LAGENCODER: A NON-PARAMETRIC METHOD FOR REPRESENTATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Non-parametric encoders offer advantages in interpretability and generalizability. However, they often perform significantly worse than deep neural networks on many challenging recognition tasks, and it remains unclear how to effectively apply these techniques to such tasks. In this work, we introduce LagEncoder, a non-parametric, training-free feature extraction method based on finite element basis functions. Our encoder features a universal architecture that can be applied to various types of raw data and recognition tasks. We found that LagEncoder effectively overcomes the limitations of neural networks in regression problems, particularly when fitting multi-frequency functions. The LagEncoder-based model converges quickly and requires low training costs, as only the head is trained. Additionally, LagEncoder provides an efficient parameter-efficient fine-tuning approach. Our experiments on the ImageNet dataset demonstrate that pre-trained models using LagEncoder achieve performance improvements within just one training epoch. Furthermore, it does not require adjustments to the original training recipe, and the model's total parameters remain nearly unchanged. Our evaluation of the scaling law for model performance indicates that using LagEncoder is more cost-effective than merely increasing the model size.

1 INTRODUCTION

Neural networks have played a pivotal role in the evolution of artificial intelligence, particularly excelling in challenge recognition tasks, where their performance has, in some cases, surpassed human-level capabilities. Furthermore, advances in transfer learning have demonstrated that neural network encoders exhibit notable domain adaptation properties, allowing them to generalize across disparate tasks and data distributions. For instance, an encoder pre-trained for an image classification task can be effectively transferred to an object detection task, even when the underlying data distributions differ, while still maintaining robust feature extraction capabilities.

However, traditional machine learning models often demonstrate better domain adaptation. They provided such feature extractors that require no training and are independent of specific recognition tasks (Pearson, 1901; Sparck Jones, 1972). For example, the kernel functions used in Support Vector Machines (SVM) are unrelated to data labels, resulting in non-trainable feature extractors that inherently provide complete domain adaptation. While traditional models typically offer higher interpretability, they often perform significantly worse than neural networks in specific, highly challenging recognition tasks. It is worth noting that many neural network architectures are based on mathematical methods that do not depend on labeled data. For instance, the linear layer and the attention layer rely on inner products for feature extraction, where the vector basis in linear algebra is fixed. Similarly, convolution layers perform feature extraction through convolution, in the Fourier transform, the kernel function is predefined and independent of the target.

We focus on constructing an encoder that is practical for challenging recognition tasks with complete domain adaptation. Through our extensive experiments, we found that, as Fourier transforms, neural networks often struggle to fit multi-frequency functions effectively (see Section 3.1.1). Specifically, when fitting functions with sharp transitions or localized features, a large number of Fourier terms may be required, which can reduce computational efficiency and potentially lead to overfitting. Similarly, neural networks often need to be much deeper, resulting in a significant increase in model size, to fit such functions properly. In numerical simulation problems, Finite Element Method (FEM)

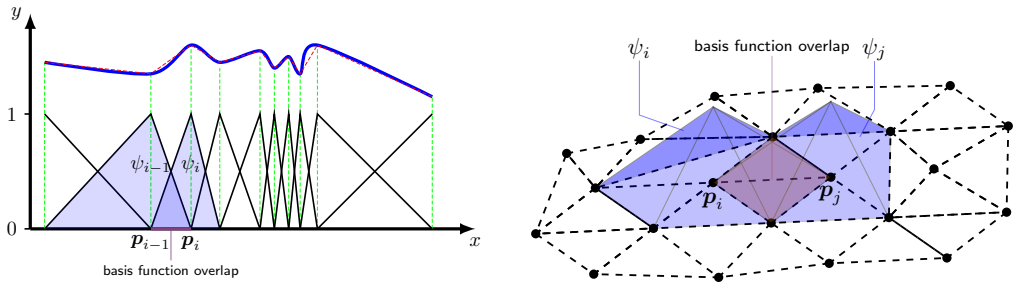


Figure 1: Left - The $F(\boldsymbol{x})$ (solid blue line) is approximated with $f(\boldsymbol{x};\boldsymbol{\theta})$ (dashed red line), which is a linear combination of linear basis functions (ψ_i is represented by the solid black lines). Right - Tent-shaped linear basis functions that have a value of 1 at the corresponding node and zero on all other nodes. Two base functions that share an element have a basis function overlap.

can be used to fit more complex functions and is widely applied in fields such as engineering and mathematical modeling. Because FEM's local basis functions can easily handle multi-scale problems, allowing it to represent features at varying scales, from coarse to fine resolution (see Fig. 1, left), making it a powerful tool for extracting multi-scale features. In this work, we employed the Lagrange basis function from the FEM as a feature extractor, which we call LagEncoder.

Compared to other encoders, LagEncoder offers a universal architecture applicable across various recognition domains, including regression, image and text classification, and image super-resolution tasks, all without the need for training or fine-tuning. Additionally, LagEncoder serves as an efficient parameter-efficient fine-tuning (PEFT) approach (see Fig. 3 (c)). Our experiments on the ImageNet dataset demonstrate that pre-trained models using LagEncoder achieve performance improvements within just one training epoch. Furthermore, it does not require adjustments to the original training recipe, and the model's total parameters remain nearly unchanged. Our evaluation of the scaling law for model performance shows that using LagEncoder is more cost-effective than simply scaling the model.

2 METHOD

In general, neural networks provide a mapping from input to predicted output, and the training process involves minimizing empirical risk to determine the parameters of an empirical model that fits the data:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \tag{1}$$

where L is the cost function, $f(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$ is the predicted output given input $\boldsymbol{x}$, $\boldsymbol{y}$ is the target output, $\boldsymbol{\theta}$ represents the model parameters, and m is the number of training examples. Specifically, in most neural networks, the projection head is realized as a linear layer. The predicted output can be expressed as a linear combination of the features and the weights:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_i \boldsymbol{\theta}_i^{(\text{head})} \cdot \psi_i(\boldsymbol{x};\boldsymbol{\theta}^{(\text{encoder})}) \tag{2}$$

where $\psi_i(\boldsymbol{x};\boldsymbol{\theta}^{(\text{encoder})})$ is a feature and $\boldsymbol{\theta}^{(\text{head})}$ are the parameters of the model head. The function approximation formula in FEM has a similar format:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_i \boldsymbol{\theta}_i^{(\text{linear})} \cdot \psi_i(\boldsymbol{x}) \tag{3}$$

where $\psi_i(\boldsymbol{x})$ is a Lagrange basis function. Fig. 1 (left) shows 10 basis functions for a one-dimensional input space, while the right side presents two basis functions for a two-dimensional input space. These tent-shaped linear basis functions have a value of 1 at their corresponding node and 0 at all other nodes. The shape of the basis functions depends on the structure of the mesh. For

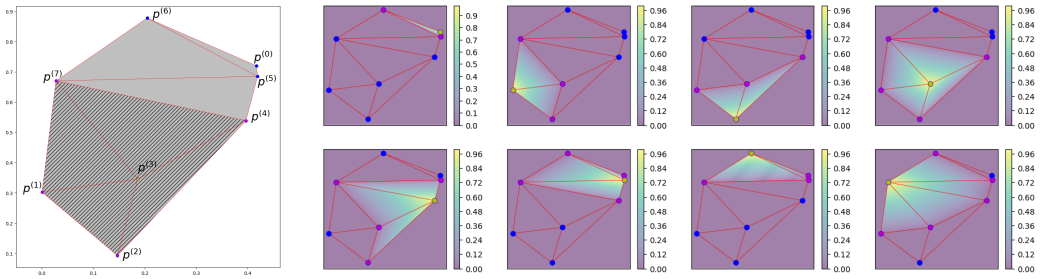


Figure 2: Left - Mesh with eight nodes and seven triangles. Right - Contours of eight Lagrange basis functions, linear variation of ψ_i associated with node $\boldsymbol{p}^{(i)}$ across all triangles.

example, in Fig. 1 (left), the mesh consists of nine intervals of varying lengths along the x-axis, whereas in Fig. 1 (right), the mesh is composed of 27 triangles in the plane. In Section 2.2, we will provide a detailed definition of meshes in arbitrary dimensions and the specific form of the basis functions.

As we mentioned earlier, these basis functions are highly effective at extracting multi-scale features, and 1 (left) illustrates this principle. In regions with multiple small peaks, more nodes are allocated to capture sharp transitions or localized features, while in smoother regions, fewer nodes are used to reduce the overall model size. Formally, these basis functions are *Lipschitz continuous* and piecewise linear, making the linear combination Eqn.(3) capable of approximating any continuous function with arbitrary precision. Given a function $F(\boldsymbol{x})$, we can bound the approximation error as follows:

$$|f(\boldsymbol{x};\boldsymbol{\theta}) - F(\boldsymbol{x})| \leq \max_{\boldsymbol{\xi}\in\Omega} \|\nabla f(\boldsymbol{\xi})\| \cdot h. \tag{4}$$

Here, $h = \max_i \max_j \mathbf{1}_{\text{dist}(\boldsymbol{p}^{(i)},\boldsymbol{p}^{(j)})=1} \cdot \|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}\|$ represents the maximum length of mesh edges (see Section 2.1) and Ω is the domain over which the approximation occurs. Comparing Eqn.(2) and Eqn.(3), we can see that the LagEncoder is parameter-free.

2.1 MESH

In the context of FEM, elements often serve as the fundamental building blocks of the triangulation *mesh*, taking the form of *simplices* created by connecting *nodes*. For instance, in 1D FEM, simplices are intervals (see Fig. 1, left); in 2D FEM, triangles with three nodes are commonly used (see Fig. 1, right), while 3D FEM often employs tetrahedra with four nodes. This concept is visually depicted in Fig. 2 (left), where the mesh consists of eight nodes and seven triangles. This type of mesh is established by specifying the coordinates of discrete nodes and the vertex indices of simplices. Let d represent the dimensions, $\{\boldsymbol{p}^{(i)}\}_{i=0}^{n-1}$ denote the grid nodes, and introduce a matrix $\boldsymbol{P}$ to store the node coordinates:

$$\boldsymbol{P}_{i,j} = \boldsymbol{p}_j^{(i)}.$$

Additionally, utilize a matrix $\boldsymbol{T}$ to store the indices of nodes constituting the simplices within the triangulation. Specifically, access the j-th sorted vertex of the i-th simplex in this mesh as $\boldsymbol{P}_{\boldsymbol{T}_{i,j},:}$. Fig. 2 (left) illustrate a matrix $\boldsymbol{T}$ takes the following form:

$$\boldsymbol{T} = \begin{bmatrix} 6 & 0 & 3 & 2 & 7 & 3 & 4 \\ 7 & 6 & 7 & 3 & 4 & 4 & 3 \\ 5 & 5 & 1 & 1 & 5 & 7 & 2 \end{bmatrix}^T.$$

This matrix serves to describe all seven simplices within the mesh, such as the first simplex $\triangle \boldsymbol{p}^{(6)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(5)}$ and the last simplex $\triangle \boldsymbol{p}^{(4)}\boldsymbol{p}^{(3)}\boldsymbol{p}^{(2)}$.

The first-order Lagrange basis studied in this article, denoted as $\{\psi_0(\boldsymbol{x}), \cdots, \psi_{n-1}(\boldsymbol{x})\} \subset P_1(\mathbb{R}^d)$, are piecewise linear polynomials associated with nodes $\{\boldsymbol{p}^{(0)}, \cdots, \boldsymbol{p}^{(n-1)}\}$. These functions are defined such that $\psi_i(\boldsymbol{p}^{(j)}) = \mathbf{1}_{\text{i=j}}$. Fig. 2 (right) illustrates this: $\psi_i(\boldsymbol{x})$ corresponds to node $\boldsymbol{p}^{(i)}$, exhibiting linear variation across all elements. Its support encompasses the union of all neighboring elements of node $\boldsymbol{p}^{(i)}$ (refer to Appendix B for a 3-dimensional visualization). For example, $\text{supp}(\psi_3) = \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(4)}\boldsymbol{p}^{(7)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(1)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(1)}\boldsymbol{p}^{(2)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(2)}\boldsymbol{p}^{(4)}$.

2.2 LAGENCODER

Now, we formulate the Lagrange basis from its original definition to establish the foundational architecture of LagEncoder. It is important to highlight that the traditional Lagrange basis involves unbalanced computing of barycentric coordinates, which may not be well-suited for parallel deep learning platforms (see Appendix C for details on the traditional definition of the Lagrange basis). Consequently, in this subsection, we re-derive the Lagrange basis to enhance parallel computing.

Let n_t represent the number of simplices in the multiscale mesh. We introduce the Parameters Tensor $\mathbf{S}$ defined as:

$$\mathbf{S}_{j,:,:} = \begin{bmatrix} \boldsymbol{p}_0^{(\boldsymbol{T}_{j,0})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,0})} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d-1})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d-1})} & 1 \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d})} & 1 \end{bmatrix}^{-1}, \quad j = 0, \cdots, n_t - 1.$$

Additionally, we introduce the Node Membership tensor $\mathbf{M}$ defined as:

$$\mathbf{M}_{i,j,k} = \begin{cases} 1, & \text{if the } i\text{-th node matches the } k\text{-th vertex of the } j\text{-th simplex,} \\ 0, & \text{other cases.} \end{cases}$$

By defining:

$$\boldsymbol{U}_{j,k}(\boldsymbol{x}) = \sum_{\tau=0}^{d-1} \mathbf{S}_{j,\tau,k} \cdot \boldsymbol{x}_\tau + \mathbf{S}_{j,d,k}, \quad j = 0, \cdots, n_t - 1,\ k = 0, \cdots, d.$$

We will demonstrate in Appendix A that the following function qualifies the definition of Lagrange basis:

$$\psi_i(\boldsymbol{x}) = \frac{\sum_{j=0}^{n_t-1} \sum_{k=0}^{d} \mathbf{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0} \cdot \mathbf{M}_{i,j,k} \cdot \boldsymbol{U}_{j,k}(\boldsymbol{x})}{\max(\sum_{j=0}^{n_t-1} \sum_{k=0}^{d} \mathbf{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0} \cdot \mathbf{M}_{i,j,k}, 1)}, \quad i = 0, \cdots, n-1. \tag{5}$$

So far, we have successfully constructed :

$$\begin{aligned} \text{LagEncoder} :& \mathbb{R}^d \to [0,1]^n, \\ & \boldsymbol{x} \mapsto (\psi_0(\boldsymbol{x}), \cdots, \psi_{n-1}(\boldsymbol{x})). \end{aligned}$$

The above basis is in the format of $P_1(\mathbb{R}^d)$, which is very useful for low-dimensional regression tasks (see Section 3.1). Furthermore, if Eqn.(3) is a decoupled system, we can decompose the input space into a direct sum of d one-dimensional subspaces. In this case, Eqn.(5) simplifies to the Lagrange basis $P_1(\mathbb{R}^1)$:

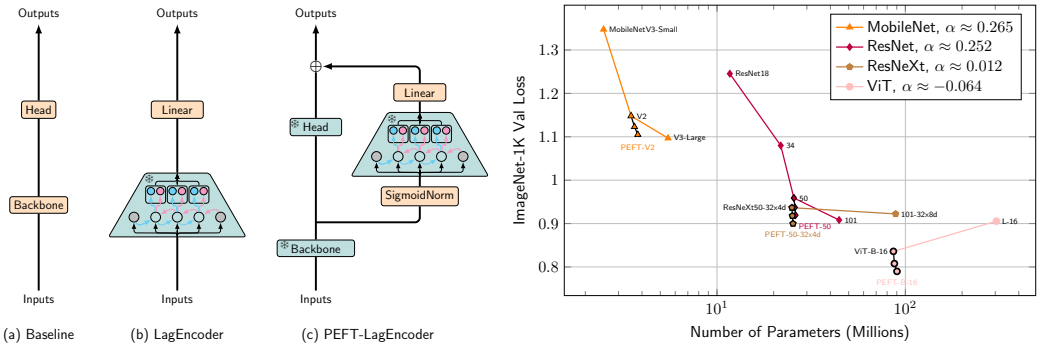
$$\psi_i(\boldsymbol{x}) = \min\left(\frac{\text{ReLU}(\boldsymbol{x} - p_{i-1})}{p_i - p_{i-1}}, \frac{\text{ReLU}(p_{i+1} - \boldsymbol{x})}{p_{i+1} - p_i} \right), \quad i = 0, 1, \cdots, n-1. \tag{6}$$

The $P_1(\mathbb{R}^1)$ basis has exceptionally low computational complexity, making it well-suited for recognition tasks on large-scale datasets. In the next section, we will introduce the specific approach in detail. We use the $P_1(\mathbb{R}^1)$ basis to implement an adaptive method for learning a decoupled residual system. This method can enhance the performance of pre-trained models on large datasets.

2.3 PEFT-LAGENCODER

LagEncoder demonstrates strong interpretability, and our experiments show that it performs exceptionally well in cases of low-dimensional input data (see Section 3.1). However, as the output dimension of LagEncoder increases with the data dimension d, the output grows factorially according to Eqn.(5), leading to an output size of $\mathcal{O}(d!)$. Consequently, when handling high-dimensional data (such as large images), the computational cost becomes extremely high. This high computational demand makes applying LagEncoder in resource-constrained environments particularly challenging.

To address this problem, we can apply the LagEncoder as a module to PEFT methods. Now, suppose there exists a perfect model $F(\boldsymbol{x})$ with 100% test accuracy and a given pre-trained model $f(\boldsymbol{x}; \boldsymbol{\theta})$.



(a) Baseline (b) LagEncoder (c) PEFT-LagEncoder

Figure 3: **Left:** (a) shows a plain network example; (b) illustrates a LagEncoder followed by a trainable linear layer; (c) demonstrates an application of LagEncoder to the PEFT method, where the pre-trained model is frozen, and the SigmoidNorm layer, used for data normalization, can be either trainable or frozen. The linear layer connected to LagEncoder is trainable. **Right:** The colored solid lines represent the pre-trained models with different scales, and the black solid lines represent our adaptive method. Our adaptation method significantly outperforms model scaling, enabling additional performance gains with minimal changes to model size.

If the pre-trained model has high test accuracy, then the residual $F(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\theta})$ must be very flat. For different inputs $\boldsymbol{x}$, this residual is generally close to zero. This implies that the support set of this residual is sparsely distributed in the input space, making the dimensionality reduction process on the pre-trained model's features likely reversible. This enables us to use the LagEncoder.

In Fig. 3 (c), the SigmoidNorm layer reduces the dimension and normalizes the output representation vectors $\boldsymbol{u}$ of the pre-trained backbone:

$$\boldsymbol{v} \leftarrow \text{Sigmoid}(\text{PCA}(\boldsymbol{u}))$$

where $\boldsymbol{v}$ represents the reduced-dimensionality feature. Here, we can generate the PCA model using standard unsupervised methods or adjust the weight matrix in the PCA through training. Next, we use the $P_1(\mathbb{R}^1)$ LagEncoder to compute the residual and combine it with the pre-trained model to estimate the prediction:

$$F(x) \approx f(\boldsymbol{x};\boldsymbol{\theta}) + \sum_i \boldsymbol{\theta}_i^{(\text{linear})} \cdot \psi_i(\boldsymbol{v}). \tag{7}$$

Since the support set of the residual $F(x) - f(\boldsymbol{x};\boldsymbol{\theta})$ is sparsely distributed, the added branch converges fast during the training phase. Our experiments show that our adaptation method hardly scales the model size and can improve the model performance within one epoch of training (see Table 1). This approach enables us to effectively fine-tune the model and save computing resources.

2.4 Multiscale Domain Decomposition Method

In our earlier discussion, we introduced the interpolation Eqn.(3) and its associated error-bound formula Eqn.(4). However, this tool is not appropriate for machine learning modeling, since we face a crucial challenge: the "given function $F(\boldsymbol{x})$ to be fitted" represents a ground truth that remains unknown. Instead, in the scenario of machine learning, a typical dataset provides us with a collection of input-target pairs. For any given simplex, select a subset $\{(x^{(k_0)}, y^{(k_0)}), \cdots, (x^{(k_{m'-1})}, y^{(k_{m'-1})})\}$ from the training set $\{(x^{(0)}, y^{(0)}), \cdots, (x^{(m-1)}, y^{(m-1)})\}$ where m' is the cardinality of subset, m is the cardinality of subset, $\{k_i\}_{i=0}^{m'-1} \subseteq \{i\}_{i=0}^{m-1}$, and all subset elements reside within the given simplex. Our goal now is to assess the error of $f(\boldsymbol{x};\boldsymbol{\theta})$ within this simplex.

Crucially, due to the linearity of basis functions $\{\psi_0(\boldsymbol{x}), \cdots, \psi_{n-1}(\boldsymbol{x})\}$ within each simplex, their linear combination also remains linear within these simplices. As a result, in the given simplex, there exists a set of coefficients $\boldsymbol{\beta}$ that:

$$\sum_k \boldsymbol{\theta}_k^{(\text{linear})} \cdot \psi_k(\boldsymbol{x}^{(i)}) = \boldsymbol{\beta}_0 \boldsymbol{x}_0^{(i)} + \cdots + \boldsymbol{\beta}_{d-1} \boldsymbol{x}_{d-1}^{(i)} + \boldsymbol{\beta}_d.$$

Algorithm 1 Domain decomposition method of generating multiscale mesh.

Input: Maximum degrees of freedom N to perform. Depending on the size of the training set.
Input: Initial Simplex Indices Matrix $\boldsymbol{T}$ of shape $(1, d+1)$ with $\boldsymbol{T}_{0,i} = i$.
Input: Initial Node Matrix $\boldsymbol{P}$ containing coordinates of $d+1$ points forming a simplex covering all training raw data.
Output: The updated Node Matrix $\boldsymbol{P}$ and Simplex Indices Matrix $\boldsymbol{T}$ of the refined mesh.

- $n \leftarrow d+1$

while $n < N$ **do**

- Create the binary Longest Edge Matrix $\boldsymbol{M}$ where $\boldsymbol{M}_{i,j} = 1$ indicates that the i-th edge is the longest side of the j-th simplex.
- Formulate the binary Edge Membership Matrix $\boldsymbol{E}$ where $\boldsymbol{E}_{i,j} = 1$ indicates that the i-th edge is a side of the j-th simplex.
- Establish the binary Data-Simplex Membership Matrix $\boldsymbol{B}$ where $\boldsymbol{B}_{i,j} = 1$ signifies that the i-th raw data falls within the j-th simplex.
- Compute the index of the priority edge:

$$\arg\min_i \frac{\sum_j \sum_k \boldsymbol{M}_{i,j}\boldsymbol{B}_{k,j}}{\max(\sum_j \sum_k \boldsymbol{E}_{i,j}\boldsymbol{B}_{k,j}, 1)}.$$

The priority edge is the longest side among many simplices, and these relevant simplices cover a substantial portion of the raw data.

- Insert a new node at the midpoint of the priority edge and update the Node Matrix $\boldsymbol{P}$.
- Update the Simplex Indices Matrix $\boldsymbol{T}$ and utilize it to update mesh edges.
- $n \leftarrow n+1$

end while

Therefore we can obtain the following error bound by solving a *Ordinary Least Squares* problem

$$\sum_{i=0}^{m'-1} \left| y^{(i)} - \sum_k \boldsymbol{\theta}_k^{(\text{linear})} \cdot \psi_k(\boldsymbol{x}^{(i)}) \right|^2 \leq \sum_{i=0}^{m'-1} |y^{(i)} - (\hat{\boldsymbol{\beta}}_0 \boldsymbol{x}_0^{(i)} + \cdots + \hat{\boldsymbol{\beta}}_{d-1}\boldsymbol{x}_{d-1}^{(i)} + \hat{\boldsymbol{\beta}}_d)|^2, \quad (8)$$

where $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$, $\boldsymbol{X}_{i,:d} = \boldsymbol{x}^{(i)}$, $\boldsymbol{X}_{i,d} = 1$, and $\boldsymbol{y}_i = y^{(i)}$. This formula shows the error bound reduced to 0 when $m' \leq d+1$. By combining this conclusion with the global error-bound formula Eqn.(4), we can summarize two critical goals for mesh generation in modeling:

1. Each simplex in the mesh should ideally contain as few original data points from the training set as possible. When each simplex covers no more than $d+1$ raw data examples, the model perfectly fits the training set.
2. Decreasing the bound of mesh edge lengths results in a reduced bound of error.

Algorithm 1 is designed to achieve these two goals. It describes the process of generating a multiscale mesh and serves as the initial step in constructing the LagEncoder architecture. To enhance readability, this brief pseudocode traverses all training raw data, simplices, and edges in each iteration. For acceleration, we use divide-and-conquer techniques in the program. In each iteration of Algorithm 1, a new fine node is added to the grid, and several coarse simplices are subdivided into finer simplices. In Appendix D illustrates the process of refining a mesh (see Fig. 7, left). As the number of nodes (degrees of freedom) increases, each simplex (triangle) covers a reduced amount of raw data. Additionally, the secondary objective is to split the longest side of each simplex, progressively generating more acute triangles to minimize the value of h in Eqn.(4).

When applying LagEncoder to the PEFT method, Algorithm 1 can be replaced by the simpler V-cycle algorithm from FEM. The V-cycle updates the mesh by estimating the empirical distribution of examples in the training set. It allocates more nodes in high-frequency (dense) regions and fewer nodes in low-frequency (sparse) regions. Notably, LagEncoder is plug-and-play, it automatically updates the mesh during inference. We have integrated the V-Cycle into LagEncoder's forward method, so users do not need to manually update the mesh, just as they do not need to manually update the mean and variance in the Batch Normalization (Ioffe, 2015). The underlying mathematical principles of this algorithm are relatively complex, and a visualized workflow is provided in Appendix D for better understanding (see Fig. 7, right).

Algorithm 2 V-Cycle.

Require: The pre-trained model $f(\boldsymbol{x};\boldsymbol{\theta})$ with frozen parameter $\boldsymbol{\theta}$.
Require: Initial parameter $\boldsymbol{\theta}^{(\text{linear})}$ of the linear head.
Require: Top-K threshold, K_1 and K_2 (Suggested defaults: $0.1 \times \text{batch}_{\text{size}}$ and 1 respectively).
Require: Mesh updating frequency N.

- $k \leftarrow 1$
- **Generate PDF**: Construct a histogram on the interval $[-1, 1]$ to represent the Probability Density Function (PDF) of the empirical data distribution, initially using equal-width binning. The grid $\boldsymbol{P}$ contains $n-1$ bins, with n coarse nodes.

while stopping criterion not met **do**

- Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
- Compute corresponding losses: $l_i \leftarrow L(f(\boldsymbol{x};\boldsymbol{\theta}) + \sum_i \boldsymbol{\theta}_i^{(\text{linear})} \cdot \psi_i(\boldsymbol{v}^{(i)}), \boldsymbol{y}^{(i)})$.
- **Generate PDF**: Estimate a temporary PDF using the current m examples: specifically, select the top K_1 examples with the highest loss values and calculate the proportion of these examples in each bin.
- **Update PDF**: Update the histogram using this temporary PDF and Exponential Moving Average (EMA).

if $k \bmod N \equiv 0$ **then**

- **Mesh Refinement**: Mark the top K_2 bins with the highest values in the histogram as coarse elements, and add their midpoints as fine nodes. The degrees of freedom increase to $n + K_2$.
- **Interpolation**: Use linear interpolation to compute the values of the linear head parameters at each fine node.
- **Mesh Coarsening**: Solve a system of equations to transform the current histogram back into equal-width binning, reducing the grid's degrees of freedom back to n.
- **Interpolation**: Use linear interpolation to compute the values of the linear head parameters at each new node.

end if

$k \leftarrow k + 1$

end while

3 EXPERIMENTS

In the upcoming sections, we present a comprehensive series of experiments to showcase the effectiveness and universality of LagEncoder across various tasks. Our exploration begins with an analysis of its performance in regression tasks, followed by examinations in image and text recognition. Finally, we apply it to the PEFT method and assess its performance on large datasets.

3.1 REGRESSION TASKS

3.1.1 THE LIMITATIONS OF TRADITIONAL REGRESSORS AND NEURAL NETWORKS

Traditional regressors often struggle with automatically overcoming overfitting, while neural networks address this issue. However, neural networks tend to perform poorly when fitting multi-frequency functions, a phenomenon referred to as the *frequency principle* by Xu et al. (2019). Fig. 4 clearly illustrates this conflict: on the left, Support Vector Regression (Platt et al., 1999) successfully fits dataset $\mathbb{B}$ but overfits on dataset $\mathbb{A}$, whereas the neural network performs well on $\mathbb{A}$ but underfits on $\mathbb{B}$. We found that neither MLPs, CNNs, nor Transformers could effectively fit regions with sharp transitions in dataset $\mathbb{B}$. We hypothesize that when input data moves within a localized area while the target changes drastically, neural networks may treat this region as noise to avoid overfitting.

Our LagEncoder combines the strengths of traditional regressors and neural networks. As discussed in Section 2, it leverages a multiscale mesh, enabling it to adaptively fit data in high-frequency regions by using fine simplices. Fig. 4 (right) showcases the ability of the LagEncoder-based model to handle the challenging dataset $\mathbb{C}$. Additionally, Fig. 9 in Appendix E.3 further illustrates the gradual fitting process of the LagEncoder-based model during training.

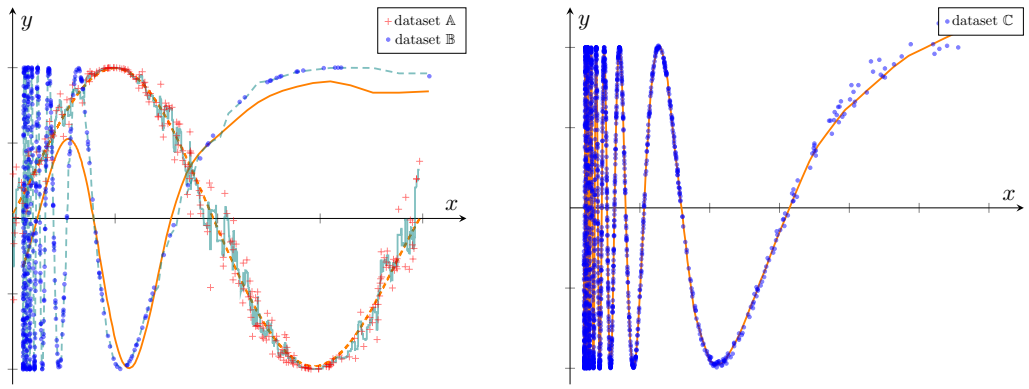


Figure 4: Left - illustrates the performance of traditional regressors in fitting the high-noise dataset $\mathbb{A} = \{(x, y)|X \sim u(0, \frac{\pi}{4}), Y \sim N(\sin 8x, |\cos 8x|)\}$ and the multi-frequency dataset $\mathbb{B} = \{(x, y)|\frac{1}{X} \sim u(0.02, 0.5), y = \sin \frac{1}{x}\}$. The dashed teal curve shows that the traditional regressor (*e.g.*, Support Vector Regression) succeeds in fitting dataset $\mathbb{B}$. However, when applied to the high-noise dataset $\mathbb{A}$, shown by the solid teal curve, the traditional regressor exhibits overfitting, struggling to generalize automatically. Conversely, we also showcase the performance of a neural network (*e.g.*, Multi-Layer Perceptron) in fitting dataset $\mathbb{A}$, exemplified by the dashed orange curve. However, as depicted by the solid orange curve, this neural network faces challenges in fitting multi-frequency dataset $\mathbb{B}$, indicative of underfitting. Right - We present the exceptional adaptability of a LagEncoder-based model in fitting the dataset $\mathbb{C} = \{(x, y)|\frac{1}{X} \sim u(0.02, 0.5), Y \sim N(\sin \frac{1}{x}, 0.5x^2)\}$, which is both high-noise and multi-frequency.

Some comparison experiments between the LagEncoder-based model and traditional regressors (Thiel, 1950; Cantzler, 1981; Zhang, 2004; Hilt & Seegrist, 1977; Stone, 1974; Jain et al., 2018; Murphy, 2012; Platt et al., 1999; Friedman, 2001; Breiman, 2001) can be found in Appendix E.1. The LagEncoder-based model consistently achieves high R^2 scores across all test sets, demonstrating its robustness and effectiveness, even on datasets with high noise levels and multiple frequency components.

3.1.2 Scaling Law and Error-Bound Formula

Our LagEncoder exhibits strong interpretability, and this is quantitatively supported by experimental results. The experiment shown in Fig. 5 demonstrates that the LagEncoder-based model adheres to the empirical scaling law. Additionally, since the number of parameters in the model's linear head is proportional to the number of simplices n_t in the mesh, and $(n_t/d!)^{1/d} = O(h^{-1})$, this experiment further verifies that our model satisfies the error bound formula described by Eqn.(4).

3.2 Natural Language Processing

In this section, we delve into the practical application of LagEncoder for text feature extraction using the AG News dataset (Zhang et al., 2015) for classification tasks. Our experimental findings reveal that LagEncoder can directly extract features from raw text. However, recognizing that tokens are unordered categorical variables, we add a non-parametric preprocessing layer to enhance performance, converting each token into a four-dimensional vector representing its proportion across four categories.

Throughout our experiment, we set the degree of freedom n of LagEncoder to 64. We employed the SGD optimizer to minimize the cross-entropy loss, starting with a learning rate of 5.0 and reducing it by a factor of 0.1 every two epochs. The batch size was set to 32. This experiment demonstrates the neural network achieving 90.01% test accuracy after the first epoch, with 90.4% test accuracy reached within just five epochs.

The LagEncoder-based model possesses a unique characteristic in text classification tasks: the number of its parameters remains independent of the token count. Compared to word2vec-based net-

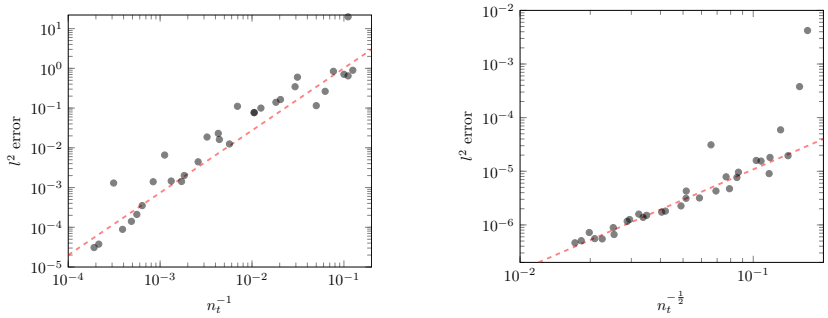


Figure 5: Left - the results of 32 experiments fitting a 1-dimensional function $y = \sin \frac{1}{x}$, with each gray point representing an experiment and showing the relationship between n_t^{-1} and Mean Square Error (MSE). Right - 32 experiments fitting a 2-dimensional function $y = \sum_{i=1}^{2} \sin \frac{\boldsymbol{x}_i}{2\pi}$, mirroring the left side with gray points signifying individual experiments and demonstrating the relationship between $n_t^{-1/2}$ and MSE.

works, our model performs equally well in classification but boasts only 256 model parameters, a significant reduction from the word2vec-based network with more than 6.13 million parameters.

3.3 Computer Vision

As described in Section 2.3, LagEncoder can serve as a non-parametric module for the PEFT method. The new model includes a residual branch where the SigmoidNorm layer and linear head contain trainable parameters (see Fig. 3 (c)). In this section, we illustrate the effectiveness of PEFT-LagEncoder through ablation studies. To ensure a fair comparison with baseline models, we adopt a stricter experimental setup: *the original training recipe of the pre-trained model cannot be modified when training the residual branch*.

This restriction is crucial to ablation studies because the training recipes for pre-trained models often have room for optimization (Wightman et al., 2021). Changes to batch size, optimizer, weight decay rate, or the order of applying data augmentations could potentially improve the performance of the pre-trained model (Touvron et al., 2019; 2021). To avoid such effects, we selected pre-trained models from TorchVision with publicly available training recipes as our baselines. The weights of these pre-trained models closely reproduce the results from the original papers on the ImageNet-1K dataset (Russakovsky et al., 2015), with recipes available at (TorchVision Contributors, 2024). Table 1 presents the fine-tuning results of our adaptation method.

Model	d	n	# Params	Acc@1 (%)	Acc@5 (%)
MobileNet-V2	32	8	0.298 M (+8.51%)	$71.920_{\pm.016}(+.045)$	$90.302_{\pm.021}(+.016)$
ResNet50	32	8	0.324 M (+1.27%)	$76.334_{\pm.054}(+.204)$	$92.976_{\pm.024}(+.114)$
ResNeXt50	32	8	0.324 M (+1.30%)	$77.796_{\pm.063}(+.178)$	$93.653_{\pm.038}(+.178)$
ViT-B-16	8	32	1.032 M (+1.19%)	$81.092_{\pm.012}(+.020)$	$95.316_{\pm.004}(-.002)$

Table 1: Classification accuracy on the ImageNet validation set, with all methods adhering to the raw training recipes. We report the changes in the number of model parameters and their ratio of changes and total. For each model, we conducted multiple experiments and show the accuracy as "mean$_{\pm\text{std}}$(mean $-$ baseline)".

The trainable parameters in our adaptation method come from the SigmoidNorm and linear layer (see Fig. 3(c)), the number of parameters can be precisely calculated as $(C_{\text{in}} + 1) \times d + d \times n \times C$, where C_{in} is the input dimension to SigmoidNorm, d is the output dimension after PCA dimensionality reduction, n is the number of nodes in the mesh (degrees of freedom), and C is the total output dimension. Our experiments reveal that, generally, the larger the values of d and n for the ImageNet dataset, the better our adaptation method performs. Since there are diminishing returns,

we empirically limit d and n to 64. As shown in Table 1, our method is highly stable, with nearly no variation across multiple experiments. In our approach, the number of trainable parameters is minimal, and the model converges within just one epoch.

We also study empirical scaling laws for model performance on cross-entropy loss to demonstrate how our adaptation method fully exploits the potential of pre-trained models. (Kaplan et al., 2020) proposed an empirical formula where the loss scales as a power-law with model size:

$$L(N) = \left(\frac{c}{N}\right)^{\alpha}$$

where $L(N)$ is the cross-entropy loss on the validation dataset, N is the number of model parameters, and c and α are constants related to the model type. Fig. 3 (right) shows that our adaptation method is more effective than simply scaling the model size. Our method increases the α for MobileNet (Howard et al., 2017) from 0.265 to 0.473, for ResNet He et al. (2016) from 0.252 to 2.05, for ResNeXt Xie et al. (2017) from 0.012 to 3.06, and for ViT (Dosovitskiy et al., 2020) from -0.064 to 1.20.

Compared to other model scaling and adaptation methods, our approach requires no adjustments to the training recipe, has low computational demands (with very few trainable parameters), and achieves convergence within a single epoch. On 4x A6000 GPUs, it takes about 20 minutes to enhance the performance of a ViT-B-16 model, offering significant practical value.

4 Future Directions

It is important to recognize that LagEncoder has certain limitations. As described in Section 2.3, the high computational cost of extracting features from high-dimensional data restricts its direct application to large-scale datasets. Currently, we address this by incorporating LagEncoder as part of an adaptation method, but we aim to improve the encoder in the future to be able to extract features directly from large images. Additionally, the experiments in Section 3.3 show that while our adaptation method significantly reduces cross-entropy on the ImageNet-1K validation set and consistently improves validation accuracy, the magnitude of improvement of accuracy is relatively modest, which we intend to explore further. We did not compare LagEncoder with models like SVM or KAN, which offer strong interpretability, because their underlying principles are entirely different. Moreover, we prioritize performance on challenging tasks where most interpretable models often lack relevant experimental results. We also did not conduct comparative studies with other PEFT methods, as model scaling has proven more effective than PEFT methods for image recognition tasks. However, we will continue to monitor developments in the PEFT field and plan to include comparative experiments if relevant research emerges.

5 Conclusion

We discussed domain adaptation in transfer learning and hypothesized that if an encoder possesses full generalization ability and is not dependent on a specific dataset. We further hypothesized that it is possible to develop a non-parametric encoder that requires no training. In response, we proposed a non-parametric encoder with a universal architecture capable of accommodating diverse types of raw data and recognition tasks. It extracts features by depicting the distribution of raw data, eliminating the need to consider the underlying meaning of the data. This inherent characteristic enables its generalization.

Our experiments demonstrated several key advantages of the LagEncoder-based model: 1) **Strong mathematical explainability**. Our results show that the performance of these models aligns perfectly with the theoretical error-bound formula and scaling law, providing a clear understanding of the representation learning process. 2) **Fast training**. These models contain only one linear layer for training, typically converging within just 1 to 2 epochs, which is particularly beneficial for learning large datasets.

In conclusion, our research introduces a novel encoder derived from established mathematical theory, which eliminates the need for extensive training, fine-tuning, and heavy computational resources. Our LagEncoder stands as an explainable and non-black-box encoder, holding significant value in representation learning research.

REFERENCES

Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

H Cantzler. Random sample consensus (ransac). Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh, 1981.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. Annals of statistics, pp. 1189–1232, 2001.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

Donald E Hilt and Donald W Seegrist. Ridge, a computer program for calculating ridge regression estimates, volume 236. Department of Agriculture, Forest Service, Northeastern Forest Experiment ..., 1977.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent for least squares regression. In Conference On Learning Theory, pp. 545–604. PMLR, 2018.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.

Kevin P Murphy. Machine learning: a probabilistic perspective. MIT press, 2012.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11):559–572, 1901.

John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers, 10(3):61–74, 1999.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3):211–252, 2015.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. Journal of documentation, 28(1):11–21, 1972.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. Journal of the royal statistical society: Series B (Methodological), 36(2):111–133, 1974.

H Thiel. A rank-invariant method of linear and polynomial regression analysis. i, ii, iii. Nederl. Akad. Wetensch., Proc. 53, pp. 386–392, 1950.

TorchVision Contributors. Torchvision models. `https://pytorch.org/vision/master/models.html`, 2024. Accessed: 2024-08-09.

Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. Advances in neural information processing systems, 32, 2019.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In International conference on machine learning, pp. 10347–10357. PMLR, 2021.

Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. arXiv preprint arXiv:2110.00476, 2021.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1492–1500, 2017.

Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. In International Conference on Neural Information Processing, pp. 264–274. Springer, 2019.

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning, pp. 116, 2004.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. Advances in neural information processing systems, 28, 2015.

A PROOF OF LAGRANGE BASIS EXPRESSION

We will now demonstrate, in three concise steps, that Eqn.(5) qualifies as a Lagrange basis function.
Piecewise Linear: Since $\{\boldsymbol{U}_{j,0}, \cdots, \boldsymbol{U}_{j,d}\}$ are piecewise linear functions, their linear combination L_i is also piecewise linear.
Kronecker Delta: From the definition of $\boldsymbol{U}$ and $\mathbf{S}$, we have the following equation:

$$[\boldsymbol{x}_0 \quad \cdots \quad \boldsymbol{x}_{d-1} \quad 1] = [\boldsymbol{U}_{j,0}(\boldsymbol{x}) \quad \cdots \quad \boldsymbol{U}_{j,d}(\boldsymbol{x})] \begin{bmatrix} \boldsymbol{p}_0^{(\boldsymbol{T}_{j,0})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,0})} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d-1})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d-1})} & 1 \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d})} & 1 \end{bmatrix}.$$

Decomposing this equation, we obtain $\boldsymbol{x} = \sum_{k=0}^{d} \boldsymbol{U}_{j,k}(\boldsymbol{x}) \boldsymbol{p}^{(\boldsymbol{T}_{j,k})}$ and $\sum_{k=0}^{d} \boldsymbol{U}_{j,k}(\boldsymbol{x}) = 1$. This implies two important conclusions: $\boldsymbol{U}_{j,k'}(\boldsymbol{p}^{(\boldsymbol{T}_{j,k''})}) = \mathbf{1}_{k'=k''}$ and $\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0$ is true if and only if $\boldsymbol{x}$ belongs to the j-th simplex. Therefore, we have

$$\begin{cases} \mathbf{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{p}^{(i)}) \geq 0} = \mathbf{M}_{i,j,k} = \boldsymbol{U}_{j,k}(\boldsymbol{p}^{(i)}) = 1, & \text{if } i = \boldsymbol{T}_{j,k} \\ \mathbf{M}_{i,j,k} = \boldsymbol{U}_{j,k}(\boldsymbol{p}^{(i)}) = 0, & \text{if } i \neq \boldsymbol{T}_{j,k} \end{cases}$$

This proves that $\psi_i(\boldsymbol{p}^{(j)}) = \mathbf{1}_{i=j}$.
Globally Continuity: Lastly, since ψ_i is inherently linear within all simplices and exhibits continuity across all grid nodes, we can conclude that ψ_i is globally continuous.

B VISUALIZATION OF LAGRANGIAN BASIS

In section 2, we introduced first-order Lagrange basis functions, a set of piecewise linear functions defined on a mesh. Each basis function corresponds to a node.

Consider the grid depicted in Fig. 6 (left). Taking the node $\boldsymbol{p}^{(20)}$ as an example, it has a total of four neighboring nodes: $\boldsymbol{p}^{(5)}$, $\boldsymbol{p}^{(0)}$, $\boldsymbol{p}^{(7)}$, and $\boldsymbol{p}^{(4)}$. By connecting these nodes, we can determine the support of the basis function ψ_{20}.

In Fig. 6 (middle), we present the function graphs of ψ_{20} and ψ_7. It can be observed that these functions exhibit linear variations on each mesh triangle. Taking ψ_{20} as an example, its function value at $\boldsymbol{p}_{20}$ is 1, and 0 at all other nodes. Similarly, ψ_7 has a function value of 1 at $\boldsymbol{p}_7$ and 0 at other nodes.

In Fig. 6 (right), the orange triangles represent the function graph of $f(\boldsymbol{x}; \boldsymbol{\theta})$ on the domain $\triangle \boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}$, where the function values of $f(\boldsymbol{x}; \boldsymbol{\theta})$ at the vertices $(\boldsymbol{p}^{(0)}, \boldsymbol{p}^{(7)}, \boldsymbol{p}^{(9)})$ are $(\boldsymbol{\theta}_0, \boldsymbol{\theta}_7, \boldsymbol{\theta}_9)$, respectively. The green dots represent a subset of training set, where the projections (raw data) fall on $\triangle \boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}$. As shown in equation (2), when the number of green points does not exceed three, there exists a solution of $(\boldsymbol{\theta}_0, \boldsymbol{\theta}_7, \boldsymbol{\theta}_9)$ such that all green points lie on the surface of $f(\boldsymbol{x}; \boldsymbol{\theta})$, such that MSE reach a minimum value of 0.

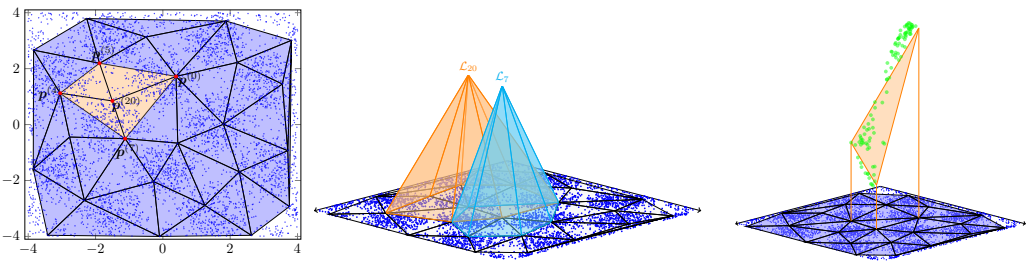


Figure 6: Data visualization. Left - an example of 2-dimensional mesh. Middle - the graphs of basis functions ψ_{20} and ψ_7. Right - the graphs of the function $f(\boldsymbol{x}; \boldsymbol{\theta})$ and a subset of training set.

C THE TRADITIONAL EXPRESSION OF LAGRANGE BASIS

Given the complexity of FEM as a numerical method, to enhance understanding, we start with a two-dimensional case and a triangulated mesh to illustrate the traditional expression of basis functions. Let triangle $\triangle$ be defined by nodes $\{\boldsymbol{p}^{(i)}, \boldsymbol{p}^{(j)}, \boldsymbol{p}^{(k)}\}$. The following barycentric coordinates $\{\lambda_{\triangle,i}, \lambda_{\triangle,j}, \lambda_{\triangle,k}\}$ are three first-degree polynomials of $\boldsymbol{x}$

$$\begin{bmatrix} \lambda_{\triangle,i}(\boldsymbol{x}) \\ \lambda_{\triangle,j}(\boldsymbol{x}) \\ \lambda_{\triangle,k}(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_0^{(i)} & \boldsymbol{p}_0^{(j)} & \boldsymbol{p}_0^{(k)} \\ \boldsymbol{p}_1^{(i)} & \boldsymbol{p}_1^{(j)} & \boldsymbol{p}_1^{(k)} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ 1 \end{bmatrix}.$$

Referring to the instance depicted in Fig. 2 (left), the mesh consists of eight nodes and seven triangles. Specifically, let $\{\triangle^{(j)} = \triangle \boldsymbol{p}^{(\boldsymbol{T}_{j,0})} \boldsymbol{p}^{(\boldsymbol{T}_{j,1})} \boldsymbol{p}^{(\boldsymbol{T}_{j,2})} | j = 0, \cdots, 6\}$. We will now verify that the following ψ_3 corresponds to the third basis function in this mesh

$$\psi_3(\boldsymbol{x}) = \frac{1}{\max(\sum_{i\in\{2,3,6,5\}} \mathbf{1}_{\boldsymbol{x}\in\triangle^{(i)}}, 1)} \sum_{i\in\{2,3,6,5\}} \mathbf{1}_{\boldsymbol{x}\in\triangle^{(i)}} \lambda_{\triangle^{(i)},3}(\boldsymbol{x}).$$

First, ψ_3 possesses values of *Kronecker Delta*:

$$\psi_3(\boldsymbol{x}) = \begin{cases} \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(0)}, \\ \frac{1}{\max(2,1)}(1\cdot 0 \quad +1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(1)}, \\ \frac{1}{\max(2,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+1\cdot 0 \quad +1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(2)}, \\ \frac{1}{\max(4,0)}(1\cdot 1 \quad +1\cdot 1 \quad +1\cdot 1 \quad +1\cdot 1) \quad =1, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(3)}, \\ \frac{1}{\max(2,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+1\cdot 0 \quad +1\cdot 0) \quad =0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(4)}, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(5)}, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(6)}, \\ \frac{1}{\max(2,1)}(1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+1\cdot 0) \quad =0, & \text{if } \boldsymbol{x}=\boldsymbol{p}^{(7)}, \end{cases}$$

Then, ψ_3 is a first-degree polynomial in every triangle and $\operatorname{supp}\psi_3 = \operatorname{inn} \cup_{i\in\{2,3,6,5\}} \triangle^{(i)}$:

$$\psi_3(\boldsymbol{x}) = \begin{cases} \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_0, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_1, \\ \frac{1}{\max(1,1)}(1\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(2)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_2, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(3)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_3, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_4, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(5)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_5, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(6)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\operatorname{inn}\boldsymbol{T}_6, \end{cases}$$

Finally, since ψ_3 is continuous in all nodes and first-degree in all triangles, it is globally continuous.

D VISUALIZATION OF MESH REFINEMENT

In this work, we utilize the triangle mesh for constructing $P_1(\mathbb{R}^d)$ Lagrange basis. As the mesh is refined, each simplex will contain the same number of points, so Algorithm 1 is an equal-frequency binning method in d-dimensional space.

(Fig. 7, left) depicts a mesh variety during the iterations of Algorithm 1. In the coarse mesh, simplices containing more points will be refined, and simplies containing fewer points will be retained. When we detect that the number of points (training input data) contained in all simplexes is not much different, the mesh will stop being refined.

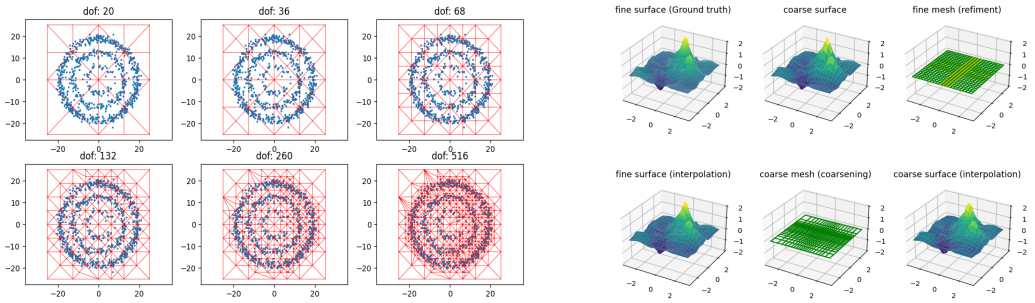


Figure 7: Left - Each triangle represents a 2-dimensional simplex, with increasing degrees of freedom indicating higher levels of refinement. Discrete points denote the raw data. Right - V-cycle transforms a uniform mesh into a multiscale mesh by inspecting residuals, redistributing nodes from flat regions to improve the mesh's representation capacity.

E Additional Experiments and Applications

E.1 Regression

As mentioned earlier, neural networks often struggle to fit multi-frequency datasets effectively. Therefore, our primary focus is comparing the LagEncoder-based model with traditional regressors. To evaluate the effectiveness and generalization of the LagEncoder-based model, we have devised four diverse datasets, each generated from distinct probability distributions:

1. $\mathbb{A}^1$: Generated from the distribution $\{(x, y)|X \sim U(-\pi, \pi), Y \sim \mathcal{N}(\sin x, \frac{1}{5}\cos^2 x)\}$, with 1000 training examples and 200 test examples. The LagEncoder-based model was trained with a learning rate of 0.1.
2. $\mathbb{B}^1$: Generated from the distribution $\{(x, y)|\frac{1}{X} \sim U(0.02, 1.0), Y \sim \mathcal{N}(\sin \frac{1}{x}, 0.01)\}$, with 1000 training examples and 200 test examples. We trained the LagEncoder-based model with a learning rate of 0.9.
3. $\mathbb{A}^2$: Generated from the distribution $\{(\boldsymbol{x}, y)|\boldsymbol{X}_i \sim U(-\pi, \pi), Y_i \sim \mathcal{N}(\sin \boldsymbol{x}_i, \frac{1}{10}\cos^2 \boldsymbol{x}_i)$ $, Y = \frac{1}{2}(Y_1 + Y_2)\}$, with 7,500 training examples and 1,500 test examples. The LagEncoder-based model was trained with a learning rate of 0.1.
4. $\mathbb{B}^2$: Generated from the distribution $\{(\boldsymbol{x}, y)|\frac{1}{\boldsymbol{X}_i} \sim U(0.05, 0.5), Y_i \sim \mathcal{N}(\sin \frac{1}{\boldsymbol{x}_i}, 0.01), Y = \frac{1}{2}(Y_1 + Y_2)\}$, with 50,000 training examples and 10,000 test examples. The training utilized a learning rate of 0.9.

Table 2 displays the coefficient of determination (R^2) scores for the LagEncoder-based model and traditional regressors (Thiel, 1950; Cantzler, 1981; Zhang, 2004; Hilt & Seegrist, 1977; Stone, 1974; Jain et al., 2018; Murphy, 2012; Platt et al., 1999; Friedman, 2001; Breiman, 2001) across fitting the four datasets. The LagEncoder-based model consistently achieves high R^2 scores across all test sets, demonstrating the effectiveness of the InterpolationNet on both high-noise and multi-frequency datasets. Furthermore, the minimal gap between training and test set evaluations underscores the robustness of the LagEncoder-based model, indicating its capability of generalization.

E.2 Fitting high-noise dataset

In this section, we conduct the LagEncoder-based model on fitting the dataset $\mathbb{A} = \{(x, y)|X \sim U(-4, 4), Y \sim \mathcal{N}(\sin x, 0.2\cos^2 x)\}$. A comprises 6000 examples, with 5000 for training and 1000 for testing. Fig. 8 shows the training progress.

METHOD	$\mathbb{A}^1$		$\mathbb{B}^1$		$\mathbb{A}^2$		$\mathbb{B}^2$	
OLS Linear	0.037	0.042	0.963	0.951	0.085	0.092	0.984	0.984
Theil-Sen	-44.7	-54.4	0.958	0.946	-0.41	-3.81	0.982	0.982
RANSAC	-1.21	-1.43	0.963	0.951	-27.0	-27.1	0.983	0.983
Huber	0.036	0.041	0.962	0.949	0.085	0.092	0.984	0.984
Ridge	0.031	0.038	0.963	0.951	0.055	0.061	0.984	0.984
RidgeCV	0.037	0.042	0.963	0.951	0.085	0.092	0.984	0.984
SGD	0.009	0.01	0.962	0.95	0.005	0.004	0.983	0.983
KRR	0.0036	0.04	0.97	0.962	0.056	0.051	0.993	0.992
SVR	0.11	0.101	0.97	0.962	0.29	0.308	0.992	0.992
Gradient Boosting	0.964	0.962	0.98	0.96	0.989	0.988	0.992	0.99
Random Forests	1.0	0.999	0.995	0.945	1.0	0.999	0.999	0.99
Voting	0.852	0.852	0.942	0.917	0.869	0.868	0.951	0.946
Net	1.0	1.0	0.971	0.963	0.999	0.999	0.992	0.992

Table 2: A comprehensive comparison between the LagEncoder-based model and traditional regressors. The left half of each paired column displays the training R^2 score, while the right half showcases the corresponding test R^2 score.

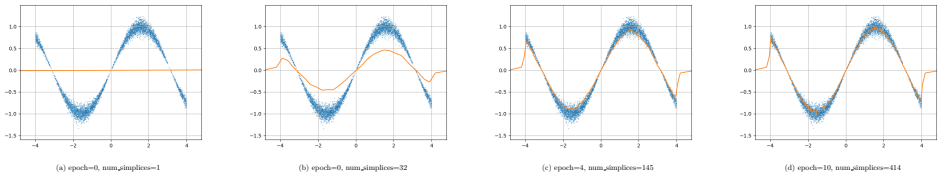


Figure 8: Blue dots represent the training set, while the orange curve represents the network.

E.3 FITTING MULTI-FREQUENCY DATASET

In this section, we conduct the LagEncoder-based model on fitting the dataset $\mathbb{A} = \{(x, y)|\frac{1}{x} \sim U(0.02, 0.5), y = \sin\frac{1}{x}\}$. A comprises 6000 examples, with 5000 for training and 1000 for testing. Fig. 9 illustrates the training progress. Remarkably, after just 4 epochs of training, the neural network outputs closely approximate the target values. By the 32nd epoch's conclusion, the neural network outputs and target values are nearly indistinguishable.

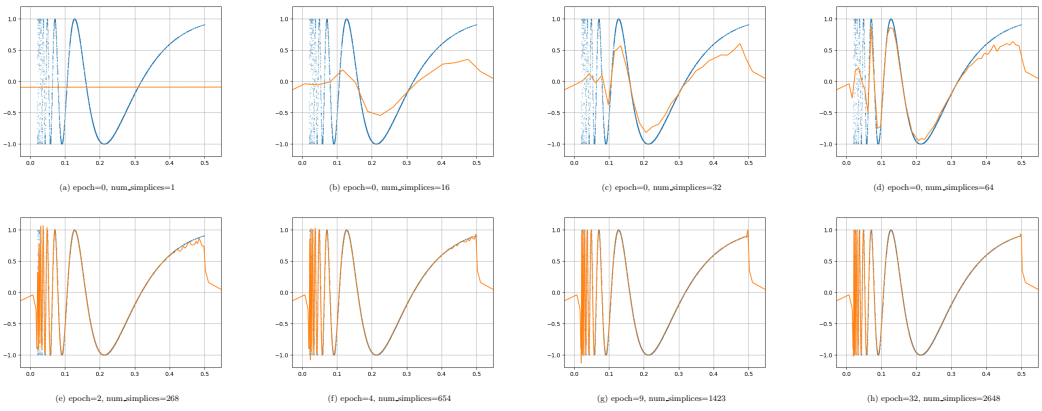


Figure 9: Blue dots represent the training set, while the orange curve represents the network.

E.4 Fit a Vector-Valued Function

In this instance, we utilize the LagEncoder-based model to fit spherical harmonics. Our dataset denoted as $\mathbb{A} = \{(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{x} = (\theta, \phi), \boldsymbol{y} = (\text{Real}(Y_4^2(\theta, \phi)), \text{Imag}(Y_4^2(\theta, \phi))), \Theta \sim U(0, 2\pi), \Phi \sim U(0, \pi)\}$, comprises 48,000 examples, with 40,000 allocated for training and an additional 8,000 for testing. Fig. 10 shows the training progress.

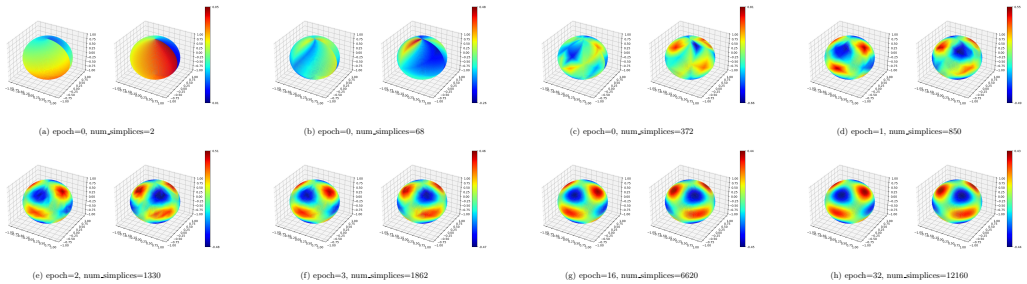


Figure 10: In each block, the left panel represents the real part of our model output, while the right panel represents the imaginary part of the model output.

E.5 Solve PDEs

In this section, we utilize the LagEncoder-based model to address the following partial differential equations (PDEs):

$$\begin{cases} \Delta u + (u - \beta)^2 = (\alpha \cos x \sin y - 1)^2 + 1, & (x, y) \in \Omega; \\ u = \beta, & (x, y) \in \partial\Omega. \end{cases}$$

Here, $\Omega = [0, 1] \times [0, 1]$. We construct a dataset that takes (α, β) as input data and assigns the corresponding numerical solution of the PDEs as the target output. This dataset comprises 12,000 examples, with α randomly selected from the distribution $U(-\pi/2, \pi/2)$ and β randomly chosen from the distribution $U(0, 2\pi)$. We then split the dataset into two parts: 10,000 for training and 2,000 for testing. Fig. 11 illustrates how well the network predicts the exact solution.

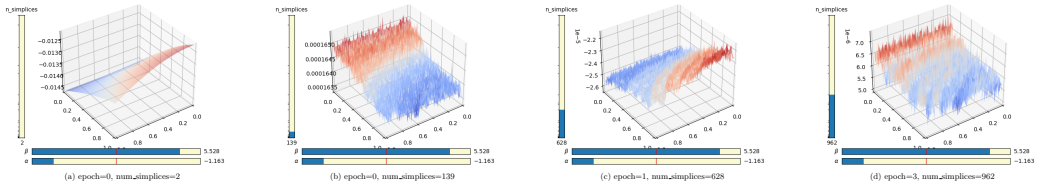


Figure 11: Residual - The gap between the exact solution and the model output.