Efficient and Robust Point Cloud Embedding: Theories, Algorithms and Applications

Preliminary Oral Exam (Thesis Proposal)

Dehao Yuan
Department of Computer Science
University of Maryland, College Park
dhyuan@umd.edu

Dissertation Proposal submitted to:
Department of Computer Science
University of Maryland, College Park, MD 20742

July 2, 2024

Advisory Committee:

Dr. Yiannis Aloimonos	Chair	University of Maryland, College Park
Dr. Furong Huang	Dean's Rep	University of Maryland, College Park
Dr. Cornelia Fermüller	Member	University of Maryland, College Park
Dr. Bruno Olshausen	Member	University of California, Berkeley

Abstract

This thesis proposal seeks to advance point cloud embedding by focusing on two critical areas: computational and memory efficiency, and robustness to noise and density variations. Existing methods, such as PointNet and KPConv, rely heavily on data-driven approaches that require extensive training to capture geometric features. These approaches, while effective in certain respects, fall short in terms of inherent robustness against environmental noise and data density fluctuations, and often require substantial computational resources. These limitations restrict their application in scenarios where speed and resource constraints are critical, such as in event camera stream processing and drone navigation.

In response, this proposal introduces novel methodologies that utilize kernel methods to enhance both the efficiency and robustness of point cloud embeddings, grounded in a strong theoretical framework. It further explores the application of these advanced embeddings in two distinct domains: real-time processing of event camera streams and numeric encoding in tabular data. These case studies demonstrate the versatility and potential impact of the proposed methods across various technological fields.

The thesis is structured to methodically address these challenges, presenting a comprehensive approach from foundational theories and algorithms to practical applications. This includes detailed discussions on the mathematical modeling of point clouds, development of efficient and robust embedding techniques using kernel methods, and their implementation in diverse settings.

Contents

1 Introduction

Overview. This thesis proposal advances point cloud embedding by enhancing two perspectives: 1) computation and memory efficiency, 2) robustness to noise and density variation.

Existing work, such as PointNet [1] and KPConv [2], are mainly data-driven. These involve learnable parameters optimized through extensive training for tasks like classification, segmentation, and masked pretraining [3]. These methods assume that sufficient training allows embeddings to accurately represent geometric features.

However, data-driven techniques lack inherent robustness against noise and variations in data density. Attempts to overcome these challenges typically involve data augmentation. Yet, they do not guarantee consistent robustness. Additionally, these models demand substantial computational resources. This restricts their use in speed-critical tasks like event camera stream processing and in resource-constrained environments like drone navigation.

In response, this proposal addresses two major challenges in point cloud embeddings: 1) computation and memory efficiency, 2) robustness to noise and density variation. Furthermore, it explores how advanced point cloud embeddings can be applied across two distinct domains: event camera stream processing and numeric encoding in tabular data. These applications illustrate the versatility of the advanced embeddings, demonstrating its potential to impact a wide range of technological fields significantly.

The introductory chapter of this thesis is structured as a concise overview of the entire proposal. Section 1.1 defines the problem of point cloud embedding. Section 1.2 emphasizes the requirement for efficiency and robustness in this context. Section 1.3 introduces the theories and algorithms developed to achieve these goals. Finally, Section 1.4 discusses the proposed applications that can benefit from these advancements.

The entire thesis proposal is structured as followed. Chapter 2 introduces the background information, including the mathematical formulation of point clouds, and several fields that are relevant and inspires the development of the thesis. Chapter 3 focuses on the efficiency and robustness to noise in point cloud embedding. Chapter 4 focuses on the robustness to density variation in point cloud embedding. Chapter 5 unifies Chapter 3 and 4, producing an elegant point cloud embedding algorithm that are 1) computation and memory efficiency, 2) robustness to noise and density variation. Chapter 6 and Chapter 7 discusses two applications of the unified algorithm presented in Chapter 5, demonstrating how the efficient and robust point embedding algorithms can benefit applications in diverse domains.

1.1 Point Cloud Embedding

Point Cloud Embedding transforms a point cloud—a set of data points in Euclidean space—into vector representations that capture the essential features of the data. Mathematically, this involves taking an unordered set of points $\mathbf{X} \in \mathbb{R}^{n\times 3}$ and outputting a vector $\mathbf{G} \in \mathbb{R}^{n\times d}$. Each element $\mathbf{G}[j] \in \mathbb{R}^{d}$ represents the geometric features of the point $\mathbf{X}[j]$, conditioned on the point cloud $\mathbf{X}$:

$$\mathbf{G}[j] = \text{Point\_Cloud\_Embedding}(\mathbf{X}[\text{j}] \mid \mathbf{X})$$

This initial definition provides a broad overview, intentionally omitting specific design details. We hope to demonstrate the variety of ways point cloud embedding tailored to different tasks. To illustrate this variety, several examples demonstrate how the point cloud embedding module is adapted for different types of tasks in Figure 1.1.

Local regression tasks focus on the Euclidean local neighborhood of a point $\mathbf{X}[j]$. In applications such as normal estimation and normal flow estimation, local geometric features are encoded and transformed into output vectors using point-wise Multilayer Perceptrons (MLPs).

Non-local regression tasks, in contrast, require $\mathbf{G}[j]$ to aggregate information from both nearby and distant points within the cloud. Such formulation is crucial for tasks like segmentation and detection, where understanding extensive spatial relationships enhances performance. Here, techniques like convolution and attention mechanisms are usually employed to aggregate these widespread features.

Condensation tasks aggregate all points into a single vector, for example, classification. Typically, a max-pooling operation is used to compress features from either local or non-local regression into a compact representation that encapsulates the overall characteristics of the point cloud.

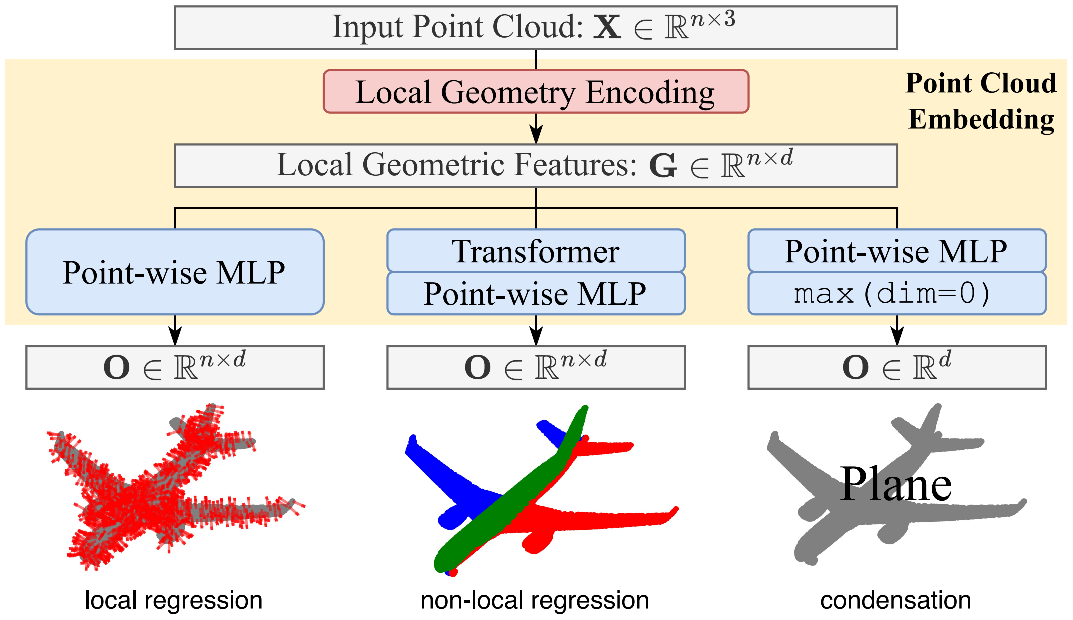

Input Point Cloud: $\mathbf{X} \in \mathbb{R}^{n\times 3}$
Point Cloud Embedding
Local Geometry Encoding
Local Geometric Features: $\mathbf{G} \in \mathbb{R}^{n\times d}$
Point-wise MLP
Transformer
Point-wise MLP
Point-wise MLP
max(dim=0)
$\mathbf{O} \in \mathbb{R}^{n\times d}$
$\mathbf{O} \in \mathbb{R}^{n\times d}$
$\mathbf{O} \in \mathbb{R}^{d}$
Plane
local regression
non-local regression
condensation

Figure 1.1: Different designs of point cloud embedding given different types of tasks.

1.2 Efficiency and Robustness

Point cloud embedding is fundamental for point cloud processing. Improving the efficiency and robustness will benefit many important applications. We illustrate with examples here.

Efficiency: In autonomous robotics, the ability to quickly interpret complex 3D environments is essential. Efficient point cloud embedding enables robots, such as those navigating through manufacturing plants, to identify obstacles and interact safely with their surroundings without delay. This rapid processing is vital for a robotic arm that must place items precisely on an assembly line. Similarly, drones, which often operate under significant power and computational limitations, benefit from efficient point cloud processing. In scenarios like disaster area mapping, drones need to analyze environmental data swiftly to conserve battery while providing vital information, ensuring they complete their missions before power depletion.

Robustness: Robustness addresses the challenges posed by data imperfections, such as sensor noise and density variations, which are common in real-world applications. In autonomous vehicles, robust point cloud embeddings help filter out noise from sensor inaccuracies, which is crucial during adverse weather conditions like heavy rain. Such robustness ensures that the vehicle maintains accurate spatial analysis, critical for safe navigation. In event camera stream processing, different camera configurations, lighting conditions will produce events with different densities. Robustness to density variation enables the model to generalize better to different environments.

1.3 Roadmap – Theories and Algorithms

The theories behind this thesis proposal comes from an elegant philosophy, which can be summarized in the following theorem:

Theorem 1. *A Gaussian kernel mixture* $f(\mathbf{x}) = \sum_{k=1}^{n} \alpha_k \exp(-\alpha^2 ||\mathbf{x} - \mathbf{x}_k||^2/2)$ *can be represented by* $\mathbf{F} = \sum_{k=1}^{n} \alpha_k \exp(i\mathbf{x}_k \mathbf{A}) \in \mathbb{C}^d$*, where* $A \in \mathbb{R}^{3\times d}$ *is a matrix where all elements are drawn from normal distribution* $\mathcal{N}(0, \alpha^2)$*. Besides, the representation is reconstructive and isometric.*

The Gaussian kernel mixture $f(\mathbf{x})$ is a non-parametric function that does not have a fixed dimension. So it cannot be fed to neural networks like MLP for machine learning tasks. Fortunately, this theorem claims that **we can almost losslessly transform the Gaussian kernel mixture $f(\mathbf{x})$ into a fixed-length vector** through pre-defined operations. Consequently, if a type of data can be described by a Gaussian kernel mixture $f(\mathbf{x})$, we immediately obtain the representation. The entire thesis is developed from this simple philosophy.

Chapter 2 introduces the background behind this theorem and gives the proof. Chapter 3, 4, 5 present the developed theories and algorithms for efficient and robust point cloud embedding from the philosophy. Specifically, Chapter 3 focuses on the efficiency, and Chapter 4 focuses on the robustness. Chapter 5 unifies both chapters and produce a point cloud embedding algorithm that is efficient and robust, with solid theoretical foundation.

1.4 Roadmap – Applications

Following Chapter 5, we apply the unified algorithm to two very distinct domains: optical flow estimation from event camera (Chapter 6) and numeric encoding for tabular pretraining (Chapter 7). In both chapters, we present our current methodologies and current preliminary experimental results. The applications will be the main focus after the preliminary exam.

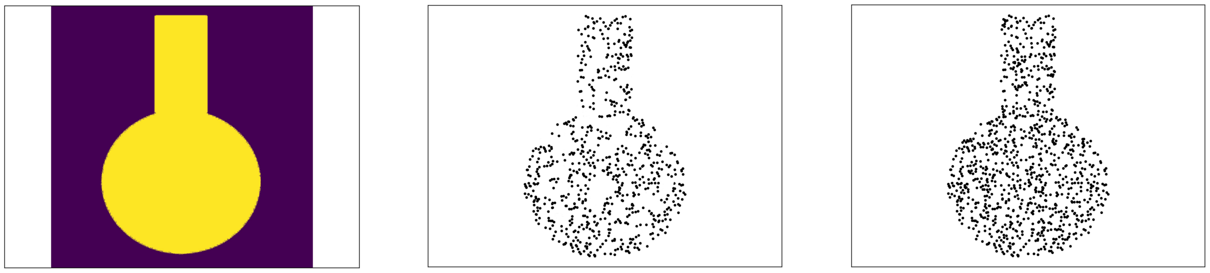
2 Background

2.1 Point Cloud and its Mathematical Modeling

A point cloud is an unordered collection of points in Euclidean space, represented as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$. These are typically generated by various sensing technologies. For example, 3D scanners collect detailed three-dimensional information from objects by emitting laser beams and measuring the return time of the light; this distance data is then converted into a spatial point cloud. Another source is event cameras, which detect changes in light intensity rather than capturing static frames. These cameras produce data streams where each event marks a change in light at a specific pixel, forming a temporal point cloud.

To develop point cloud embedding algorithms with theoretical support, it is essential to model point clouds mathematically. In this thesis proposal, point clouds are considered as samples from a continuous function. Specifically, an object's shape or an event surface is modeled as an (unknown) probability density function $f : \mathbb{R}^3 \to \mathbb{R}^+$, where $f(\mathbf{x})$ represents the probability density that a point $\mathbf{x}$ is on the shape. Thus, the point cloud is viewed as random samples of this function. To design a point cloud embedding module, we are essentially constructing the representation of the function f using the random samples.

Modeling point clouds in this manner provides a clear definition of density and noise. Noise in a point cloud means the observed data is $\mathbf{x}_k + \epsilon_k$, where $\mathbf{x}_k$ are samples from the function and ϵ_k represents noise. Density variation indicates that the shape function f is sampled with a non-uniform distribution. The research challenge then focuses on how to build a consistent representation of the function f with the presence of noise and density variations. This approach allows for the development of a robust point cloud embedding module with theoretical support. Figure 2.1 illustrates the mathematical modeling of point cloud.



(a) underlying function f. (b) point cloud (sparser). (c) point cloud (denser).

Figure 2.1: Mathematical modeling of point cloud. The point cloud is viewed as random samples from an underlying function f, which characterizes the object shape or event surface. Point cloud embedding aims to construct a consistent representation of function f using samples that may be corrupted with noise or density variation.

2.2 Hyper-Dimensional Computing

2.2.1 Overview

Hyper-dimensional Computing (HDC), also known as Vector Symbolic Architecture (VSA) [4], uses very high dimensional vectors, called hypervectors, to perform symbolic operations. By manually defining operations on hypervectors, we can design encoders that are descriptive and computationally efficient without any training.

Hypervectors d-dimensional random vectors are called hypervectors if they are drawn from a distribution $\mathcal{H}$ such that any two vectors are very likely orthogonal to each other. That is, for any $\epsilon > 0$, $\text{Prob.}\big(|cos(\mathbf{x}, \mathbf{y})| < \epsilon\big) \to 1$ when $d \to \infty$, where $\mathbf{x}, \mathbf{y} \sim \mathcal{H}$ and cos is the cosine similarity between two vectors. For example, a hypervector $\mathbf{x}$ can be generated by independently choosing a random value for each element of the vector from the Bernoulli distribution, which produces binary hypervectors. Hypervectors can be manipulated through two atomic operations, superposition and binding:

Superposition A binary operation $+ : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a superposition if:

1. Commutative property: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$.
2. Associative property: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$.

Superposition combines vectors into a single vector of the same dimension, and the superposed vector is similar to all of its components. For real vectors, superposition is done by summing or averaging vectors.

Binding A binary operation $\otimes : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a binding operation if:

1. Commutative property: $\mathbf{x} \otimes \mathbf{y} = \mathbf{y} \otimes \mathbf{x}$.
2. Distributive property: $\mathbf{x} \otimes (\mathbf{y} + \mathbf{z}) = \mathbf{x} \otimes \mathbf{y} + \mathbf{x} \otimes \mathbf{z}$.
3. Similarity preserving: $cos(\mathbf{x} \otimes \mathbf{y}_1, \mathbf{x} \otimes \mathbf{y}_2) = cos(\mathbf{y}_1, \mathbf{y}_2)$.
4. $\mathbf{x} \otimes \mathbf{y}$ has the same distribution of $\mathbf{x}$ and $\mathbf{y}$. In other words, $\mathbf{x} \otimes \mathbf{y} \in \mathcal{H}$.

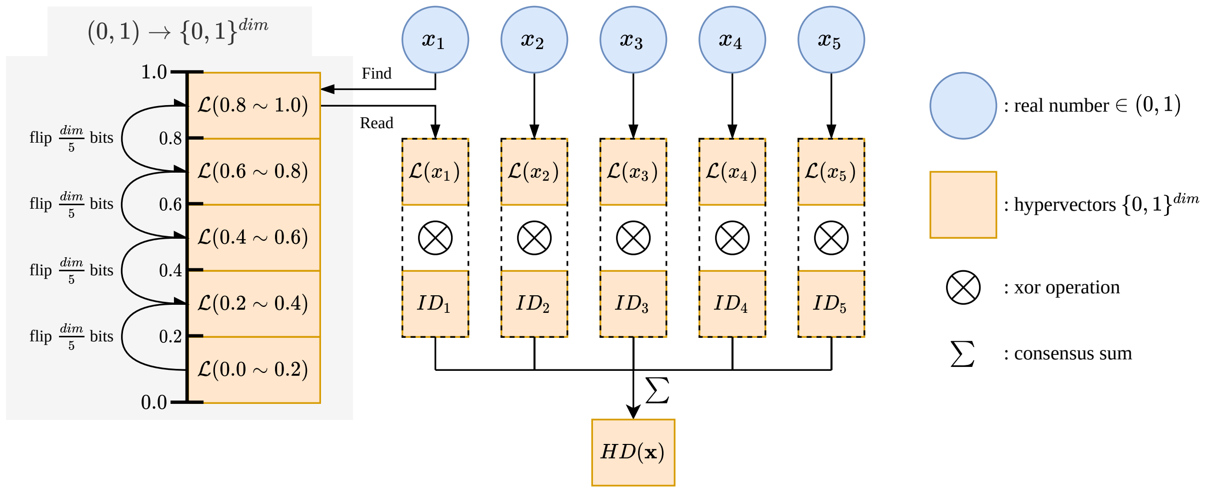
where $cos(\cdot, \cdot)$ is the cosine similarity between two hypervectors, and $\mathbf{x}$, $\mathbf{y}_1$ and $\mathbf{y}_2$ are hypervectors.

2.2.2 Examples of HDC Encoders

In Hyper-dimensional computing (HDC), an object is represented by a hyper-dimensional (HD) vector. However, obtaining the representation is not a trivial question. In this section, we give examples of converting a real vector to a binary HD vector.

Converting a real vector to an HD vector is essentially converting a signal to a symbol. Such conversion is significant because it allows symbolic operations on signals. As an analogy, a 1-d signal can be converted to its frequency domain by Fourier transformation. Such conversion offers many new options for signal processing such as denoising in the frequency domain. Similarly, converting a real vector (a signal) to an HD vector (a symbol) enables, for example, binding and bundling of two signals in the symbolic space.

There are two algorithms to convert a real vector to an HD vector: record-based encoding [5] and N-gram-based encoding [6]. For simplicity, we assume the real vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ has been normalized so that all elements x_i are in $(0, 1)$.



$(0,1) \to \{0,1\}^{dim}$
1.0
0.8
0.6
0.4
0.2
0.0
flip $\frac{dim}{5}$ bits
$\mathcal{L}(0.8 \sim 1.0)$
$\mathcal{L}(0.6 \sim 0.8)$
$\mathcal{L}(0.4 \sim 0.6)$
$\mathcal{L}(0.2 \sim 0.4)$
$\mathcal{L}(0.0 \sim 0.2)$
Find
Read
x_1 x_2 x_3 x_4 x_5
$\mathcal{L}(x_1)$ $\mathcal{L}(x_2)$ $\mathcal{L}(x_3)$ $\mathcal{L}(x_4)$ $\mathcal{L}(x_5)$
ID_1 ID_2 ID_3 ID_4 ID_5
$\sum$
$HD(\mathbf{x})$
: real number $\in (0,1)$
: hypervectors $\{0,1\}^{dim}$
: xor operation
: consensus sum

Figure 2.2: Pipeline of the record-based encoding.

Record-based Encoding Figure 2.2 shows the pipeline of the record-based encoding. First, we construct a mapping $\mathcal{L} : (0,1) \to \{0,1\}^d$ that converts a real number in $(0,1)$ to an HD vector: the interval $(0,1)$ is quantized into q sub-intervals. Each sub-interval is assigned an HD vector and all the real numbers in the sub-interval are mapped to the same HD vector. The HD representation of the first sub-interval is randomly generated, and the HD representation of the k-th sub-interval is obtained by randomly flipping $1/q$ bits of the $(k-1)$-th sub-interval. Under this construction scheme, neighboring sub-intervals will have similar HD representations and the HD representations of 0.0 and 1.0 will be orthogonal.

Using the mapping, each element x_i in $\mathbf{x}$ can be converted to an HD vector $\mathcal{L}(x_i)$. Then the HD representations of $\mathbf{x}$ is obtained by binding and bundling:

$$HD(\mathbf{x}) = \sum_i \left[\mathcal{L}(x_i) \oplus ID_i\right]$$

where $\otimes$ is the xor operation, ID_i are random identity HD vectors associated with the entry i, $\sum$ is the consensus sum.

N-gram-based Encoding N-gram-based encoding is similar to the record-based encoding. The same mapping $\mathcal{L} : (0,1) \to \{0,1\}^d$ is constructed and the HD representation of $\mathbf{x}$ is obtained by xor-permute operation:

$$HD(\mathbf{x}) = \bigoplus_i \left[\Pi^i \mathcal{L}(x_i)\right]$$

where $\bigoplus$ denotes the xor operation, Π^i denotes permuting an HD vector by i times.

2.2.3 Case Study: Neural Network Ensemble using Hyper-Dimensional Computing

We present an application of HDC encoders on neunal network ensemble. This case study is to demonstrate HDC encoders can yield meaningful representation if designed properly. The ensemble method discussed is named HD-Glue [7].

In hyper-dimensional computing (HDC), the most important operations are binding and bundling, which enable any object to be combined and associated on a symbolic level. With the current

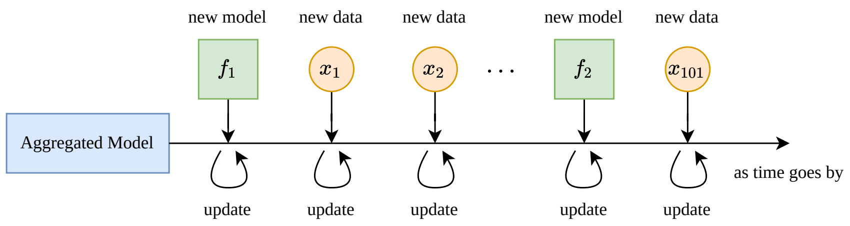


Figure 2.3: HD-Glue: Online learning setting.

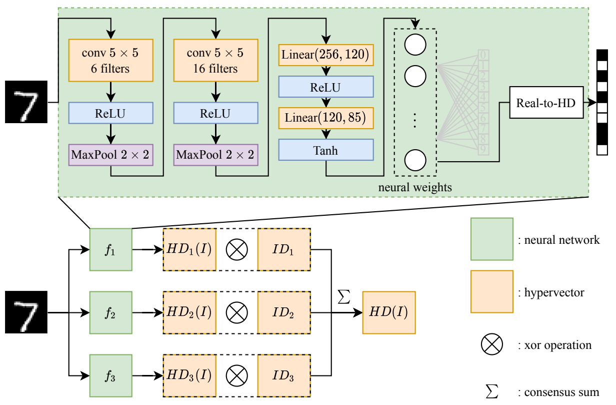


Figure 2.4: HD-Glue pipeline for static aggregation.

development of neural networks, it is natural to ask: can we ensemble multiple neural networks together so that they can work better as a group?

Although combining neural networks to improve the overall performance is already widely studied, known as neural network ensemble [8], the ensemble algorithm usually targets a better voting scheme, instead of a better representation of the input. In this proposal, we introduce HD-Glue, which aggregates neural networks on the symbolic level, and demonstrates that such aggregation is meaningful and useful.

Another novelty of HD-Glue is that it also supports online learning. As more data and more models are seen, the aggregated model can update itself continuously, without inducing heavy computational cost. In addition, the aggregated model also welcomes new classes to be added, and the learning of new classes is fast. Figure 2.3 shows the online learning pipeline.

In the following sections, we will introduce the details of HD-Glue. First, in *Static Aggregation*, we introduce how HD-Glue works when many models and data are presented simultaneously. Then, in *Dynamic Aggregation*, we introduce how HD-Glue works when models and data are presented sequentially. Finally, in *Experiment*, we show some experiments on different benchmark datasets under the two settings.

Static Aggregation Figure 2.4 shows the pipeline of static aggregation. For each neural network (for illustration purpose we assume it is an image classifier), its final softmax regression layer is removed so that it can serve as an image encoder. With these image encoders, an image can be transformed into multiple embedding vectors. The embedding vectors are then converted to HD vectors by the record-based encoding algorithm and then bound and bundled to form a single HD vector. Now that an image is transformed to its HD representation, we can apply the normal hyper-dimensional classification algorithm in which we build the class prototypes.

The static aggregation is a symbolic aggregation. Instead, one may choose to aggregate the networks by the simple average of the image embeddings, which is a signal aggregation. In some situations, symbolic aggregation enjoys more benefits over signal aggregation. For example, when aggregating models with different modalities, symbolic aggregation can be applied without introducing new parameters. Using signal aggregation, on the other hand, one must project the two modalities onto one common space, which requires learning new parameters.

Dynamic Aggregation There are two scenarios in dynamic aggregation. First, when a new training sample is seen, the training sample is converted to its HD representation and summed to its class prototypes. The summation is frequent and cheap. Second, when a new model is seen, the class prototypes are reconstructed using static aggregation of all the historical data and models. This step is expensive but rarely happens.

Experiment Table 2.1 demonstrates that HD-Glue can aggregate neural networks with different embedding sizes with high accuracy. Table 2.2 shows the accuracy of dynamically aggregating multiple neural networks, tested on MNIST. New classes are added sequentially. It demonstrates HD-Glue can quickly learn to classify new classes.

CIFAR-100 Different Embedding Sizes			
	Length		
	512		**256**
Model	***VGG11***	***VGG13***	***ResNet18***
Accuracy	65.8%	67.8%	72.3%
Model	***VGG16***	***VGG19***	***ResNet34***
Accuracy	65.4%	60.3%	74.5%
	HD-glue		
Size	**dim=4000**	**dim=8000**	**dim=12000**
100	64.1%	68.4%	70.5%
500	72.7%	**74.8%**	**75.2%**
1000	73.1%	**75.5%**	**75.9%**
5000	74.1%	**76.2%**	**76.3%**

Table 2.1: Testing accuracy when the embedding sizes of networks are different

MNIST Online Learning Results					
	Number of classes with 100 new examples each				
Class	**2×100**	**4×100**	**6×100**	**8×100**	**10×100**
Digit 0	99.9%	99.3%	98.1%	94.1%	94.3%
Digit 1	100.0%	98.1%	97.6%	98.2%	97.9%
Digit 2		86.9%	90.8%	86.7%	82.9%
Digit 3		98.9%	89.2%	82.4%	80.0%
Digit 4			98.0%	95.2%	84.4%
Digit 5			82.6%	82.1%	79.1%
Digit 6				94.9%	95.0%
Digit 7				92.9%	89.0%
Digit 8					85.5%
Digit 9					83.2%
All	99.9%	95.8%	93.0%	91.0%	87.3%

Table 2.2: Testing accuracy under dynamic aggregation setting.

2.3 Kernel Methods

Gaussian Kernel, also known as the Radial Basis Function (RBF) kernel, is a prominent kernel used in machine learning for transforming data into a higher-dimensional space. The kernel function is defined as:

$$\mathcal{G}_\alpha(\mathbf{x}, \mathbf{y}) := \exp\big(- \frac{\alpha^2 ||\mathbf{x} - \mathbf{y}||^2}{2} \big)$$

Here, $\mathbf{x}$ and $\mathbf{y}$ represent feature vectors in the input space, $||\mathbf{x} - \mathbf{y}||$ is the Euclidean distance between these vectors, and α is the kernel's bandwidth parameter, which controls the scale of the feature space transformation. The exponential decay component of the Gaussian kernel makes it sensitive to the distance between data points, effectively capturing complex, non-linear relationships by mapping the input data into a potentially infinite-dimensional space.

Support Vector Machine. Gaussian kernel is employed to enable the classifier to construct non-linear decision boundaries. SVMs operate by finding the optimal separating hyperplane that maximizes the margin between different classes in the transformed feature space. When using a Gaussian kernel, the SVM model computes the decision function:

$$f(x) = \sum_{k=1}^{n} \alpha_k y_k \mathcal{G}(\mathbf{x}_k, \mathbf{x}) + b$$

where $\mathbf{x}_k$ are support vectors, α_k are Lagrange multipliers (non-zero for support vectors), y_k are the class labels, and b is the bias. The decision function $f(x)$ represents the inner product of the input vector $\mathbf{x}$ with each support vector in the higher-dimensional space, weighted by α_k and y_k, and adjusted by the bias b. The decision function $f(x)$ allows the SVM to classify data points based on their position relative to the decision boundary in the new feature space.

One-Class Support Vector Machine is a variation of the traditional Support Vector Machine that is used for anomaly detection, where the primary goal is to identify how well a new data point fits within the distribution of an existing dataset. Unlike traditional SVMs that differentiate between two or more classes based on a labeled dataset, One-Class SVM works with a single-class dataset to determine the data boundary.

The Gaussian kernel plays a crucial role in One-Class SVM by enabling the mapping of input data into a higher-dimensional space, where the separation from the origin is feasible. Given its ability to handle non-linear patterns, the Gaussian kernel is particularly effective when the distribution of data in the original space is not spherical or when the data contains complex structures. Using the Gaussian kernel, the decision function for One-Class SVM becomes:

$$f(x) = \sum_{k=1}^{n} \alpha_k \mathcal{G}(\mathbf{x}, \mathbf{x}_k) + b$$

This function will predict positive values for data points that are considered "normal", and negative values for outliers or anomalies, based on their position relative to the learned boundary in the feature space.

Bochner's Theorem states that the Gaussian kernel $\mathcal{G}(\mathbf{x}, \mathbf{y})$ can be approximated by the inner product of finite-length vectors $e^{i\mathbf{x}\mathbf{A}}$ and $e^{i\mathbf{y}\mathbf{A}}$, where $\mathbf{A}$ is a matrix. This is a very useful theorem when we build representation of point clouds, which we will demonstrate in the following section after we prove the theorem.

Lemma 1 (Bochner's Theorem)**.** *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$, $\mathbf{A} \in \mathbb{R}^{3\times d}$. All elements in $\mathbf{A}$ are drawn from normal distribution $\mathcal{N}(0, \alpha^2)$. Then as $d \to \infty$,*

$$\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle \to \mathcal{G}_\alpha(\mathbf{x}, \mathbf{y}) := \exp\big(-\frac{\alpha^2||\mathbf{x}-\mathbf{y}||^2}{2}\big)$$

Proof. Let $\mathbf{a} \in \mathbb{R}^3$ where $\mathbf{a} \sim \mathcal{N}(\mathbf{0}, \alpha^2\mathbf{I}_{3\times3})$ be one column of the matrix $\mathbf{A}$, we claim that $\mathbb{E}\big[\Re\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\big] = \mathcal{G}_\alpha(\mathbf{x}, \mathbf{y})$:

$$\begin{aligned}
\mathbb{E}\Big[\Re\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\Big] &= \mathbb{E}\Big[\cos\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)\Big] \\
&= \mathbb{E}\Big[\sum_{k=0}^{\infty}(-1)^k\frac{\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)^{2k}}{(2k)!}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\mathbb{E}\Big[\big(\sum_{j=1}^{3}(\mathbf{x}_j-\mathbf{y}_j)\mathbf{a}_j\big)^{2k}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\mathbb{E}\Big[\big(\alpha||\mathbf{x}-\mathbf{y}||Z\big)^{2k}\Big] \quad \text{where } Z \in \mathcal{N}(0,1) \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\cdot\alpha^{2k}||\mathbf{x}-\mathbf{y}||^{2k}\cdot\frac{(2k)!}{k!2^k} \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{k!2^k}\cdot\alpha^{2k}||\mathbf{x}-\mathbf{y}||^{2k} \\
&= \exp\big(-\frac{\alpha^2||\mathbf{x}-\mathbf{y}||^2}{2}\big)
\end{aligned}$$

On the other hand, $\mathbb{E}\big[\Im\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\big] = 0$ because normal distribution is a symmetric distribution around 0:

$$\begin{aligned}
\mathbb{E}\Big[\Im\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\Big] &= \mathbb{E}\Big[\sin\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)\Big] \\
&= \mathbb{E}\Big[\sum_{k=0}^{\infty}(-1)^k\frac{\big(\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})\big)^{2k+1}}{(2k+1)!}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\mathbb{E}\Big[\big(\sum_{j=1}^{3}(\mathbf{x}_j-\mathbf{y}_j)\mathbf{a}_j\big)^{2k+1}\Big] \\
&= \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\mathbb{E}\Big[\big(\alpha||\mathbf{x}-\mathbf{y}||Z\big)^{2k+1}\Big] \quad \text{where } Z \in \mathcal{N}(0,1) \\
&= 0 \quad \text{because } \mathbb{E}(Z^{2k+1}) = 0
\end{aligned}$$

Therefore, when we randomize d rows of such $\mathbf{a}$ vector, the inner product $\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle = \frac{1}{d}\sum_{k=1}^{d} e^{i\mathbf{a}_k\cdot(\mathbf{x}-\mathbf{y})}$ will converge to $\mathcal{G}_\alpha(\mathbf{x}, \mathbf{y})$ thanks to the Central Limit Theorem. □

Vector Function Architecture. Now with all the pieces of tools mentioned above, we are ready to see how the tools can be assembled to build powerful representation. We found that the decision functions returned by SVM or one-class SVM are both in analytic form of

$$f(\mathbf{x}) = \sum_{k=1}^{n} \alpha_k \mathcal{G}(\mathbf{x}, \mathbf{x}_k) + b$$

Currently, the function $f(\mathbf{x})$ is an non-parametric function that does not have a fixed-length dimension. Fortunately, **with the help of Bochner's theorem and vector function architecture (VFA) [9], we can transform the decision function $f(\mathbf{x})$ into a fixed-length vector.** The entire theories behind this thesis proposal is built upon this simple idea:

Theorem 2 (VFA + Bochner's Theorem)**.** $f(\mathbf{x}) = \sum_{k=1}^{n} \alpha_k \exp(-\alpha^2 ||\mathbf{x} - \mathbf{x}_k||^2/2)$ *can be represented by* $\mathbf{F} = \sum_{k=1}^{n} \alpha_k \exp(i\mathbf{x}_k \mathbf{A}) \in \mathbb{C}^d$*, where* $A \in \mathbb{R}^d$ *is a matrix where all elements are drawn from normal distribution* $\mathcal{N}(0, \alpha^2)$*. The representation is reconstructive and isometric.*

The proof of the theorem is an immediate corollary from VFA and Bochner's Theorem. It shows how a Gaussian kernel mixture can be represented into a vector. Consequently, if a type of data can be described by a Gaussian kernel mixture $f(\mathbf{x})$, we immediately obtain the representation. The entire thesis is developed from this simple philosophy. Now we are ready to dive into the theories, algorithms and methodology derived from this philosophy.

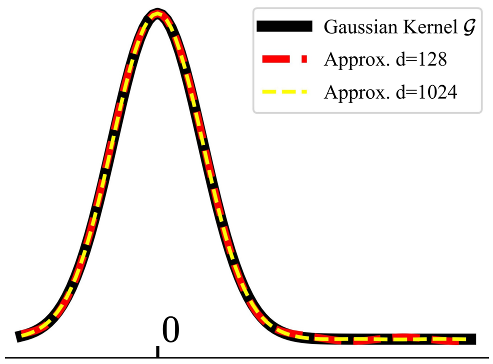


Figure 2.5: Approximation of Gaussian kernel with Bochner's theorem.

3 Theories & Algorithms I (Efficiency): A Linear Time and Space Local Point Cloud Geometry Encoder via Vectorized Kernel Mixture

This chapter introduces VecKM, which focuses on the computation and memory efficiency in point cloud embedding.

3.1 Introduction

In point cloud analysis, encoding local geometry is a fundamental step. In both low-level tasks such as feature matching and normal estimation, and high-level tasks such as classification, segmentation, and detection, encoding local geometry is usually required before passing the point cloud into any deep network. Much effort has been placed into the design of local geometry encoders, which can be loosely divided into two categories: hand-crafted features and learnable encoders. Hand-crafted features [10] are manually defined features based on domain expertise, and learnable encoders require computationally expensive processing through trainable structures such as multi-layer perceptrons (MLP) [1; 11] or convolutions [12; 2].

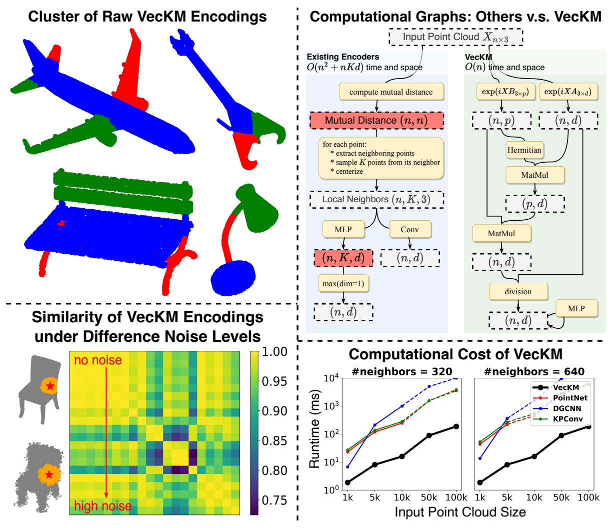


Figure 3.1: Our VecKM encoding is descriptive, robust to noise, and efficient in runtime and memory cost. **Upper Left**: Raw VecKM encodings, *without any training*, already capture rich geometric features such as orientations and shapes. **Lower Left**: Under varying levels of noise, VecKM encodings remain highly consistent. **Upper Right**: Existing encoders face memory costs of $(n^2 + nKd)$, while VecKM costs only $(nd + np)$ memory. Existing encoders compute nK MLPs, whereas VecKM only computes n MLPs. **Lower Right**: VecKM is 10x$\sim$100x faster than existing encoders in wall-clock time and scalable to large point cloud inputs.

These local geometry encoders follow a similar pipeline. They first group the input point cloud into neighborhoods and then process each neighborhood individually. As illustrated in Figure 3.1 (upper right), the pipeline involves computing the mutual distance between points. Then for each point, a number of K points are sampled from its neighborhood, and MLP or convolution are used to transform the sampled neighborhood. In this pipeline, grouping the point cloud into neighborhoods requires n^2 time and space, and the MLP-based architectures, in particular, requires a sequence of MLPs to transform nK vectors and reaches an intermediate stage of (n, K, d). The pipeline results in bottlenecks in both computation and memory. Consequently, they usually resort to downsampling the local point clouds (i.e. reducing K), which can lead to inadequate representation of the local point cloud.

In this work, we address the computation and memory bottlenecks faced by the existing encoders, reducing the memory cost from $(n^2 + nKd)$ to $(nd + np)$ and only computing n MLPs. Besides, our representation is constructed from all the neighboring points without downsampling, and hence is more descriptive. Our approach is inspired by [9; 13], which converts continuous functions into fixed-length vectors. Building on this concept, we introduce VecKM, which conceptualizes local point clouds as kernel mixtures (a form of continuous function) and vectorizes them. Under this formulation, we prove the local geometry encoding is reconstructive and isometric to the local point cloud, which guarantees the descriptiveness of the representation. One essential advantage of VecKM is its factorizable property, which eliminates the need of explicitly grouping the neighborhoods and reuses many computations.

The VecKM encodings can subsequently be passed to deep cloud point models, such as PointNet++ [14] and transformers [15]. VecKM's light representation and ease of computation significantly speed up the inference, while still achieving on-par or improved performance than other networks in classification and segmentation tasks. Our contributions are summarized below:

- We present VecKM, a local geometry encoder that is descriptive and efficient. VecKM costs only $nd + np$ memory and computes only n MLPs. This is achieved through a novel approach of vectorizing kernel mixtures, coupled with its unique factorizability. VecKM is the only existing local geometry encoder that costs linear time and space.
- Unlike existing encoders downsampling the local point cloud, VecKM constructs the local geometry encoding using all neighboring points, and hence is more descriptive.
- We evaluate our VecKM on multiple point cloud tasks. In normal estimation, VecKM is $>$ 100x faster and achieves $>$ 16% lower error than other widely-used learnable encoders and demonstrates the strongest robustness against different types of data corruption. In classification and segmentation tasks, integrating VecKM as a preprocessing module achieves consistently better performance than the PointNet, PointNet++, and point transformer baselines, and runs consistently faster by up to 10 times.

3.2 Methodology

3.2.1 Problem Definition and Main Theorems

Problem Definition. Let the input point cloud be $X = \{\mathbf{x}_k\}_{k=1}^n$. Denote the centerized neighbor of the point $\mathbf{x}_k$ as $\mathfrak{N}(\mathbf{x}_k) := \{\mathbf{x}_j - \mathbf{x}_k : ||\mathbf{x}_j - \mathbf{x}_k|| < r\}$. The output is the set of dense local geometric features $G = \{\mathbf{g}_k\}_{k=1}^n$, where $\mathbf{g}_k = E\big(\mathfrak{N}(\mathbf{x}_k)\big) \in \mathbb{C}^d$. We look for an encoder E that maps the local point cloud into a fixed-length vector, which captures the underlying shape" sampled by the point cloud.

To better formalize the heuristic expression of capturing the underlying shape", we think of the local shape around the point x_k as a distribution function $f_k : \mathbb{R}^3 \to \mathbb{R}^+$, where $f_k(\mathbf{x})$ gives the probability density that a point $\mathbf{x}$ is on the local shape. We then think of the centerized local point cloud $\mathfrak{N}(\mathbf{x}_k)$ as random samples from the distribution function f_k. We expect the local point cloud encoding $E\big(\mathfrak{N}(\mathbf{x}_k)\big) \in \mathbb{C}^d$ to represent the distribution function f_k. For a good representation, we consider two natural properties: 1. the distribution function can be reconstructed from the encoding; 2. the correlation of the distribution functions is preserved by the similarity of the encodings.

Pointwise Local Geometry Encoding. Under the problem definition, we present the formula for encoding the local geometry around a single point. Unless specified otherwise, all input points $\mathbf{x}_j$ are assumed to be three-dimensional.

Theorem 3 (Pointwise Local Geometry Encoding)**.** *Denote the neighbors of the point* $\mathbf{x}_0$ *as* $\mathfrak{N}(\mathbf{x}_0) := \{\mathbf{x}_k - \mathbf{x}_0\}_{k=1}^n$. *The local geometry encoding of* $\mathbf{x}_0$ *is computed as*

$$E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big) = \frac{1}{n}\sum_{k=1}^{n} \exp\big(i(\mathbf{x}_k - \mathbf{x}_0)\mathbf{A}_{3\times d}\big) \tag{3.1}$$

where i *is the imaginary unit and* $\mathbf{A} \in \mathbb{R}^{3\times d}$ *is a fixed random matrix where each element follows the normal distribution* $\mathcal{N}(0, \alpha^2)$. As to be shown in Section 3.2.2, $E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big)$ is fundamentally vectorizing a kernel mixture about $\mathfrak{N}(\mathbf{x}_0)$, so we name the encoding VecKM. Next, we present two propositions that claim VecKM encoding produces a good representation of the local shape:

Proposition 1 (Reconstruction)**.** *WLOG, let* f *be the distribution function characterizing the local shape of* $\mathbf{0}$, $X = \{\mathbf{x}_k\}_{k=1}^n$ *be the random samples drawn from the distribution function* f, *and* $\mathbf{g}_n = \frac{1}{n}\sum_{k=1}^n \exp\big(i\mathbf{x}_k\mathbf{A}\big)$ *be the VecKM encoding given by Eqn.* (3.1). $\mathbf{A} \in \mathbb{R}^{3\times d}$ *is a fixed matrix whose entries are drawn from* $\mathcal{N}(0, \alpha^2)$. *Then at all points* $\mathbf{x}$ *where* $f(\mathbf{x})$ *is continuous, as* $n \to \infty$ *and* $\alpha^2 \to 0$,

$$\langle \mathbf{g}_n, \exp(i\mathbf{x}\mathbf{A})\rangle \to f(\mathbf{x})$$

where $\langle\cdot,\cdot\rangle$ *denotes the inner product between two complex vectors.* The proposition states that under a suitable selection of the parameter α^2, the distribution function f can be approximately reconstructed from the VecKM encoding $\mathbf{g}_n$.

Proposition 2 (Similarity Preservation)**.** *Let* f_1, f_2 *be two distribution functions characterizing two local shapes and* X_1, X_2 *be the random samples from the two distribution functions.* $\mathbf{g}_1$, $\mathbf{g}_2$ *are the VecKM encodings given by Eqn.* (3.1) *with* X_1, X_2 *as inputs.* $\mathbf{A}$ *is a fixed matrix whose entries are drawn from* $\mathcal{N}(0, \alpha^2)$. *Then the function similarity is preserved by the VecKM encodings: as* $n \to \infty$ *and* $\alpha^2 \to 0$,

$$\langle \mathbf{g}_1, \mathbf{g}_2\rangle \to \langle f_1, f_2\rangle = \int_{\mathbb{R}^3} f(x)g(x)dx$$

where $\langle\cdot,\cdot\rangle$ *denotes the inner product between two complex vectors.* The proposition states that under a suitable selection of the parameter α^2, the correlation of functions (i.e. shapes) is approximately preserved by the VecKM encoding.

In brief, Theorem 3 presents the formula for encoding the local geometry around a single point. Proposition 1, 2 assert that VecKM well represents the underlying local geometry. In Section 3.2.2, we will explain the mechanism behind Theorem 3 and prove Proposition 1, 2 in detail.

Dense Local Geometry Encoder With Eqn. (3.1), we can already compute the local geometry encoding for each point individually by grouping their neighborhoods. However, VecKM has a unique factorizable property that enables us to reuse computations and eliminate the intermediate step:

Theorem 4 (Dense Local Geometry Encoding)**.** *Denoting the input point cloud as a matrix* $\mathbf{X}_{n\times 3} = [\mathbf{x}_1; \mathbf{x}_2; \cdots; \mathbf{x}_n]$, *the dense local geometry encoding* $\mathbf{G}_{n\times d}$ *is computed by*

$$\begin{aligned}
\mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d})\\
\mathcal{B}_{n\times p} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p})\\
\mathbf{G}_{n\times d} &= \mathit{normalize}\big((\mathcal{B}\times\mathcal{B}^H\times\mathcal{A})\ ./\ \mathcal{A}\big)
\end{aligned} \tag{3.2}$$

where $\mathbf{A}$ *and* $\mathbf{B}$ *are two random fixed matrix whose entries are drawn from* $\mathcal{N}(0,\alpha^2)$ *and* $\mathcal{N}(0,\beta^2)$. $\times$ *denotes the matrix multiplication, and ./ denotes the elementwise division.* As to be explained in Section 3.2.3, computing the dense local geometry encoding using Eqn. (3.2) has almost the same effect as computing the pointwise local geometry encoding using Eqn. (3.1). However, Eqn. (3.2) only takes $\Theta(npd)$ time and $(np+nd)$ space to compute, where p, to be shown, is a marginal factor. The computation graph is visualized in Figure 3.1 (upper right).

Structure of Proof. In Section 3.2.2, we explain the mechanism behind Theorem 3 and prove our assertion that VecKM produces a good representation of the local geometry. In Section 3.2.3, we explain why Eqn. (3.2) has almost the same effect as Eqn. (3.1) and the mechanism behind Theorem 4. In Section 3.2.4, we introduce how to incorporate VecKM encodings into deep point cloud architectures.

3.2.2 Pointwise Local Geometry Encoder

In this section, we introduce why VecKM (Eqn. 3.1) produces a good representation of the local geometry. The key idea, as illustrated in Figure 3.2, is that (i) VecKM vectorizes a Gaussian kernel mixture associated with the local point cloud, where (ii) the associated kernel mixture can approximate the local shape distribution function. Therefore, VecKM effectively represents the local shape. We will separately validate assertion (i) and (ii).

(i) VecKM vectorizes a kernel mixture. We first present a lemma stating that VecKM embodies a Gaussian kernel $\mathcal{G}$:

Lemma 1 (VecKM embodies a Gaussian kernel)**.** *Let* $\mathbf{x},\mathbf{y}\in\mathbb{R}^3$, $\mathbf{A}\in\mathbb{R}^{3\times d}$. *All elements in* $\mathbf{A}$ *are drawn from normal distribution* $\mathcal{N}(0,\alpha^2)$. *Then as* $d\to\infty$,

$$\frac{1}{d}\langle e^{i\mathbf{x}\mathbf{A}}, e^{i\mathbf{y}\mathbf{A}}\rangle \to \mathcal{G}_\alpha(\mathbf{x},\mathbf{y}) := \exp\big(-\frac{\alpha^2\|\mathbf{x}-\mathbf{y}\|^2}{2}\big)$$

Lemma 1 is a corollary from the Bochner's theorem [16; 17], which we provide a proof in Chapter 2. Importantly, the Gaussian kernel $\mathcal{G}$ is approximated by the inner product of finite-length vectors $e^{i\mathbf{x}\mathbf{A}}$ and $e^{i\mathbf{y}\mathbf{A}}$. This approximation is important in vectorizing the kernel mixture and ensures the reconstructive and isometric properties in Proposition 1, 2, as detailed in the subsequent two lemmas. Unless otherwise specified, all entries in $\mathbf{A}$ are drawn from $\mathcal{N}(0,\alpha^2)$ and $\mathcal{G}$ means $\mathcal{G}_\alpha$. The proofs are borrowed from [9].

Lemma 2 (Reconstruction)**.** *Let* $\mathbf{g}=\frac{1}{n}\sum_{k=1}^n \exp(i\mathbf{x}_k\mathbf{A})$ *be the VecKM encoding, where all entries in* $\mathbf{A}\in\mathbb{R}^{3\times d}$ *are drawn from* $\mathcal{N}(0,\alpha^2)$. $\hat{f}(\mathbf{x})=\frac{1}{n}\sum_{k=1}^n\mathcal{G}_\alpha(\mathbf{x},\mathbf{x}_k)$ *be the associated Gaussian kernel mixture. Then* $\langle\exp(i\mathbf{x}\mathbf{A}),\mathbf{g}\rangle\to\hat{f}(\mathbf{x})$ *as* $d\to\infty$.

The lemma is derived from the linearity of the inner product:

$$\begin{aligned}\langle\exp(i\mathbf{x}\mathbf{A}),\mathbf{g}\rangle &= \frac{1}{n}\sum_{k=1}^n\langle\exp(i\mathbf{x}\mathbf{A}),\exp(i\mathbf{x}_k\mathbf{A})\rangle\\ &\to \frac{1}{n}\sum_{k=1}^n\mathcal{G}(\mathbf{x},\mathbf{x}_k)=\hat{f}(\mathbf{x})\end{aligned}$$

The lemma states that the Gaussian kernel mixture can be approximately reconstructed from the VecKM encoding $\mathbf{g}$, which theoretically shows that VecKM is equivalent to the Gaussian kernel mixture when d is large.

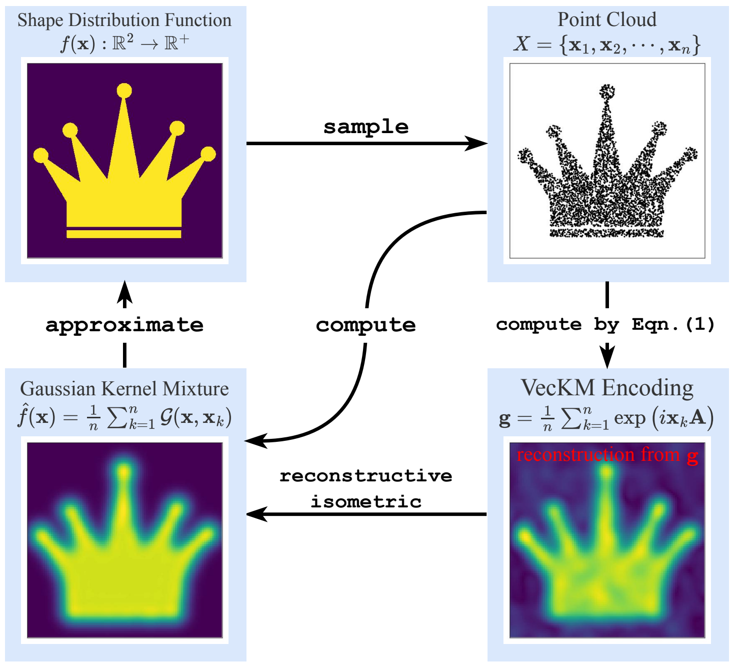

Shape Distribution Function
$f(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^+$
Point Cloud
$X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$
sample
approximate
compute
compute by Eqn. (1)
Gaussian Kernel Mixture
$\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{k=1}^{n}\mathcal{G}(\mathbf{x}, \mathbf{x}_k)$
VecKM Encoding
$\mathbf{g} = \frac{1}{n}\sum_{k=1}^{n}\exp(i\mathbf{x}_k\mathbf{A})$
reconstruction from g
reconstructive isometric

Figure 3.2: Theoretical outline of VecKM illustrated by 2d shapes. A point cloud, sampled from a shape distribution function, is associated with a Gaussian kernel mixture and a corresponding VecKM encoding, where the VecKM encoding is proved to be reconstructive and isometric to the Gaussian kernel mixture. Since the Gaussian kernel mixture can approximate the shape function, the VecKM encoding yields a good representation of the shape.

Lemma 3 (Similarity Preservation)**.** *Let* $\mathbf{g}_1$*,* $\mathbf{g}_2$ *be two VecKM encodings and* f_1*,* f_2 *be their associated Gaussian kernel mixtures. Then* $\langle \mathbf{g}_1, \mathbf{g}_2 \rangle \to \langle f_1, f_2 \rangle$ *as* $d \to \infty$*.*

The lemma states that the VecKM encoding preserves the similarity/correlation between kernel mixtures, which further verifies that the encoding is not only equivalent but also isometric to the Gaussian kernel mixture. The lemma is derived from the linearity of integration:

$$\begin{aligned}\langle f_1, f_2 \rangle &= \int_{\mathbf{x}\in\mathbb{R}^3} \Big(\frac{1}{n}\sum_{p=1}^{n}\mathcal{G}(\mathbf{x}, \mathbf{x}_p)\Big)\Big(\frac{1}{m}\sum_{q=1}^{m}\mathcal{G}(\mathbf{x}, \mathbf{x}'_q)\Big) d\mathbf{x} \\ &= \frac{1}{mn}\sum_{p,q}\int_{\mathbf{x}\in\mathbb{R}^3}\mathcal{G}(\mathbf{x}, \mathbf{x}_p)\mathcal{G}(\mathbf{x}, \mathbf{x}'_q) d\mathbf{x} \\ &= \frac{1}{mn}\sum_{p,q}\mathcal{G}(\mathbf{x}_p, \mathbf{x}'_q) \leftarrow \langle \mathbf{g}_1, \mathbf{g}_2 \rangle\end{aligned}$$

Lemma 1-3 complete the argument that the VecKM encoding is equivalent and isometric to the kernel mixture when d is large. In practice, the selection of d is independent of the size of the point cloud. d as small as 256 yields good encoding in many scenarios.

(ii) The Gaussian kernel mixture associated with the point cloud approximates the shape function. This is derived from the one-class support vector machine (SVM). The input to the one-class SVM is a collection of points and a user-defined kernel function, where the Gaussian (a.k.a. radial basis function) kernel is a common choice. The output of the one-class SVM is a kernel mixture which estimates the distribution of the input point set. [18] proves that with an appropriately chosen parameter α^2 (defined in Lemma 1), a Gaussian kernel mixture can approximate the distribution function. This validates the assertion that the kernel mixtures associated with VecKM can approximate the shape distribution function. Coupled with Lemma 2, 3, we prove Proposition 1, 2, which reveal that VecKM effectively represents the local geometry.

3.2.3 Dense Local Geometry Encoder

In the previous section, we explained why Eqn. 3.1 well represents the underlying local shape. In this section, we introduce the unique factorizable property that enables efficient computation of the dense local geometry encoding.

The geometry encoding in Eqn. (3.1) can be factorized into:

$$\begin{aligned}
E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big) &= \frac{1}{n}\sum_{k=1}^{n} \exp\big(i(\mathbf{x}_k - \mathbf{x}_0)\mathbf{A}_{3\times d}\big) \\
&= \frac{1}{n}\Big[\sum_{k=1}^{n} \exp(i\mathbf{x}_k\mathbf{A})\Big] \ ./\ \exp(i\mathbf{x}_0\mathbf{A})
\end{aligned}$$

Under this observation, we can write the dense local geometry encoding in terms of matrix computation:

$$\begin{aligned}
\mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d}) \\
\mathbf{G}_{n\times d} &= [\mathbf{J}_{n\times n}\mathcal{A}_{n\times d}] \ ./\ \mathcal{A}_{n\times d}
\end{aligned} \tag{3.3}$$

$\mathbf{J}_{n\times n}$ is the adjacency matrix of the point cloud $\mathbf{X}_{n\times 3}$, where $\mathbf{J}[j,k] = 1$ if $||\mathbf{x}_j - \mathbf{x}_k|| < r$ and 0 otherwise. Under this formulation, we still require n^2 time and space to compute the adjacency matrix $\mathbf{J}$ and $(n^2 d)$ FLOPs to compute $\mathbf{G}$. But one important idea can be applied to speed up the computation: Instead of adopting a sharp threshold r to define the adjacency relation, we employ an exponential decay function to establish this relationship:

$$\hat{\mathbf{J}}[j,k] = \exp(-\beta^2||\mathbf{x}_j - \mathbf{x}_k||^2/2)$$

where $\hat{\mathbf{J}}[j,k]$ decays from 1 to 0 as $||\mathbf{x}_j - \mathbf{x}_k||$ increases and the parameter β controls the speed of decaying. As comparison, $\mathbf{J}[j,k]$ drops sharply from 1 to 0 when $||\mathbf{x}_j - \mathbf{x}_k||$ reaches $> r$. The parameter β in $\hat{\mathbf{J}}$ has the same functionality as the parameter r in $\mathbf{J}$, which is controlling the receptive field of the local neighbors. Arguably, $\mathbf{J}$ and $\hat{\mathbf{J}}$ behave similarly and it is natural to substitute $\mathbf{J}$ with $\hat{\mathbf{J}}$ in Eqn. (3.3). The motivation of this substitution is that $\hat{\mathbf{J}}$ can be factorized into a matrix multiplication:

$$\begin{aligned}
&\mathcal{B}_{n\times p} = \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p}) \\
&\hat{\mathbf{J}}_{n\times n} \leftarrow \mathcal{B}\times\mathcal{B}^H \quad \text{as } p\to\infty
\end{aligned}$$

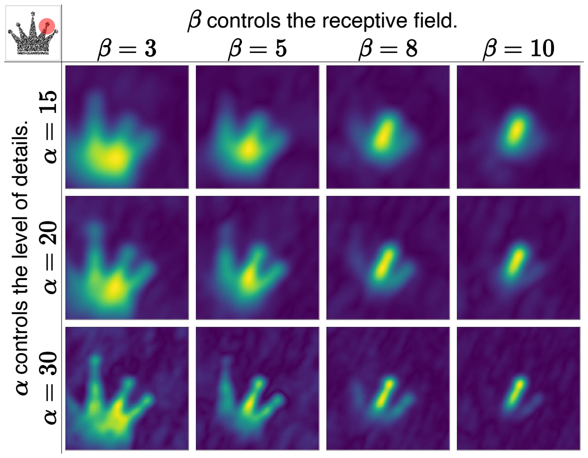
where all entries in $\mathbf{B}\in\mathbb{R}^{3\times p}$ follow $\mathcal{N}(0,\beta^2)$. Such approximation is, again, guaranteed by Lemma 1. With such approximation, Eqn. (3.3) can be rewritten as

$$\begin{aligned}
\mathbf{G}_{n\times d} &= [\hat{\mathbf{J}}_{n\times n}\mathcal{A}_{n\times d}] \ ./\ \mathcal{A}_{n\times d} \\
&\approx [\mathcal{B}_{n\times p}\times(\mathcal{B}^H\times\mathcal{A})_{p\times d}] \ ./\ \mathcal{A}_{n\times d}
\end{aligned}$$

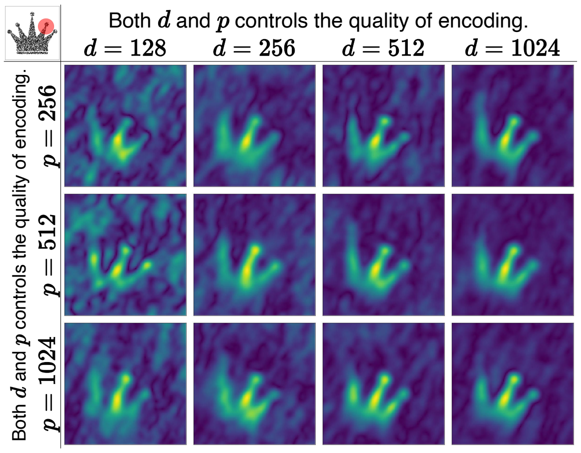
By computing $\mathcal{B}^H\times\mathcal{A}$ first, the computation cost is reduced to $\Theta(npd)$. A large point cloud size usually requires a larger p to reduce the noise, but the value p is much smaller than n. For a point cloud with size 100k, $p = 4096$ is sufficient. A large p improves the quality of the encoding, but does not increase the size of the encoding, and hence does not increase the cost of subsequent processings. Such approximation-and-factorization trick is inspired from [19], which accelerates the attention computation in transformers. This concludes the proof of Theorem 4.

Effect of α and β. We perform a qualitative analysis of the effect of the parameters α and β in Theorem 4. In short, α controls the level of details and β controls the receptive field of the local neighbor. As illustrated in the following figures, when α is larger, more high-frequency details are preserved in the encoding, and meanwhile the local geometry encodings tends to be dissimilar to each other. A larger α is usually preferred in tasks that require refined local geometry, such as normal estimation. A smaller α is usually preferred in high-level tasks, such as classification and segmentation. More quantitative analysis will be presented in Section 3.3.5.

Effect of d and p. The parameters d and p control the quality of the encoding. Higher values lead to better quality of encoding. The following figures provide the qualitative analysis.



(a) Effect of the parameters α and β in Theorem 4.



(b) Effect of the parameters d and p in Theorem 4.

Uniqueness of VecKM. VecKM cannot be established without two important properties: 1. VecKM embodies a kernel function (Lemma 1); 2. VecKM is factorizable. Importantly, the family of exponential functions is the only family of functions that has the factorizability property with respect to multiplication and division: $f(x - y) = f(x)/f(y)$. But if we use the real exponential functions, the computation is not numerically stable, and meanwhile, the inner product between the constructed vectors will not induce a kernel, i.e. Lemma 1 will not hold. Therefore, VecKM is the only choice to enable both properties, i.e. both being factorizable and inducing a kernel function. *Therefore, we conjecture that VecKM may be the only possible linear local geometry encoder.* Fortunately, we are blessed with the advantages offered by complex vectors, which provide the necessary descriptiveness and efficiency for VecKM.

3.2.4 VecKM in Point Cloud Deep Learning

VecKM can seamlessly be integrated into widely-used deep point cloud architectures, including PointNet [1], PointNet++ [14], and transformers [20; 21]. Typically, these architectures compute the dense local geometry in the first layer, often utilizing mini-PointNet or sequences of KPConvs [2]. To use VecKM in those architectures, we simply replace the dense local geometry modules with our VecKM encodings.

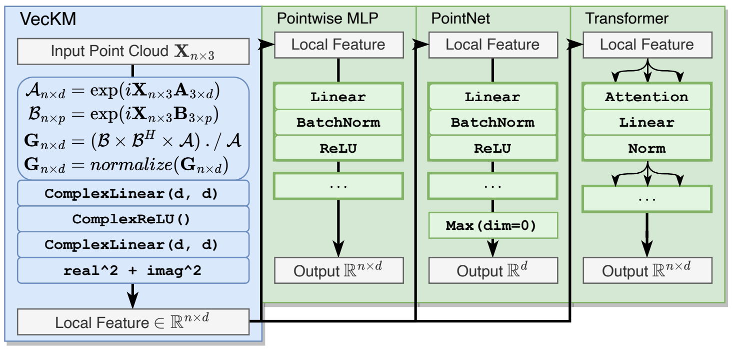

VecKM
Input Point Cloud $\mathbf{X}_{n\times 3}$
$\mathcal{A}_{n\times d} = \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d})$
$\mathcal{B}_{n\times p} = \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p})$
$\mathbf{G}_{n\times d} = (\mathcal{B} \times \mathcal{B}^H \times \mathcal{A})\ ./\ \mathcal{A}$
$\mathbf{G}_{n\times d} = normalize(\mathbf{G}_{n\times d})$
ComplexLinear(d, d)
ComplexReLU()
ComplexLinear(d, d)
real^2 + imag^2
Local Feature $\in \mathbb{R}^{n\times d}$
Pointwise MLP
Local Feature
Linear
BatchNorm
ReLU
...
Output $\mathbb{R}^{n\times d}$
PointNet
Local Feature
Linear
BatchNorm
ReLU
...
Max(dim=0)
Output $\mathbb{R}^{d}$
Transformer
Local Feature
Attention
Linear
Norm
...
Output $\mathbb{R}^{n\times d}$

Figure 3.4: VecKM can be seamlessly integrated into deep point cloud architectures, improving both accuracy and efficiency.

Note that VecKM produces complex vector outputs. To effectively utilize this in subsequent layers, we employ a series of complex linear layers and complex ReLU layers [22] to process the encodings. Finally, we cast the complex vectors into real vectors by calculating the squared norm of the complex vectors, thereby making the output compatible with standard architecture requirements. Figure 3.4 presents several examples of integrating VecKM into deep point cloud architectures, which are capable of solving many tasks involving point cloud inputs.

3.3 Experiments

3.3.1 Normal Estimation on PCPNet Dataset

We compare our VecKM against other local geometry encoders in four dimensions: accuracy, computational cost, memory cost, and robustness to noise. We select local point cloud normal estimation as our evaluation task because of its inherent challenges. This task requires the geometry encoders to adequately understand the local geometry. Moreover, it presents significant challenges in terms of memory and time complexity, given the large number of points in the input and the large number of neighboring points that need to be considered. As to be shown, **VecKM outperforms other encoders in all four dimensions by large margins.**

Dataset and Metrics. We use PCPNet [23] as the evaluation dataset. PCPNet includes 8 shapes in the training set and 19 shapes in the test set. Each shape is sampled with 100,000 points and their ground-truth normals are derived from the original meshes. PCPNet provides two types of data corruption for testing: (1) point perturbations: adding Gaussian noise to the point coordinates. (2) point density variation: resampling the point cloud under two scenarios, where *gradient* simulates the effects of varying distances from a sensor and *strips* simulates the occlusion effect. We use the root mean squared angle error (RMSE) in degrees as the evaluation metrics.

Compared Encoders. We compare our VecKM against several widely-used local geometry encoders: PointNet [1], KPConv [2] and DGCNN [24]. **PointNet**. The input point cloud is first grouped into the shape of $(n, K, 3)$ and transformed into the shape of (n, K, d) by multi-layer perceptrons. Finally, a maxpooling operation shapes the data into (n, d). K is the number of neighboring points, which we attempt different values. **KPConv**. KPConv convolutes the local neighbors through a set of kernel points and transforms the convoluted features through a fully-connected layer. KPConv has a tunable parameter: the number of kernel points, which

Table 3.1: Normal estimation RMSE on the PCPNet dataset.

	Perturbations				Density Variation		Average
	None	Low	Med	High	Gradient	Stripe	
KPConv, #kp=16	22.68	23.09	25.21	29.05	34.40	25.61	26.67
KPConv, #kp=32	22.74	22.21	24.08	28.25	32.24	24.94	25.74
KPConv, #kp=64	22.09	22.12	23.90	28.45	28.60	24.05	24.86
DGCNN, #nbr=32	24.08	24.04	25.19	28.24	27.12	27.55	26.03
DGCNN, #nbr=64	23.21	25.34	25.66	26.01	28.86	28.20	26.21
DGCNN, #nbr=128	18.46	18.71	20.38	25.62	23.01	21.29	21.24
PointNet, #nbr=300	14.98	16.30	20.19	26.83	23.68	19.00	20.17
PointNet, #nbr=500	16.10	16.54	21.38	26.93	26.06	18.89	20.99
PointNet, #nbr=700	15.59	16.25	20.99	26.21	24.66	17.87	20.27
VecKM (Ours)	**13.59**	**13.99**	**18.04**	**22.21**	**18.98**	**17.20**	**17.34**

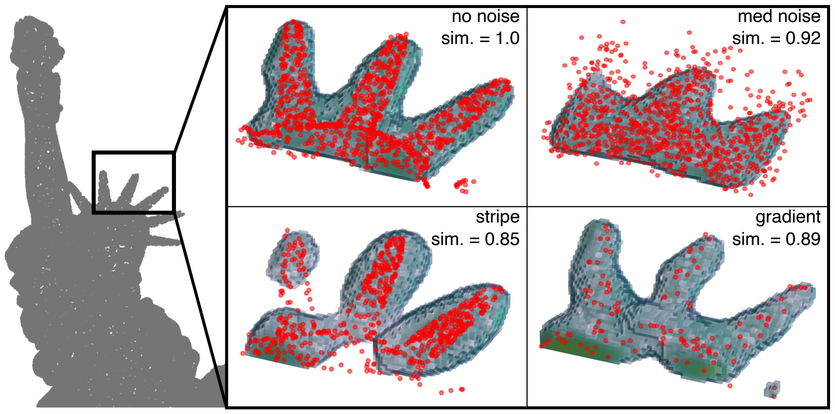


Figure 3.5: VecKM's robustness to data corruptions. VecKM can reconstruct the local shape under corrupted inputs. The VecKM encodings remain highly similar under data corruptions.

we attempt different values. **DGCNN** models the neighboring points as dynamic graphs and performs edge convolution to aggregate the local feature. We adopt the architecture in the original paper, which consists of five layers of edge convolution. DGCNN has a tunable parameter: the number of neighbors being convoluted, which we attempt different values. **VecKM** (Ours): We adopt a multi-scale of $\alpha = 60$ and $\beta = [10, 20]$. Since the size of the point cloud is large, we implement VecKM by Eqn. (3.2). We set d as 256 and p as 4096. We ensure the number of neighboring points considered by each encoder to be within 500$\sim$1000, which is sufficient to estimate the local normals. After encoding the local geometry, three layers of neural network are applied to predict the normals.

Training Details. Each model is trained with a batch size of 200 for a total of 200 epochs. We use the Adam optimizer, setting the learning rate at 10^{-3}. For data augmentation, Gaussian noise is added to the input point cloud. The input point cloud and their normals are randomly rotated.

As shown in Table 3.1, **VecKM achieves $>$ 16% lower errors than all the compared encoders and performs the best under all data corruption settings.** This reveals that VecKM effectively captures the local geometry and is more robust to input perturbation and density variation. The effectiveness of VecKM can be attributed to its reconstructive and isometric properties, and its noise robustness is derived from the robustness inherent in the kernel mixture. Figure 3.5 visualizes the explanation, which shows that even under corruptions, VecKM can still reconstruct local shapes and the VecKM encodings are consistent. In the case of the *stripe* corruption setting, while the reconstruction may appear less accurate, the downstream neural network compensates for this discrepancy. This is evidenced by the relatively stable RMSE of the *stripe* setting in Table 3.1, indicating that the overall impact on performance is not substantial.

As shown in Figure 3.6, **VecKM is $>$ 100x faster than all the compared encoders and is scalable to large point cloud inputs.** Even when the input size is as large as 100k, VecKM only takes 150 ms to run. For memory cost, PointNet and DGCNN easily incur memory outrage when the neighbor size K is large because they require an intermediate step of (n, K, d) to compute the encoding. KPConv can be memory efficient through careful parallel programming, but existing implementations are not scalable to the settings we experiment with. VecKM, however, thanks to its unique factorizable property, only costs less than 8GB memory even with pure PyTorch implementation.

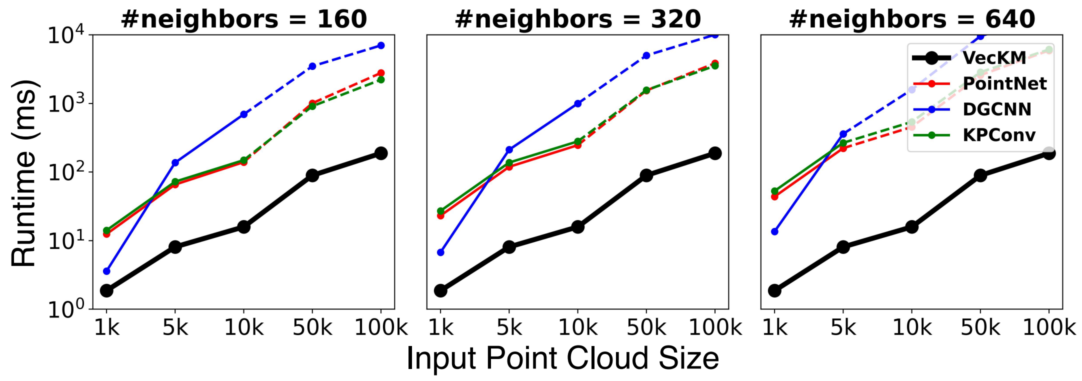


Figure 3.6: Runtime of local geometry encoders under different input point cloud size and neighbor size. All models are tested on an RTXA-5000 with 24 GB memory. Dash lines mean the memory is not sufficient to process all the points in one batch and has to process the points batches by batches.

3.3.2 Classification on ModelNet40 Dataset

We evaluate our VecKM on 3D object classification using the ModelNet40 dataset [25]. We compare classification accuracy and inference time with the baselines.

Training Details. We use the same training setting for all the methods. We use the official split with 9,843 objects for training and 2,468 for testing. Each point cloud is uniformly sampled to 1,024 points. During training, random translation in $[-0.2, 0.2]$, and random scaling in $[0.67, 1.50]$ are applied. We set the batch size to 32 and train the models for 250 epochs. We use the Adam optimizer, setting the initial learning rate as 0.001, with a cosine annealing scheduler. All models are trained and tested on an RTXA-5000.

Baselines. For our experiments, we select three widely-used point cloud architectures: PointNet [1], PointNet++ [14] and the Point Cloud Transformer (PCT) [20]. We integrate VecKM encoding into these architectures as outlined in Sec. 3.2.4, which involves adding or replacing the original local geometry encoding modules with VecKM with $\alpha = 30$, $\beta = 6$ and $d = 256$. Since the size of the point cloud is small, we implement VecKM by Eqn. (3.3). We also compare VecKM-based architectures with the another light-weight network PointMLP [11].

Specifically, for PointNet, since it does not have a local geometry encoding module, we *add* the

Table 3.2: Classification performance on the ModelNet40 dataset. VecKM $\rightarrow$ PN means *adding* VecKM as a preprocessing module to PointNet, so the runtime is expected to be longer than the PointNet baseline. VecKM $\leftrightharpoons$ PN++/PCT means *replacing* the original dense local geometry encoding in the original architectures with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance Accuracy	Avg. Class Accuracy	Inference Time (ms) (1 batch)	# parameters
PointMLP	93.2%	90.1%	325.85	13.2M
PointNet	90.8%	87.1%	3.04	1.61M
VecKM $\rightarrow$ PN	92.9%	89.7%	14.32	9.06M
Difference	$\uparrow$ 2.1%	$\uparrow$ 2.6%	not comparable	+7.61M
PointNet++	92.7%	89.4%	117.13	1.48M
VecKM $\leftrightharpoons$ PN++	93.0%	89.7%	65.78	3.94M
Difference	$\uparrow$ 0.3%	$\uparrow$ 0.3%	78% faster	+2.46M
PCT	92.9%	89.8%	149.72	2.88M
VecKM $\leftrightharpoons$ PCT	93.1%	90.6%	21.44	5.07M
Difference	$\uparrow$ 0.2%	$\uparrow$ 0.8%	5.98x faster	+2.19M

VecKM module before the PointNet, which means the PointNet receives the geometry encoding as input instead of the raw point coordinates. Since we *add* (denoted by $\rightarrow$) VecKM as an additional module, the runtime is going to be longer. For PointNet++, we *replace* (denoted by $\leftrightharpoons$) the first set abstraction layer with our VecKM encoding and leave the rest unchanged. For PCT, we *replace* the initial input embedding module with the VecKM while retaining the transformer modules. For PointNet++ and PCT, since we *replace* the dense local geometry encoding module with the more efficient VecKM, the runtime is expected to decrease.

As demonstrated in Table 3.2, **architectures based on VecKM consistently outperform their baseline counterparts in accuracy while also benefiting from significantly reduced runtime.** When VecKM is integrated with PointNet++ and PCT, not only is performance enhanced, but the speed of operation is also faster compared to the baselines. When comparing VecKM $\rightarrow$ PN against PointNet, there is a notable improvement in accuracy by 2.1% and 2.6%, with only a minimal increase in runtime. This is significant since the VecKM $\rightarrow$ PN architecture exhibits superior performance compared to both PointNet++ and PCT, and meanwhile operating 7.18x and 9.5x faster, respectively. Compared with PointMLP, VecKM-based architectures are even more efficient, achieving on-par accuracies.

3.3.3 Part Segmentation on ShapeNet Dataset

We evaluate our VecKM on 3D object part segmentation. Our experiment utilizes the ShapeNet [26] dataset. Similar to the classification experiment, we compare the IoU and inference time with PointNet, PointNet++, PCT, and PointMLP. The baselines and their VecKM counter-parts are obtained like the classification experiment in Section 3.3.2. The parameters of VecKM are selected as $\alpha = 30, \beta = 9$ and $d = 256$. Since the size of the point cloud is small, we implement VecKM by Eqn. (3.3).

Training Details. We use the same training setting for all the methods. We use the official split with 14,006 3D models for training and 2,874 for testing. Each point cloud is uniformly sampled to 2,048 points. During training, random translation in $[-0.2, 0.2]$, and random scaling in $[0.67, 1.50]$ are applied. We set the batch size to 16 and train the model for 250 epochs. We use the Adam optimizer, setting the initial learning rate as 0.001, with a cosine annealing scheduler.

Table 3.3: Part segmentation performance on the ShapeNet dataset. Similar to the classification, $\rightarrow$ means adding VecKM as a preprocessing module, so the runtime is expected to be longer. $\leftrightharpoons$ means replacing the dense local geometry encoding module with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance mIoU	Avg. Class mIoU	Inference Time (ms) (1 batch)	# parameters
PointMLP	85.1%	82.1%	240.39	16.76M
PointNet	83.1%	77.6%	15.1	8.34M
VecKM $\rightarrow$ PN	84.9%	81.8%	40.8	1.29M
Difference	$\uparrow$ 1.8%	$\uparrow$ 4.2%	not comparable	+7.05M
PointNet++	85.0%	81.9%	130.8	1.41M
VecKM $\leftrightharpoons$ PN++	85.3%	82.0%	65.9	1.50M
Difference	$\uparrow$ 0.3%	$\uparrow$ 0.1%	98% faster	+0.09M
PCT	85.7%	82.6%	145.2	1.63M
VecKM $\leftrightharpoons$ PCT	85.8%	82.6%	46.6	1.71M
Difference	$\uparrow$ 0.1%	0.0%	2.11x faster	+0.08M

As demonstrated in Table 3.3, similar to the classfication experiment, architectures based on VecKM consistently outperform their baseline counter-parts in accuracy while also benefiting from significantly reduced runtime.

3.3.4 Semantic Segmentation on S3DIS Dataset

We evaluate our VecKM on 3D semantic segmentation. We use the S3DIS dataset [27], which is an indoor scene dataset. It contains 6 areas and 271 rooms. Each point in this dataset is classified into one of 13 categories. Each scene contains around 10,000∼100,000 points. We use the same training setting as [21].

Baselines. We select PointNet++ and Point Transformer [21] as the baselines. For PointNet++, in its first layer, PointNet++ first downsamples the point cloud by 1/4 and for each sampled point, 32 neighboring points are sampled and transformed by a PointNet. The VecKM $\rightarrow$ PN++ counter-part is obtained by *adding* the dense local geometry encoder before the first layer. Consequently, the PointNet in the first layer will transform the local geometry encoding instead of the raw 3d coordinates. Because of the downsampling operation in PointNet++, its inference time is much shorter. Therefore, PN++ and VecKM $\rightarrow$ PN++ are not comparable in terms of inference time. For Point Transformer, its first layer is a dense local geometry encoder with PointNet. We *replace* the dense local geometry encoder with our VecKM encoding to obtain the PT $\leftrightharpoons$ VecKM architecture. In both architectures, since the size of the point cloud is large, we implement VecKM by Eqn. (3.2). We set $\alpha = 30, \beta = 9, d = 256, p = 2048$, and we use a sequence of two complex linear layers to transform the local geometry encoding from $\mathbb{C}^{256}$ to $\mathbb{C}^{64}$.

As shown in Table 3.4, **VecKM improves PointNet++ baseline significantly**. This is because the downsampling of the point cloud induces information loss in the PointNet++ baseline, while the dense VecKM encoding effectively bridges the gap. On the other hand, **VecKM improves the inference speed of point transformer**, which is expected given the efficiency of VecKM especially on large point cloud input. Regarding why VecKM $\leftrightharpoons$ PT does not yield better accuracy, it is possibly because the heavy-weight point transformer architecture already adequately reasons on the geometry. Unlike PointNet++, the local geometry encoding is not a bottleneck for point transformer. Since the subsequent processing costs the majority of the running time, the acceleration is not as significant as the previous experiments.

Table 3.4: Semantic segmentation performance on the S3DIS dataset. Similar to the classification experiment, $\rightarrow$ means adding VecKM as a preprocessing module. Since PointNet++ downsamples the point cloud at the first layer while VecKM $\rightarrow$ PN++ does not, their inference time is not comparable. $\leftrightharpoons$ means replacing the original dense local geometry encoding module with VecKM. Since VecKM is more efficient, the runtime is reduced.

	Instance mIoU	Avg. Class mIoU	Overall Accuracy	Inference Time (ms) (per scene)	#parameters
PointNet++	64.05	71.52	87.92	96	0.968M
VecKM $\rightarrow$ PN++	67.48	73.53	89.33	391	1.11M
Difference	$\uparrow$ 3.43	$\uparrow$ 2.01	$\uparrow$ 1.41	not comparable	+0.142M
Point Transformer	69.29	75.66	90.36	559	7.77M
VecKM $\leftrightharpoons$ PT	69.53	75.84	90.39	447	7.93M
Difference	$\uparrow$ 0.24	$\uparrow$ 0.18	$\uparrow$ 0.03	20% faster	+0.16M

3.3.5 Ablation Studies

In Section 3.2.3, we qualitatively analyze the effect of the parameters α and β in Theorem 4. In this section, we quantitatively analyze the effect of the parameters in the context of the ModelNet40 classification experiment, with the VecKM $\rightarrow$ PN architecture. For α selection, when the input point cloud is normalized within a unit ball, setting α in the range of $(20, 35)$ yields good performance. As shown in Table 3.5, appropriate selections of α and β are important to yield a good performance on the downstream tasks.

We study how many fully-connected layers are needed for transforming the VecKM encoding, in the context of normal estimation tasks. As shown in Figure 3.7, two layers are sufficient for stably satisfactory performance, highlighting the inherent descriptiveness of VecKM encoding.

Notice that the selection of the α parameter varies across different tasks. In classification, where refined local geometry is less critical, a smaller α is used to abstract away finer details. For normal estimation tasks, where accurate local shape representation is crucial, a larger α is employed to retain essential details. These findings demonstrate VecKM's adaptability in meeting the diverse requirements of various tasks, adjusting to the specific level of detail needed.

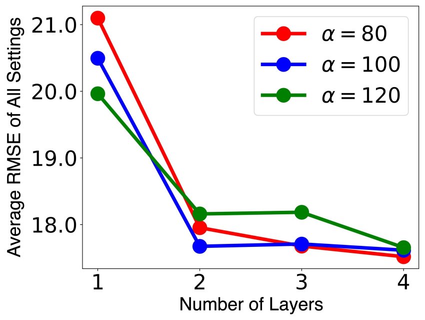


Figure 3.7: Average RMSE of normal estimation trained with different numbers of layers.

Table 3.5: Ablation study on the selection of the parameters α and β in Theorem 4, in the context of ModelNet40 classification experiment. Numbers greater than 92.5% are bolded.

	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 35$
$\beta = 4$	91.73%	91.94%	91.73%	91.77%
$\beta = 6$	**92.59%**	92.14%	**92.87%**	**92.50%**
$\beta = 9$	92.18%	**92.71%**	**92.95%**	**92.50%**
$\beta = 12$	92.10%	**92.54%**	**92.59%**	92.38%

3.4 Conclusion

VecKM, our novel local point cloud encoder, stands out for its efficiency and noise robustness. VecKM vectorizes a kernel mixture associated with the local point cloud, providing a solid theoretical foundation for its descriptiveness and robustness. Thanks to its special formulation, VecKM is the only existing local geometry encoder that costs linear time and space. Through extensive experiments, VecKM has demonstrated significant improvements in speed and accuracy across a variety of point cloud processing tasks. VecKM has many potential applications due to its notable features. Its efficiency facilitates faster inference, ideal for time-critical tasks like event data processing.

4 Theories & Algorithms II (Robustness): Decodable and Sample Invariant Continuous Object Encoder

This chapter introduces HDFE (hyper-dimensional function encoding), which focuses on the robustness to density variation in point cloud embedding.

4.1 Introduction

Continuous objects are objects that can be sampled with arbitrary distribution and density. Examples include point clouds [28], event-based vision data [29], and sparse meteorological data [30]. A crucial characteristic of continuous objects, which poses a challenge for learning, is that their sample distribution and size varies between training and test sets. For example, point cloud data in the testing phase may be sparser or denser than that in the training phase. A framework that handles this inconsistency is essential for continuous object learning.

When designing the framework, four properties are desirable: (1) *Sample distribution invariance*: the framework is not affected by the distribution from which the samples are collected. (2) *Sample size invariance*: the framework is not affected by the number of samples. (3) *Explicit representation*: the framework generates outputs with fixed dimensions, such as fixed-length vectors. (4) *Decodability*: the continuous object can be reconstructed at arbitrary resolution from the representation.

Sample invariance (properties 1 and 2) ensures that differently sampled instances of the same continuous objects are treated consistently, thereby eliminating the ambiguity caused by variations in sampling. An explicit representation (property 3) enables a neural network to receive continuous objects as inputs, by consuming the encodings of the objects. Decodability (property 4) enables a neural network to predict a continuous object, by first predicting the representation and then decoding it back to the continuous object. Fig. 4.1 illustrates the properties and their motivations.

However, existing methodologies, which we divide into three categories, are limited when incorporating the four properties. (1) *Discrete framework.* The methods discretize continuous objects and process them with neural networks. For example, [31] uses a 3D-CNN to process voxelized point clouds, [32] uses an RNN to predict particle trajectories. These methods are not sample invariant – the spatial and temporal resolution must be consistent across the training and testing phases. (2) *Mesh-grid-based framework.* They operate on continuous objects defined on mesh grids and achieve discretization invariance (the framework is not affected by the resolution of the grids). Examples include the Fourier transform [33] and the neural operator [34]. But they do not apply to sparse data like point clouds. (3) *Sparse framework.* They operate on sparse samples drawn from the continuous object. Kernel methods [35] work for non-linear regression, classification, etc. But they do not provide an explicit representation of the function. PointNet [1] receives sparse point cloud input and produces an explicit representation, but the representation is not decodable. In addition, all the frameworks require extra training of the encoder, which is undesired in some scarce data scenarios.

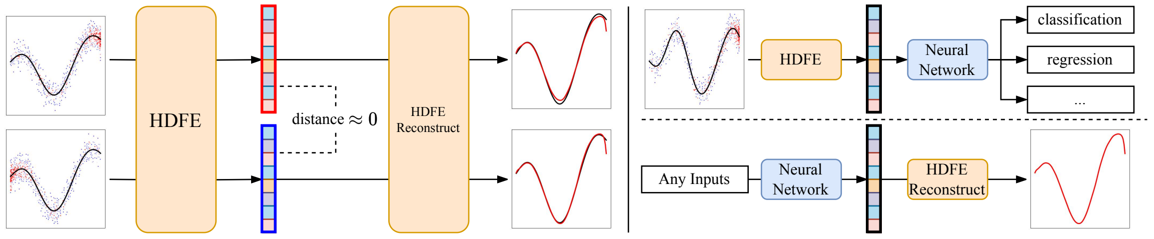


Figure 4.1: **Left**: HDFE encodes continuous objects into fixed-length vectors without any training. The encoding is not affected by the distribution and size with which the object is sampled. The encoding can be decoded to reconstruct the continuous object. **Right**: Applications of HDFE. HDFE can be used to perform machine learning tasks (e.g. classification, regression) on continuous objects. HDFE also enables neural networks to regress continuous objects by predicting their encodings.

Currently, only the vector function architecture (VFA) [36] can encode an explicit function into a vector through sparse samples, while preserving all four properties. However, VFA is limited by its strong assumption of the functional form. VFA requires the input function to conform to $f(x) = \sum_k \alpha_k \cdot K(x, x_k)$, where $K : X \times X \to \mathbb{R}$ is a kernel defined on X. If the input function does not conform to the form, VFA cannot apply or induces large errors. In practice, such requirement is rarely satisfied. For example, $f(x)$ cannot even approximate a constant function $g(x) = 1$: to approximate the constant function, the kernel K must be constant. But with the constant kernel, $f(x)$ cannot approximate other non-constant functions. Such limitation greatly hinders the application of VFA.

We propose hyper-dimensional function encoding (HDFE), which does not assume any explicit form of input functions but only requires Lipschitz continuity. Consequently, HDFE can encode *a much larger class of functions*, while holding all four properties without any training. Thanks to the relaxation, HDFE can be applied to multiple real-world applications that VFA fails, which will be elaborated on in the experiment section. HDFE maps the samples to a high-dimensional space and computes weighted averages of the samples in that space to capture collective information of all the samples. A challenge in HDFE design is maintaining sample invariance, for which we propose a novel iterative refinement process to decide the weight of each sample. The contributions of our paper can be summarized as follows:

- We present HDFE, an encoder for continuous objects without any training that exhibits sample invariance, decodability, and distance-preservation. To the best of our knowledge, HDFE is the only algorithm that can encode Lipschitz functions while upholding all the four properties.
- We provide extensive theoretical foundation for HDFE. We prove that HDFE is equipped with all the desirable properties. We also verify them with empirical experiments.
- We evaluate HDFE on mesh-grid data and sparse data. In the mesh-grid data domain, HDFE achieves competitive performance as the specialized state-of-the-art (SOTA) in function-to-function mapping tasks. In the sparse data domain, replacing PointNet with HDFE leads to average error decreases of 12% and 15% in two benchmarks, and incorporating HDFE into the PointNet-based SOTA architecture leads to average error decreases of 2.5% and 1.7%.

4.2 Problem Definition and Methodology

Let F be the family of c-Lipschitz continuous functions defined on a compact domain X with a compact range Y. In other words, $\forall f \in F$, $f : X \to Y$ and $d_Y\big(f(x_1), f(x_2)\big) \leq c \cdot d_X\big(x_1, x_2\big)$, where (X, d_X) and (Y, d_Y) are metric spaces, and c is the Lipschitz constant. Our goal is to find a representation algorithm that can encode a function $f \in F$ into a vector representation $\mathbf{F} \in \mathbb{C}^N$. To construct it, we will feed samples of the function mapping $\big\{\big(x_i, f(x_i)\big)\big\}$ to the representation algorithm, which will generate the vector representation based on these samples.

We require the function representation to satisfy the following: (1) Sample distribution invariance: the function representation is "not affected" by the distribution from which the samples are collected. (2) Sample size invariance: the function representation is "not affected" by the number of samples. (3) Fixed-length representation: all functions are represented by fixed-length vectors. (4) Decodability: as new inputs query the function representation, it can reconstruct the function values.

To better formalize the heuristic expression of "not affected" in Properties 1 and 2, we introduce the definition of asymptotic sample invariance to formulate an exact mathematical expression:

Definition 5 (Asymptotic Sample Invariance)**.** Let $f : X \to Y$ be the function to be encoded, $p : X \to (0, 1)$ be a probability density function (pdf) on X, $\{x_i\}_{i=1}^n \sim p(X)$ be n independent samples of X. Let $\mathbf{F}_n$ be the representation computed from the samples $\{x_i, f(x_i)\}_{i=1}^n$, asymptotic sample invariance implies $\mathbf{F}_n$ converges to a limit $\mathbf{F}_\infty$ independent of the pdf p.

In this definition, sample size invariance is reflected because the distance between $\mathbf{F}_m$ and $\mathbf{F}_n$ can be arbitrarily small as m, n become large. Sample distribution invariance is reflected because the limit $\mathbf{F}_\infty$ does not depend on the pdf p, as long as p is supported on the whole input space X.

With the problem definition above, we present our hyper-dimensional function encoding (HDFE) approach. Sec. 4.2.1 introduces how HDFE encodes explicit functions. Sec. 4.2.2 generalizes HDFE to implicit function encoding. Sec. 4.2.3 realizes HDFE for vector-valued function encoding. Finally, Sec. 4.2.4 establishes the theorems that HDFE is asymptotic sample invariant and distance-preserving. Throughout the section, we assume the functions are c-Lipschitz continuous. The assumption will also be explained in Section 4.2.4.

4.2.1 Explicit Function Encoding

Encoding HDFE is inspired by the methodology of hyper-dimensional computing (HDC) [37], where one encodes an indefinite number of data points into a fixed-length vector. The common practice is to first map the data points to a high-dimensional space and then average the data point representations in that space. The resulting superposed vector can represent the distribution of the data. Following the idea, we represent an explicit function as the superposition of its samples:

$$\mathbf{F} = \sum_i w_i \cdot E(x_i, y_i) \tag{4.1}$$

where E maps function samples to a high-dimensional space $\mathbb{C}^N$. The question remains (a) how to design the mapping to make the vector decodable; (b) how to determine the weight of each sample w_i so that the representation is sample invariant. We will answer question (a) first and leave question (b) to the iterative refinement section.

Regarding the selection of $E(x, y)$, a counter-example is a linear mapping, where the average of the function samples in the high-dimensional space will degenerate to the average of the function samples, which does not represent the function. To avoid degeneration, the encodings of the samples should not interfere with each other if they are far from each other. Specifically, if $d_X(x_1, x_2)$ is larger than a threshold ϵ_0, their function values $f(x_1), f(x_2)$ may be significantly different. In this case, we want $E(x_1, y_1)$ to be orthogonal to $E(x_2, y_2)$ to avoid interference. On the other hand, if $d_X(x_1, x_2)$ is smaller than the threshold ϵ_0, by the Lipschitz continuity, the distance between their function values $d_Y(f(x_1), f(x_2))$ is bounded by $c\epsilon_0$. In this case, we want $E(x_1, y_1)$ to be similar to $E(x_2, y_2)$. We call the tunable threshold ϵ_0 the *receptive field* of HDFE, which will be discussed in Sec. 4.2.4. Denoting the similarity between vectors as $\langle\cdot,\cdot\rangle$, the requirement can be formulated as:

$$\langle E(x,y), E(x',y')\rangle \begin{cases} \approx 1 & d_X(x,x') < \epsilon_0 \\ \text{decays to 0 quickly} & d_X(x,x') > \epsilon_0 \end{cases} \tag{4.2}$$

In addition to avoiding degeneration, we also require the encoding to be decodable. This can be achieved by factorizing $E(x, y)$ into two components: We first map x_i and y_i to the high-dimensional space $\mathbb{C}^N$ through two different mappings E_X and E_Y. To ensure Eqn. (4.2) is satisfied, we require $\langle E_X(x), E_X(x')\rangle \approx 1$ when $d_X(x, x') < \epsilon_0$ and that it decays to 0 otherwise. The property of E_Y will be mentioned later in the discussion of decoding. Finally, we compute the joint embedding of x_i and y_i through a *binding* operation $\otimes$: $E(x_i, y_i) = E_X(x_i) \otimes E_Y(y_i)$.

We will show that the representation is decodable if the binding operation satisfies these properties:

1. commutative: $x \otimes y = y \otimes x$
2. distributive: $x \otimes (y + z) = x \otimes y + x \otimes z$
3. similarity preserving: $\langle x \otimes y, x \otimes z\rangle = \langle y, z\rangle$.
4. invertible: there exists an associative, distributive, similarity preserving operator that undoes the binding, called *unbinding* $\oslash$, satisfying $(x \otimes y) \oslash z = (x \oslash z) \otimes y$ and $(x \otimes y) \oslash x = y$.

The binding and unbinding operations can be analogous to multiplication and division, where the difference is that binding and unbinding operate on vectors and are similarity preserving.

Decoding With the properties of the two operations, the decoding of the function representation can be performed by a similarity search. Given the function representation $\mathbf{F} \in \mathbb{C}^N$, and a query input $x_0 \in X$, the estimated function value $\hat{y_0}$ is computed by:

$$\hat{y_0} = \text{argmax}_{y\in Y} \langle \mathbf{F} \oslash E_X(x_0), E_Y(y)\rangle \tag{4.3}$$

The distributive property allows the unbinding operation to be performed sample-by-sample. The invertible property allows the unbinding operation to recover the encoding of the function values: $E_X(x_i) \otimes E_Y\big(f(x_i)\big) \oslash E_X(x_0) \approx E_Y\big(f(x_i)\big) \approx E_Y\big(f(x_0)\big)$ when $d_X(x_0, x_i)$ is small. The similarity preserving property ensures that $[E_X(x_i) \oslash E_X(x_0)] \otimes E_Y(f(x_i))$ produces a vector orthogonal to $E_Y(f(x_0))$ when the distance between two samples is large, resulting in a summation of noise. The following formula illustrates the idea.

$$\begin{aligned} \mathbf{F} \oslash E_X(x_0) &= \sum_i w_i \cdot \left[E_X(x_i) \otimes E_Y\big(f(x_i)\big) \oslash E_X(x_0)\right] \\ &= \underbrace{\sum_{d(x_0,x_i)<\epsilon_0} w_i \cdot E_Y\big(f(x_i)\big)}_{\approx E_Y(f(x_0))} + \underbrace{\sum_{d(x_0,x_i)>\epsilon_0} w_i \cdot [E_X(x_i) \oslash E_X(x_0)] \otimes E_Y(f(x_i))}_{\text{noise, since orthogonal to } E_Y(f(x_0))} \end{aligned}$$

After computing $\mathbf{F} \oslash E_X(x_0)$, we search for $y \in Y$ such that the cosine similarity between $E_Y(y)$ and $\mathbf{F} \oslash E_X(x_0)$ is maximized. We desire that $\frac{\partial}{\partial y}\langle E_Y(y), E_Y(y')\rangle > 0$ for all y and y' so that the optimization can be solved by gradient descent. See Supp. 4.5.1 for detailed formulation.

Since the decoding only involves measuring cosine similarity, in the last step, we normalize the function representation to achieve sample size invariance without inducing any loss:

$$\mathbf{F} = \mathit{normalize}\Big(\sum_i w_i \cdot \big[E_X(x_i) \otimes E_Y(f(x_i))\big]\Big) \tag{4.4}$$

Iterative refinement for sample distribution invariance In Eqn. (4.4), we are left to determine the weight of each sample so that the representation is sample invariant. To address this, we propose an iterative refinement process to make the encoding invariant to the sample distribution. We initialize $w_i = 1$ and compute the initial function vector. Then we compute the similarity between the function vector and the encoding of each sample. We then add the sample encoding with the lowest similarity to the function vector and repeat this process until the lowest similarity no longer increases. By doing so, the output will be *the center of the smallest ball containing all the sample encodings*. Such output is asymptotic sample invariant because the ball converges to the smallest ball containing $\cup_{x \in X}[E_X(x) \otimes E_Y(f(x))]$ as the sample size goes large, where the limit ball only depends on the function. We left the formal proof to the Supp. 4.5.2.

Algorithm 1 Iterative Refinement

```
$z_i \leftarrow E_X(x_i) \otimes E_Y(f(x_i))$ for all $i$.
$\mathbf{F} = \sum_i z_i$
while $\min_i \langle \mathbf{F}, z_i \rangle$ still increases do
    $j = \text{argmin}_i \langle \mathbf{F}, z_i \rangle$
    $\mathbf{F} \leftarrow \mathbf{F} + z_j$
end while
```

One-Shot Refinement In Algorithm 1, the motivation of the iterative refinement is to balance the weights between dense and sparse samples. By iterative refinement, we adjust the function encoding so that the sparse samples also contribute to the encoding. Such motivation can be achieved by another cheaper one-shot refinement. After obtaining the initial function encoding by averaging the sample encoding, we compute the similarity between this initial encoding and all the sample encodings. The similarity can serve as a rough estimation of the sample density at a particular point. Therefore, if we were to balance the weights between dense and sparse samples, we can simply recompute the weights by the inverse estimated density. Algorithm 2 illustrates the procedure. Although one-shot refinement is not strictly sample distribution invariant, it is a quite good approximation of the sample invariant function encoding and is very cheap to compute.

Algorithm 2 One-Shot Refinement

```
$z_i \leftarrow E_X(x_i) \otimes E_Y(f(x_i))$ for all $i$.
$\mathbf{F} = \sum_i z_i$
for i do
    $w_i = \langle \mathbf{F}, z_i \rangle$ ▷ $w_i$ is an estimation of the density at $(x_i, f(x_i))$.
    $w_i = \max(\epsilon, w_i)$ ▷ Ensure numerical stability.
    $w_i = w_i^{-1} / \sum_{j=1}^{n} w_j^{-1}$ ▷ Compute inverse density.
end for
$\mathbf{F} = \sum_i w_i \cdot z_i$
```

4.2.2 Implicit Function Encoding

Generalizing HDFE to implicit functions is fairly straightforward. Without loss of generality, we assume an implicit function is represented as $f(x) = 0$. Then it can be encoded using Eqn. (4.5), where the weights w_x are determined by the iterative refinement:

$$\mathbf{F}_{f=0} = \mathit{normalize}\Big(\sum_{x:f(x)=0} w_x \cdot E_X(x) \Big) \tag{4.5}$$

The formula can be understood as encoding an explicit function g, where $g(x) = 1$ if $f(x) = 0$ and $g(x) = 0$ if $f(x) \neq 0$. Then by choosing $E_Y(1) = 1$ and $E_Y(0) = 0$ in Eqn. (4.4), we can obtain Eqn. (4.5). The formula can be interpreted in a simple way: a continuous object can be represented as the summation of its samples in a high-dimensional space.

4.2.3 Vector-Valued Function Encoding

In the previous sections, we established a theoretical framework for encoding c-Lipschitz continuous functions. In this section, we put this framework into practice by carefully choosing appropriate input and output mappings E_X, E_Y, the binding operator $\otimes$, and the unbinding operator $\oslash$ in Eqn. (4.4). We will first state our choice and then explain the motivation behind it.

Formulation Let $(\mathbf{x}, y)$ be one of the function samples, where $\mathbf{x} \in \mathbb{R}^m$ and $y \in \mathbb{R}$, the mapping $E_X : X \to \mathbb{C}^N$, $E_Y : \mathbb{R} \to \mathbb{C}^N$ and the operations $\otimes$ and $\oslash$ are chosen as:

$$E_X(\mathbf{x}) := \exp\big(i \cdot \alpha \frac{\Phi \mathbf{x}}{m}\big) \qquad E_Y(y) := \exp\big(i\beta\Psi y\big) \tag{4.6}$$

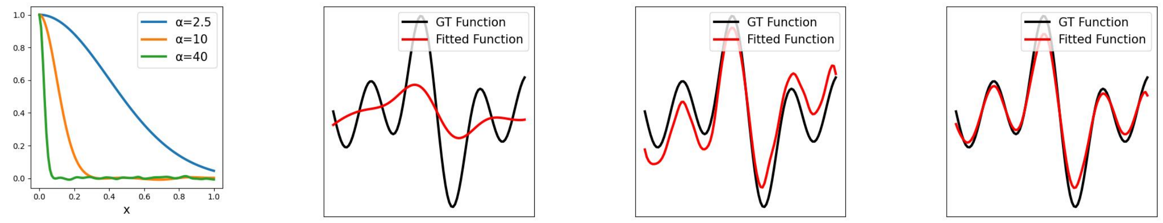
$$E_X(\mathbf{x}) \otimes E_Y(y) := \exp\big(i \cdot \alpha \frac{\Phi \mathbf{x}}{m} + i\beta\Psi y\big)$$

$$E_X(\mathbf{x}) \oslash E_Y(y) := \exp\big(i \cdot \alpha \frac{\Phi \mathbf{x}}{m} - i\beta\Psi y\big)$$

where i is the imaginary unit, $\Phi \in \mathbb{R}^{N\times m}$ and $\Psi \in \mathbb{R}^N$ are random fixed matrices where all elements are drawn from the standard normal distribution. α and β are hyper-parameters controlling the properties of the mappings.

Motivation The above way of mapping real vectors to high-dimensional spaces is modified from [38], known as fractional power encoding (FPE). We introduce the motivation for adopting this technique here.

First, the mappings are continuous, which can avoid losses when mapping samples to the embedding space. Second, the receptive field of the input mapping E_X (the ϵ_0 in Eqn. (4.2)) can be adjusted easily through manipulating α. Fig. 4.2a demonstrates how manipulating α can alter the behavior of E_X. Typically, α has a magnitude of 10 for capturing the high-frequency component of the function. Thirdly, the decodability of the output mapping E_Y can easily be achieved by selecting appropriate β values. We select β such that $\langle E_Y(0), E_Y(1)\rangle$ is equal to 0 to utilize the space $\mathbb{C}^N$ maximally while keeping the gradient of $\langle E_Y(y_1), E_Y(y_2)\rangle$ non-zero for all y_1 and y_2. Per the illustration in Fig. 4.2a, the optimal choice for β is 2.5. Finally, the binding and unbinding operators are defined as the element-wise multiplication and division of complex vectors, which satisfy the required properties.



(a) Similarity between $E_X(x)$ and $E_X(0)$.

(b) Large recept. field. Dimension = 1000.

(c) Small recept. field. Dimension = 1000.

(d) Small recept. field. Dimension = 2000.

Figure 4.2: (a): How α and β in Eqn. (4.6) affects the receptive field of HDFE. (b)-(d): Functions can be reconstructed accurately given a suitable receptive field and encoding dimension. To capture the high-frequency component of the function, a small receptive field and a high dimension are required.

4.2.4 Properties of HDFE

HDFE produces an explicit decodable representation of functions. In this section, we state a theorem on the asymptotic sample invariance, completing the claim that HDFE satisfies all four desirable properties. We study the effect of the receptive field on the behavior of HDFE. We also state that HDFE is distance-preserving and discuss the potential of scaling HDFE to high-dimensional data. We leave the proofs to Supp. 4.5.2, 4.5.3 and verify the claims with empirical experiments in Section 4.3.1. We include several empirical experiments of HDFE in Sec. 4.3.2, 4.3.3, including the cost of the iterative refinement and one-shot refinement, and the effectiveness of sample invariance in a synthetic regression problem.

Theorem 6 (Sample Invariance)**.** *HDFE is asymptotic sample invariant (**Definition** 5).*

HDFE being sample invariant ensures functions realized with different sampling schemes are treated invariantly. The proof is in Supp. 4.5.2.

Theorem 7 (Distance Preserving)**.** *Let $f, g : X \to Y$ be both c-Lipschitz continuous, then their L2-distance is preserved in the encoding. In other words, HDFE is an isometry:*

$$||f - g||_{L_2} = \int_{x \in X} \left| f(x) - g(x) \right|^2 dx \approx b - a\langle \mathbf{F}, \mathbf{G} \rangle$$

HDFE being isometric indicates that HDFE encodes functions into a organized embedding space, which can reduce the complexity of the machine learning architecture when training downstream tasks on the functions. The proof is in Supp. 4.5.3.

Effect of receptive field Fig. 4.2 shows the reconstruction results of a 1d function $f : \mathbb{R} \to \mathbb{R}$, which demonstrates that HDFE can reconstruct the original functions given a suitable receptive field and a sufficiently large embedding space. When using a large receptive field (Fig. 4.2b), the high-frequency components will be missed by HDFE. When using a small receptive field (Fig. 4.2c), the high-frequency components can be captured, but it may cause incorrect reconstruction if the dimension of the embedding space is not large enough. Fortunately, reconstruction failures can be eliminated by increasing the dimension of the embedding space (Fig. 4.2d).

4.3 Experiment

In this section, we present two applications of HDFE. Sec. 4.3.1 verifies the properties claimed in Sec. 4.2.4 with empirical experiments. Sec. 4.3.2 analyzes the computation cost of the refinements. Sec. 4.3.3 examines the efficacy of the sample invariance through a synthetic function regression problem. Sec. 4.3.4 showcases how HDFE can be leveraged for solving partial differential equations (PDE). This exemplifies how HDFE can enhance neural networks to receive function inputs and produce function outputs. In Sec. 4.3.5, we apply HDFE to predict the surface normal of point clouds. This demonstrates how HDFE can enhance neural networks to process implicit functions and extract relevant attributes.

4.3.1 Empirical Experiment of HDFE

Sample Invariance. In Fig. 4.3, we demonstrate that the function encoding produced by HDFE remains invariant of both the sample distribution and sample density. Specifically, we sample function values from three distinct input space distributions, namely left-skewed, right-skewed, and uniform distribution, each with sample sizes of either 5000 or 1000. We then calculate the similarity between the function vectors generated from these six sets of function samples. Before tuning the function vectors, the representation is influenced by the sample distributions (Fig. 4.3 Mid). However, after the tuning process, the function vector becomes immune to the sample distribution (Fig. 4.3 Right).

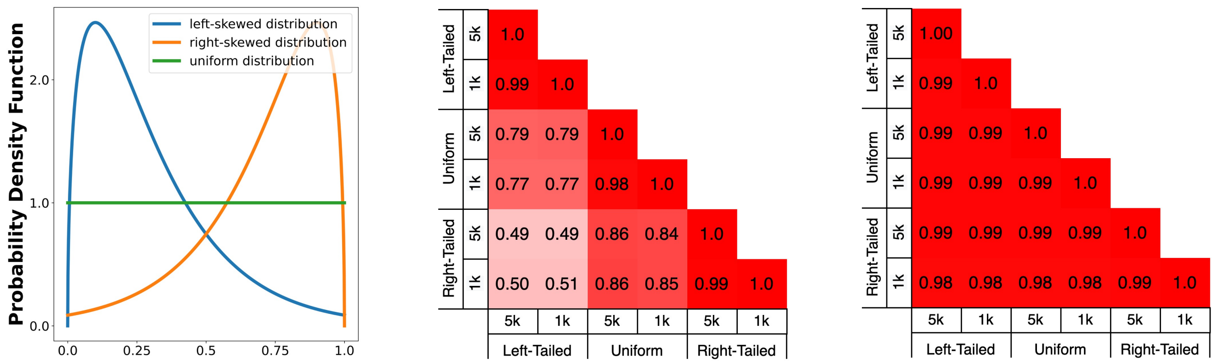


Figure 4.3: HDFE is invariant of sample distribution and sample size. **Left**: Three distributions where the function samples are drawn from. For each distribution, the sample size is either 5000 or 1000. **Mid, Right**: Similarity among the function vectors generated by the six sets of function samples, before and after the function vector tuning process, respectively.

Isometry In Fig. 4.4, we generate pairs of random functions and compute their function encodings through HDFE. We plot the L2-distance between the functions and the similarity between their encodings. We discover a strong correlation between them. This coincides the isometric property claimed in Theorem 7.

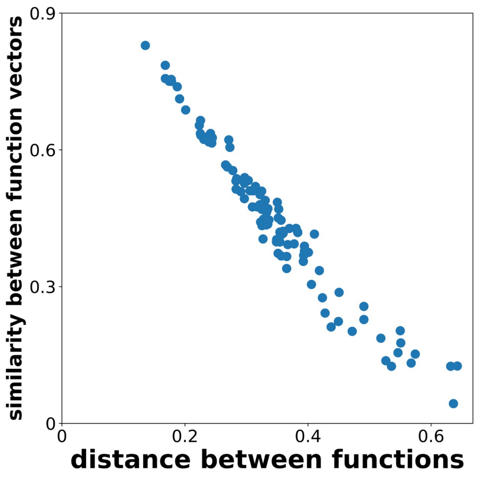


Figure 4.4: HDFE is a distance preserving transformation. The L2-distance between functions is proportional to the negative similarity between their encodings.

4.3.2 Cost of Refinements

We perform a comparison among no refinement, one-shot refinement and iterative refinement with synthetic data, where we encode the same function sampled with two different distributions and compute the similarity between the two encodings. Specifically, we generate a random function by

$$f(x) = \frac{1}{2} + \frac{1}{8}\sum_{k=1}^{4} a_k \sin(2\pi k x) \tag{4.7}$$

where $a_k \sim Uniform(0,1)$ are the parameters controlling the generation and $f(x) \in (0,1)$. We construct the encoding of the function by samples from two different sample distributions. The first distribution is computed by $x_i \sim Uniform(0,1)$ and $x_i \leftarrow x_i^2$. The second distribution is computed by $x_i \sim Uniform(0,1)$ and $x_i \leftarrow 1-x_i^2$. Consequently, the first distribution is left-tailed, and the second distribution is right-tailed. Then we compare the similarity of function encodings generated by no iterative refinement, one-shot refinement, and iterative refinement. Figure 4.5 shows the comparison, which demonstrates that one-shot refinement is a quite good approximation of the sample invariant function encoding (the similarity increases from ~ 0.5 to 0.98 after one-shot refinement).

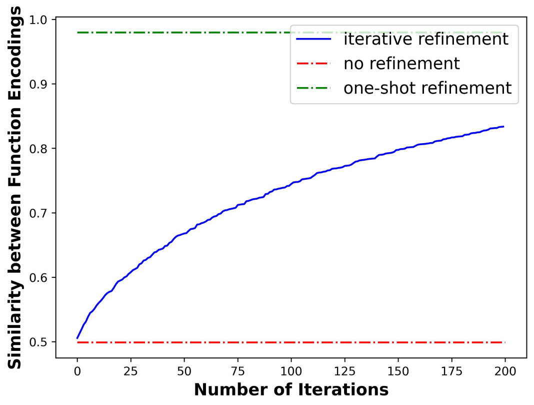


Figure 4.5: When encoding the same function under two different sample distributions, one-shot refinement can approximate the sample invariant function encoding well. It takes 75/90/1500 ms to encode 5000 samples on a CPU, and 7.5/8.0/250 ms on an NVIDIA Titan-X GPU when performing no refinement/one-shot refinement/200-step iterative refinement.

4.3.3 Effectiveness of Sample Invariance

In this section, we examine the effectiveness of HDFE's sample invariance property through an synthetic function regression problem. We first generate random functions by Eqn. (4.7), where $a_k \sim Uniform(0, 1)$ are the parameters controlling the generation. The task is to regress the coefficients $[a_1, a_2, a_3, a_4]$ from the function samples $\{x_i, f(x_i)\}$. Regarding the sample size, the number of function samples is 5000 in the training phase and 2500 in the testing phase. Regarding sample distribution, we consider two different settings:

Setting 1 (No Sample Distribution Variation): The sample distribution is consistent between training phase and testing phase. We let $x_i \sim Uniform(0, 1)$ in both training phase and testing phase.

Setting 2 (Sample Distribution Variation): The sample distribution is different between the training phase and testing phase. Specifically, in the training phase, we let $x_i \sim Uniform(0, 1)$ and $x_i \leftarrow x_i^2$. In the testing phase, we let $x_i \sim Uniform(0, 1)$ and $x_i \leftarrow 1 - x_i^2$. Consequently, the sample distribution in the training phase is left-tailed, while in the testing phase is right-tailed.

We compare our HDFE with PointNet in terms of mean squared error (MSE) and the R-squared (R^2) metrics. For HDFE, we compare the performance among no refinement, one-shot refinement, and 200-step iterative refinement. Table 4.1 shows the comparison.

In Setting 1, when there is no distribution variation, HDFE achieves significantly lower error than PointNet. This is because HDFE is capable of capturing the entire distribution of functions, while PointNet seems to struggle on that.

In Setting 2, when there is distribution variation, PointNet fails miserably, while HDFE, even without iterative refinement, already achieves fairly good estimation, and even better than the PointNet in Setting 1. In addition, the experiment also shows that both the one-shot refinement and the iterative refinement are effective techniques to improve the robustness to distribution variation.

Table 4.1: Performance of function parameters regression. PointNet fails when sample distribution varies between training and testing phases, while HDFE is robust to the sample distribution variation.

		PointNet	HDFE		
			No Refinement	One-Shot Refinement	200-Step Iter. Ref.
No Distr. Var.	MSE	0.0037	< 0.0005	< 0.0005	< 0.0005
	R^2	0.978	> 0.9975	> 0.9975	> 0.9975
Distr. Var.	MSE	0.0717	0.003	< 0.0005	0.001
	R^2	0.513	0.982	> 0.9975	0.992

4.3.4 PDE Solver

Several neural networks have been developed to solve partial differential equations (PDE), such as the Fourier neural operator [34]. In this section, we compare our approach using HDFE against the current approaches and show that we achieve on-par performance. VFA does not apply to the problem since the input and output functions do not conform to the form that VFA requires.

Architecture To solve PDEs using neural networks, we first encode the PDE and its solution into their vector embeddings using HDFE. Then, we train a multi-layer perceptron to map the embedding of the PDE to the embedding of its solution. The optimization target is the cosine similarity between the predicted embedding and the true embedding. Since the embeddings are complex vectors, we adopt a Deep Complex Network [39] as the architecture of the multi-layer perceptron. Once the model is trained, we use it to predict the embedding of the solution, which is then decoded to obtain the actual solution.

Dataset We use 1d Burgers' Equation [40] and 2d Darcy Flow [41] for evaluating our method. The error is measured by the absolute distance between the predicted solution and the ground-truth solution. The benchmark [42] has been used to evaluate neural operators widely. For the 1d Burgers' Equation, it provides 2048 PDEs and their solutions, sampled at a 1d mesh grid at a resolution of 8192. For the 2d Darcy Flow, it provides 2048 PDEs and their solutions, sampled at a 2d mesh grid at a resolution of 241 × 241.

Baselines We evaluate our HDFE against other neural network PDE-solving methods. These include: PCANN [43]; MGKN: Multipole Graph Neural Operator [42]; FNO: Fourier Neural Operator [34].

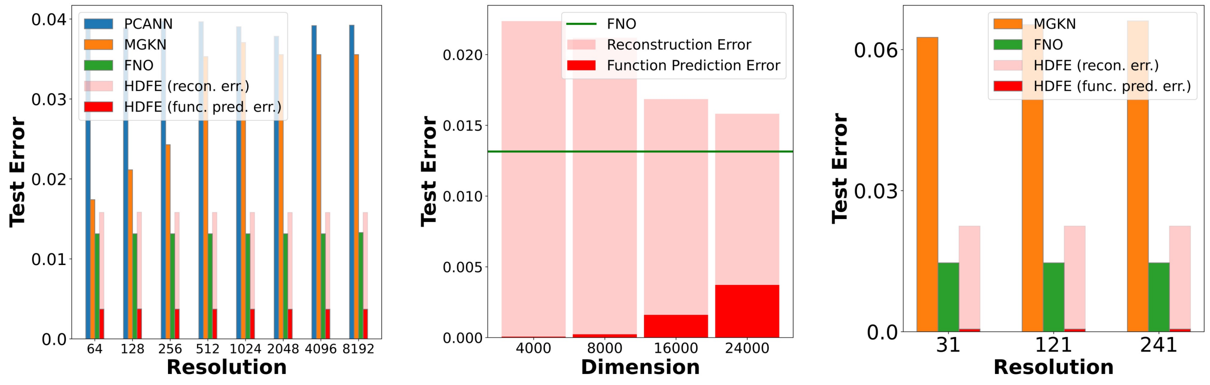


Figure 4.6: HDFE solves a PDE by predicting the encoding of its solution and then reconstructing at points, so the error consists of a function encoding prediction error and a reconstruction error. **Left**: Prediction error of different methods under different testing resolutions, evaluated on the 1d Burgers' equation. **Mid**: The reconstruction error (in HDFE) dominates the function encoding prediction error, while the reconstruction error can be reduced by increasing the dimensionality of the embedding. **Right**: Prediction error of different methods evaluated on 2d Darcy Flow.

When decoding is required, **our approach achieves $\sim$ 55% lower prediction error than MGKN and PCANN and competitive performance to FNO**. The error of HDFE consists of two components. The first is the error arising from predicting the solution embedding, and the second is the reconstruction error arising when decoding the solution from the predicted embedding. In contrast, FNO directly predicts the solution, and hence, does not suffer from reconstruction error. If we consider both errors, HDFE achieves comparable performance to FNO. Fig. 4.6 shows the comparison.

On the other hand, *when decoding is not required*, **our approach achieves lower error than FNO**. Such scenarios happen frequently when we use functions only as input, for example, the local geometry prediction problem in Experiment 4.3.5. Despite the presence of reconstruction error, **the reconstruction error can be reduced by increasing the embedding dimension**, as shown in Figure 4.6 (Mid). Increasing the embedding dimension may slightly increase the function prediction error, possibly because the network is not adequately trained due to limited training data and some overfitting. We conjecture that this prediction error can be reduced with more training data.

In addition to comparable performance, **HDFE overcomes two limitations of FNO**. First, HDFE provides an explicit function representation, resolving the restriction of FNO, which only models the mappings between functions without extracting attributes from them. Second, HDFE not only works for grid-sampled functions but also for sparsely sampled functions.

4.3.5 Unoriented Surface Normal Estimation

Next, we apply HDFE to extract attributes of functions, a setting where neither neural operators nor VFA applies, because neural operators do not consume sparse samples and VFA does not encode implicit functions. We predict the unoriented surface normal from 3d point cloud input.

Baselines We compare our HDFE with two baselines. In the first baseline, we compare the vanilla HDFE with the PCPNet [23], which is a vanilla PointNet [1] architecture. We replace the PointNet with our HDFE appended with a deep complex network [39]. In the second baseline, we incorporate HDFE into HSurf-Net [44], which is the state-of-the-art PointNet-based normal estimator. In both settings, we compare the effect of data augmentation in the HDFE module, where we add noise to the weight of each sample when generating the patch encoding by HDFE.

Dataset and metrics We use the root mean squared angle error (RMSE) as the metrics, evaluated on the PCPNet [23] and FamousShape [45] datasets. We compare the robustness for two types of data corruption: (1) point density: sampling subsets of points with two regimes, where *gradient* simulates the effects of distance from the sensor, and *strips* simulates local occlusions. (2) point perturbations: adding Gaussian noise to the point coordinates. Table 4.2 reports normal angle RMSE comparison with the baselines on PCPNet and FamousShape.

Table 4.2: Unoriented normal RMSE results on datasets PCPNet and FamousShape. Replacing from PointNet to HDFE improves performance. Integrating HDFE with the SOTA estimator improves its performance. Applying data augmentation to HDFE improves its performance.

	PCPNet Dataset (↓ means improvement)							**FamousShape Dataset (↓ means improvement)**						
	Noise				Density		Average	Noise				Density		Average
	None	Low	Med	High	Stripe	Gradient		None	Low	Med	High	Stripe	Gradient	
PCPNet [23]	9.64	11.51	18.27	22.84	11.73	13.46	14.58	18.47	21.07	32.60	39.93	18.14	19.50	24.95
PCPNet - PointNet + HDFE	9.48	11.05	**17.16**	**22.53**	11.61	10.19	13.67	15.66	**17.92**	31.24	38.89	17.26	14.55	22.59
Difference	0.16 ↓	0.46 ↓	1.11 ↓	0.31 ↓	0.12 ↓	3.27 ↓	0.91 ↓	2.81 ↓	3.15 ↓	1.36 ↓	1.04 ↓	0.88 ↓	4.95 ↓	2.36 ↓
PCPNet - PointNet + HDFE + Aug.	**7.97**	**10.72**	17.69	22.76	**9.47**	**8.67**	**12.88**	**13.04**	17.99	**31.23**	**38.57**	**14.01**	**12.13**	**21.16**
Difference	1.67 ↓	0.79 ↓	0.58 ↓	0.08 ↓	2.26 ↓	4.79 ↓	1.70 ↓	5.43 ↓	3.08 ↓	1.37 ↓	1.36 ↓	4.13 ↓	7.37 ↓	3.79 ↓
HSurf-Net [44]	4.30	8.78	16.15	**21.64**	5.18	5.03	10.18	7.54	15.56	29.47	38.61	7.82	7.44	17.74
HSurf-Net + HDFE	4.13	**8.64**	**16.14**	**21.64**	5.02	4.87	10.07	7.46	**15.50**	**29.42**	**38.56**	7.77	7.35	17.68
Difference	0.17 ↓	0.14 ↓	0.01 ↓	0.00	0.16 ↓	0.16 ↓	0.11 ↓	0.08 ↓	0.06 ↓	0.04 ↓	0.05 ↓	0.05 ↓	0.09 ↓	0.06 ↓
HSurf-Net + HDFE + Aug.	**3.89**	8.78	**16.14**	21.65	**4.60**	**4.51**	**9.93**	**7.11**	15.57	29.44	38.57	**6.97**	**6.98**	**17.44**
Difference	0.41 ↓	0.00	0.01 ↓	0.01 ↑	0.58 ↓	0.52 ↓	0.25 ↓	0.43 ↓	0.01 ↑	0.03 ↓	0.04 ↓	0.85 ↓	0.46 ↓	0.30 ↓

HDFE significantly outperforms the PointNet baseline. When processing the local patches, we replace PointNet with HDFE followed by a neural network. This replacement leads to an average reduction in error of 1.70 and 3.79 on each dataset. This is possibly because HDFE encodes the distribution of the local patch, which is guaranteed by the decodability property of HDFE. PointNet, on the other hand, does not have such guarantee. Specifically, PointNet aggregates point cloud features through a max-pooling operation, which may omit points within the point cloud and fail to adequately capture the patch's distribution. Consequently, in tasks where modeling the point cloud distribution is crucial, such as normal estimation, PointNet exhibits higher error compared to HDFE.

HDFE, as a plug-in module, improves the SOTA baseline significantly. HSurf-Net [44], the SOTA method in surface normal estimation, introduces many features, such as local aggregation layers, and global shift layers specifically for the task. Notably, HDFE does not compel such features. We incorporate HDFE into HSurf-Net, where it leads to average error reductions of 0.25/0.30 on each dataset. Notably, such incorporation can be performed on any PointNet-based architecture across various tasks. Incorporating HDFE to other PointNet-based architectures for performance and robustness gain can be a future research direction.

HDFE promotes stronger robustness to point density variance. In both comparisons and both benchmarks, HDFE exhibits stronger robustness to point density variation than its PointNet counterpart, especially in the Density-Gradient setting (error reduction of 4.79/7.37/0.52/0.46). This shows the effectiveness of the HDFE's sample invariance property and the embedding augmentation. Sample invariance ensures a stable encoding of local patches when the point density changes. The embedding augmentation is a second assurance to make the system more robust to density variation.

4.4 Conclusion

Hyper-Dimensional Function Encoding (HDFE) constructs vector representations for continuous objects. The representation, without any training, is sample invariant, decodable, and isometric. These properties position HDFE as an interface for the processing of continuous objects by neural networks. Our study demonstrates that the HDFE-based architecture attains significantly reduced errors compared to PointNet-based counterparts, especially in the presence of density perturbations. This reveals that HDFE presents a promising complement to PointNet and its variations for processing point cloud data. Adapting HDFE (e.g. imposing rotational invariance to HDFE) to tasks like point cloud classification and segmentation offers promising avenues for exploration. Still, HDFE does possess limitations in encoding capacity. For functions defined over large domains or highly non-linear functions, HDFE can experience underfitting. The exploration of techniques to enhance HDFE's capacity remains promising research. Regardless, HDFE already shows strong applicability in low-dimensional (1D, 2D, 3D) inputs.

4.5 Supplementary Materials

4.5.1 Gradient Descent for Decoding Function Encoding

In Eqn. (4.3), HDFE decodes the function encoding by a similarity maximization. Given the function encoding $\mathbf{F} \in \mathbb{C}^N$, and a query point $x_0 \in X$, the function value $\hat{y}_0$ is reconstructed by:

$$\hat{y}_0 = \text{argmax}_{y \in Y} \langle \mathbf{F} \oslash E_X(x_0), E_Y(y) \rangle$$

When E_X and E_Y are chosen as the fractional power encoding (Eqn. (4.6)), the optimization can be solved by gradient descent. In this section, we detail the gradient descent formulation. We assume the output space $Y = \mathbb{R}$.

By setting E_Y as the fractional power encoding, the optimization can be rewritten as:

$$\hat{y}_0 = \text{argmax}_{y \in (0,1)} \langle \mathbf{F} \oslash E_X(x_0), \exp(i\Psi y) \rangle$$[1]

where $\Psi \in \mathbb{R}^N$ is a random fixed vector where all elements are drawn from the normal distribution. Since $\mathbf{F} \oslash E_X(x_0)$ is a constant vector, the optimization can be further simplified as:

$$\hat{y}_0 = \text{argmax}_{y \in (0,1)} \langle \mathbf{z}, \exp(i\Psi y) \rangle \tag{4.8}$$

where $\mathbf{z} = \mathbf{F} \oslash E_X(x_0) \in \mathbb{C}^N$. We write $\mathbf{z}$ into its polar form:

$$\mathbf{z} = \left[a_1 e^{i\theta_1}, a_2 e^{i\theta_2}, \cdots, a_N e^{i\theta_N} \right]$$

where $a_k \in [0, +\infty)$ and $\theta_k \in [0, 2\pi)$. We first simplify Eqn. (4.8) and then compute its gradient with respect to y.

$$\begin{aligned}
\langle \mathbf{z}, \exp(i\Psi y) \rangle &= \frac{1}{N^2} ||\bar{\mathbf{z}} \cdot \exp(i\Psi y)||^2 \\
&= \frac{1}{N^2} \Big|\Big| \sum_{k=1}^{N} a_k e^{i\theta_k} e^{-i\Psi_k y} \Big|\Big|^2 \\
&= \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} a_p a_q e^{i\left[(\theta_p - \theta_q) - (\Psi_p - \Psi_q) y\right]} \\
&= \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} a_p a_q \cos\left[(\theta_p - \theta_q) - (\Psi_p - \Psi_q) y\right]
\end{aligned}$$

The gradient can be computed easily by taking the derivative:

$$\frac{d}{dy} \langle \mathbf{z}, \exp(i\Psi y) \rangle = \frac{1}{N^2} \sum_{p=1}^{N} \sum_{q=1}^{N} a_p a_q (\Psi_p - \Psi_q) \sin\left[(\theta_p - \theta_q) - (\Psi_p - \Psi_q) y\right] \tag{4.9}$$

In practice, N can be a large number like 8000. Computing the gradient by Eqn. (4.9) can be expensive. Fortunately, since Ψ is a random fixed vector, where all elements are independent of

[1]In Eqn. (4.6), $E_Y(y) = \exp(i\beta\Psi y)$. Since β and n are two constant numbers, we rewrite $\beta\Psi$ as Ψ for notation purpose.

each other, we can unbiasedly estimate the gradient by sampling a small number of entries in the vector. The decoding can be summarized by the pseudo-code below.

Algorithm 3 Gradient Descent for Decoding Function Encoding

```
Input: $\mathbf{F}$, $x_0$, $E_X$, $\Psi$
▷ $\mathbf{F}$ is the function encoding. $x_0$ is the query point.
▷ $E_X$ is the input mapping. $\Psi$ is the parameter in $E_Y$.
$\mathbf{z} = \mathbf{F} \oslash E_X(x_0)$
$\mathbf{z} = \left[a_1 e^{i\theta_1}, a_2 e^{i\theta_2}, \cdots, a_N e^{i\theta_N}\right]$
▷ Convert $\mathbf{z}$ into its polar form.
while $\langle \mathbf{z}, \exp(i\Psi y)\rangle$ still increases do
    Randomly select a subset of $[0, N) \cap \mathbb{Z}$. Denote as $S$.
    ▷ We choose $|S| = 500$.
    $g = |S|^{-2} \cdot \sum_{p,q \in S} a_p a_q (\Psi_p - \Psi_q) \sin\left[(\theta_p - \theta_q) - (\Psi_p - \Psi_q) y\right]$
    $y \leftarrow y - \alpha \cdot g$
    ▷ $\alpha$ is the learning rate.
end while
```

4.5.2 Proof of Asymptotic Sample Invariance

In the main paper, we claim that the iterative refinement (Algorithm 1) will converge to *the center of the smallest ball containing all the sample encodings* and therefore, HDFE leads to an asymptotic sample invariant representation. In this section, we detail the proof of the argument. To facilitate the understanding, Figure 4.7 sketches the proof from a high-level viewpoint. Recall the definition of asymptotic sample invariance (definition 5):

Definition 1 (Asymptotic Sample Invariance)**.** Let $f : X \to Y$ be the function to be encoded, $p : X \to (0, 1)$ be a probability density function (pdf) on X, $\{x_i\}_{i=1}^n \sim p(X)$ be n independent samples of X. Let $\mathbf{F}_n$ be the representation computed from the samples $\{x_i, f(x_i)\}_{i=1}^n$, asymptotic sample invariance implies $\mathbf{F}_n$ converges to a limit $\mathbf{F}_\infty$ independent of the pdf p.

Proof. We begin by showing the iterative refinement converges to

$$\mathbf{F}_n = \text{argmax}_{||z||=1} \min_{i=1}^{n} \langle z, E(x_i, f(x_i))\rangle \tag{4.10}$$

where $E(x, f(x))$ is defined at Eqn. (4.6) in the original paper. It maps a function sample to a high-dimensional space $\mathbb{C}^N$.

To show the convergence of the iterative refinement, it follows from the gradient descent: since $\nabla[-\min_i \langle z, E(x_i, f(x_i))\rangle] = -\text{argmin}_i \langle z, E(x_i, f(x_i))\rangle$, the gradient descent is formulated as $z \leftarrow z + \alpha \cdot \text{argmin}_i \langle z, E(x_i, f(x_i))\rangle$, which aligns with the iterative refinement in the paper.

Then we will prove Eqn. (4.10) produces a sample invariant encoding by proving $\mathbf{F}_n$ converges to

$$\mathbf{F}_\infty = \text{argmax}_{||z||=1} \min_{x \in X} \langle z, E(x, f(x))\rangle \tag{4.11}$$

Throughout the proof, we use the following definitions:

$S \subset \mathbb{C}^N$	$\triangleq$	$\bigcup_{x\in X} E(x, f(x))$
$S_n \subset \mathbb{C}^N$	$\triangleq$	$\bigcup_{i=1}^{n} E(x_i, f(x_i))$
$\lvert\lvert \cdot \rvert\rvert$	$\triangleq$	L2-norm of a complex vector.
d_H	$\triangleq$	$\max_{q\in Q} \min_{p\in P} \lvert\lvert p - q\rvert\rvert$. Hausdorff distance between two compact sets $P \subset Q \subset \mathbb{C}^N$.
$Ball(P)$	$\triangleq$	the smallest solid ball that contains the compact set P.
$center(Ball(\cdot))$	$\triangleq$	center of the ball.

Recall from Eqn. (4.6), $E(x, y) = \mathcal{F}^{-1}(e^{i(\Phi x+\Psi y)})$, so $||E(x, y)|| = 1$ for all x and y. Since $||z_1 - z_2||^2 = ||z_1||^2 + ||z_2||^2 - 2\langle z_1, z_2\rangle$, we have $\langle z, E(x, y)\rangle = 1 - \frac{1}{2}||z - E(x, y)||^2$ if $||z||^2 = 1$. Therefore, Eqn. (4.10) is equivalent to

$$\mathbf{F}_n = \text{argmin}_{||z||=1} \max_{i=1}^{n} ||z - E(x_i, f(x_i))|| \tag{4.12}$$

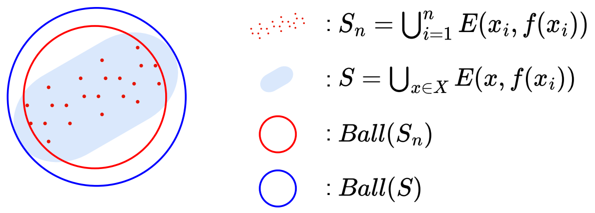
Note that Eqn. (4.12) implies **$\mathbf{F}_n$ is the center of the smallest ball containing** S_n:

$$\mathbf{F}_n = center(Ball(S_n)) \tag{4.13}$$

because if we were to construct balls containing S_n with a center $\mathbf{F}' \neq \mathbf{F}_n$, the radius of the ball must be larger than the radius of $Ball(S_n)$.

When $n \to \infty$, the Hausdorff distance between $Ball(S_n)$ and $Ball(S)$ goes to 0 with probability one. First, it is easy to see that $d_H(Ball(S_n), Ball(S))$ is a decreasing sequence and is positive, so the limit exists. Assume the limit is strictly positive, then there exists a point $p' \in S$ such that $\min_{q\in S_n} ||p' - q|| > c$ for some constant $c > 0$ as $n \to \infty$. This means no sample is drawn from the ball $B_c(p')$. This is contradictory to the definition of $p : X \to (0, 1)$: p is positive over the input space X.

Finally, we conclude by $||\mathbf{F}_n - \mathbf{F}_\infty|| \le d_H(Ball(S_n), Ball(S))$. Since $Ball(S_n) \subset Ball(S)$, from elementary geometry, if $A \subset B$ are two balls, then $||center(B) - center(A)|| \le radius(B) - radius(A) \le d_H(B, A)$. Therefore, $||center(Ball(S_n)) - center(Ball(S))|| \le d_H(Ball(S_n), Ball(S))$. Therefore, $||\mathbf{F}_n - \mathbf{F}_\infty|| \le d_H(Ball(S_n), Ball(S))$, which decays to 0 as $n \to \infty$. $\square$



: $S_n = \bigcup_{i=1}^n E(x_i, f(x_i))$
: $S = \bigcup_{x\in X} E(x, f(x_i))$
: $Ball(S_n)$
: $Ball(S)$

Figure 4.7: Proof of asymptotic sample invariance (overview). $Ball(S)$ and $Ball(S_n)$ are the smallest ball containing S and S_n. As $n \to \infty$, the Hausdorff distance between the two balls goes to zero with probability one. From elementary geometry, $||center(Ball(S_n)) - center(Ball(S))|| \le d_H(Ball(S_n), Ball(S))$. So the distance between the centers of the two balls goes to 0.

4.5.3 Proof of Isometry

In this section, we complete the proof that HDFE is an isometry.

Theorem 1. *Let $f, g : X \to Y$ be both c-Lipschitz continuous, then their L2-distance is preserved in the encoding. In other words, HDFE is an isometry:*

$$||f - g||_{L_2} = \int_{x \in X} \left| f(x) - g(x) \right|^2 dx \approx b - a\langle \mathbf{F}, \mathbf{G} \rangle$$

Lemma 4. $\langle x \otimes y, x \otimes z \rangle = \langle y, z \rangle$.

Proof. Let $x = e^{i\mathbf{x}}$, $y = e^{i\mathbf{y}}$, $z = e^{i\mathbf{z}}$,

$$\begin{aligned} \langle x \otimes y, x \otimes z \rangle &= \langle e^{i(\mathbf{x}+\mathbf{y}))}, e^{i(\mathbf{x}+\mathbf{z})} \rangle \\ &= e^{i(\mathbf{x}+\mathbf{y})} \cdot e^{-i(\mathbf{x}+\mathbf{z})} \\ &= \langle e^{i\mathbf{y}}, e^{i\mathbf{z}} \rangle \\ &= \langle y, z \rangle \end{aligned}$$

□

Proof.

$$\begin{aligned} \langle \mathbf{F}, \mathbf{G} \rangle &= \int_x \int_{x'} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &= \int_{|x-x'|<\epsilon} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &\qquad + \int_{|x-x'|>\epsilon} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &= \int_x \langle E_X(x) \otimes E_Y(f(x)), E_X(x) \otimes E_Y(g(x)) \rangle dx + noise \\ &\approx \int_x \langle E_Y(f(x)), E_Y(g(x)) \rangle dx \qquad \text{by Lemma 4.} \\ &= \int_x \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} \beta^{2k} (f(x) - g(x))^{2k} dx \qquad \text{by Bochner's theorem} \\ &\approx b - a \int_x |f(x) - g(x)|^2 dx \qquad \text{by taking the first and second order terms} \end{aligned}$$

The second line holds because when $d_X(x, x')$ is larger than the receptive field ϵ_0, $E_X(x)$ and $E_X(x')$ will be orthogonal, so the similarity between $E_X(x) \otimes E_Y(f(y))$ and $E_X(x') \otimes E_Y(g(y))$ will be close to zero and they will be summed as noise. □

5 Theories and Algorithms: A Two-Page Summary

We believe that the most elegant summary of the two chapters, spanning 31 pages, is encapsulated in 74 lines of PyTorch code.

Chapter 3 introduces a local point cloud geometry encoder that is efficient to compute. Chapter 4 introduces an iterative refinement algorithm to obtain make the function encoding invariant to sampling distribution. In this chapter, we unify the techniques in both algorithms to produce a local geometry encoder that is 1) efficient to compute, and 2) robust to noise and density variation. This will conclude the theories and algorithms chapter.

```python
import torch
import torch.nn as nn
import numpy as np
from scipy.stats import norm

def strict_standard_normal(d):
    # this function generate very similar outcomes as torch.randn(d)
    # but the numbers are strictly standard normal, no randomness.
    y = np.linspace(0, 1, d+2)
    x = norm.ppf(y)[1:-1]
    np.random.shuffle(x)
    x = torch.tensor(x).float()
    return x

class VecKM(nn.Module):
    def __init__(self, d=256, alpha=6, beta=1.8, p=4096):
        super().__init__()
        self.alpha, self.beta, self.d, self.p = alpha, beta, d, p
        self.sqrt_d = d ** 0.5

        self.A = torch.stack(
            [strict_standard_normal(d) for _ in range(3)],
            dim=0
        ) * alpha
        self.A = nn.Parameter(self.A, False)            # Real(3, d)

        self.B = torch.stack(
            [strict_standard_normal(p) for _ in range(3)],
            dim=0
        ) * beta
        self.B = nn.Parameter(self.B, False)            # Real(3, d)

    def forward(self, pts):
        """ Compute the dense local geometry encodings of the point cloud.
        Args:
            pts: (bs, n, 3) or (n, 3) tensor, the input point cloud.
        Returns:
            G: (bs, n, d) or (n, d) tensor. the dense local geometry
                encodings.
        """
        """ VecKM START """
        pA = pts @ self.A                                # Real(..., n, d)
        pB = pts @ self.B                                # Real(..., n, p)
        eA = torch.concatenate(
            (torch.cos(pA), torch.sin(pA)), dim=-1)      # Real(..., n, 2d)
        eB = torch.concatenate(
            (torch.cos(pB), torch.sin(pB)), dim=-1)      # Real(..., n, 2p)

        """ HDFE START """
        density_emb = eB.mean(dim=-2, keepdim=True)      # Real(..., 1, 2p)
        density = eB @ density_emb.transpose(-1,-2)      # Real(..., n, 1)
        scaled_eA = eA / density                         # Real(..., n, 2d)
        """ HDFE END """

        G = torch.matmul(
            eB,                                          # Real(..., n, 2p)
            eB.transpose(-1,-2) @ scaled_eA              # Real(..., 2p, 2d)
        )                                                # Real(..., n, 2d)
        G = torch.complex(
            G[..., :self.d], G[..., self.d:]
        ) / torch.complex(
            eA[..., :self.d], eA[..., self.d:]
        )                                                # Comp(..., n, d)
        """ VecKM END """

        G = torch.concatenate((G.real, G.imag), dim=-1) # Real(..., n, 2d)
        G = G / torch.norm(G, dim=-1, keepdim=True) * self.sqrt_d
        return G

if __name__ == '__main__':
    vkm = VecKM()
    pts = torch.rand((10,1000,3))
    G = vkm(pts) # it will be a Real(10,1000,512) tensor.
    pts = torch.rand((1000,3))
    G = vkm(pts) # it will be a Real(1000,512) tensor.
```

6 Proposed Application I: Real-Time Optical Flow Estimation from Events Camera

6.1 Introduction

Estimating motion fields (optical flow or normal flow) from event cameras is a well-established challenge. Traditional approaches often transform the event stream into frames and apply conventional feature matching techniques to estimate the flow. This process, however, negates the inherent asynchronous advantage of event cameras. Alternatively, some event-based methods employ contrast maximization to derive flow from raw events, though this approach is computationally intensive. While there are real-time algorithms available, they lack sufficient accuracy. Our proposed application leverages a robust and efficient point cloud embedding algorithm to achieve dense, real-time motion field estimation.

6.2 Methodology

Our methodology is straightforward. Every 20 ms, we gather events within this interval and compute dense local feature embeddings. These embeddings are then processed through a fully-connected network that predicts the flow, with the network trained end-to-end.

6.3 Experiment Plan

Motion Field Estimation Accuracy. We aim to benchmark the accuracy of our motion field estimation against existing methods by comparing estimation errors. Additionally, we will measure and compare the runtime and latency of each method, and qualitatively assess the visualized motion fields.

Performance of Downstream Tasks. We will evaluate the impact of our motion field estimation on downstream tasks, such as ego-motion estimation and moving object segmentation, to demonstrate the practical accuracy of these applications.

7 Proposed Application II: Distribution-Aware Numeric Number Encoding in Tabular Data

7.1 Introduction

Encoding numeric values into vectors remains a significant challenge for language models and tabular data. Although recent efforts have explored various encoding strategies, demonstrating their effectiveness, consensus on the optimal approach is still lacking. This makes the exploration of more effective numeric value encoding a vibrant area of research.

Current architectures for handling tabular data and language models employ diverse methods for encoding numeric values. These methods include scaling with feature vectors, piece-wise linear mapping, periodic activation functions, and quantization encoding. Research by [46] compares these methods and shows that effective numeric encoding can significantly enhance model performance.

However, these methods often overlook a crucial aspect: they typically encode numeric values based solely on the numbers themselves, disregarding the contextual information that could be gleaned from the data, such as the mean and standard deviation of a column. Incorporating this contextual information could potentially enhance model performance.

For example, when we encode the temperature $20°C$, different context can produce very different interpretation of the temperature. If it is in Iceland, then $20°C$ means an abnormal high temperature. But if it is in Brazil, then it means a normal temperature. Therefore, it is perceivably beneficial to encode a numeric value conditioned on some contexts.

For tabular data, such context can be obtained easily because the columns of the data provide context. Specifically, we want to encode the entry of a column conditioned on the distribution of the entire column.

For instance, the temperature $20°C$ can be interpreted differently depending on the context. In Iceland, $20°C$ might signify an abnormally high temperature, whereas in Brazil, it would be considered normal. Recognizing the value of context-dependent encoding, we propose to encode numeric values based on the distribution of the data within their respective columns.

7.2 Methodology

Denote a column of a table as $\{x_1, x_2, \cdots, x_n\}$, the encoding of an entry is $E(x_j) = \sum_{k=1}^{n} \exp[i(x_k - x_j)A]$, where $\sum_{k=1}^{n} \exp(ix_kA)$ encodes the distribution of the column and $E(x_j)$ encodes the number x_j conditioned on the distribution.

7.3 Experiment Plan

End-to-End Panel Data Training. Following the experimental design outlined in [46], this study will compare whether incorporating distributional priors can improve model performance, with no pre-training involved.

Single-Table Pre-Training and Finetuning. According to the methodology in [47], the model will be pre-trained and finetuned on a large singular table to assess the effectiveness of the encoding within pre-training pipelines.

Multiple-Table Pre-Training and Finetuning. Following the experiment design by [48], this scenario tests the robustness of the encoding strategy across different tables, a more demanding and complex task. This will evaluate the transferability of the encoding across diverse datasets.

8 Reading List

8.1 Proposed Application I.

1. Shiba, S., Aoki, Y., & Gallego, G. (2022, October). Secrets of event-based optical flow. In European Conference on Computer Vision (pp. 628-645). Cham: Springer Nature Switzerland.

2. Dalgaty, T., Mesquida, T., Joubert, D., Sironi, A., Vivet, P., & Posch, C. (2023). Hugnet: Hemi-spherical update graph neural network applied to low-latency event-based optical flow. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3953-3962).

3. Liu, H., Chen, G., Qu, S., Zhang, Y., Li, Z., Knoll, A., & Jiang, C. (2023). TMA: Temporal motion aggregation for event-based optical flow. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9685-9694).

4. Paredes-Vallés, F., Scheper, K. Y., De Wagter, C., & De Croon, G. C. (2023). Taming contrast maximization for learning sequential, low-latency, event-based optical flow. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9695-9705).

5. Huang, X., Zhang, Y., & Xiong, Z. (2023). Progressive spatio-temporal alignment for efficient event-based motion estimation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 1537-1546).

6. Shiba, S., Aoki, Y., & Gallego, G. (2022). Fast event-based optical flow estimation by triplet matching. IEEE Signal Processing Letters, 29, 2712-2716.

7. You, H., Cao, Y., Yuan, W., Wang, F., Qiao, N., & Li, Y. (2024). Vector-Symbolic Architecture for Event-Based Optical Flow. arXiv preprint arXiv:2405.08300.

8. Gehrig, M., Millhäusler, M., Gehrig, D., & Scaramuzza, D. (2021, December). E-raft: Dense optical flow from event cameras. In 2021 International Conference on 3D Vision (3DV) (pp. 197-206). IEEE.

9. Lu, Y., Wang, Q., Ma, S., Geng, T., Chen, Y. V., Chen, H., & Liu, D. (2023). Transflow: Transformer as flow learner. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 18063-18073).

10. Wan, Z., Dai, Y., & Mao, Y. (2022). Learning dense and continuous optical flow from an event camera. IEEE Transactions on Image Processing, 31, 7237-7251.

8.2 Proposed Application II.

1. Golkar, S., Pettee, M., Eickenberg, M., Bietti, A., Cranmer, M., Krawezik, G., ... & Ho, S. (2023). xval: A continuous number encoding for large language models. arXiv preprint arXiv:2310.02989.
2. Thawani, A., Pujara, J., Szekely, P. A., & Ilievski, F. (2021). Representing numbers in NLP: a survey and a vision. arXiv preprint arXiv:2103.13136.
3. Dong, H., Cheng, Z., He, X., Zhou, M., Zhou, A., Zhou, F., ... & Zhang, D. (2022). Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. arXiv preprint arXiv:2201.09745.
4. Yan, J., Zheng, B., Xu, H., Zhu, Y., Chen, D., Sun, J., ... & Chen, J. (2024). Making pre-trained language models great on tabular prediction. arXiv preprint arXiv:2403.01841.
5. Ye, C., Lu, G., Wang, H., Li, L., Wu, S., Chen, G., & Zhao, J. (2024, May). Towards Cross-Table Masked Pretraining for Web Data Mining. In Proceedings of the ACM on Web Conference 2024 (pp. 4449-4459).
6. Wang, Z., & Sun, J. (2022). Transtab: Learning transferable tabular transformers across tables. Advances in Neural Information Processing Systems, 35, 2902-2915.
7. Zhu, B., Shi, X., Erickson, N., Li, M., Karypis, G., & Shoaran, M. (2023). Xtab: Cross-table pretraining for tabular transformers. arXiv preprint arXiv:2305.06090.
8. Padhi, I., Schiff, Y., Melnyk, I., Rigotti, M., Mroueh, Y., Dognin, P., ... & Altman, E. (2021, June). Tabular transformers for modeling multivariate time series. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3565-3569). IEEE.
9. Zhang, D., Wang, L., Dai, X., Jain, S., Wang, J., Fan, Y., ... & Zhang, W. (2023, October). Fata-trans: Field and time-aware transformer for sequential tabular data. In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (pp. 3247-3256).
10. Gorishniy, Y., Rubachev, I., & Babenko, A. (2022). On embeddings for numerical features in tabular deep learning. Advances in Neural Information Processing Systems, 35, 24991-25004.

8.3 Other Potentially Relevant Papers

1. Pang, Y., Wang, W., Tay, F. E., Liu, W., Tian, Y., & Yuan, L. (2022, October). Masked autoencoders for point cloud self-supervised learning. In European conference on computer vision (pp. 604-621). Cham: Springer Nature Switzerland.

2. Li, Z., Gao, Z., Tan, C., Ren, B., Yang, L. T., & Li, S. Z. (2024). General Point Model Pretraining with Autoencoding and Autoregressive. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 20954-20964).

3. Yan, S., Yang, Y., Guo, Y., Pan, H., Wang, P. S., Tong, X., ... & Huang, Q. (2023). 3d feature prediction for masked-autoencoder-based point cloud pretraining. arXiv preprint arXiv:2304.06911.

4. Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., & Lu, J. (2022). Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 19313-19322).

5. Zhang, Y., Lin, J., He, C., Chen, Y., Jia, K., & Zhang, L. (2022). Masked surfel prediction for self-supervised point cloud learning. arXiv preprint arXiv:2207.03111.

6. Liu, H., Cai, M., & Lee, Y. J. (2022, October). Masked discrimination for self-supervised learning on point clouds. In European Conference on Computer Vision (pp. 657-675). Cham: Springer Nature Switzerland.

7. Wu, X., Jiang, L., Wang, P. S., Liu, Z., Liu, X., Qiao, Y., ... & Zhao, H. (2024). Point Transformer V3: Simpler Faster Stronger. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4840-4851).

8. Wu, X., Lao, Y., Jiang, L., Liu, X., & Zhao, H. (2022). Point transformer v2: Grouped vector attention and partition-based pooling. Advances in Neural Information Processing Systems, 35, 33330-33342.

9. Wei, C., Fan, H., Xie, S., Wu, C. Y., Yuille, A., & Feichtenhofer, C. (2022). Masked feature prediction for self-supervised visual pre-training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 14668-14678).

10. Yan, S., Yang, Z., Li, H., Song, C., Guan, L., Kang, H., ... & Huang, Q. (2023). Implicit autoencoder for point-cloud self-supervised representation learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 14530-14542).

9 Publications

1. **Yuan, D.**, Fermüller, C., Rabbani, T., Huang, F., & Aloimonos, Y. (2024). A Linear Time and Space Local Point Cloud Geometry Encoder via Vectorized Kernel Mixture (VecKM). In 2024 International Conference on Machine Learning.

2. **Yuan, D.**, Huang, F., Fermüller, C., & Aloimonos, Y. (2023). Decodable and Sample Invariant Continuous Object Encoder. In 2024 International Conference on Learning Representations.

3. **Yuan, D.***, Sutor, P.*, Summers-Stay, D., Fermuller, C., & Aloimonos, Y. (2022, July). Gluing neural networks symbolically through hyperdimensional computing. In 2022 International Joint Conference on Neural Networks (IJCNN) (pp. 1-10). IEEE.

4. Amrouch, H., Imani, M., Jiao, X., Aloimonos, Y., Fermuller, C., **Yuan, D.**, ... & Sutor, P. (2022, October). Brain-inspired hyperdimensional computing for ultra-efficient edge ai. In 2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS) (pp. 25-34). IEEE.

10 Timelines

- August 20th, 2024: Thesis Proposal
- September 28th, 2024: Submit Proposed Application II to ICLR2025.
- November 10th, 2024: Submit Proposed Application I to CVPR2025.
- August 2024 - May 2025: Potential Resubmissions and New Projects.
- May 2025: Thesis and Dissertion Defense.

Bibliography

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[2] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6411–6420, 2019.

[3] Y. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, and L. Yuan, "Masked autoencoders for point cloud self-supervised learning," in *European conference on computer vision*, pp. 604–621, Springer, 2022.

[4] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys (CSUR)*, 2021.

[5] D. Kleyko and E. Osipov, "Brain-like classifier of temporal patterns," in *2014 International Conference on Computer and Information Sciences (ICCOINS)*, pp. 1–6, IEEE, 2014.

[6] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *International Symposium on Quantum Interaction*, pp. 265–274, Springer, 2016.

[7] P. Sutor, D. Yuan, D. Summers-Stay, C. Fermuller, and Y. Aloimonos, "Gluing neural networks symbolically through hyperdimensional computing," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, IEEE, 2022.

[8] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[9] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations (in brief)," in *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, pp. 115–122, 2022.

[10] X.-F. Han, Z.-A. Feng, S.-J. Sun, and G.-Q. Xiao, "3d point cloud descriptors: state-of-the-art," *Artificial Intelligence Review*, pp. 1–51, 2023.

[11] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, "Rethinking network design and local geometry in point cloud: A simple residual mlp framework," *arXiv preprint arXiv:2202.07123*, 2022.

[12] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," *Advances in neural information processing systems*, vol. 31, 2018.

[13] D. Yuan, F. Huang, C. Fermüller, and Y. Aloimonos, "Decodable and sample invariant continuous object encoder," *arXiv preprint arXiv:2311.00187*, 2023.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] S. Bochner, *Harmonic analysis and the theory of probability.* Courier Corporation, 2005.

[17] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in neural information processing systems*, vol. 20, 2007.

[18] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Advances in neural information processing systems*, vol. 12, 1999.

[19] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. A. Smith, and L. Kong, "Random feature attention," *arXiv preprint arXiv:2103.02143*, 2021.

[20] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, vol. 7, pp. 187–199, 2021.

[21] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16259–16268, 2021.

[22] C. Trabelsi, O. Bilaniuk, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," *CoRR*, vol. abs/1705.09792, 2017.

[23] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, "Pcpnet learning local shape properties from raw point clouds," in *Computer graphics forum*, vol. 37, pp. 75–85, Wiley Online Library, 2018.

[24] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[25] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[26] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[27] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2d-3d-semantic data for indoor scene understanding," *arXiv preprint arXiv:1702.01105*, 2017.

[28] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 12, pp. 4338–4364, 2020.

[29] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, *et al.*, "Event-based vision: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 1, pp. 154–180, 2020.

[30] X. Lu, D. Yuan, Y. Chen, and J. C. Fung, "Impacts of urbanization and long-term meteorological variations on global pm2. 5 and its associated health burden," *Environmental Pollution*, vol. 270, p. 116003, 2021.

[31] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 399–404, IEEE, 2017.

[33] S. Salih, *Fourier Transform: Signal Processing.* BoD–Books on Demand, 2012.

[34] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[35] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," 2008.

[36] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations," *arXiv preprint arXiv:2109.03429*, 2021.

[37] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, “A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges,” *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–52, 2023.

[38] B. Komer and C. Eliasmith, “Efficient navigation using a scalable, biologically inspired spatial representation.,” in *CogSci*, 2020.

[39] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, “Deep complex networks,” *arXiv preprint arXiv:1705.09792*, 2017.

[40] C. H. Su and C. S. Gardner, “Korteweg-de vries equation and generalizations. iii. derivation of the korteweg-de vries equation and burgers equation,” *Journal of Mathematical Physics*, vol. 10, no. 3, pp. 536–539, 1969.

[41] M. Tek, “Development of a generalized darcy equation,” *Journal of Petroleum Technology*, vol. 9, no. 06, pp. 45–47, 1957.

[42] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar, “Multipole graph neural operator for parametric partial differential equations,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 6755–6766, 2020.

[43] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, “Model reduction and neural networks for parametric pdes,” *The SMAI journal of computational mathematics*, vol. 7, pp. 121–157, 2021.

[44] Q. Li, Y.-S. Liu, J.-S. Cheng, C. Wang, Y. Fang, and Z. Han, “Hsurf-net: Normal estimation for 3d point clouds by learning hyper surfaces,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 4218–4230, 2022.

[45] Q. Li, H. Feng, K. Shi, Y. Gao, Y. Fang, Y.-S. Liu, and Z. Han, “Shs-net: Learning signed hyper surfaces for oriented normal estimation of point clouds,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13591–13600, 2023.

[46] Y. Gorishniy, I. Rubachev, and A. Babenko, “On embeddings for numerical features in tabular deep learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24991–25004, 2022.

[47] I. Padhi, Y. Schiff, I. Melnyk, M. Rigotti, Y. Mroueh, P. Dognin, J. Ross, R. Nair, and E. Altman, “Tabular transformers for modeling multivariate time series,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3565–3569, IEEE, 2021.

[48] J. Yan, B. Zheng, H. Xu, Y. Zhu, D. Chen, J. Sun, J. Wu, and J. Chen, “Making pre-trained language models great on tabular prediction,” *arXiv preprint arXiv:2403.01841*, 2024.