APPENDIX

A NOTATION TABLE

Table 2: Table of Notation

		Notations of Continuous Functions
F	$\triangleq$	family of c-Lipschitz continuous functions
c	$\triangleq$	Lipschitz constant of the continuous function
X	$\triangleq$	input space, a bounded domain of the function
Y	$\triangleq$	output space, a bounded range of the function
d_X	$\triangleq$	distance metrics in the input space X
d_Y	$\triangleq$	distance metrics in the output space Y
$\{(x_i, f(x_i)\}$	$\triangleq$	collection of function samples.
m	$\triangleq$	dimensionality of the input space X.
		Notations of HDFE
ϵ_0	$\triangleq$	receptive field of HDFE.
$\mathbf{F}$	$\triangleq$	N-dimensional complex vector. Encoding of the explicit function f.
$\mathbf{F}_{f=0}$	$\triangleq$	N-dimensional complex vector. Encoding of the implicit function $f(x) = 0$.
N	$\triangleq$	dimensionality of the function encoding.
E_X, E_Y	$\triangleq$	the input and output mapping from X, Y to the encoding space $\mathbb{C}^N$.
$\otimes$	$\triangleq$	binding operation.
$\oslash$	$\triangleq$	unbinding operation.
$\langle \cdot, \cdot \rangle$	$\triangleq$	cosine similarity between two complex vectors.

B POINTNET AS FUNCTION ENCODER

In this section, we present empirical evidence highlighting PointNet's limitations in generating decodable function encodings. Our designed pipeline, following the popular style of training reconstruction networks, demonstrates PointNet's inability to produce even reasonable reconstructions. Consequently, we infer that without substantial modifications, PointNet is unlikely to acquire the necessary capability for effective function encoding.

Specifically, we generate random functions by

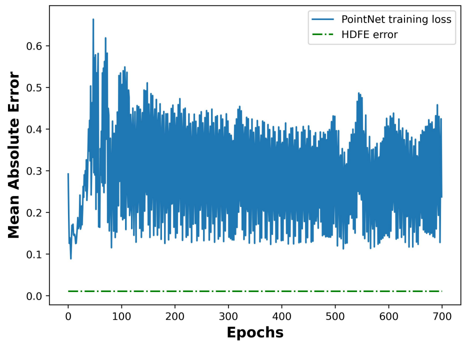
$$f(x) = \frac{1}{2} + \frac{1}{8} \sum_{k=1}^{4} a_k \sin(2\pi k x)$$

where $a_k \sim Uniform(0, 1)$ are the parameters controlling the generation and $f(x) \in (0, 1)$. We randomly sample $\{x_i, y_i\}_{i=1}^{5000}$ where $y_i = f(x_i) + \epsilon_i$, $x_i \sim Uniform(0, 1)$ and ϵ_i is white noise with variance 1e-4. Then we stack x_i and y_i into a $(5000, 2)$ matrix and feed it to a standard PointNet to generate a 1024-dimensional vector. After encoding the function samples, we introduce a decoder that retrieves the function values from the encoding when receiving function inputs. We optimize the PointNet encoder and the decoder by minimizing the reconstruction loss. The procedure can be summarized as:

$$\mathbf{F} = \texttt{PointNet}\big(\{(x_i, y_i)\}_{i=1}^{n}\big), \quad \hat{y_i} = \texttt{Decoder}(\mathbf{F}, x_i), \quad \mathcal{L} = \sum_{i=1}^{n} |y_i - \hat{y}_i|^2$$

The decoder is designed as followed: x_i is lifted to a 1024-dimensional vector $\tilde{\mathbf{x}}_i$ through sequential layers of `Linear(1,64)`, `Linear(64,128)`, `Linear(128,1024)` with `BatchNorm` and

ReLU inserted between the linear layers. The retrieved function value $\hat{y}_i$ is computed by the dot product between $\mathbf{F}$ and $\tilde{\mathbf{x}}_i$. Figure 4 (**Left**) plots the training curve and Figure 4 (**Right**) visualizes the reconstruction, which shows PointNet fails to learn a decodable representation.



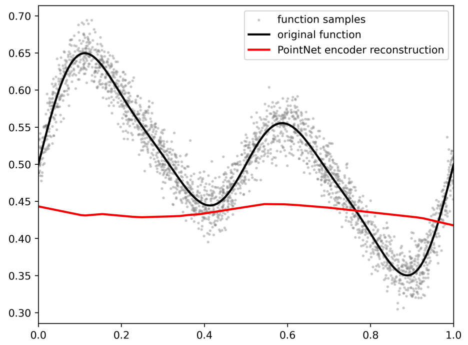


Figure 4: PointNet fails to learn an effective function encoder. **Left**: Training curve of the PointNet function encoder. The error is still significantly higher than HDFE though training for 700 epochs. **Right**: The PointNet function encoder does not produce a reasonable reconstruction.

C VECTOR FUNCTION ARCHITECTURE

In this section, we will give a brief introduction of VFA and its relation and comparison with HDFE.

Vector Function Architecture (VFA) A function of the form

$$f(x) = \sum_k \alpha_k \cdot K(x, x_k) \tag{7}$$

is represented by $\mathbf{F} = \sum_k \alpha_k \cdot E(x_k)$, where the kernel $K(\cdot, \cdot)$ and the encoder $E(\cdot)$ satisfy $K(x, y) = \langle E(x), E(y) \rangle$. Given the function encoding $\mathbf{F}$, the function value at a query point x_0 can be retrieved by $\hat{f}(x_0) = \langle \mathbf{F}, E(x_0) \rangle$.

VFA satisfies the four desired properties: VFA can encode an explicit function into a fixed-length vector, where the encoding is sample invariant and decodable. Besides, the function encoding will preserve the overall similarity of the functions: $\langle \mathbf{F}, \mathbf{G} \rangle = \int_x f(x)g(x)dx$. Note that when encoding multiple functions, the kernel $K(\cdot, \cdot)$ and the encoder $E(\cdot)$ must be the same. Otherwise, the function encoding generated by VFA will be meaningless.

However, it is a strong assumption that a function can be approximated by equation 7. In other words, under a fixed selection of $K(\cdot, \cdot)$ and $E(\cdot)$, equation 7 can only approximate a small set of functions. There are a large set of functions that cannot be approximated by equation 7. In Fig. 5, we show a failure case of VFA. We generate a function $f(x) = 0.1 + \sum_{k=1}^{10} \alpha_k \cdot K(x, x_k)$, where $x_k \sim N(0, 1)$, $\alpha_k \sim Uniform(0, 1)$. $K(x, y) = \exp[-20(x - y)^2]$ is an RBF kernel. We generate 1000 function samples and add white noise to the output values to form the training set. The experiment shows that VFA fails to reconstruct the function but HDFE succeeds.

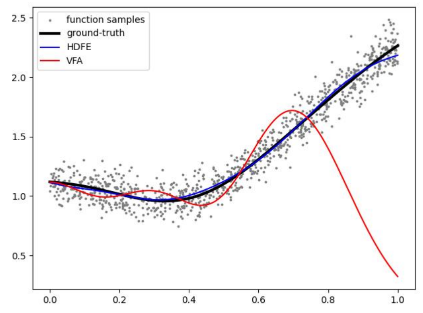


Figure 5: VFA fails to reconstruct the function but HDFE succeeds.

In this paper, HDFE is applicable for all c-Lipchitz function, without compromising the four desired properties. Thanks to the improvement, we can apply function encoding to real-world applications. The first application, PDE solving, cannot be solved by VFA because the input and output functions cannot be approximated by equation 7. The second application, local geometry prediction, has to deal with implicit function encoding, which cannot be solved by VFA either because VFA only encodes explicit functions.

D SUITABLE INPUT TYPES FOR HDFE

We mentioned HDFE can encode Lipschitz functions. In this section, we will discuss several data types that exhibits Lipschitz continuity, making them well-suited as inputs for HDFE. This enumeration is not exhaustive of all potential suitable inputs for HDFE. Our objective is to furnish a conceptual understanding of the characteristics that define a suitable input, thereby guiding future considerations in this domain.

Point Cloud Local Geometry In many 3D vision problems, point cloud local features are important, such as point cloud registration (Huang et al., 2021), scene matching (Han et al., 2023). The local geometry around a point in a point cloud is usually a continuous surface, which inherently exhibits Lipschitz continuity. This characteristic makes HDFE suitable for encoding this type of data.

Meteorological Measurements In the field of meteorology, machine learning plays a pivotal role in various applications, including pollutant estimation (Lu et al., 2020) and weather forecasting (Chen et al., 2022). A standard practice in these applications involves inputting meteorological measurements from neighboring areas (for instance, windows of 100 km^2) into machine learning models. HDFE is a suitable tool for encoding these neighboring meteorological measurements, primarily due to the inherent physical constraints that ensure these measurements do not rapidly change over time and space.

Time Series Data: Some time series data, especially in fields like event logging (Chen et al., 2021) or network traffic (Abbasi et al., 2021), are sparse in nature. Such sparse events can be treated as implicit functions and therefore HDFE is well-suited for it.

E DERIVATION OF DECODING

In this section, we complete the details why $\mathbf{F} \oslash E_X(x_0)$ can be decomposed into the summation of $E_Y(f(x_0))$ and noises. This is an immediate outcome from the similarity preserving property. When $d_X(x_0, x_i)$ is large, $\langle E_X(x_0) \otimes E_Y(f(x_0)), E_X(x_i) \otimes E_Y(f(x_i))\rangle \approx 0$. Since the unbinding operation is similarity preserving, we have $\langle E_Y(f(x_0)), E_X(x_i) \otimes E_Y(f(x_i)) \oslash E_X(x_0)\rangle \approx 0$. Therefore, $\mathbf{F} \oslash E_X(x_0)$ can be decomposed into $E_Y(f(x_0))$ and the sum of vectors that are orthogonal to it. Consequently, when we search for the y to maximize the $\langle \mathbf{F} \oslash E_X(x_0), E_Y(y)\rangle$, those orthogonal vectors will not bias the optimization.

F GRADIENT DESCENT FOR DECODING FUNCTION ENCODING

In equation 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 (equation 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:

$$\hat{y}_0 = \text{argmax}_{y \in (0,1)} \langle \mathbf{z}, \exp(i\Psi y)\rangle \tag{8}$$

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

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

[1]In equation 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.

where $a_k \in [0, +\infty)$ and $\theta_k \in [0, 2\pi)$. We first simplify equation 8 and then compute its gradient with respect to y.

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

The gradient can be computed easily by taking the derivative:

$$\frac{d}{dy}\langle \mathbf{z}, \exp(i\Psi y)\rangle = \frac{1}{N^2}\sum_{p=1}^{N}\sum_{q=1}^{N} a_p a_q (\Psi_p - \Psi_q) \sin\big[(\theta_p-\theta_q)-(\Psi_p-\Psi_q)y\big] \tag{9}$$

In practice, N can be a large number like 8000. Computing the gradient by equation 9 can be expensive. Fortunately, since Ψ is a random fixed vector, where all elements are independent of 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 2 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 $E_Y$.
$\mathbf{z} = \mathbf{F} \oslash E_X(x_0)$
$\mathbf{z} = \big[a_1 e^{i\theta_1}, a_2 e^{i\theta_2}, \cdots, a_N e^{i\theta_N}\big]$        ▷ 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 $|S| = 500$.
    $g = |S|^{-2} \cdot \sum_{p,q\in S} a_p a_q (\Psi_p - \Psi_q) \sin\big[(\theta_p-\theta_q)-(\Psi_p-\Psi_q)y\big]$
    $y \leftarrow y - \alpha \cdot g$        ▷ $\alpha$ is the learning rate.
end while
```

G Relation between FPE and RBF Kernel

In the main paper, we mention that we adopt fractional power encoding (FPE) as the mapping for the input and output space. In this section, we explain the rational behind our choice by revealing its close relationship between the radial basis function (RBF) kernel. For explanation purpose, we discuss the single-variable case. It is straight-forward to generalize to multi-variable scenarios.

RBF kernel is a similarity measure defined as $K(x, y) = \exp\big(-\gamma(x-y)^2\big)$. The feature space of the kernel has an infinite number of dimensions:

$$\exp\big(-\gamma(x-y)^2\big) = \sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\gamma^{2k}(x-y)^{2k} = \langle\phi(x), \phi(y)\rangle$$

where $\phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{\gamma}{1!}}x, \sqrt{\frac{\gamma^2}{2!}}x^2, \sqrt{\frac{\gamma^3}{3!}}x^3, \cdots\right]$.

We will then show a theorem that tells that FPE maps real numbers into finite complex vectors, where the similarity between the vectors can approximate a heavy-tailed RBF kernel. However, the feature space of the heavy-tailed RBF kernel still has an infinite number of dimensions and therefore inherits all the blessings of the RBF kernel. Notably, FPE provides a finite encoding, while approximating an infinite dimensional feature space.

Theorem 3. *Let $x, y \in \mathbb{R}$, $E(x) = e^{i\gamma\Theta x}$, where $\Theta \in \mathbb{R}^N$ and $\forall \theta \in \Theta, \theta \sim \mathcal{N}(0, \frac{1}{2})$, then as $N \to \infty$,*

$$\langle E(x), E(y) \rangle = \langle e^{i\gamma\Theta x}, e^{i\gamma\Theta y} \rangle \to \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!!} \gamma^{2k} (x-y)^{2k} = \langle \phi(x), \phi(y) \rangle$$

where $\phi(x) = e^{-\gamma x^2} \left[1, \sqrt{\frac{\gamma}{1!!}} x, \sqrt{\frac{\gamma^2}{2!!}} x^2, \sqrt{\frac{\gamma^3}{3!!}} x^3, \cdots \right]$.

Proof.

$$\begin{aligned}
\langle e^{i\gamma\Theta x}, e^{i\gamma\Theta y} \rangle &= \frac{1}{N^2} || \exp(i\gamma\Theta x) \cdot \exp(-i\gamma\Theta y) ||^2 \\
&= \frac{1}{N^2} \Big|\Big| \sum_{k=1}^{N} e^{i\gamma\theta_k (x-y)} \Big|\Big|^2 \\
&= \frac{1}{N^2} \Big[\sum_{p=1}^{N} 1 + \sum_{p \neq q} e^{i\gamma(\theta_p - \theta_q)(x-y)} \Big] \\
&= \frac{1}{N} + \frac{1}{N^2} \sum_{p \neq q} \cos \left[\gamma(\theta_p - \theta_q)(x-y) \right] \\
&= \frac{1}{N} + \frac{1}{N^2} \sum_{p \neq q} \sum_{k=0}^{\infty} (-1)^k \frac{\left[\gamma(\theta_p - \theta_q)(x-y) \right]^{2k}}{(2k)!} \\
&= \frac{1}{N} + \sum_{k=0}^{\infty} (-1)^k \Big[\sum_{p \neq q} \frac{(\theta_p - \theta_q)^{2k}}{N^2} \Big] \frac{\gamma^{2k} (x-y)^{2k}}{(2k)!} \\
&\to \sum_{k=0}^{\infty} (-1)^k (2k-1)!! \frac{\gamma^{2k} (x-y)^{2k}}{(2k)!} \\
&= \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!!} \gamma^{2k} (x-y)^{2k}
\end{aligned}$$

Since $\theta_p, \theta_q \sim \mathcal{N}(0, \frac{1}{2})$, $\theta_p - \theta_q \sim \mathcal{N}(0, 1)$, and therefore, $\mathbb{E}[(\theta_p - \theta_q)^{2n}] = (2n-1)!!$. So $\sum_{p \neq q} \frac{(\theta_p - \theta_q)^{2n}}{N^2}$ converges to $(2n-1)!!$. □

H PROOF OF THE HDFE'S PROPERTIES

H.1 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 6 sketches the proof from a high-level viewpoint. Recall the definition of asymptotic sample invariance (definition 1):

Definition (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

$$\mathbf{F}_n = \text{argmax}_{||z||=1} \min_{i=1}^{n} \langle z, E(x_i, f(x_i)) \rangle \tag{10}$$

where $E(x, f(x))$ is defined at equation 6 in the original paper. It maps a function sample to a high-dimensional space $\mathbb{C}^N$.

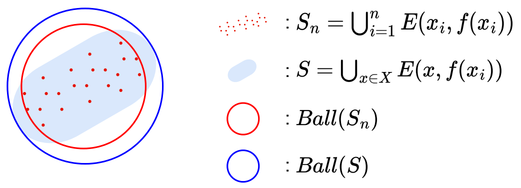

$: S_n = \bigcup_{i=1}^{n} E(x_i, f(x_i))$
$: S = \bigcup_{x \in X} E(x, f(x_i))$
$: Ball(S_n)$
$: Ball(S)$

Figure 6: 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, $||center(Ball(S_n)) - center(Ball(S))|| \leq d_H(Ball(S_n), Ball(S))$. So the distance between the centers of the two balls goes to 0.

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 $z \leftarrow z + \alpha \cdot \text{argmin}_\text{i} \langle z, E(x_i, f(x_i)) \rangle$, which aligns with the iterative refinement in the paper.

Then we will prove equation 10 produces a sample invariant encoding by proving $\mathbf{F}_n$ converges to

$$\mathbf{F}_\infty = \text{argmax}_{||z||=1} \min_{x \in X} \langle z, E(x, f(x)) \rangle \tag{11}$$

Throughout the proof, we use the following definitions:

$S \subset \mathbb{C}^N$	$\triangleq$	$\bigcup_{x \in X} E(x, f(x))$
$S_n \subset \mathbb{C}^N$	$\triangleq$	$\bigcup_{i=1}^{n} E(x_i, f(x_i))$
$\|\| \cdot \|\|$	$\triangleq$	L2-norm of a complex vector.
d_H	$\triangleq$	$\max_{q \in Q} \min_{p \in P} \|\|p - q\|\|$. 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 equation 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, equation 10 is equivalent to

$$\mathbf{F}_n = \text{argmin}_{||z||=1} \max_{i=1}^{n} ||z - E(x_i, f(x_i))|| \tag{12}$$

Note that equation 12 implies $\mathbf{F}_n$ **is the center of the smallest ball containing** S_n:

$$\mathbf{F}_n = center(Ball(S_n)) \tag{13}$$

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

H.2 Isometry

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

Theorem. *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.

$$\begin{aligned}\langle \mathbf{F}, \mathbf{G} \rangle &= \int_x \int_{x'} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &= \int_{|x-x'|<\epsilon} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &\qquad + \int_{|x-x'|>\epsilon} \langle E_X(x) \otimes E_Y(f(x)), E_X(x') \otimes E_Y(g(x')) \rangle dx' dx \\ &= \int_x \langle E_X(x) \otimes E_Y(f(x)), E_X(x) \otimes E_Y(g(x)) \rangle dx + noise \\ &\approx \int_x \langle E_Y(f(x)), E_Y(g(x)) \rangle dx \qquad \text{by Lemma 4.} \\ &= \int_x \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!!} \beta^{2k} (f(x) - g(x))^{2k} dx \qquad \text{by Theorem 3} \\ &\approx b - a \int_x |f(x) - g(x)|^2 dx \qquad \text{by taking the first and second order terms}\end{aligned}$$

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. □

I Empirical Experiment of HDFE

In this section, we verify the properties claimed in Sec. 2.4 with empirical experiments.

I.1 Sample Invariance

In Fig. 7, 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. 7 Mid). However, after the tuning process, the function vector becomes immune to the sample distribution (Fig. 7 Right).

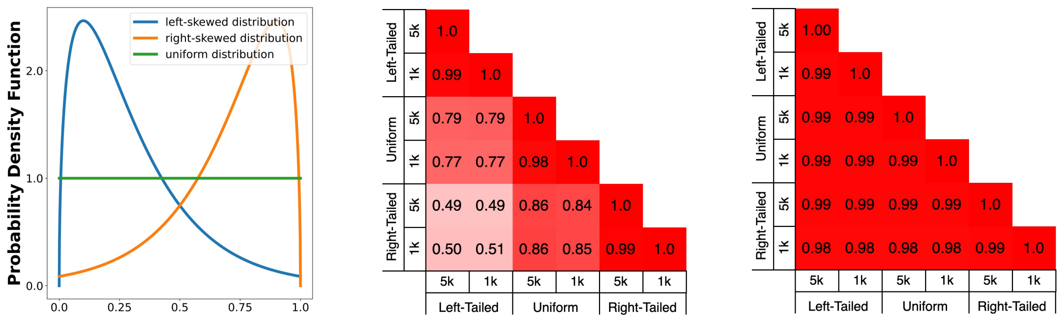


Figure 7: HDFE is invariant of sample distribution and sample size. **Left**: Three distributions where the function samples are drawn from. For each distribution, the sample size is either 5000 or 1000. **Mid, Right**: Similarity among the function vectors generated by the six sets of function samples, before and after the function vector tuning process, respectively.

I.2 Isometry

In Fig. 8, 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 2.

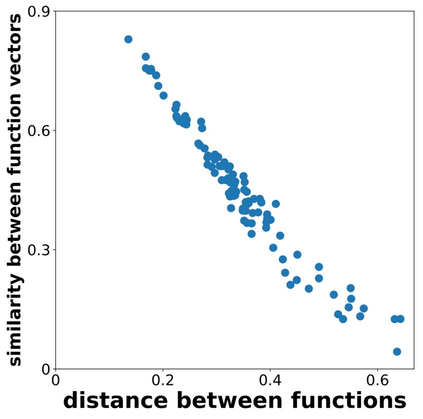


Figure 8: HDFE is a distance preserving transformation. The L2-distance between functions is proportional to the negative similarity between their encodings.

I.3 Practical Consideration of Iterative Refinement

In Algorithm 1, we propose one implementation of iterative refinement, which iteratively adds the sample encoding that has the minimum similarity with the function encoding. This implementation is a conservative implementation that guarantees asymptotic sample invariance. However, in practical applications, strict sample invariance may not be necessary. For example, achieving a similarity threshold of 0.99 when constructing with different samples might not be required. This relaxation is viable because the downstream neural network possesses an inherent ability to handle some level of inconsistency. Therefore, we consider a practical adaptation of Algorithm 1 by introducing slight modifications to accommodate these real-world considerations.

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 3 illustrates the procedure.

Algorithm 3 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
$\mathbf{F} = \sum_i w_i \cdot z_i$
```

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. 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

$$f(x) = \frac{1}{2} + \frac{1}{8} \sum_{k=1}^{4} a_k \sin(2\pi k x) \tag{14}$$

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 9 shows the comparison, which demonstrates that one-shot refinement is a quite good approximation of the sample invariant function encoding (the similarity increases from $\sim$ 0.5 to 0.98 after one-shot refinement). In Appendix I.4, Table 3, we also compare the effectiveness of the three refinement schemes in a synthetic regression problem.

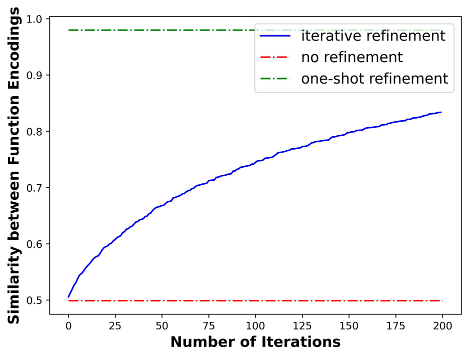


Figure 9: When encoding the same function under two different sample distributions, one-shot refinement can approximate the sample invariant function encoding well. It takes 75/90/1500 ms to encode 5000 samples on a CPU, and 7.5/8.0/250 ms on an NVIDIA Titan-X GPU when performing no refinement/one-shot refinement/200-step iterative refinement.

I.4 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 equation 14, where $a_k \sim Uniform(0, 1)$ are the parameters controlling the generation. The task is to regress the

Table 3: 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

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 (introduced in Appendix I.3), and 200-step iterative refinement. Table 3 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.

I.5 Information Loss of HDFE

In this section, we analyze the information loss when encoding continuous objects with HDFE. It is intuitive that a larger encoding dimensionality induces smaller information loss, and encoding a function changing more rapidly induces larger information loss. We attempt to quantify the relation through empirical experiments. We generate random functions and we measure the "function complexity" by the integral of the absolute gradient: $\text{complexity}(f) = \int_0^1 |f'(x)|dx$. Consequently, functions changing more rapidly yield a higher $\text{complexity}(f)$.

We study the relation between the reconstruction mean absolute error (MAE) / R-squared (R^2) and the function complexity under different encoding dimensions. Figure 10 reveals the MAE exhibits a linear relation with the input function complexity, while the R-squared seems not to be affected by the function complexity when the dimension is large enough.

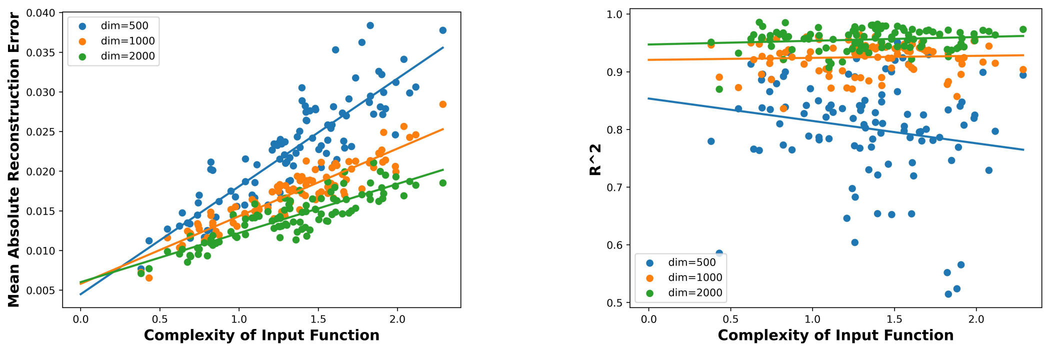


Figure 10: Empirical information loss when encoding functions of different complexities.

I.6 Low-Rank High-Dimensional Scenarios

We generate random functions by first randomizing $x_k \in \mathbb{R}^d$ and $\alpha_k \in \mathbb{R}$, the random function is constructed by:

$$f(x) = \sum_{k=1}^{n} \alpha_k \cdot K(x, x_k)$$

where n can measure the complexity of the function, and d is the dimension of the function input. The testing samples are generated by $x_k + noise$ for all $k \in [n]$.

In Fig. 11, the reconstruction error increases as n increases, which indicates the encoding quality is negatively correlated to the complexity of the function. However, the reconstruction error does not change as d increases, which indicates that the encoding quality does not depend on the explicit dimension of the function input.

This empirical experiment shows that HDFE has the potential to operate on high-dimensional data, because the encoding quality of HDFE does not depend on the dimension of the function input, but only depends on the complexity of the function.

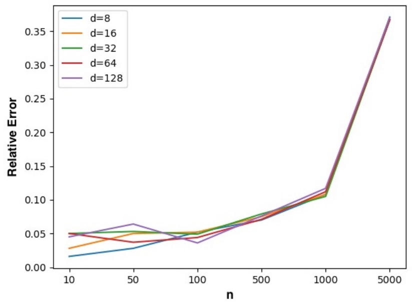


Figure 11: The reconstruction error of HDFE is negatively correlated to the complexity of the function, but does not depend on the dimension of the function input.

J Experiment Details

J.1 PDE Solver

The PDE and the solution are encoded into two embedding vectors with length N. A deep complex network is trained to learn the mapping between two vectors. The architecture is a sequence of layers: `[ComplexLinear(N,256), ComplexReLU(), ComplexLinear(256,256), ComplexReLU(), ComplexLinear(256,256), ComplexReLU(), ComplexLinear(256,N)]`. The network is trained with Adam optimizer with a learning rate of 0.001 for 20,000 iterations. The α value in equation 6 is 15, 25, 42, 45 for $N = 4000, 8000, 16000, 24000$ and the β value is 2.5.

J.2 Surface Normal Estimation

The architecture is a sequence of layers: `[ComplexLinear(N,256), ComplexBatchNorm() ComplexReLU(), ComplexLinear(256,256), ComplexBatchNorm(), ComplexReLU(), ComplexLinear(256,128), ComplexBatchNorm(), ComplexReLU()]`. After the sequence of layers, it will produce a 128-dimensional complex vector `z`. Since we desire a 3-dimensional real vector output (normal vector in $\mathbb{R}^3$), we use two `RealLinear(128,3)` layers `L_real` and `L_imag`. The final output normal vector is `L_real(z.real) + L_imag(z.imag)`. The network is trained with Adam optimizer with learning rate 0.001 for 270 epochs. The α is chosen as 20 and the dimensionality is chosen as 4096.

J.3 Adding HDFE Module to HSurf-Net

Denote the input local patch as P with shape $(B, N, 3)$, where B is the batch size, N is the number of points in a local patch. HSurf-Net uses their novel space transformation module to extract n keypoints and their K nearest neighbors. The resulting data is denoted as P_sub with shape $(B, n, K, 3)$. HSurf-Net uses a PointNet to process P_sub, by first lifting the dimension to (B, n, K, C) and then doing a maxpooling to shape the data into (B, n, C). We add our HDFE module here: we use HDFE to shape the data from $(B, n, K, 3)$ to (B, n, C), by first lifting the dimension from $(B, n, K, 3)$ to $(B, n, K, 512)$ using equation 6 and then average the embedding across the neighbors to shape it into $(B, n, 512)$. Then we use a fully-connected layer to map the data into (B, n, C), which becomes the HDFE feature. Then we sum the features generated by HSurf-Net and HDFE into a (B, n, C) matrix and pass to the output layer as HSurf-Net does.

K Ablation Studies of Surface Normal Estimation

Table 4: Ablation Studies on the PCPNet dataset.

Dimension	2048							4096						
Noise Level	None	Low	Med	High	Stripe	Gradient	Average	None	Low	Med	High	Stripe	Gradient	Average
$\alpha = 10$	8.98	10.80	**17.40**	**22.18**	10.62	9.92	13.32	9.00	**10.60**	**17.40**	**22.48**	10.50	9.87	13.31
$\alpha = 15$	8.98	11.02	17.73	22.72	10.85	9.46	13.46	8.29	10.77	17.46	22.63	10.17	9.10	13.07
$\alpha = 20$	**8.14**	**10.53**	17.86	23.07	**9.93**	**8.81**	**13.06**	**7.97**	10.72	17.69	22.76	**9.47**	**8.67**	**12.88**
$\alpha = 25$	8.86	11.43	17.94	22.85	10.42	9.33	13.47	8.40	11.23	17.66	22.74	9.89	8.95	13.15

Table 5: Ablation Studies on the FamousShape dataset.

Dimension	2048							4096						
Noise Level	None	Low	Med	High	Stripe	Gradient	Average	None	Low	Med	High	Stripe	Gradient	Average
$\alpha = 10$	15.12	18.43	30.86	**38.66**	15.52	13.82	22.07	15.06	18.15	**30.61**	38.50	16.81	13.71	22.14
$\alpha = 15$	14.42	**17.97**	**30.45**	38.92	15.47	13.81	21.84	13.68	**17.77**	31.17	38.79	14.83	13.06	21.55
$\alpha = 20$	**13.37**	18.43	31.40	39.03	**13.92**	**12.54**	**21.45**	**13.04**	17.99	31.23	**38.57**	**14.01**	**12.13**	**21.16**
$\alpha = 25$	14.70	19.71	31.52	38.95	15.04	13.56	22.25	14.03	19.16	31.39	38.65	14.38	13.22	21.81

In general, when the dimensionality is higher, the error is lower. When the receptive field is large (α is small), HDFE performs better when the noise level is high. When the receptive field is small (α is large), HDFE performs better when the noise level is low. This coincides with the analysis in Fig. 2. A large receptive field tends to filter out the perturbations and therefore is more robust to noise. A small receptive field can capture the high-frequency details and therefore is more accurate.

L Noise Robustness

In this section, we further analyze why HDFE is robust to point perturbations. For visualization purposes, we perform the analysis on 2d data, while the analysis generalizes well to higher dimensions. We randomly sample 1000 points from the unit circle $x^2 + y^2 = 1$ and add Gaussian noises to the samples. Then we encode the points into vectors using HDFE with different receptive fields ($\alpha = 10, 15, 20, 25$) and reconstruct the implicit function. The pseudo-color plot in Fig. 12 visualizes the likelihood that a point lies on the unit circle.

When α is small, the visualization shows that the reconstructions under different noise levels are similar, which tells that the encodings under different noise levels are similar. So it demonstrates a robustness to point perturbations. On the other hand, when α is large, the reconstruction is more refined and captures the high-frequency details, but as a price, it is more sensitive to the point perturbations.

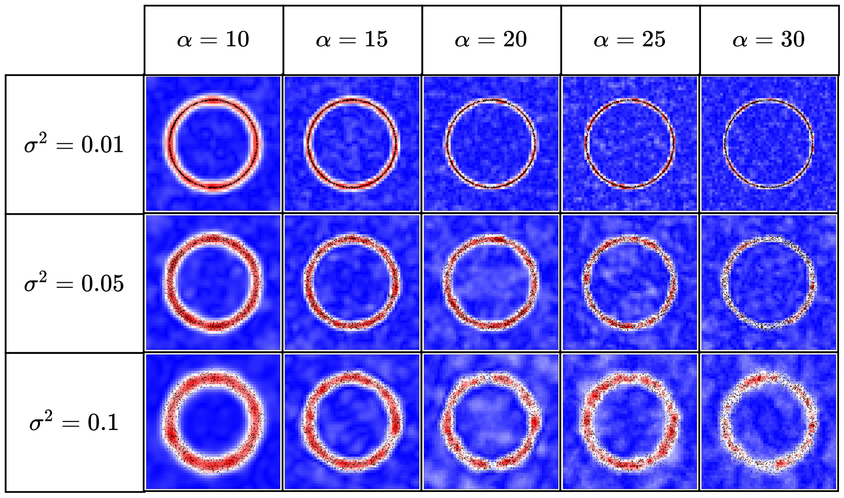


Figure 12: Reconstruction of implicit functions sampled with noisy inputs under different choices of receptive field.