFMM model being trained on the ‘Anisotropic 40Hz‘ dataset.

PDEBench Table 2 shows the results of the CloudNFMM against the Darcy Flow benchmarks. These results for the baseline architectures are referenced from the HAMLET paper (Bryutkin et al. 2024), this was done so that we could best compare the CloudNFMM against other graph and transformer-based neural operators. They were all trained with the hyperparameters from the default implementation of baseline methods found within their respective papers[10], and trained and evaluated on 64×64 grids.

Analysis

The results indicate that the CloudNFMM approach, outlined in this paper, is a potentially powerful performance-to-parameter efficiency class of neural operators. This is most

[10]The authors note that the, ”...dataset-specific hyperparameters follow the PDEBench setting, while model-specific hyperparameters follow the default setting of baseline methods suggested by the code repositories or their papers.”

Table 2: Results of Neural Operators on the 2D Darcy Flow datasets.

Darcy Flow β	OFormer	GeoFNO	HAMLET	CloudNFMM
$\beta = 0.01$	**2.21e-03**	2.70e-03	2.45e-03	2.53e-01
$\beta = 0.1$	2.55e-03	4.15e-03	**2.60e-03**	1.06e-01
$\beta = 1.0$	3.00e-03	6.20e-03	**2.74e-03**	4.50e-02
$\beta = 10.0$	7.32e-03	2.08e-02	**5.51e-03**	1.39e-02
$\beta = 100.0$	4.91e-02	1.65e-01	3.37e-02	**1.18e-02**

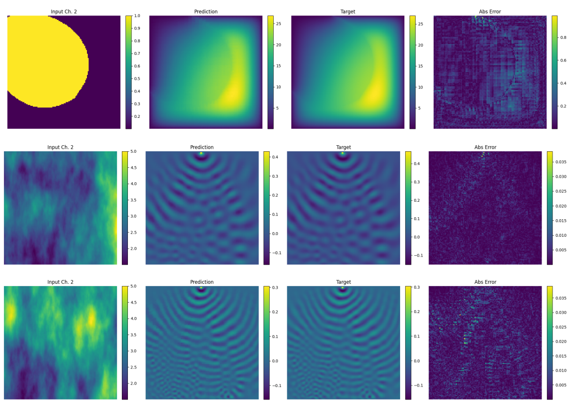

Figure 4: Examples from the validation set of: (*Top*) Darcy Flow $\beta = 100$, (*Middle*) Helmholtz Anisotropic 20Hz, and (*Bottom*) Helmholtz Anisotropic 40Hz

apparent from the WaveBench dataset results – see Table 1, Figure 4, and Figure 3. Here, the model demonstrates exceptional performance, consistently outperforming most baseline models – particularly at higher frequencies – while having the lowest parameter count (1.8M) by a significant margin. This suggests that the FMM-inspired hierarchical structure is highly effective at capturing the complex, this is supported by the first few epochs within training – as seen in Figure 3. Here the model quickly builds robust representations for far-field interactions using the hierarchical component, while the local transformer layer uses these to produce a smooth field. Conversely, the model's performance on the PDEBench Darcy Flow dataset – see Table 2 – is less competitive, especially for low to moderate values of the permeability coefficient β. Here the current CloudNFMM approach struggles compared to fully transformer-based operators – like OFormer and HAMLET.

Limitations and Next Steps

However, it is noteworthy that as β increases to 100.0 – representing a very high-contrast problem – the CloudNFMM's performance improves to best-in-class. This result suggests that the current mechanism for exchanging information between the hierarchical FMM tree and the local attention operator may be a cause of the performance bottleneck. This suggests that the mechanism linking the coarse FMM tree and the fine-grained leaf interactions needs refinement. The current $\mathbf{T}^{\text{ofs}}$ and $\mathbf{T}^{\text{tfi}}$ operators might prevent local operator from building good representations for classes of PDEs with very smooth solutions.

Secondly, the current architecture is only formulated as a direct solver for time-harmonic problems[11]. We suggest addressing this using the same mechanism used by Bryutkin et al. (2024) in HAMLET, and originally outlined by Li, Meidani, and Farimani (2022) for the OFormer – incorporating a recurrent structure in the latent space after the final layer of the model (before we apply $\mathcal{Q}$ in (2)), which would propagate solutions through time.

Finally, while we deploy RoPE within the local attention layer, we aim to build representations which are informed by a learnt relative positional encoding. The absence of this information[12] may be preventing the CloudNFMM from learning fine-grained, geometry-dependent physical laws within the local neighbourhood.

Additional directions we aim to address in future work are: experimenting with replacing the MLPs in the CloudNFMM with SIREN (Sitzmann et al. 2020) – and KAN (Liu et al. 2024b) – models, and more comprehensive testing of the CloudNFMM on variable point densities and irregular samplings.

Conclusion

In this work, we introduced the Cloud Neural Fast Multipole Method (CloudNFMM), a discretisation-agnostic neural operator designed to solve time-harmonic PDEs on variable point densities (point cloud) data. We achieve this by adapting the information flow of the FMM, creating a hybrid method containing a hierarchical and local component. This allows our model to efficiently capture both long-range and short-range physical interactions without being constrained to a regular grid. Our experiments demonstrate that the CloudNFMM achieves state-of-the-art performance and remarkable parameter efficiency on challenging, highly oscillatory wave propagation problems. While its performance on smooth problems is an area of improvement in the coupling of local and global information flow, the results underscore the significant potential of this simple approach. The CloudNFMM represents a promising step towards creating scalable, discretisation-agnostic neural operators, indicating that fusion of principled numerical algorithms and deep learning architectures is a fruitful path for the future of scientific machine learning.

Acknowledgments

We acknowledge the use of Google's LLM, Gemini, as a writing assistant. The model was used to enhance grammar, improve phrasing, and improve the overall clarity of the text – but was not used for retrieval, discovery, or ideation. The authors carefully reviewed, revised, and take full responsibility for the scientific integrity and final content of this paper.

[11] This could be formulated as a fixed time horizon solver as well.

[12] This already included as an input to $\mathcal{P}$ in (2), but this indicates that this is a weak signal to the local attention operator.

References

Berner, J.; Liu-Schiaffini, M.; Kossaifi, J.; Duruisseaux, V.; Bonev, B.; Azizzadenesheli, K.; and Anandkumar, A. 2025. Principled approaches for extending neural architectures to function spaces for operator learning.

Boullé, N.; and Townsend, A. 2023. A Mathematical Guide to Operator Learning. ArXiv:2312.14688 [cs, math].

Bryutkin, A.; Huang, J.; Deng, Z.; Yang, G.; Schönlieb, C.-B.; and Aviles-Rivero, A. 2024. HAMLET: Graph transformer neural operator for partial differential equations.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2020. An image is worth 16x16 words: Transformers for image recognition at scale.

Fan, Y.; Feliu-Faba, J.; Lin, L.; Ying, L.; and Zepeda-Nunez, L. 2019a. A multiscale neural network based on hierarchical nested bases. ArXiv:1808.02376.

Fan, Y.; Lin, L.; Ying, L.; and Zepeda-Nunez, L. 2019b. A multiscale neural network based on hierarchical matrices. arXiv:1807.01883.

Fognini, E. M.; Betcke, M. M.; and Cox, B. T. 2025. Learning greens operators through hierarchical neural networks inspired by the Fast Multipole Method.

Janny, S.; Béneteau, A.; Nadri, M.; Digne, J.; Thome, N.; and Wolf, C. 2023. Eagle: Large-scale learning of turbulent fluid dynamics with mesh transformers.

KHOO, Y.; LU, J.; and YING, L. 2020. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 1–15.

Kovachki, N.; Li, Z.; Liu, B.; Azizzadenesheli, K.; Bhattacharya, K.; Stuart, A.; and Anandkumar, A. 2021a. Neural Operator: Learning Maps Between Function Spaces. *arXiv:2108.08481 [cs, math]*. ArXiv: 2108.08481.

Kovachki, N. B.; Lanthaler, S.; and Stuart, A. M. 2024. Operator learning: Algorithms and analysis.

Kovachki, N. B.; Li, Z.; Liu, B.; Azizzadenesheli, K.; Bhattacharya, K.; Stuart, A. M.; and Anandkumar, A. 2021b. Neural Operator: Learning Maps Between Function Spaces. *CoRR*, abs/2108.08481.

Li, Z.; Huang, D. Z.; Liu, B.; and Anandkumar, A. 2022. Fourier neural operator with learned deformations for PDEs on general geometries.

Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; and Anandkumar, A. 2020. Multipole graph neural operator for parametric partial differential equations.

Li, Z.; Meidani, K.; and Farimani, A. B. 2022. Transformer for partial differential equations' operator learning.

Liu, T.; Benitez, J. A. L.; Faucher, F.; Khorashadizadeh, A.; de Hoop, M. V.; and Dokmanić, I. 2024a. WaveBench: Benchmarking Data-driven Solvers for Linear Wave Propagation PDEs. *Transactions on Machine Learning Research*.

Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T. Y.; and Tegmark, M. 2024b. KAN: Kolmogorov-Arnold Networks.

Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; and Karniadakis, G. E. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3): 218–229. Number: 3 Publisher: Nature Publishing Group.

Martinsson, P.-G. 2019. *Fast Direct Solvers for Elliptic PDEs*. Society for Industrial and Applied Mathematics. ISBN 9781611976045.

Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; and Battaglia, P. W. 2020. Learning mesh-based simulation with graph networks.

Rokhlin, V. 1985. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2): 187–207.

Sitzmann, V.; Martel, J. N. P.; Bergman, A. W.; Lindell, D. B.; and Wetzstein, G. 2020. Implicit neural representations with periodic activation functions.

Su, J.; Lu, Y.; Pan, S.; Murtadha, A.; Wen, B.; and Liu, Y. 2021. Roformer: Enhanced Transformer with Rotary Position Embedding.

Su, J.; Lu, Y.; Pan, S.; Wen, B.; and Liu, Y. 2022. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint arXiv:2104.09864*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need.

Ying, L.; Biros, G.; and Zorin, D. 2004. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2): 591–626.

The FMM

The FMM is originally an efficient, hierarchical, numerical algorithm for computation of long-range forces in N-Body problems within gravitational and electrostatic fields developed by Rokhlin (1985) and has been extended by Ying, Biros, and Zorin (2004) to apply the FMM to any Elliptic PDE with a Green's kernel. We will outline a high-level discussion of the FMM's information flow, for a deeper handling and derivation of the FMM we refer you to Martinsson (2019) for reference material. The FMM belongs to a family with linear or close to linear complexity for evaluating all pairwise interactions between n-particles, which is achieved by using two key ideas: a low-rank decomposition of the kernel, as seen in Figure 6, and hierarchically partitioning the spatial domain.[13] The FMM was originally designed to solve a N-Body interaction problems of the form (6), with $G(x, y)$ being the Green's kernel of the underlying physical problem, x_i the set of point locations, ϕ_i the set of corresponding sources, and $u(x_i)$ being the set of potentials we wish to compute for all $1 \leq i \leq N$.

The functionality of the FMM stems from approximating far-field interactions using translation operators while directly computing only near-field interactions. We can best describe how these translation operators work together to

[13] This is done via a Quadtree in $2D$ and an OctTree in $3D$.

compute a full level of the FMM by inspecting the operators needed to compute the interaction between two sufficiently separated boxes β_σ and β_τ. Here the τ and σ subscripts indicate the target box and source box respectively, and ‘sufficiently separated’ means that $2b \leq \|c_\tau - c_\sigma\|$ with c_τ and c_σ being the centre of β_τ and β_σ respectively and b being the length of a box at a given coarseness.

We denote $\mathcal{F}_\tau = \{\beta_\sigma | 2b \leq \|c_\tau - c_\sigma\|_1\}$ and $\mathcal{N}_\tau = \{\beta_\sigma | 2b > \|c_\tau - c_\sigma\|_1\}$ to be the *far-field* and *near-field* of β_τ respectively. We can compute v_τ from ϕ_σ by either a direct evaluation of $G(x,y)$ or compute it approximately by using the operators defined in equations (7) and (10).

$$u(x_i) = \sum_{j=1}^{N} G(x_i, x_j)\phi_j, \quad i = 1, 2, 3, ..., N \qquad (6)$$

During the *upwards pass*, the sources ϕ_σ within a region β_σ are translated into a single, compact outgoing vector, $q_\sigma \in \mathbb{R}^m$. Next, the *downward pass* maps this outgoing vector to a compact incoming vector, $h_\tau \in \mathbb{R}^m$, which is then propagated from the root down to the leaf level. Finally, the *leaf level pass* expands the far-field vector h_τ into approximate potentials v_τ and combines them with the direct evaluation of $G(x,y)$ for near-field particles.

How to Construct a Quad-Tree

The FMM uses a quad-tree to hierarchically decompose a 2D domain, it is constructed in the following way:

- Let the total domain be represented by a unit square $[0,1] \times [0,1]$.
- This domain is recursively and uniformly subdivided down to a fixed depth, d.
- The number of patches, M, at this finest level, is given by $M = 4^d$.
- Each patch $m \in \{1, \ldots, M\}$ is a square region of side length $l = 1/2^d$.

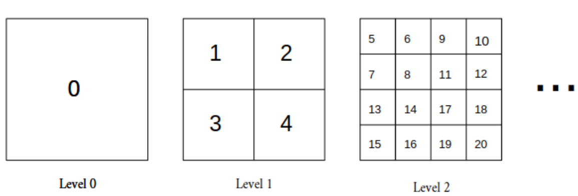

Figure 5: The Subdivision of the domain $\mathcal{D}$ into a QuadTree.

Upwards pass

We begin with (7), which embeds the source terms into an outward potential in $\mathbb{R}^m$. To avoid repeated computation, this potential, $q_{\tau;l}$,[14] is translated from level l in the tree, to level $l-1$. This is done by another operator,[15] $\mathbf{T}^{\text{ofo}}$, which combines the four child outward potentials into one outward potential for the parent box, β_Σ, with the potential centred at the centre of the parent box. This is mathematically represented in (8), letting $\mathcal{C}_\Sigma$ denote the children boxes of β_Σ.

$$q_\sigma = \mathbf{T}^{\text{ofs}}(\phi_\sigma) \qquad (7)$$

$$q_\Sigma = \sum_{\tau \in \mathcal{C}_\Sigma} \mathbf{T}_l^{\text{ofo}}(q_\tau) \qquad (8)$$

Downward pass

Starting at level 2 and propagating down to the leaf level[16], denoted as level l, we combine potentials from the far-field. As moving down the quad-tree allows for finer spatial refinement, we can split up the incoming potential for a box, $h_{\tau,k} = h_{\tau,k}^{\mathcal{P}} + h_{\tau,k}^{\mathcal{N}}$, into two distinct components to reuse computation from the previous level. These two distinct components correspond to potential from the previous level of refinement, $h_{\tau,k}^{\mathcal{P}}$, and a component corresponding to the increased refinement from descending the tree, $h_{\tau,k}^{\mathcal{N}}$. To compute $h_{\tau,k}^{\mathcal{N}}$ we apply $\mathbf{T}_k^{\text{ifo}}$ to every sufficiently separated box not within the previous level of refinement, denoted $\mathcal{U}_\tau$. The incoming potential from the parent box corresponds to $h_{\tau,k}^{\mathcal{P}}$, this is shifted from the parent box, β_T, to the children boxes, β_τ, by $\mathbf{T}^{\text{ifi}}$. We define $\mathcal{D}_k$ to be the downward pass for level k, which is mathematically represented for a single target box, β_τ, in (9).

$$h_\tau = \mathbf{T}_k^{\text{ifi}} h_T + \sum_{\sigma \in \mathcal{U}_\tau} \mathbf{T}_k^{\text{ifo}} q_\sigma \qquad (9)$$

Leaf level Pass

At the finest level of refinement, the leaf level, we are left to calculate the contribution from both h_τ and from the points in the near-field, $\mathcal{N}_\tau$. We apply $\mathbf{T}^{\text{tfi}}$ to expand h_τ into the far-field contribution of v_τ. Those sources which lie within $\mathcal{N}_\tau$ we may compute by directly evaluating $G(x,y)$ between the sources in the near-field. This process is mathematically represented in (10).

$$v_\tau(x_i) = \mathbf{T}^{\text{tfi}}(h_\tau) + \sum_{\substack{j \in I_\tau \\ i \neq j}} G(x_i, x_j) + \sum_{\substack{\sigma \in \mathcal{N}_\tau \\ j \in I_\sigma}} G(x_i, x_j) \qquad (10)$$

The Neural FMM

The efficiency of the FMM stems from approximating far-field interactions using translation operators while computing near-field interactions directly. However, a fundamental limitation of the traditional FMM is the requirement for an explicit, analytically available Green’s kernel to derive

[14] The number corresponds to which level of the tree that the operator or vector corresponds to. For example $\mathbf{T}_2^{\text{ifo}}$ correspond to $\mathbf{T}^{\text{ifo}}$ on level 2 of the Tree.

[15] We simplify this process by only having one operator for $\mathbf{T}^{\text{ofo}}$ and $\mathbf{T}^{\text{ifi}}$.

[16] As the spatial resolution in the higher levels, levels 0 and 1, is too coarse to allow for separation of the near-field and far-field.

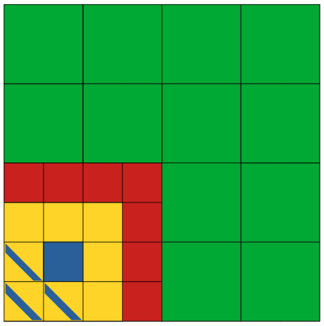

Figure 6: A visualisation of the decomposition of $\mathcal{D}$ about β_τ, the blue square. Here we can see the different interaction sets of β_τ, with the dashed blue and yellow boxes being the parent of β_τ, β_T. The yellow boxes being $\mathcal{N}_\tau$, the green boxes being the far field of β_T, $\mathcal{F}_T$. The red and green (technically the children of the green boxes, but they define the same region) boxes being the far field of β_τ, $\mathcal{F}_\tau$ and the red boxes being its unique far field, $\mathcal{U}_\tau$.

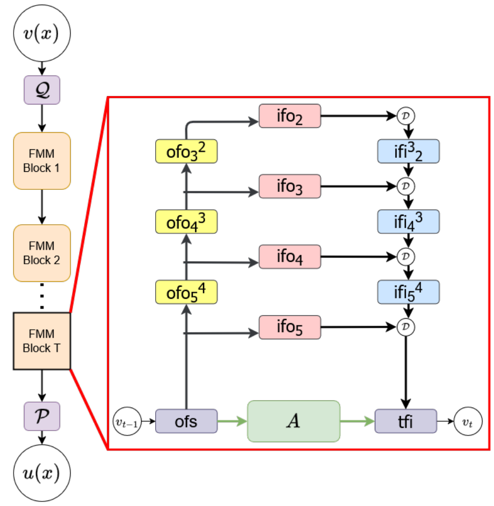

Figure 7: The Deep Neural FMM Architecture – We follow the neural operator framework from (2), with a lifting function $\mathcal{Q}$ and a projection function $\mathcal{P}$, while replacing each $\mathcal{K}_t(v_t))$ with a Neural FMM Block (outlined within the red box). Within the Neural FMM Block, v_{t-1} passes up the Tree via the Upward Pass (7) and (8). This we then apply $\mathbf{T}^{\text{ifo}}_l$ to convert the outgoing potentials to their incoming potentials followed by propagating the information down the Tree via the Downward pass (9). The Leaf Pass then computes the contribution from the near-field, $\mathcal{N}_\tau$, using a MLP.

these operators, which is difficult for problems in heterogeneous domains or where the kernel is unknown. Building upon the discussion of neural operators, our contribution is the **Neural Fast Multipole Method**, which integrates the information flow of the FMM while replacing the handcrafted, kernel-dependent translation operators with learnt operators parameterised by a learnable operators. We leverage the FMM's hierarchical partitioning and computation flow, outlined in (7) and (10), to split up and learn representations of local and far-field interactions. These passes are integrated into a single computational unit, the **Neural FMM Block**, with multiple of these blocks stacked together to form a deeper model, the **Deep Neural FMM**, enhancing expressivity of the model and mirroring (2).

Neural FMM implementation

The Neural FMM deviates from this by replacing each translation operator with an MLP while still following the non-local information flow for computing the far-field contribution, namely summing contributions from sufficiently separated boxes, rather than using a local operation at each level. The largest deviation from the FMM has been using one operator per level, $\mathbf{T}^{\text{ifo}}_{\theta;k}$, to represent the family of linear maps which are derived from the translation function formula[17] from level k, represented by $\mathbf{T}^{\text{ifo}}_{\tau,\sigma}$; with a different matrix operator for each τ, σ pair.

Position encoding As the operators are applied channel-wise to every element in out domain at once, the network does not inherently know the spatial position of each element, which is core to how the Multipole-to-Local translation formula in the FMM performs the translation from outgoing potentials to incoming potentials. This required the inclusion of a spatial encoding scheme to reintroduce this spatial dependence for our MLP's. This was handled by the use of Rotary Position Embeddings (RoPE) (Su et al. 2021, 2022) applied to the vectors corresponding to each box, using the position of each box in the 1D Morton ordering as the position for RoPE. This approach was chosen as it was found to encode position information more directly, leading to better preservation of the spatial relationships between boxes when compared to additive sinusoidal encodings, and simpler to implement than a custom position encoding scheme.

[17]This is also called the Multipole-to-Local translation formula in the literature.

Downward pass implementation The summations over interaction sets, particularly in the Downward Pass for the unique far-field $\mathcal{U}_\tau$ (9) is computationally intensive due to the non-local/non-contigous locations of the boxes within $\mathcal{U}_\tau$ with respect to the location of β_τ. In order to increase the efficiency of the aggregation within the downward pass, we pre-compute masks corresponding to these interaction sets at the initialisation of the architecture.

Attention as Message Passing

Now that we have outlined that self-attention and cross-attention can be thought of as working on K_N (a directed complete graph) or $\mathbf{K}_{N,M}$ (a complete undirected bipartite graph) respectively, we will now outline the corresponding form of:

- The Message Function: $K_\theta(\bullet, \bullet)$
- The Aggregation Function: $\square(\bullet, \cdots, \bullet)$
- The Update Function: $U(\bullet, \bullet)$

Without loss of generality, we will only consider the self-attention mechanism applied to a sequence of tokens

$\{t_1, \ldots, t_N\}$ with input embeddings $\{x_1, \ldots, x_N\}$, as the only difference between cross and self attention is the topology of the underlying graph[18] while the mechanism itself is the same.

Message Function

In a general MPNN, the message function $M_{vw}^{(t+1)} = M_t(x_v^t, x_w^t, e_{vw})$ computes a *message* dependent on the source node w, the target node v, and the edge features e_{vw}. Suppose we are on a graph without edge embeddings, so the adjacency matrix is a binary matrix indicating if nodes are connected together or not; the message function would then drop the dependence on e_{vw}. If we constrict the class of message functions to some kernel function, $K(\bullet, \bullet) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. Within the attention mechanism, this K is actually a bilinear product parameterised by two affine transformations, Key - $\mathbf{K}$ and Query - $\mathbf{Q}$, which means K has the following form:

$$a_{i,j} = K(x_i, x_j) = (\mathbf{K}x_i)^T(\mathbf{Q}x_j) \tag{11}$$

Aggregation Function

Since attention is a weighted sum, the aggregation function is summation, to produce the *context vector* c_i for token/node x_i. However, within attention, there are two additional permutation invariant functions we apply. Firstly, we apply Softmax to the attention scores to compute a distribution, secondly we then apply the affine transformation - $\mathbf{V}$ - to the vector associated with the node/token, so the aggregation function has the following form:

$$c_i = \sum_{j \in N(x_i)} \left[\frac{\exp(a_{i,j})}{\sum_{k \in N(x_i)} \exp(a_{i,k})} \mathbf{V} \right] x_j \tag{12}$$

$$c_i = \sum_{j \in N(x_i)} \left[\frac{\exp\left(K(x_i, x_j)\right)}{\sum_{k \in N(x_i)} \exp\left(K(x_i, x_k)\right)} \mathbf{V} \right] x_j \tag{13}$$

$$c_i = \sum_{j \in N(x_i)} \left(\mathbf{Softmax}_{N(x_i)} \left[K(x_i, x_j) \right] \right) \mathbf{V} x_j \tag{14}$$

Where $\mathbf{Softmax}_{N(x_i)}$ is Softmax normalised with respect to the neighbourhood of edges from x_i. As summation, Softmax, and matrix multiplication are permutation invariant, this is a valid message passing aggregation function.

Multiple Heads Multi-head attention does not change the core message passing framework, but instead parallelises the core logic over h *heads*, which are subspace projections in which we perform attention independently. Each head $h \in \{1, \ldots, H\}$ contain their own set of affine transformations, $(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h) : \mathbb{R}^{\frac{D}{H}} \to \mathbb{R}^{\frac{D}{H}}$, although in practice one large affine transformation is used for the token sequence, which is then reshaped to match the number of heads. The individual head outputs are concatenated to form a single, larger aggregated message, $c_i^H = c_i^{h_1} \| c_i^{h_2} \| \cdots \| c_i^{h_H}$. This concatenated vector is then passed through a final affine transformation, parameterised by $\mathbf{P}^O$, to produce the final output embedding:

$$c_i = \mathbf{P}^O \left(c_i^H \right) = \mathbf{P}^O \left(c_i^{h_1} \| c_i^{h_2} \| \cdots \| c_i^{h_H} \right) \tag{15}$$

In this view, multi-head attention is equivalent to computing multiple directional edges between each of the token/node embeddings, with the output of each of these subspace attention mechanisms being combined in the aggregation step.

$$\begin{aligned}
K_h(x_i, x_j) &= (x_i \mathbf{K})^T (\mathbf{Q} x_j) \\
c_i^h &= \sum_{j \in N(x_i)} \left[\frac{\exp\left(K_h(x_i, x_j)\right)}{\sum_{k \in N(x_i)} \exp\left(K_h(x_i, x_k)\right)} \mathbf{V}_h \right] x_j \\
c_i &= \mathbf{P}^O \left(c_i^{h_1} \| c_i^{h_2} \| \cdots \| c_i^{h_H} \right)
\end{aligned}$$

Since the update function is applied to the context vectors c_i, multi-headed attention only modifies the message and aggregation function of the attention mechanism.

Update Function

Once we have computed the context vector c_i via the message and aggregation function, we then need to update the representation of the token/node with respect to the context vector. In standard Transformer architectures the update function is a MLP with residual connections before and after the MLP, there are typically several normalisation operations[19], so the form of the update function is:

$$U(x_i, c_i) = x_i + \mathbf{MLP}[x_i + c_i] \tag{16}$$

Thus, we can see that the attention mechanism implements a message passing scheme on a graph, where each weight is determined via the kernel message function $a_{ij} = K(x_i, x_j)$ - the query-key mechanism.

CloudNFMM Implementation Details

Metrics and Loss Functions

Relative $\mathbf{L}_p$ The relative loss function – $\mathcal{L}_p^{\text{rel}}$ – computes a l_p norm between the predicted and ground truth values, which is normalised by the l_p norm of the ground truth – as originally outlined by N. Kovachki et al. (Kovachki et al. 2021b). For a predicted output tensor, v, and ground truth tensor, u, the relative error loss and relative error metric – $\mathcal{E}_p^{\text{rel}}$ – are computed as:

$$\mathcal{L}_p^{\text{rel}}(v, u) = \frac{1}{N} \sum_{i=1}^{N} \frac{\|v_i - u_i\|_p}{\|u_i\|_p} \tag{17}$$

$$\mathcal{E}_p^{\text{rel}}(v, u) = \frac{\|v - u\|_p}{\|u\|_p} \tag{18}$$

[18] However, assuming $\mathbf{S}$ or $\mathbf{T}$ are disjoint sets does allow for more efficient computation.

[19] These are typically either a pre-/post-application of LayerNorm between the residual connection and application of the MLP, but I have removed them for simplicity as they are added to improve stability during training.

where p represents the order of the norm, and N is the batch size. This loss provides a scale-invariant measure of error, particularly useful when dealing with solutions that may vary significantly in magnitude.

Training Protocols

Default Training Protocol We found that training using Table 3 provided satisfactory results on a large class of problems.

Table 3: Default Training Protocol for the CloudNFMM

Parameter	Value
# of Epochs	50
Optimiser	AdamW
Scheduler	OneCycle
lr_{start}	3e-4
lr_{end}	3e-6
Train/Test Split	$80/20$

WaveBench Training Protocol This is the training protocol outlined in the WaveBench paper can be seen in Table 4.

Table 4: WaveBench Training Protocol

Parameter	Value
# of Epochs	50
Optimiser	AdamW
Scheduler	Cosine Annealing
lr_{start}	1e-4
lr_{end}	1e-6
Train/Test Split	$99/1$

Note: In the original WaveBench paper, they have $lr_{\text{start}} =$ 1e-3 & $lr_{\text{end}} =$ 1e-5. However, due to the training instabilities of transformer-based architectures at high learning rates, we reduced both lr_{start} & lr_{end} by an order of magnitude.