ABSTRACT

Title of Dissertation: VECTOR KERNEL REPRESENTATION THEORY: APPLICATIONS IN POINT CLOUDS AND EVENT CAMERAS

Dehao Yuan
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Yiannis Aloimonos
Dr. Cornelia Fermüller
Department of Computer Science

Point set representation is a fundamental problem in fields such as point cloud processing and event-based vision. A point set is an unordered collection of points in Euclidean space, typically generated by sensing technologies like 3D scanners or event cameras. The goal of point set representation is to transform these unordered points into fixed-length vectors suitable for machine learning tasks. A key challenge in this process lies in handling point sets of varying sizes and uneven distributions. This requires representation methods that scale effectively to large point sets while remaining resilient to noise and variations in point density.

Existing point set representation methods generally fall into two categories: point-based and voxel-based approaches. Point-based methods, like PointNet and its variants, use multi-layer perceptrons to convert point clouds into fixed-length vectors. However, they often struggle with noise and variations in point density, especially when these variations

were unseen in the training set. These methods can also be computationally expensive and do not scale well to large point sets, such as event streams from event cameras. Voxel-based methods, on the other hand, voxelize point sets into regular grid structures, allowing for convolutional operations. While methods like E-RAFT show promise, voxel-based approaches often suffer from performance drops when point set densities vary and face challenges with generalizability across different domains.

This thesis introduces Vector Kernel Representation (VKR), a novel framework for point set representation that addresses the limitations of existing methods. We first establish Vector Kernel Representation Theorem, proving that Gaussian kernel mixtures can be represented as fixed-length complex vectors through a predefined operation. Building on top of the theorem, VKR is applied to various tasks, demonstrating superior accuracy and computational efficiency. In point cloud encoding, VKR significantly improves performance and reduce computational costs. When extended to event-based normal flow estimation, VKR significantly improves accuracy and generalizability compared to voxel-based methods. The thesis concludes by highlighting potential future directions, including time series modeling, integrating VKR with temporal architectures for event-based tasks, and advancing moving object segmentation through robust normal flow estimation.

VECTOR KERNEL REPRESENTATION THEORY: APPLICATIONS IN POINT CLOUDS AND EVENT CAMERAS

by

Dehao Yuan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Yiannis Aloimonos, Chair/Advisor
Dr. Cornelia Fermüller, Co-Advisor
Professor Ramani Duraiswami
Professor Furong Huang
Professor Bruno Olshausen
Dean's Representative:
Professor Timothy Horiuchi

© Copyright by
Dehao Yuan
2025

Foreword

If you only take away one thing from this thesis, this is it:

If your data can somehow be described by a Gaussian kernel mixture $f : \mathbb{R}^d \to \mathbb{R}$:

$$f(\mathbf{x}) = \frac{1}{n} \sum_{k=1}^{n} \exp\big(- \alpha^2 ||\mathbf{x} - \mathbf{x}_k||^2 / 2 \big)$$

then your data can be represented by a D-dimensional complex vector by:

$$\text{VectorKernelRepresentation}(f) = \frac{1}{n} \sum_{k=1}^{n} \exp(i \mathbf{x}_k \mathbf{A})$$

where i is the imaginary unit, $\mathbf{A}$ is a fixed d-by-D matrix with all elements drawn independently from $\mathcal{N}(0, \alpha^2)$.

This thesis is developed on top of this.

Acknowledgments

I deeply appreciate my advisor Prof. John Yiannis Aloimonos and my co-advisor Dr. Cornelia Fermüller. Throughout my PhD journey, they always provided the strongest support for my research, career, and personal growth. The PhD path is never easy, but their encouragement, guidance, and belief in me made the challenges feel much more manageable, and even exciting at times. I deeply appreciate my committees, Prof. Timothy Horiuchi, Prof. Furong Huang, Prof. Ramani Duraiswami, and Prof. Bruno Olshausen, for generously dedicating their time to reviewing my thesis. Their thoughtful feedback and insightful suggestions have been invaluable in shaping this work.

My labmates and friends at the UMD PRG lab have enriched my life and research greatly. I highly value many interesting and inspiring discussion with them. It is a great fortune to work with talented and brilliant researchers at PRG lab: Jingxi Chen, Levi Burner, Jiayi Wu, Minghui Liu, Angelos Mavrogiannis, Yantian Zha, Xiaomin Lin, Eadom Dessalene, Zhiyuan Hua, Botao He, Pavan Mantripragada, Seong Jong Yoo, Yianni Karabatis, Vaishnavi Patil. I am deeply grateful to my friends and colleagues at UMD: Bang An, Sicheng Zhu, Xiaoyu Liu, Chenghao Deng, Mucong Ding, Tahseen Rabbani, Xiyao Wang, Yuancheng Xu, Ruijie Zheng, Haozhe An, Xijun Wang, Anli Peng, Qifeng Chen.

I owe my deepest thank to my parents 袁伟林 and 任静. They always stand by my decisions and they are my biggest cheerleaders. Every summer, they endured the exhausting

15-hour flight to visit me in the United States, just to spend time together. Their amazing home-cooked meals are the ultimate comfort, giving me the energy to go through every challenge.

I thank my wife Muyang Ye and my parents in law 叶帆 and 李丹华, who always offer me constant encouragement.

I thank my grandparents 袁长泉 and 胡少妹; 任烈 and 谢耀凤. They are the generation that study Politics and Wars, that allows me to study Mathematics and Philosophy.

I thank the Yuan's and Ren's family for giving me a wonderful childhood and a peaceful, nurturing environment to grow up in. 袁伟和, 何惠芳, 袁嘉敏, 李嘉亮；伍重熹, 袁焕珍, 伍盈盈, 柯重枢；张伟文, 袁艳珍, 张艺仪, 赵昱焜；袁伟海, 黄小玲, 袁嘉蔚, 胡建安；任勤, 袁安政, 袁翔羚；任慧, 陈扬文, 陈朗和.

Table of Contents

Foreword ii

Acknowledgements iii

Table of Contents v

List of Tables viii

List of Figures x

Chapter 1: Introduction 1
- 1.1 Point Sets: Inputs to Vector Kernel Representation . . . 1
- 1.2 Downstream Tasks and Requirements of Efficiency and Robustness . . . 3
- 1.3 Existing Methods and Their Limitations . . . 4
- 1.4 Vector Kernel Representation Theory: An Overview . . . 5
- 1.5 List of publications . . . 6

Chapter 2: Vector Kernel Representation Theorem 8
- 2.1 Vector Kernel Representation Theorem . . . 8
- 2.2 Example Usage of Vector Kernel Representation Theorem . . . 9
- 2.3 Proof From Vector Function Architecture . . . 11
- 2.4 Proof From Fourier Transform . . . 12
- 2.5 A Short Conclusion . . . 14

Chapter 3: Decodable and Sampling Invariant Continuous Object Encoder 16
- 3.1 Introduction . . . 16
- 3.2 Related Work . . . 19
  - 3.2.1 Mesh-grid-based framework . . . 19
  - 3.2.2 Sparse framework . . . 20
- 3.3 Problem Definition and Methodology . . . 21
  - 3.3.1 Explicit Function Encoding . . . 23
  - 3.3.2 Implicit Function Encoding . . . 28
  - 3.3.3 Vector-Valued Function Encoding . . . 28
  - 3.3.4 Properties of HDFE . . . 30
- 3.4 Experiment . . . 31
  - 3.4.1 Empirical Experiment of HDFE . . . 32
  - 3.4.2 Cost of Refinements . . . 33

3.4.3 Effectiveness of Sample Invariance . . . 34
3.4.4 PDE Solver . . . 36
3.4.5 Unoriented Surface Normal Estimation . . . 38
3.5 Conclusion . . . 40
3.6 Supplementary Materials . . . 42
3.6.1 Gradient Descent for Decoding Function Encoding . . . 42
3.6.2 Proof of Asymptotic Sample Invariance . . . 44
3.6.3 Proof of Isometry . . . 47

Chapter 4: A Linear Time and Space Local Point Cloud Geometry Encoder 49
4.1 Introduction . . . 49
4.2 Related Work . . . 52
4.2.1 Local Geometry Encoder . . . 52
4.2.2 Point Cloud Architectures . . . 54
4.3 Methodology . . . 55
4.3.1 Problem Definition and Main Theorems . . . 55
4.3.2 Pointwise Local Geometry Encoder . . . 58
4.3.3 Dense Local Geometry Encoder . . . 62
4.3.4 VecKM in Point Cloud Deep Learning . . . 65
4.4 Experiments . . . 66
4.4.1 Normal Estimation on PCPNet Dataset . . . 66
4.4.2 Classification on ModelNet40 Dataset . . . 70
4.4.3 Part Segmentation on ShapeNet Dataset . . . 72
4.4.4 Semantic Segmentation on S3DIS Dataset . . . 73
4.5 Ablation Studies . . . 74
4.6 Conclusion . . . 76

Chapter 5: Learning Normal Flow Directly From Event Neighborhoods 78
5.1 Introduction . . . 78
5.2 Related Work . . . 82
5.2.1 Event-Based Optical Flow Estimation . . . 82
5.2.2 Point-Based Networks for Event Cameras . . . 83
5.2.3 Normal Flow and its Applications . . . 83
5.3 Methodology . . . 85
5.3.1 Problem Definition and Overview . . . 85
5.3.2 Local Events Encoder . . . 86
5.3.3 Normal Flow Learning with a Two-Term Loss . . . 89
5.3.4 Improve Transferability by Data Augmentation . . . 90
5.3.5 Inference with Uncertainty Quantification . . . 91
5.3.6 Egomotion Estimation from Normal Flow . . . 92
5.4 Experiments on Normal Flow Estimation . . . 94
5.4.1 Evaluation on MVSEC . . . 96
5.4.2 Evaluation on EVIMO2 . . . 97
5.4.3 Evaluation on DSEC . . . 98
5.5 Ablation Studies . . . 99

5.5.1 Effect of Motion Field Loss 99
5.5.2 Effect of Uncertainty Quantification 100
5.5.3 Runtime and Memory Usage 101
5.6 Experiments on Egomotion Estimation 102
5.7 Conclusion and Future Work 103
5.8 Supplementary Materials 105
5.8.1 List of Flow Prediction Videos 105
5.8.2 Dataset Preprocessing 105
5.8.3 Implementation Details of Our Model 106
5.8.4 Per-Scene Normal Flow Evaluation on EVIMO2 107
5.8.5 Per-Scene Egomotion Evaluation on EVIMO2 107

Chapter 6: Conclusion and Future Work 111

Bibliography 113

List of Tables

3.1 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. 36

3.2 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. 39

4.1 Normal estimation RMSE on the PCPNet dataset. 68

4.2 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. 72

4.3 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. 73

4.4 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. 75

4.5 Ablation study on the selection of the parameters α and β in Theorem 6, in the context of ModelNet40 classification experiment. Numbers greater than 92.5% are bolded. 76

5.1 Quantitative results on MVSEC. The estimators are classified into model-based (MB), supervised learning (SL), self-supervised learning (SSL), frame-based (F), point-based (P). They are trained on MVSEC (M), DSEC (D), EVIMO-`imo` (E). 97

5.2 Quantitative results on the first five scenes of EVIMO2-imo's evaluation set. The complete table is in Appendix 5.8.4. 98
5.3 Comparison between the estimator trained with our motion field loss function and the one trained with the standard norm-plus-direction loss function. Using our motion field function significantly improves the model's performance. 100
5.4 Performance of the estimator under varying ensemble sizes and uncertainty thresholds. Increasing the ensemble size generally enhances results, with 4 to 6 ensembles already providing sufficiently stable outcomes. 101
5.5 Computational cost of our normal flow estimator. 102
5.6 Statistics of event density of each dataset. 102
5.7 Egomotion estimation error on EVIMO2-sfm. 104
5.8 Per-scene normal flow evaluation on EVIMO2-imo split. 108

List of Figures

1.1 Point sets in real-world. 1

1.2 Mathematical modeling of point sets. The point set is viewed as random samples from an underlying function f, which characterizes the object shape or event surface. We aim to construct a consistent representation of function f using samples that may be corrupted with noise or density variation. . . 2

2.1 Approximation of Gaussian kernel with Bochner's theorem. 13

3.1 **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. 17

3.2 (a): How α and β in Eqn. (3.6) 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. 31

3.3 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. 32

3.4 HDFE is a distance preserving transformation. The L2-distance between functions is proportional to the negative similarity between their encodings. 33

3.5 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. 34

3.6 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. 38
3.7 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, $||center(Ball(S_n)) - center(Ball(S))|| \leq d_H(Ball(S_n), Ball(S))$. So the distance between the centers of the two balls goes to 0. 47

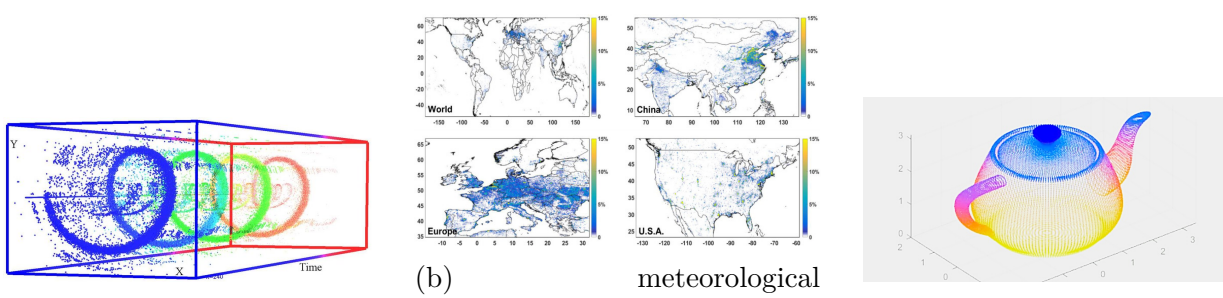
4.1 Our VecKM encoding is descriptive, robust to noise, and efficient in runtime and memory cost. **Upper Left**: Raw VecKM encodings, *without any training*, already capture rich geometric features such as orientations and shapes. **Lower Left**: Under varying levels of noise, VecKM encodings remain 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∼100x faster than existing encoders in wall-clock time and scalable to large point cloud inputs. 50
4.2 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. . 59
4.4 VecKM can be seamlessly integrated into deep point cloud architectures, improving both accuracy and efficiency. 66
4.5 VecKM's robustness to data corruptions. VecKM can reconstruct the local shape under corrupted inputs. The VecKM encodings remain highly similar under data corruptions. 68
4.6 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. . . . 70
4.7 Average RMSE of normal estimation trained with different numbers of layers. 76

5.1 We propose a point-based network for estimating normal flow from raw event data. We discover multiple key advantages of this point-based approach compared with existing learning-based approaches. An event and its neighborhood are first encoded as a fixed-dimensional vector, which is then input to a network trained in a supervised way to predict normal flow. This approach achieves high accuracy while maintaining strong transferability across different domains and datasets. Besides, we demonstrate the usefulness of the estimated normal flow in a new egomotion solver that is shown to remain robust even during aggressive camera motions. 79
5.2 Our point-based method produces accurate and sharp predictions in the presence of independently moving objects, while other methods [1, 2] fail. All models (if learning-based) are trained on DSEC and evaluated on EVIMO2. The flows are displayed in HSV color space, where the hue represents the flow direction, and the brightness represents the flow magnitude. 79
5.3 Uncertainty quantification (UQ) is important for per-event normal flow estimation, as it helps filter out less reliable predictions. For example, the normal flow predictions in Cases 1 and 2 are more reliable compared to those in Cases 3 and 4. 86
5.4 Reconstruction of a density distribution (shown in gray) from VecKM's local events encoding. The reconstructed 3D distribution closely aligns with the original (blue and red) events, demonstrating that VecKM's encoding effectively represents the event data. The examples shown are identical to those in Figure 5.3. 88
5.5 Loss maps and gradient fields of the motion field loss function. Our motion field loss function consists of radial and angular components. Given the GT optical flow $\mathbf{u}$, the radial component guides the predicted flow to lie on the circle with $\mathbf{u}$ as the diameter. The angular component guides the predicted flow to align with $\mathbf{u}$, which prevents the trivial prediction of zero flow. . . . 90
5.6 Visualization of flow prediction on EVIMO2-`imo`. The flows are displayed in HSV color space. Our method effectively preserves the boundary of the IMOs, while other methods fail. 98
5.7 The model can choose between estimating the full optical flow or normal flow depending on the texture of the local region. If the local texture is rich enough (e.g. a corner), the model will estimate full optical flow. If the local texture only contains strong edges, the model will estimate normal flow. . 100
5.8 Comparison of estimated translation velocity produced by maximizing negative depth [3] and our SVM-based method. Our approach yields more accurate predictions. 103
5.9 Screenshot of the flow prediction videos. Each row displays the norm, angle, and flow fields of both ground-truth and predicted flows. The first row visualizes the ground-truth optical flow, while subsequent rows show model predictions trained on each dataset. To illustrate the flow field, we sample 5,000 flow points for visualization. If a pixel is gray, it means the flow prediction has a high uncertainty. 109
5.10 Per-scene egomotion evaluation on EVIMO2 `sfm` split. 110

Chapter 1: Introduction

1.1 Point Sets: Inputs to Vector Kernel Representation

A point set 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. Event cameras, which detect changes in light intensity rather than capturing static frames, produce data streams where each event marks a change in light at a specific pixel, forming a temporal point set. Remote sensing collects meteorological data from various scattered locations, which can be represented as a point set consisting of measurement values and their corresponding geographic coordinates.



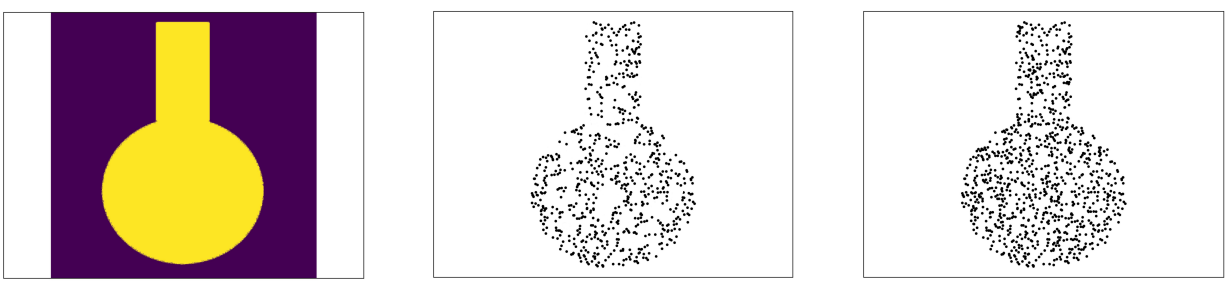
(a) event frames. (b) meteorological measurements. (c) point cloud.

Figure 1.1: Point sets in real-world.

To represent point sets with theoretical support, it is essential to model point sets

mathematically. In this thesis, point sets are considered as samples from a distribution 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 set is viewed as random samples of this function. To construct representations for point sets, we are essentially constructing the representation of the function f using the random samples.

Modeling point sets in this manner provides a clear definition of density and noise. Noise in a point set 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. Figure 1.2 illustrates the mathematical modeling of point sets.



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

Figure 1.2: Mathematical modeling of point sets. The point set is viewed as random samples from an underlying function f, which characterizes the object shape or event surface. We aim to construct a consistent representation of function f using samples that may be corrupted with noise or density variation.

1.2 Downstream Tasks and Requirements of Efficiency and Robustness

In very short, point set representation is particularly valuable when dealing with input data of **variable** size while following a **consistent pattern**.

For example, consider the following machine learning task: predict a city's precipitation based on a collection of humidity measurements. In the training set, each city has an indefinite number of humidity measurements taken at various locations and a ground-truth precipitation value. The challenge lies in mapping these varying numbers of measurements to a single precipitation value. By using point set representation, we can first transform the collection of humidity measurements into a fixed-length vector. Then we can use a standard neural network to map the fixed-length vector to the predicted precipitation. In this case, the measurements are the input data of variable size, and the underlying humidity distribution is the consistent pattern.

Having established the usefulness of point set representation, it is essential to understand why robustness and efficiency are critical in this process.

Robustness. In practice, the random samples encountered during testing will often differ from those used in training. These differences can manifest in two ways: 1) Sample size: Test cities may have fewer or more humidity measurements than those in the training set. 2) Sample distribution: The distribution of humidity measurements in test cities may differ from that of the training cities. If the point set representation method lacks robustness, such variations in sampling can degrade the performance of downstream models. Ensuring robustness helps maintain model accuracy despite discrepancies in sample size and distribution.

Efficiency and Scalability. Real-world point sets can vary dramatically in size—from hundreds of data points (e.g., meteorological measurements) to tens of thousands (e.g., data from event cameras). Therefore, it is crucial that point set representation methods are efficient and scalable, capable of processing large point sets without compromising computational performance.

1.3 Existing Methods and Their Limitations

Existing point set representation methods can be broadly categorized into two categories: point-based and voxel-based methods. As to be discussed, existing methods do not meet the robustness and efficiency requirements.

Point-Based. This is typically the dominant category and is the most widely applied. A typical example is PointNet [4], which transform point sets into fixed-length vectors using trainable networks. Many derived methods are proposed and and can be broadly classified into MLP-based and convolution-based encoders. MLP-based encoders, such as PointNet [4], CurveNet [5], PointMLP [6], utilize multilayer perceptrons to process point clouds and employ max-pooling to generate fixed-length representations. On the other hand, convolution-based encoders, including KPConv [7], PointCNN [8], PointConv [9], SpiderCNN [10], apply point or edge convolutions to achieve similar fixed-length representations.

However, point-based encoders lack inherent robustness against noise and variations in data density. Attempts to overcome these challenges typically involve data augmentation. However, they do not guarantee consistent robustness. Additionally, these methods require substantial computational resources and are not scalable (to be shown in Chapter 4). This

restricts their use in speed-critical tasks like event camera stream processing and in resource-constrained environments like drone navigation.

Voxel-Based Methods. Voxel-based methods assume that point sets lie on a regular grid, enabling the application of convolutional operations to learn point features. A prominent example is the Neural Operator, pioneered by DeepONet [11], which processes regular grids of point sets. This line of research has led to the development of several tools [12, 13, 14, 15, 16], with the Fourier Neural Operator [17] being the most well known for its impressive accuracy and computational efficiency in function-to-function mapping tasks.

When point sets do not naturally align with a regular grid, voxelization is employed to force the data onto a grid structure. This strategy is particularly useful for large point sets where point-based methods face scalability issues. In the event camera community, voxel-based representations are widely used; for instance, E-RAFT [18] voxelizes events and estimates optical flow using 3D convolutions, with many other event-based architectures adopting similar voxel-based strategies [1, 19, 20, 21, 22, 23, 24, 25].

However, voxel-based representations often lack descriptiveness due to the loss incurred by voxelization. In addition, they often suffer significant performance degradation across different domains due to variations in point set density, which can distort voxelization and lead to out-of-distribution issues during convolutional processing.

1.4 Vector Kernel Representation Theory: An Overview

This thesis introduces Vector Kernel Representation (VKR), a novel approach to point set representation. Through a combination of theoretical analysis and practical

applications, we thoroughly explore VKR's potential, demonstrating its effectiveness across various point set applications. In many cases, VKR outperforms existing methods in both accuracy and computational efficiency.

Chapter 2 lays the theoretical foundation of VKR by presenting and proving the Vector Kernel Representation Theorem, which encapsulates the core contributions of this thesis. Chapter 3 outlines the conceptual foundation and motivation behind the development of VKR. It details the inspirations and thought processes that led to its creation. In Chapter 4, we apply VKR to a fundamental point cloud task - local point cloud encoding. The application highlights VKR's computational efficiency, which significantly reduces processing costs compared to existing methods. Chapter 5 extends VKR to event camera data, addressing the normal flow estimation problem and achieving substantial improvements in accuracy and generalizability over current approaches. Finally, Chapter 6 summarizes the key contributions of this work and outlines potential future research directions.

1.5 List of publications

- **Yuan, Dehao**, Levi Burner, Jiayi Wu, Minghui Liu, Jingxi Chen, Yiannis Aloimonos, and Cornelia Fermüller. "Learning Normal Flow Directly From Event Neighborhoods." arXiv preprint arXiv:2412.11284 (2024).

- **Yuan, Dehao**, Cornelia Fermüller, Tahseen Rabbani, Furong Huang, and Yiannis Aloimonos. "A Linear Time and Space Local Point Cloud Geometry Encoder via Vectorized Kernel Mixture (VecKM)." International Conference on Machine Learning (2024).

- **Yuan, Dehao**, Furong Huang, Cornelia Fermüller, and Yiannis Aloimonos. "Decodable and Sample Invariant Continuous Object Encoder." International Conference on Learning Representation (2024).

- Sutor, Peter*, **Dehao Yuan***, Douglas Summers-Stay, Cornelia Fermuller, and Yiannis Aloimonos. "Gluing neural networks symbolically through hyperdimensional computing." International Joint Conference on Neural Networks (2022).

- Chen, Jingxi, Brandon Y. Feng, Haoming Cai, Tianfu Wang, Levi Burner, **Dehao Yuan**, Cornelia Fermuller, Christopher A. Metzler, and Yiannis Aloimonos. "Repurposing Pre-trained Video Diffusion Models for Event-based Video Interpolation." arXiv preprint arXiv: 2412.07761 (2024).

- Mavrogiannis, Angelos, **Dehao Yuan**, and Yiannis Aloimonos. "Discovering Object Attributes by Prompting Large Language Models with Perception-Action APIs." IEEE International Conference on Robotics and Automation (2025).

- Amrouch, Hussam, Mohsen Imani, Xun Jiao, Yiannis Aloimonos, Cornelia Fermuller, **Dehao Yuan**, Dongning Ma, Hamza E. Barkam, Paul R. Genssler, and Peter Sutor. "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, 2022.

Chapter 2: Vector Kernel Representation Theorem

2.1 Vector Kernel Representation Theorem

Theorem 1 (Vector Kernel Representation Theorem)**.** *Let* $\{\mathbf{x}_k\}_{k=1}^n$ *be a point set in* $\mathbb{R}^d$ *and* $f(\mathbf{x}) = \frac{1}{n}\sum_{k=1}^n \exp(-\alpha^2||\mathbf{x}-\mathbf{x}_k||^2/2)$ *be the Gaussian kernel mixture defined by the point set. Then* $f(\mathbf{x})$ *can be represented by* $\mathbf{F} = \frac{1}{n}\sum_{k=1}^n \exp(i\mathbf{x}_k\mathbf{A}) \in \mathbb{C}^D$*, where* $\mathbf{A} \in \mathbb{R}^{d\times D}$ *is a fixed matrix with all elements drawn independently from normal distribution* $\mathcal{N}(0,\alpha^2)$*.*

Vector Kernel Representation Theorem (VKRT) states that the Guassian kernel mixture defined by a point set can be represented by a fixed-length complex vector by a pre-defined operation. "Representation" here means two properties: reconstructable and isometric properties, defined as followed:

Proposition 1 (Reconstruction of Vector Kernel Representation)**.** *Let* $\mathbf{F} = \frac{1}{n}\sum_{k=1}^n \exp(i\mathbf{x}_k\mathbf{A})$ *and* $f(\mathbf{x}) = \frac{1}{n}\sum_{k=1}^n \exp(-\alpha^2||\mathbf{x}-\mathbf{x}_k||^2/2)$*, then* $\frac{1}{D}\langle\exp(i\mathbf{x}\mathbf{A}),\mathbf{F}\rangle \to f(\mathbf{x})$ *as* $D\to\infty$*.*

This means the function value $f(\mathbf{x})$ at any point $\mathbf{x}$ can be reconstructed from the representation $\mathbf{F}$ by a pre-defined operation.

Proposition 2 (Vector Kernel Representation is Isometric)**.** *Let* $\mathbf{F}_1$*,* $\mathbf{F}_2$ *be the representations of Gaussian kernel mixtures* f_1*,* f_2*, then* $\frac{1}{D}\langle\mathbf{F}_1,\mathbf{F}_2\rangle \to \langle f_1, f_2\rangle$ *as* $D\to\infty$*.*

This means the similarity (correlation) of the kernel mixtures is preserved in the encoding.

Proposition 3 (Gifted Property: Equivariance about Translation)**.** *Define*

$$\mathbf{F}\big(\{\mathbf{x}_k\}_{k=1}^n\big) = \frac{1}{n}\sum_{k=1}^{n} \exp(i\mathbf{x}_k\mathbf{A})$$

We have

$$\mathbf{F}\big(\{\mathbf{x}_k - \mathbf{x}_0\}_{k=1}^n\big) = \mathbf{F}\big(\{\mathbf{x}_k\}_{k=1}^n\big) * \exp(-i\mathbf{x}_0\mathbf{A})$$

where $*$ denotes the elementwise multiplication.

Given $\exp\big(i(\mathbf{x}-\mathbf{y})\mathbf{A}\big) = \exp(i\mathbf{x}\mathbf{A}) * \exp(-i\mathbf{y}\mathbf{A})$, the proof is straight-forward. In Chapter 4, we exploit this gifted property to solve a fundamental task in point cloud, namely local point cloud geometry encoding.

2.2 Example Usage of Vector Kernel Representation Theorem

Before proving VKRT, let's have an idea how it is useful. VKRT states that if we can somehow describe our point set by a Gaussian kernel mixture, then we automatically obtain a representation of the point set. Here are potential approaches to derive Gaussian kernel mixture from point sets:

1. Decision function of a binary classification estimated from support vector machine.

2. Decision function of a one-class support vector classifier.

3. Function fitted by support vector regression.

Let's take an example to see how we may apply the third principle in a practical scenario. Using other principles is similar. Suppose there are several meteorological measurement stations in a city, and their coordinates are $\{(x_k, y_k)\}_{k=1}^n$. They measure the temperature at a moment and obtain $(T_1, T_2, \cdots, T_n)$. How do we represent the temperature distribution in this city?

Using the third principle, we first fit a support vector regression (SVR) with Gaussian kernel, where the input is coordinate and the target is temperature. The resulting function is

$$f(x, y) = \alpha_0 + \underbrace{\sum_{k=1}^{n} \alpha_k \exp(-\frac{(x - x_k)^2 + (y - y_k)^2}{\sigma^2})}_{\text{represented by VKRT}}$$

where $\{\alpha_k\}_{k=0}^n$ is solved by the SVR. Naturally, $f(x, y)$ describes the temperature distribution. Now the second term can be represented by VKRT. We can obtain the final representation by appending α_0 to it. Now we successfully represent scattered measurements of temperature into a fixed-length vector. If we need to use it as inputs to predict other variables, this can be easily done.

Now, we proceed to prove VKRT. To provide a more comprehensive understanding, we present two proofs. The first proof is based on the Vector Function Architecture [26], which originally inspired the formulation of this theorem. The second proof relies on the Fourier transform, which I personally find easier to understand.

2.3 Proof From Vector Function Architecture

We use a lemma named **Bochner' s Theorem**, which 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 d-by-D matrix with all elements drawn from normal distribution.

Lemma 1 (Bochner' s Theorem)**.** *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\mathbf{A} \in \mathbb{R}^{d \times D}$. *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 $\mathbf{a} \in \mathbb{R}^d$ where $\mathbf{a} \sim \mathcal{N}(\mathbf{0}, \alpha^2\mathbf{I}_{d\times d})$ be one column of the matrix $\mathbf{A}$, we claim that $\mathbb{E}\big[\Re\big(e^{i\mathbf{a}\cdot(\mathbf{x}-\mathbf{y})}\big)\big] = \mathcal{G}_\alpha(\mathbf{x}, \mathbf{y})$:

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

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:

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

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 states that a function $f(x) = \sum_{k=1}^{n}\alpha_k K(x, x_k)$ can be represented by $\mathbf{F} = \sum_{k=1}^{n}\alpha_k\mathbf{v}(x_k)$, where $K(x,y)$ is a kernel and approximated by $\langle\mathbf{v}(x),\mathbf{v}(y)\rangle$. VFA proves the reconstructable and isometric property under this general formulation. With Bochner's theorem, it is clear that the choice of K and $\mathbf{v}$ here are $\exp(-\alpha^2(x-y)^2/2)$ and $\exp(ixA)$. Therefore, VKRT is proved.

2.4 Proof From Fourier Transform

We alternatively prove VKRT using Fourier transform. Specifically, we will show the Fourier transform of $\exp\big(-\alpha^2||\mathbf{x}-\mathbf{x}_k||^2/2\big)$ is $\exp(-i\mathbf{x}_k\cdot\xi)$ multiplied by a constant, where

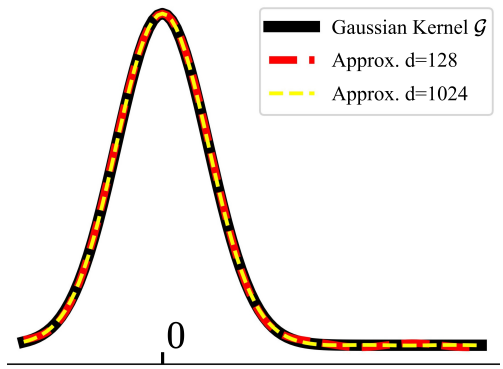


Figure 2.1: Approximation of Gaussian kernel with Bochner's theorem.

both $\mathbf{x}_k$ and ξ are in $\mathbb{R}^d$.

We seek the Fourier transform of the function

$$f(\mathbf{x}) = \exp\left(-\frac{\alpha^2}{2}\|\mathbf{x} - \mathbf{x}_k\|^2\right),$$

where $\mathbf{x}, \mathbf{x}_k \in \mathbb{R}^d$, and α is a positive scalar.

The Fourier transform of f is computed by

$$\mathcal{F}[f](\xi) = \int_{\mathbb{R}^d} \exp\left(-\frac{\alpha^2}{2}\|\mathbf{x} - \mathbf{x}_k\|^2\right) e^{-i\xi\cdot\mathbf{x}} d\mathbf{x}$$

Define $\mathbf{y} = \mathbf{x} - \mathbf{x}_k$, so that $d\mathbf{y} = d\mathbf{x}$, and rewrite the integral as

$$\mathcal{F}[f](\xi) = \int_{\mathbb{R}^d} \exp\left(-\frac{\alpha^2}{2}\|\mathbf{y}\|^2\right) e^{-i\xi\cdot(\mathbf{y}+\mathbf{x}_k)} d\mathbf{y}.$$

Since $\xi \cdot (\mathbf{y} + \mathbf{x}_k) = \xi \cdot \mathbf{y} + \xi \cdot \mathbf{x}_k$, we factor out the phase term:

$$\mathcal{F}[f](\xi) = e^{-i\xi\cdot\mathbf{x}_k} \int_{\mathbb{R}^d} \exp\left(-\frac{\alpha^2}{2}\|\mathbf{y}\|^2\right) e^{-i\xi\cdot\mathbf{y}} d\mathbf{y}.$$

The integral

$$I = \int_{\mathbb{R}^d} \exp\left(-\frac{\alpha^2}{2}\|\mathbf{y}\|^2\right) e^{-i\xi\cdot\mathbf{y}} d\mathbf{y}$$

is a well-known Gaussian Fourier transform result:

$$I = \left(\frac{2\pi}{\alpha^2}\right)^{d/2} \exp\left(-\frac{\|\xi\|^2}{2\alpha^2}\right).$$

Thus, the Fourier transform of $f(x)$ is:

$$\mathcal{F}[f](\xi) = \underbrace{\left(\frac{2\pi}{\alpha^2}\right)^{d/2} \exp\left(-\frac{\|\xi\|^2}{2\alpha^2}\right)}_{\text{constant}} e^{-i\xi\cdot\mathbf{x}_k}.$$

Now if we sample ξ from normal distribution for D times, then we obtain

$$\left[e^{-i\xi_1\cdot\mathbf{x}_k}, e^{-i\xi_2\cdot\mathbf{x}_k}, \cdots, e^{-i\xi_D\cdot\mathbf{x}_k}\right] = \exp(i\mathbf{x}_k\mathbf{A})$$

which is the representation we desire. In addition, Fourier transform is known to be invertible and isometric. Therefore, VKRT is proved.

2.5 A Short Conclusion

With the Vector Kernel Representation Theorem in mind, we are ready to explore more extensions of it in the next three chapters. Chapter 3 outlines the conceptual foundation and motivation behind the development of VKRT. It details the inspirations and thought processes that led to its creation. Chapter 4 demonstrates the application of

VKRT to a fundamental point cloud task: local point cloud encoding. This application leverages the translation equivariance to enhance efficiency and scalability. Chapter 5 extends the application of VKRT to event cameras, addressing the normal flow estimation problem, greatly improving accuracy and generalizability.

Chapter 3: Decodable and Sampling Invariant Continuous Object Encoder

3.1 Introduction

Continuous objects are objects that can be sampled with arbitrary distribution and density. Examples include point clouds [27], event-based vision data [28], and sparse meteorological data [29]. 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. 3.1 illustrates the properties and their motivations.

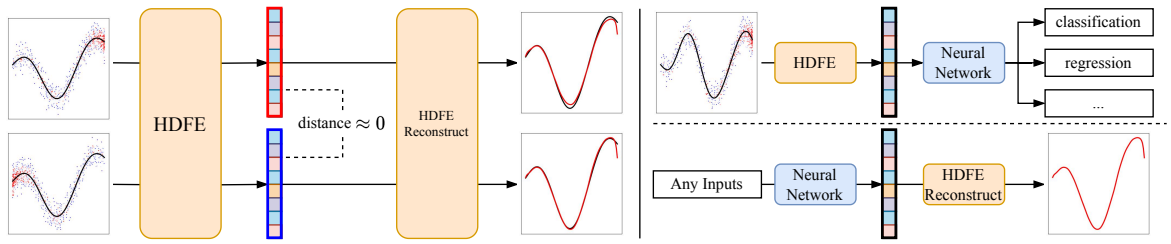


Figure 3.1: **Left**: HDFE encodes continuous objects into fixed-length vectors without any training. The encoding is not affected by the distribution and size with which the object is sampled. The encoding can be decoded to reconstruct the continuous object. **Right**: Applications of HDFE. HDFE can 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.

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, [30] uses a 3D-CNN to process voxelized point clouds, [31] 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 [32] and the neural operator [17]. 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 [33] work for non-linear regression, classification, etc. But they

do not provide an explicit representation of the function. PointNet [4] 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.

Currently, only the vector function architecture (VFA) [26] 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%.

3.2 Related Work

3.2.1 Mesh-grid-based framework

These methods operate on continuous objects (functions) defined on mesh grids with arbitrary resolution. They enjoy discretization invariance, but they do not receive sparse samples as input.

Fourier transform (FT) can map functions from their time domains to their frequency domains. By keeping a finite number of frequencies, FT can provide a vector representation of functions. 1D-FT has been a standard technique for signal processing [32] and 2D-FT has

been used for image processing [34]. FT is also incorporated into deep learning architectures [35, 36]. However, FT is not scalable since the n-dimensional Fourier transform returns an n-dimensional matrix, which is hard to process when n gets large.

Neural operator is a set of toolkits to model the mapping between function spaces. The technique was pioneered in DeepONet [11] and a series of tools were developed [12, 13, 14, 15, 16] for studying the problem. The most well-known work is the Fourier Neural Operator (FNO) [17]. The approach showed promising accuracy and computational efficiency. Though proposed in 2020, FNO is still the first choice when mapping between function spaces [37, 38, 39]. Despite their success, neural operators lack explicit function representations and their application is limited to mappings between function spaces.

3.2.2 Sparse framework

These methods, including HDFE, work with sparse samples from continuous objects.

PointNet [4] is a neural network architecture for processing point cloud data. It uses multi-layer perceptrons to capture local features of every point and then aggregates them into a global feature vector that's invariant to the order of the input points. PointNet and its variation [40, 41, 42, 43, 44, 45, 46] have been widely applied to sparse data processing, for example, for object classification [46, 47], semantic segmentation [48, 49], and object detection [50, 51] with point cloud input. However, PointNet does not produce a decodable representation. Specifically, after encoding a point cloud with PointNet, it is difficult to decide whether a point is drawn from the point cloud distribution. Besides, PointNet is also sensitive to perturbations in the input point cloud.

Kernel methods [33] are a type of machine learning algorithm that transforms data into a higher-dimensional feature space via a kernel function, such as the radial basis function (RBF) [52], which can capture nonlinear relationships. Though kernel methods can predict function values at any query input (i.e. decodable) and the prediction is invariant to the size and distribution of the training data, kernel methods do not produce an explicit representation of functions, so they are only used for fitting functions but not processing or predicting functions.

Vector function architecture VFA [26] encodes a function of the form $f(x) = \sum_k \alpha_k \cdot K(x, x_k)$ into a vector, where $K : X \times X \to \mathbb{R}$ is a kernel defined on the input space. VFA and HDFE share a similar high-level idea. They both map the samples to high-dimensional space and compute the weighted average in that space. However VFA determines the weight by relying on the assumption of the functional form. HDFE, on the other hand, uses iterative refinement to solve the weights. The iterative refinement coupled with the binding operation relaxes the assumption required by VFA and enables encoding functions across a larger class of inputs. In addition, VFA is limited to empirical experiments such as non-linear regression and density estimation, without practical applications. In comparison, HDFE is demonstrated to be applicable to real-world problems.

3.3 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 $\{(x_i, f(x_i))\}$ 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 2 (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. 3.3.1 introduces how HDFE encodes explicit functions. Sec. 3.3.2 generalizes HDFE to implicit function encoding. Sec. 3.3.3 realizes HDFE for vector-valued function encoding. Finally, Sec. 3.3.4 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 3.3.4.

3.3.1 Explicit Function Encoding

Encoding HDFE is inspired by the methodology of hyper-dimensional computing (HDC) [53], 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:

$$\mathbf{F} = \sum_i w_i \cdot E(x_i, y_i) \tag{3.1}$$

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. 3.3.4. Denoting the similarity between vectors as $\langle \cdot, \cdot \rangle$, the requirement can be formulated as:

$$\langle E(x, y), E(x', y') \rangle \begin{cases} \approx 1 & d_X(x, x') < \epsilon_0 \\ \text{decays to 0 quickly} & d_X(x, x') > \epsilon_0 \end{cases} \tag{3.2}$$

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. (3.2) 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:

$$\hat{y_0} = \text{argmax}_{y \in Y} \langle \mathbf{F} \oslash E_X(x_0), E_Y(y) \rangle \tag{3.3}$$

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.

$$\mathbf{F} \oslash E_X(x_0) = \sum_i w_i \cdot \big[E_X(x_i) \otimes E_Y\big(f(x_i)\big) \oslash E_X(x_0)\big]$$

$$= \underbrace{\sum_{d(x_0,x_i)<\epsilon_0} w_i \cdot E_Y\big(f(x_i)\big)}_{\approx E_Y(f(x_0))} + \underbrace{\sum_{d(x_0,x_i)>\epsilon_0} w_i \cdot [E_X(x_i) \oslash E_X(x_0)] \otimes E_Y(f(x_i))}_{\text{noise, since orthogonal to } E_Y(f(x_0))}$$

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. 3.6.1 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:

$$\mathbf{F} = normalize\Big(\sum_i w_i \cdot \big[E_X\big(x_i\big) \otimes E_Y\big(f(x_i)\big)\big]\Big) \tag{3.4}$$

Iterative refinement for sample distribution invariance In Eqn. (3.4), 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

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

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. 3.6.2.

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
$\mathbf{F} = \sum_i w_i \cdot z_i$

3.3.2 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. (3.5), where the weights w_x are determined by the iterative refinement:

$$\mathbf{F}_{f=0} = normalize\Big(\sum_{x:f(x)=0} w_x \cdot E_X(x)\Big) \tag{3.5}$$

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. (3.4), we can obtain Eqn. (3.5). 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.

3.3.3 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. (3.4). 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:

$$E_X(\mathbf{x}) := \exp\left(i \cdot \alpha \frac{\Phi \mathbf{x}}{m}\right) \qquad E_Y(y) := \exp\left(i\beta\Psi y\right) \tag{3.6}$$

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

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

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 [54], 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. (3.2)) can be adjusted easily through manipulating α. Fig. 3.2a 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. 3.2a, 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.

3.3.4 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. 3.6.2, 3.6.3 and verify the claims with empirical experiments in Section 3.4.1. We include several empirical experiments of HDFE in Sec. 3.4.2, 3.4.3, including the cost of the iterative refinement and one-shot refinement, and the effectiveness of sample invariance in a synthetic regression problem.

Theorem 3 (Sample Invariance)**.** *HDFE is asymptotic sample invariant (**Definition** 2).*

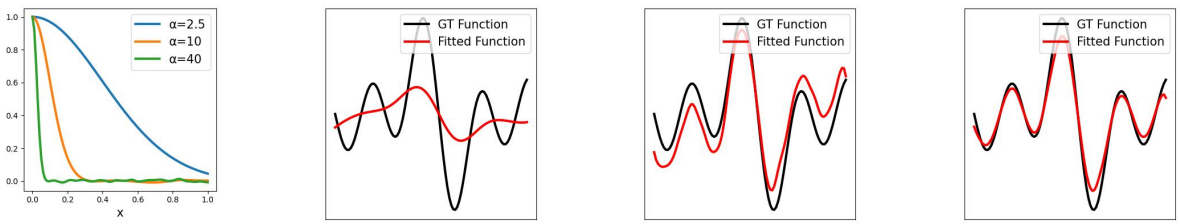
HDFE being sample invariant ensures functions realized with different sampling schemes are treated invariantly. The proof is in Supp. 3.6.2.

Theorem 4 (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. 3.6.3.

Effect of receptive field Fig. 3.2 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. 3.2b), the high-frequency components will be missed by HDFE. When using a small receptive field (Fig. 3.2c), 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. 3.2d).



(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 3.2: (a): How α and β in Eqn. (3.6) 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.

3.4 Experiment

In this section, we present two applications of HDFE. Sec. 3.4.1 verifies the properties claimed in Sec. 3.3.4 with empirical experiments. Sec. 3.4.2 analyzes the computation cost of the refinements. Sec. 3.4.3 examines the efficacy of the sample invariance through a synthetic function regression problem. Sec. 3.4.4 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. 3.4.5, 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.

3.4.1 Empirical Experiment of HDFE

Sample Invariance. In Fig. 3.3, 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. 3.3 Mid). However, after the tuning process, the function vector becomes immune to the sample distribution (Fig. 3.3 Right).

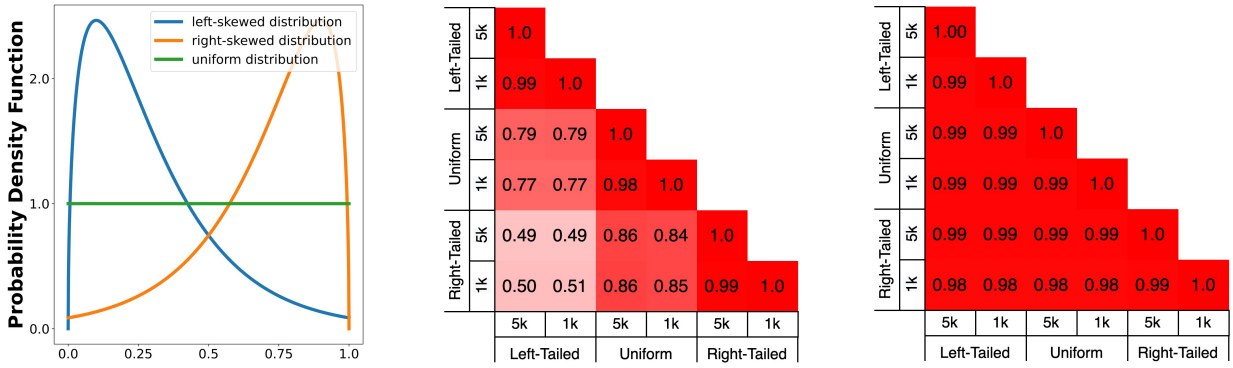


Figure 3.3: HDFE is invariant of sample distribution and sample size. **Left**: Three distributions where the function samples are drawn from. For each distribution, the sample size is either 5000 or 1000. **Mid, Right**: Similarity among the function vectors generated by the six sets of function samples, before and after the function vector tuning process, respectively.

Isometry In Fig. 3.4, 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 4.

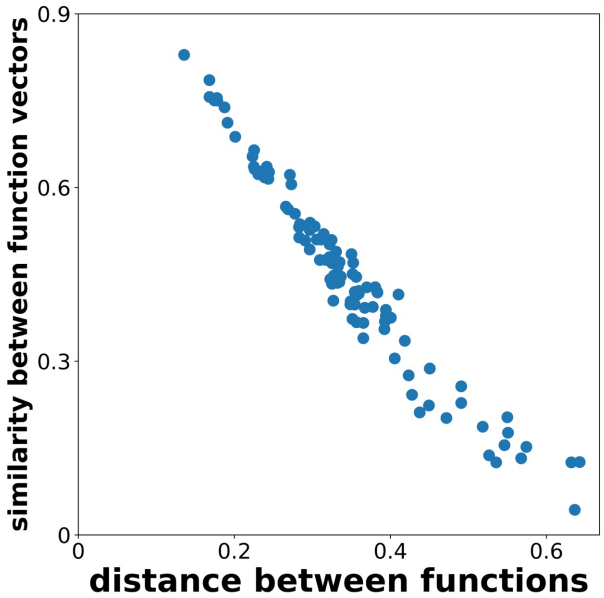


Figure 3.4: HDFE is a distance preserving transformation. The L2-distance between functions is proportional to the negative similarity between their encodings.

3.4.2 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

$$f(x) = \frac{1}{2} + \frac{1}{8}\sum_{k=1}^{4} a_k \sin(2\pi k x) \tag{3.7}$$

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 3.5 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).

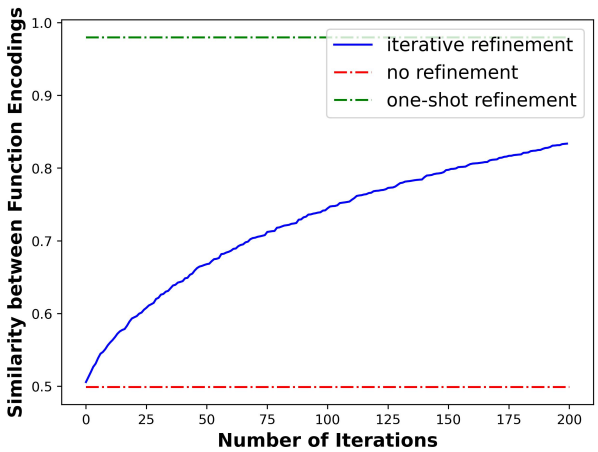


Figure 3.5: When encoding the same function under two different sample distributions, one-shot refinement can approximate the sample invariant function encoding well. It takes 75/90/1500 ms to encode 5000 samples on a CPU, and 7.5/8.0/250 ms on an NVIDIA Titan-X GPU when performing no refinement/one-shot refinement/200-step iterative refinement.

3.4.3 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. (3.7), 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 3.1 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 3.1: 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

3.4.4 PDE Solver

Several neural networks have been developed to solve partial differential equations (PDE), such as the Fourier neural operator [17]. 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 [55] 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 [56] and 2d Darcy Flow [57] for evaluating our method. The error is measured by the absolute distance between the predicted solution and the ground-truth solution. The benchmark [14] 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 [58]; MGKN: Multipole Graph Neural Operator [14]; FNO: Fourier Neural Operator [17].

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. 3.6 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 3.4.5. Despite the presence of reconstruction error, **the reconstruction error can be reduced by increasing the embedding dimension**, as shown in Figure 3.6 (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

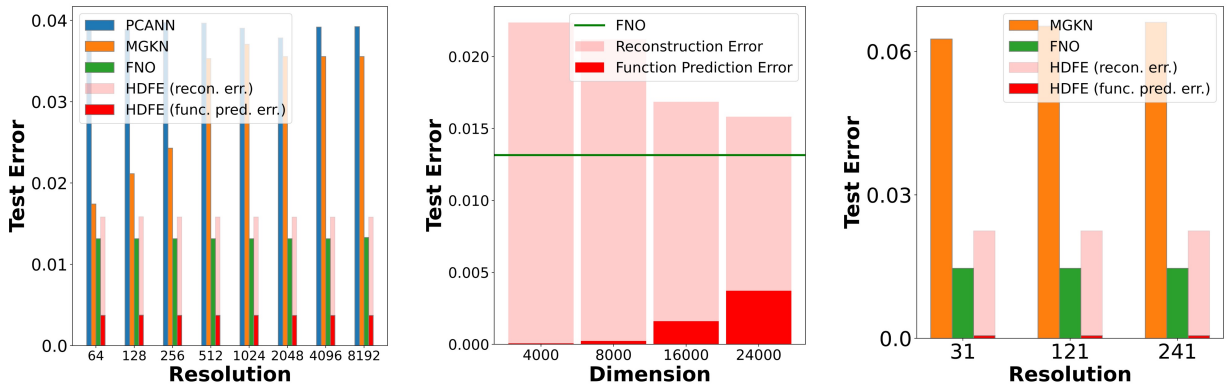


Figure 3.6: HDFE solves a PDE by predicting the encoding of its solution and then reconstructing at points, so the error consists of a function encoding prediction error and a reconstruction error. **Left**: Prediction error of different methods under different testing resolutions, evaluated on the 1d 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.

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.

3.4.5 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 [59], which is a vanilla PointNet [4] architecture. We replace the PointNet with our HDFE appended with a deep complex network [55]. In the second baseline, we incorporate HDFE into HSurf-Net [60], 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 [59] and FamousShape [61] 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 3.2 reports normal angle RMSE comparison with the baselines on PCPNet and FamousShape.

Table 3.2: 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 [59]	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 [60]	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 [60], 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.

3.5 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.

3.6 Supplementary Materials

3.6.1 Gradient Descent for Decoding Function Encoding

In Eqn. (3.3), 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. (3.6)), 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:

$$\hat{y}_0 = \text{argmax}_{y \in (0,1)} \langle \mathbf{z}, \exp(i\Psi y) \rangle \tag{3.8}$$

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 Eqn. (3.6), $E_Y(y) = \exp(i\beta\Psi y)$. 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 Eqn. (3.8) 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:

$$\frac{d}{dy}\langle \mathbf{z}, \exp(i\Psi y)\rangle = \frac{1}{N^2}\sum_{p=1}^{N}\sum_{q=1}^{N} a_p a_q (\Psi_p - \Psi_q) \sin\left[(\theta_p-\theta_q)-(\Psi_p-\Psi_q)y\right] \tag{3.9}$$

In practice, N can be a large number like 8000. Computing the gradient by Eqn. (3.9) 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 3 Gradient Descent for Decoding Function Encoding

Input: $\mathbf{F}$, x_0, E_X, Ψ

▷ $\mathbf{F}$ is the function encoding. x_0 is the query point.

▷ E_X is the input mapping. Ψ 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 $|S| = 500$.

$g = |S|^{-2} \cdot \sum_{p,q \in S} a_p a_q (\Psi_p - \Psi_q) \sin\left[(\theta_p - \theta_q) - (\Psi_p - \Psi_q) y\right]$

$y \leftarrow y - \alpha \cdot g$

▷ α is the learning rate.

end while

3.6.2 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 3.7 sketches the proof from a high-level viewpoint. Recall the definition of asymptotic sample invariance (definition 2):

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

$$\mathbf{F}_n = \text{argmax}_{||z||=1} \min_{i=1}^{n} \langle z, E(x_i, f(x_i))\rangle \tag{3.10}$$

where $E(x, f(x))$ is defined at Eqn. (3.6) 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 $z \leftarrow z+\alpha\cdot\text{argmin}_\text{i}\langle z, E(x_i, f(x_i))\rangle$, which aligns with the iterative refinement in the paper.

Then we will prove Eqn. (3.10) produces a sample invariant encoding by proving $\mathbf{F}_n$ converges to

$$\mathbf{F}_\infty = \text{argmax}_{||z||=1} \min_{x\in X}\langle z, E(x, f(x))\rangle \tag{3.11}$$

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 Eqn. (3.6), $E(x, y) = \mathcal{F}^{-1}(e^{i(\Phi x+\Psi y)})$, 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. (3.10) is equivalent to

$$\mathbf{F}_n = \text{argmin}_{||z||=1} \max_{i=1}^{n} ||z - E(x_i, f(x_i))|| \tag{3.12}$$

Note that Eqn. (3.12) implies $\mathbf{F}_n$ **is the center of the smallest ball containing**

S_n:

$$\mathbf{F}_n = center(Ball(S_n)) \tag{3.13}$$

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, $||center(Ball(S_n)) - center(Ball(S))|| \leq d_H(Ball(S_n), Ball(S))$. Therefore, $||\mathbf{F}_n - \mathbf{F}_\infty|| \leq d_H(Ball(S_n), Ball(S))$, which decays to 0 as $n \to \infty$. $\square$

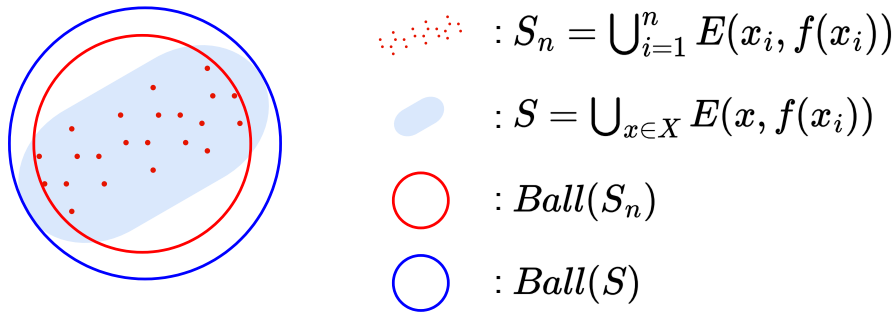


Figure 3.7: Proof of asymptotic sample invariance (overview). $Ball(S)$ and $Ball(S_n)$ are the smallest ball containing S and S_n. As $n \to \infty$, the Hausdorff distance between the two balls goes to zero with probability one. From elementary geometry, $||center(Ball(S_n)) - center(Ball(S))|| \leq d_H(Ball(S_n), Ball(S))$. So the distance between the centers of the two balls goes to 0.

3.6.3 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 1. $\langle x \otimes y, x \otimes z \rangle = \langle y, z \rangle$.

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.

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

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. □

Chapter 4: A Linear Time and Space Local Point Cloud Geometry Encoder

4.1 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 [62] are manually defined features based on domain expertise, and learnable encoders require computationally expensive processing through trainable structures such as multi-layer perceptrons (MLP) [4, 6] or convolutions [7, 8].

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 4.1 (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

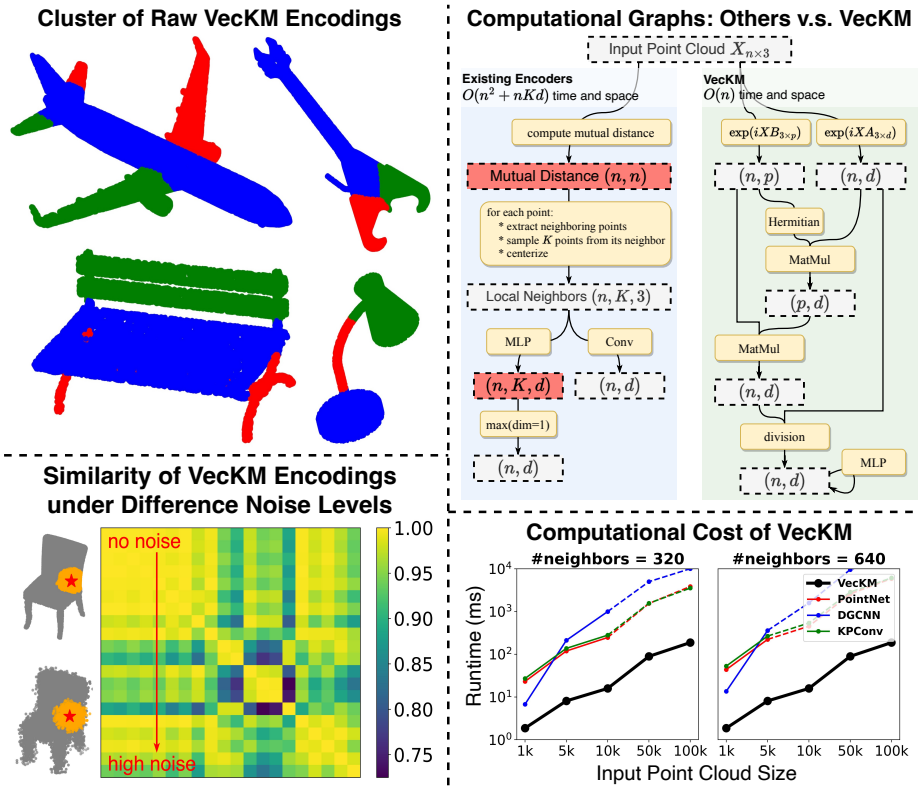


Figure 4.1: Our VecKM encoding is descriptive, robust to noise, and efficient in runtime and memory cost. **Upper Left**: Raw VecKM encodings, *without any training*, already capture rich geometric features such as orientations and shapes. **Lower Left**: Under varying levels of noise, VecKM encodings 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∼100x faster than existing encoders in wall-clock time and scalable to large point cloud inputs.

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 [63, 64], 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++ [41] and transformers [65]. 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.

4.2 Related Work

4.2.1 Local Geometry Encoder

The initial processing of raw point cloud data, an unordered collection of points, typically begins with the extraction of local geometric features. This step is essential before any further processing can occur. Existing methods for encoding local geometry can generally be categorized into two groups: hand-crafted features and learning-based encoders. All the encoders require grouping point clouds into neighborhoods.

Hand-Crafted Features are manually defined features that describe the local geometry. Domain expertise of the point cloud dataset and the task of interest are usually needed for constructing those features. We refer the reader to Han et al. [62] for a comprehensive survey of hand-crafted features.

Histogram-based features represent a significant category within hand-crafted features,

which transform the local point cloud into specific coordinate systems such as Cartesian [66], polar [67], star-shaped [68]. Then the coordinate system is quantized, and the local point cloud is binned accordingly. The resulting feature is formed by concatenating the histograms. While these features induce minimal information loss, the resolution of the point cloud affects the quality of such features.

Statistics-based features form another category within hand-crafted features, which construct statistical descriptors from geometric parameters. Examples include eigenvalues [69], covariance [70], normal orientation distribution [71], angles between normals [72], local umbrella shapes [73]. These types of features can include rich geometric features and easily achieve rotation invariance. But they tend to be lossy and are often not robust to noise and variations in density.

Learning-Based Encoders transform the raw local point clouds into fixed-length vectors through trainable networks. These encoders can be broadly categorized into MLP-based encoders and convolution-based encoders.

MLP-based encoders use MLPs to transform the local point cloud and use max-pooling to cast the point cloud into a fixed-length vector. These operations can be performed repeatedly to retrieve deep point cloud features [41]. Examples of MLP-based encoders include PointNet [4], CurveNet [5], PointMLP [6]. MLP-based encoders are usually faster to compute than convolution-based ones. But they require computing an intermediate step of (n, K, d) to perform the max-pooling operation. So they induce high memory cost when the input size n and the neighborhood size K is large.

Convolution-based encoders use point or edge convolution to transform the local point cloud into a fixed-length vector. Examples include KPConv [7], PointCNN [8], PointConv

[9], SpiderCNN [10]. Convolution-based encoders are more expensive to compute, but they do not encounter the memory bottleneck faced by MLP-based encoders.

VecKM is both a hand-crafted feature and a learning-based encoder. It not only captures the geometric features, but also faithfully encodes the point distribution. This duality allows VecKM to leverage the strengths of both approaches.

4.2.2 Point Cloud Architectures

We describe two major families of architectures for processing point clouds: PointNet++ and transformers. VecKM, as to be shown later, is compatible with both architectures.

PointNet++ [41] utilizes hierarchical neural layers to capture fine geometric details at multiple scales. Within each layer, PointNets are utilized to transform the features. Many works have been done to improve the architecture. Examples include using different learning-based local geometry encoders, as introduced in Section 4.2.1, and improving the neighborhood grouping strategies [5, 46]. PointNet++ and its derivatives tend to be faster than but less accurate than transformers.

Transformers. Given their success on a wide variety of vision tasks, along with their tolerance to permutations, many transformer-based models have been proposed for 3D point cloud processing. Models such as PCT [74], 3CROSSNet [75], and Point-BERT [76] apply transformer blocks to individual points to extract global information. Other models such as Point Transformer (PT) [77], Pointformer [78], and the Stratified Transformer [79] process local patches to extract local feature information. However, transformer-based models can suffer from computational and memory bottlenecks [62] as the attention map increases in size.

4.3 Methodology

4.3.1 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 5 (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*

$$E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big) = \frac{1}{n}\sum_{k=1}^{n} \exp\big(i(\mathbf{x}_k - \mathbf{x}_0)\mathbf{A}_{3\times d}\big) \tag{4.1}$$

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 4.3.2, $E_{\mathbf{A}}\big(\mathfrak{N}(\mathbf{x}_0)\big)$ 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 4 (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.* (4.1). $\mathbf{A} \in \mathbb{R}^{3\times d}$ *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 5 (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. (4.1) 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 5 presents the formula for encoding the local geometry around a single point. Proposition 4, 5 assert that VecKM well represents the underlying local geometry. In Section 4.3.2, we will explain the mechanism behind Theorem 5 and prove Proposition 4, 5 in detail.

Dense Local Geometry Encoder With Eqn. (4.1), 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 6 (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*

$$\begin{aligned}
\mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d}) \\
\mathcal{B}_{n\times p} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p}) \\
\mathbf{G}_{n\times d} &= normalize\big((\mathcal{B}\times\mathcal{B}^H\times\mathcal{A})\ ./\ \mathcal{A}\big)
\end{aligned} \tag{4.2}$$

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 4.3.3, computing the dense local geometry encoding using Eqn. (4.2) has almost the same effect as computing the pointwise local geometry encoding using Eqn. (4.1). However, Eqn. (4.2) 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 4.1 (upper right).

Structure of Proof. In Section 4.3.2, we explain the mechanism behind Theorem 5 and prove our assertion that VecKM produces a good representation of the local geometry. In Section 4.3.3, we explain why Eqn. (4.2) has almost the same effect as Eqn. (4.1) and the mechanism behind Theorem 6. In Section 4.3.4, we introduce how to incorporate VecKM encodings into deep point cloud architectures.

4.3.2 Pointwise Local Geometry Encoder

In this section, we introduce why VecKM (Eqn. 4.1) produces a good representation of the local geometry. The key idea, as illustrated in Figure 4.2, 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).

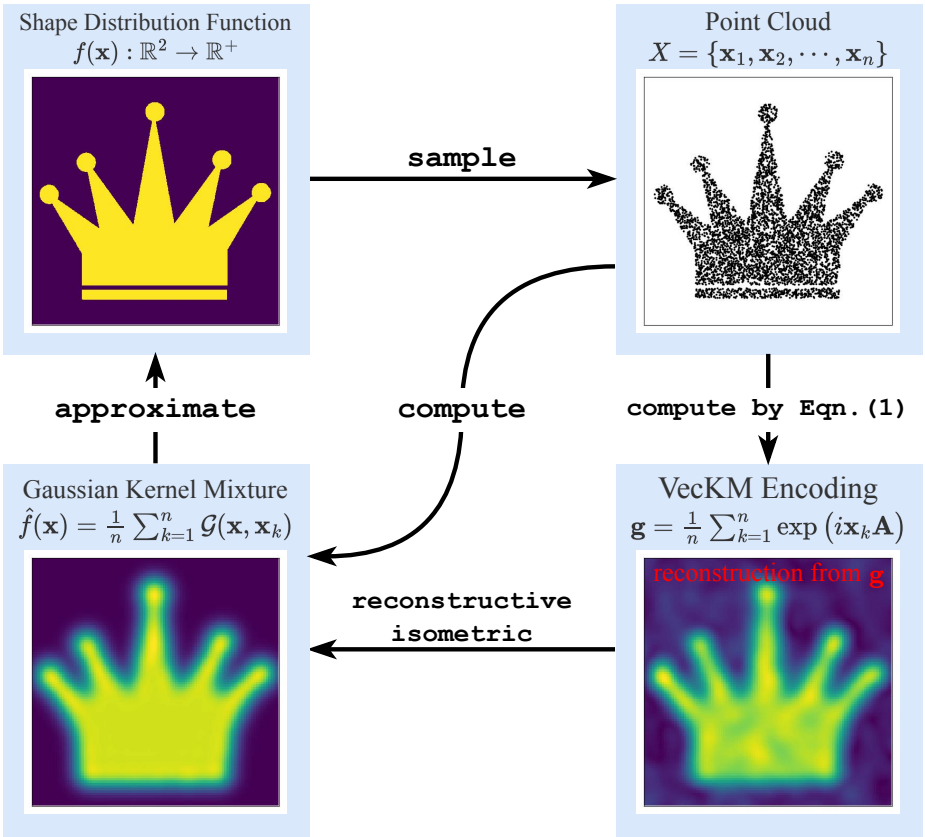


Figure 4.2: Theoretical outline of VecKM illustrated by 2d shapes. A point cloud, sampled from a shape distribution function, is associated with a Gaussian kernel mixture and a corresponding VecKM encoding, where the VecKM encoding is proved to be reconstructive and isometric to the Gaussian kernel mixture. Since the Gaussian kernel mixture can approximate the shape function, the VecKM encoding yields a good representation of the shape.

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

Lemma 2 (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 2 is a corollary from the Bochner's theorem [80, 81], 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 4, 5, 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 [63].

Lemma 3 (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.

Lemma 4 (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 2-4 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. [82] proves that with an appropriately chosen parameter α^2 (defined in Lemma 2), 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 3, 4, we prove Proposition 4, 5, which reveal that VecKM effectively represents the local geometry.

4.3.3 Dense Local Geometry Encoder

In the previous section, we explained why Eqn. 4.1 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. (4.1) 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:

$$\begin{aligned} \mathcal{A}_{n\times d} &= \exp(i\mathbf{X}_{n\times 3}\mathbf{A}_{3\times d}) \\ \mathbf{G}_{n\times d} &= [\mathbf{J}_{n\times n}\mathcal{A}_{n\times d}] \ ./ \ \mathcal{A}_{n\times d} \end{aligned} \tag{4.3}$$

$\mathbf{J}_{n\times n}$ 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. (4.3). The motivation of this substitution is that $\hat{\mathbf{J}}$ can be factorized into a matrix multiplication:

$$\mathcal{B}_{n\times p} = \exp(i\mathbf{X}_{n\times 3}\mathbf{B}_{3\times p})$$

$$\hat{\mathbf{J}}_{n\times n} \leftarrow \mathcal{B} \times \mathcal{B}^H \quad \text{as } p \to \infty$$

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 2. With such approximation, Eqn. (4.3) can be rewritten as

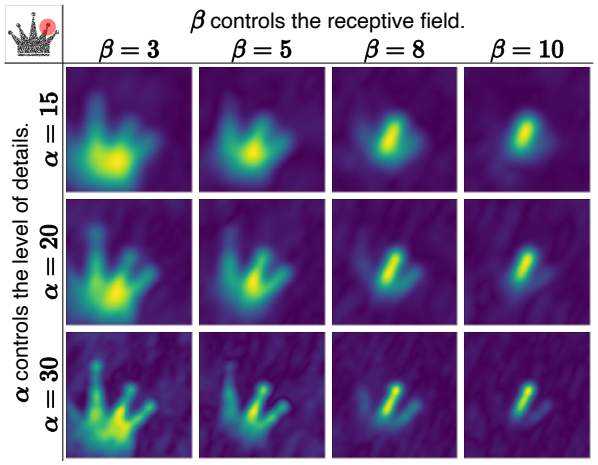
$$\begin{aligned}\mathbf{G}_{n\times d} &= [\hat{\mathbf{J}}_{n\times n}\mathcal{A}_{n\times d}] \,./\, \mathcal{A}_{n\times d} \\ &\approx [\mathcal{B}_{n\times p} \times (\mathcal{B}^H \times \mathcal{A})_{p\times d}] \,./\, \mathcal{A}_{n\times d}\end{aligned}$$

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

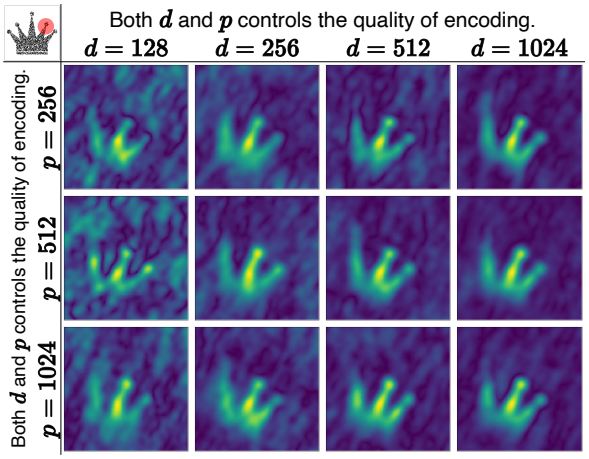
the cost of subsequent processings. Such approximation-and-factorization trick is inspired from [83], which accelerates the attention computation in transformers. This concludes the proof of Theorem 6.

Effect of α and β. We perform a qualitative analysis of the effect of the parameters α and β in Theorem 6. 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 4.5.

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 6.



(b) Effect of the parameters d and p in Theorem 6.

Uniqueness of VecKM. VecKM cannot be established without two important

properties: 1. VecKM embodies a kernel function (Lemma 2); 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 2 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.

4.3.4 VecKM in Point Cloud Deep Learning

VecKM can seamlessly be integrated into widely-used deep point cloud architectures, including PointNet [4], PointNet++ [41], and transformers [74, 77]. Typically, these architectures compute the dense local geometry in the first layer, often utilizing mini-PointNet or sequences of KPConvs [7]. To use VecKM in those architectures, we simply replace the dense local geometry modules with our VecKM encodings.

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 [84] 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 4.4 presents several examples of integrating

VecKM into deep point cloud architectures, which are capable of solving many tasks involving point cloud inputs.

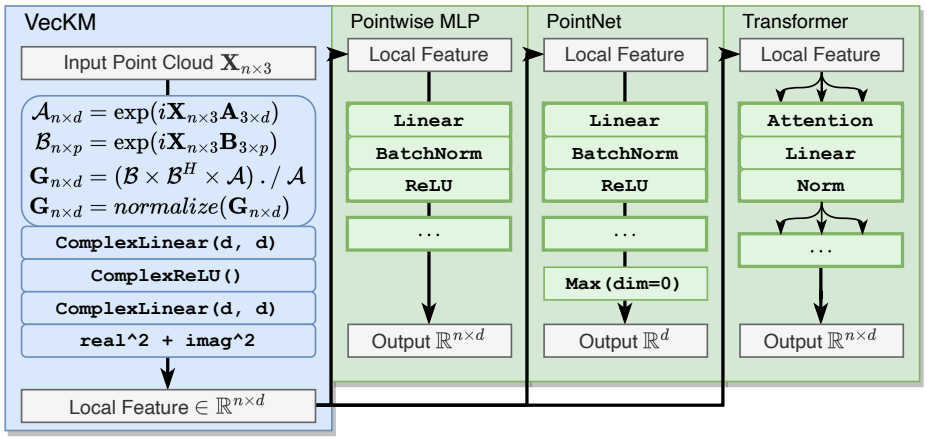


Figure 4.4: VecKM can be seamlessly integrated into deep point cloud architectures, improving both accuracy and efficiency.

4.4 Experiments

4.4.1 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 [59] 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 [4], KPConv [7] and DGCNN [85]. **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 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. (4.2). 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

Table 4.1: 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**

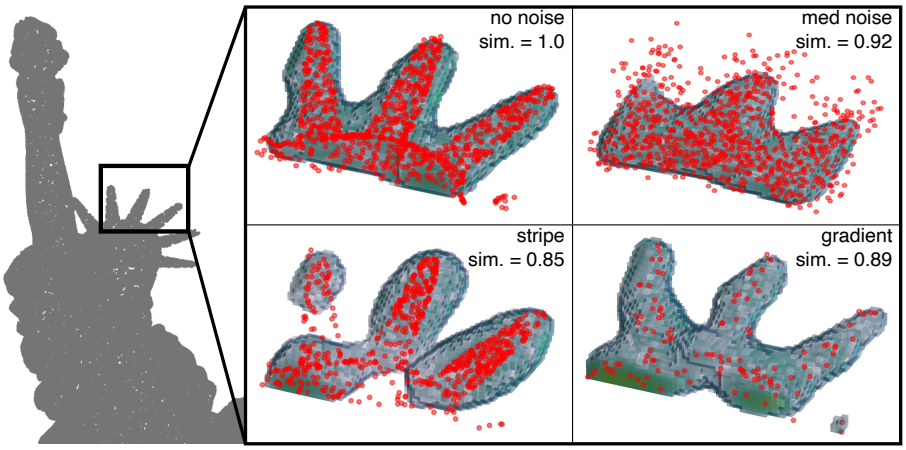


Figure 4.5: VecKM's robustness to data corruptions. VecKM can reconstruct the local shape under corrupted inputs. The VecKM encodings remain highly similar under data corruptions.

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 4.1, **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 4.5 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 4.1, indicating that the overall impact on performance is not substantial.

As shown in Figure 4.6, **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.

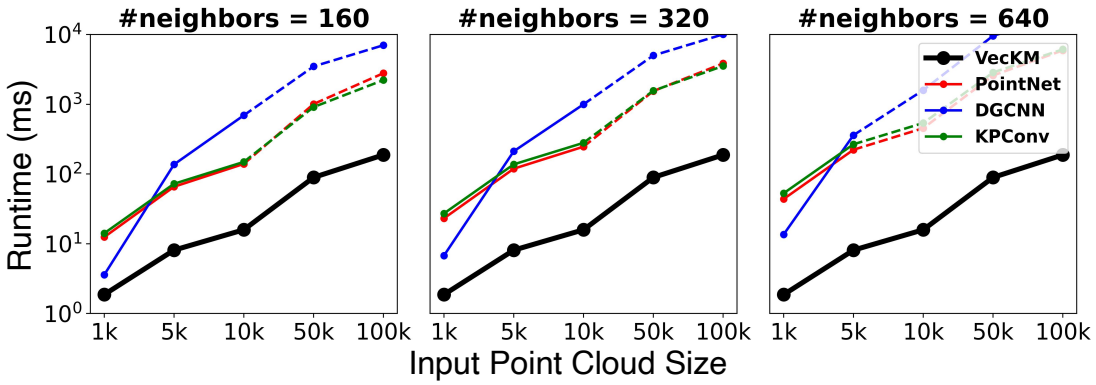


Figure 4.6: Runtime of local geometry encoders under different input point cloud size and neighbor size. All models are tested on an RTXA-5000 with 24 GB memory. Dash lines mean the memory is not sufficient to process all the points in one batch and has to process the points batches by batches.

4.4.2 Classification on ModelNet40 Dataset

We evaluate our VecKM on 3D object classification using the ModelNet40 dataset [86]. 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 [4], PointNet++ [41] and the Point Cloud Transformer (PCT) [74]. We integrate VecKM encoding into these architectures as outlined in Sec. 4.3.4, 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. (4.3). We also compare VecKM-based architectures with the another light-weight network PointMLP [6].

Specifically, for PointNet, since it does not have a local geometry encoding module, we *add* the 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 4.2, **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.

Table 4.2: 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

4.4.3 Part Segmentation on ShapeNet Dataset

We evaluate our VecKM on 3D object part segmentation. Our experiment utilizes the ShapeNet [87] 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 4.4.2. 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. (4.3).

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

Table 4.3: 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

a cosine annealing scheduler.

As demonstrated in Table 4.3, similar to the classfication experiment, architectures based on VecKM consistently outperform their baseline counter-parts in accuracy while also benefiting from significantly reduced runtime.

4.4.4 Semantic Segmentation on S3DIS Dataset

We evaluate our VecKM on 3D semantic segmentation. We use the S3DIS dataset [88], 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$\sim$100,000 points. We use the same training setting as [77].

Baselines. We select PointNet++ and Point Transformer [77] 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. (4.2). 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 4.4, **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.

4.5 Ablation Studies

In Section 4.3.3, we qualitatively analyze the effect of the parameters α and β in Theorem 6. In this section, we quantitatively analyze the effect of the parameters in the

Table 4.4: 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

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 4.5, 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 4.7, two layers are sufficient for stably satisfactory performance, highlighting the inherent descriptiveness of VecKM encoding.

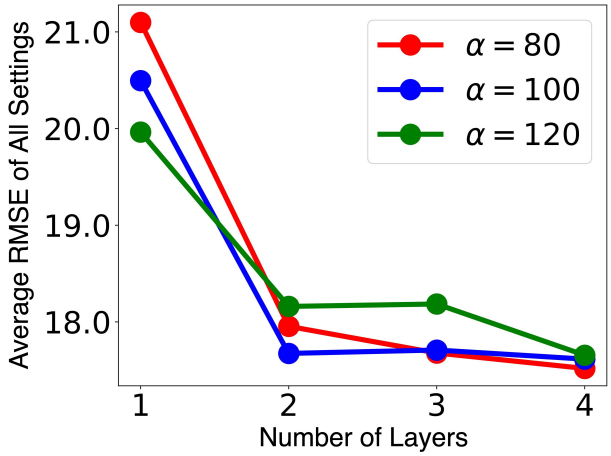


Figure 4.7: Average RMSE of normal estimation trained with different numbers of layers.

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.

Table 4.5: Ablation study on the selection of the parameters α and β in Theorem 6, 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%

4.6 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.

Chapter 5: Learning Normal Flow Directly From Event Neighborhoods

5.1 Introduction

Event-based motion field estimation is a challenging task with significant potential for visual motion interpretation tasks, primarily due to its high temporal resolution, wide dynamic range, and low latency. Image motion is critical for various applications, such as egomotion estimation [89], video interpolation [90, 91], and motion deblurring [92].

Most works on image motion focus on estimating optical flow (OF). Learning-based OF estimators like [18] perform impressively when evaluated within specific domains or datasets but suffer from accuracy degradation when applied across different domains, as shown in [23, 93]. In contrast, model-based OF estimators like [2] are more robust to domain shifts, but their accuracy is limited, particularly in scenarios where the event textures are sparse.

The lack of robustness in optical flow estimators is mainly due to the local aperture problem [94]. In regions with limited texture or strong linear edges, only the motion component perpendicular to the edges (normal flow) can be reliably measured, while the motion parallel to the edges remains ambiguous. To address this issue, optical flow estimators use CNN and RNN to enlarge their receptive fields, which makes the models vulnerable to overfitting.

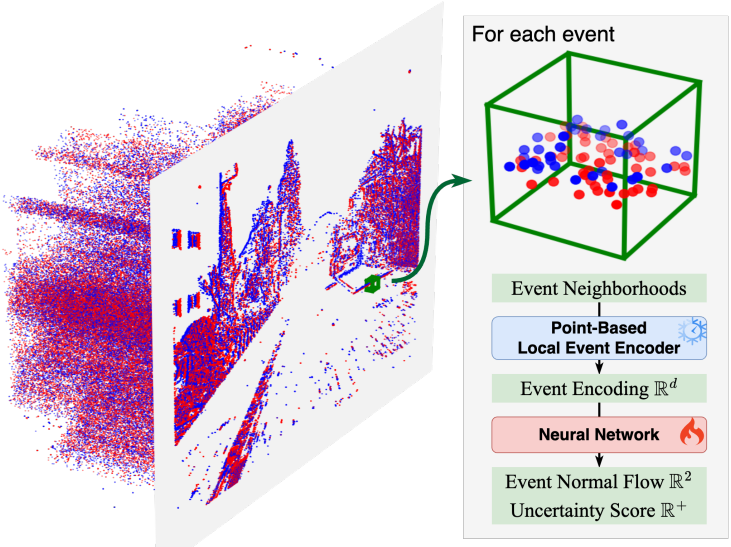


Figure 5.1: We propose a point-based network for estimating normal flow from raw event data. We discover multiple key advantages of this point-based approach compared with existing learning-based approaches. An event and its neighborhood are first encoded as a fixed-dimensional vector, which is then input to a network trained in a supervised way to predict normal flow. This approach achieves high accuracy while maintaining strong transferability across different domains and datasets. Besides, we demonstrate the usefulness of the estimated normal flow in a new egomotion solver that is shown to remain robust even during aggressive camera motions.

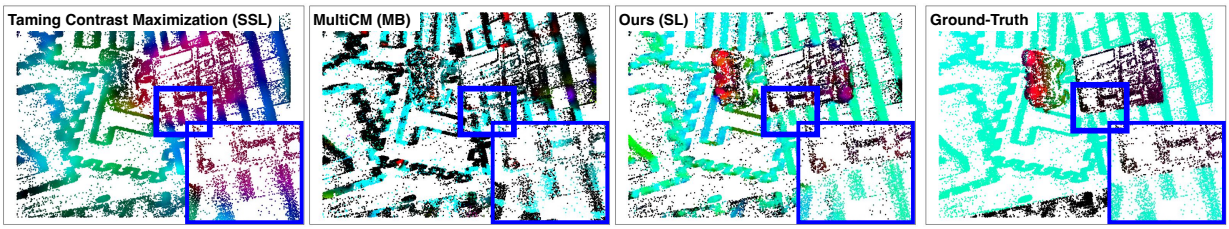


Figure 5.2: Our point-based method produces accurate and sharp predictions in the presence of independently moving objects, while other methods [1, 2] fail. All models (if learning-based) are trained on DSEC and evaluated on EVIMO2. The flows are displayed in HSV color space, where the hue represents the flow direction, and the brightness represents the flow magnitude.

With this observation, some existing methods focus on predicting normal flow (NF) and demonstrate that normal flow is useful for tasks like egomotion estimation [95, 96, 97]. However, current NF estimation approaches are predominantly model-based, relying on fitting a plane to the local space-time event surface [98, 99]. These approaches suffer from limited accuracy. Furthermore, the difficulty in obtaining ground-truth normal flow data has impeded learning-based normal flow estimators.

As shown in Figure 5.1 and 5.2, we explore supervised normal flow estimation while ensuring transferability across domains and datasets. Unlike most OF estimators that use CNNs on event frames, we propose a novel approach based on encoding 3D point sets and a novel loss function. This framework directly trains a network to estimate per-event normal flow from a local event cloud, supervised by ground-truth optical flow. To efficiently handle large event volumes, we adopt VecKM [100], a scalable and descriptive local point cloud encoder. We identify several unique advantages of estimating normal flow with this point-based network.

1. **Temporally and Spatially Sharp Predictions**. By predicting per-event flow based on the Euclidean neighborhood of every event, our method produces sharp predictions, especially for independently moving objects.

2. **Richer data augmentation.** Our point-based approach enables a wider range of augmentations, such as uniform random rotations, which significantly enhances the estimator's accuracy and robustness across various domains.

3. **Uncertainty quantification**. Our point-based method can compute prediction uncertainty with a simple ensemble inference, offering valuable insights for downstream

tasks like egomotion estimation.

4. **Strong transferability.** The method uses only event neighborhoods and we train on undistorted events in normalized camera coordinates. This improves transferability when training and testing on different datasets.

Finally, we show the effectiveness of the predicted normal flows through aggressive egomotion estimation. We introduce a novel geometric-based egomotion solver that utilizes normal flows in conjunction with IMU measurements. This completes the pipeline integrating transferable event-based normal flow estimation with egomotion estimation. Our contribution can be summarized as follows:

- We propose a learning-based, point-based normal flow estimator that offers multiple advantages over existing learning-based methods. It is more accurate than model-based normal flow estimators and more robust and transferable than learning-based optical flow estimators.

- We introduce a novel geometric-based egomotion solver only using normal flows and IMU measurements, which remains robust under aggressive egomotion scenarios.

- We extensively evaluate our point-based flow estimator on multiple datasets and multiple transfer settings.

5.2 Related Work

5.2.1 Event-Based Optical Flow Estimation

In the early stages, **model-based** methods were studied for event-based optical flow estimation. The methodologies include extensions of the Lucas-Kanade algorithm [101, 102], feature matching [103, 104, 105, 106], contrast maximization (CM) [107], Multi-CM [2], plane fitting [98, 108], filter banks [109, 110], time surface matching [111, 112, 113], iterative deblurring [114].

Recently, **learning-based frame-based** methods, such as E-RAFT [18], have dominated event-based optical flow estimation by leveraging correlation volumes. Many techniques, such as multi-modality [19, 20, 21, 22], motion aggregation [1, 23, 24, 25], synthetic datasets [93, 115, 116] have improved these models. However, their performance drops significantly when tested across different domains.

Meanwhile, **spiking neural networks** (SNNs) are applied to event-based optical flow estimation due to their efficiency in processing asynchronous data [117, 118, 119, 120, 121, 122]. SNNs offer energy efficiency and compatibility with neuromorphic hardware. However, their complex training and dependency on specialized hardware make them less practical than conventional neural networks.

Despite the existing research, **learning-based point-based** methods remain underdeveloped in the event camera community, mainly due to the challenges of training point-based estimators on large event datasets. Our work addresses this gap.

5.2.2 Point-Based Networks for Event Cameras

Point-based networks have enabled direct processing of point cloud data where PointNet [4] was the pioneering work. Subsequently, many feature extractors were developed to improve point cloud processing [6, 41, 64, 77, 123, 124]. All existing point-based networks follow a common pipeline that samples and groups input point clouds into centered neighborhoods, transforming the data from $(n, 3)$ to $(n, K, 3)$, where K is the number of neighboring points. However, this method is impractical for event data due to the huge values of n and K. Fortunately, with the invention of VecKM [100], a descriptive and scalable local point cloud encoder that eliminates the need for explicit grouping and sampling, it is now feasible to apply point-based networks to event data. We explore the potential in this paper.

Prior to the introduction of VecKM, several attempts were made to apply point-based networks to event-based vision tasks [43, 125, 126, 127, 128]. However, traditional point-based networks require downsampling of the local point cloud neighborhoods (i.e. reducing K), limiting their use to only high-level tasks like action recognition, which do not demand precise modeling of event geometry. No previous work has successfully used point-based networks for low-level tasks such as normal flow prediction, which requires an accurate representation of event geometry.

5.2.3 Normal Flow and its Applications

Normal flow refers to the component of optical flow that is perpendicular to the edges or parallel to image gradients:

$$\mathbf{n} = -\frac{\nabla I \cdot \mathbf{u}}{||\nabla I||^2} \nabla I \tag{5.1}$$

where ∇I is the image gradient, $\mathbf{u}$ is the optical flow vector, and $\mathbf{n}$ is the normal flow. *Normal flow can be estimated from a local neighborhood because it depends only on local spatial-temporal intensity changes.* Unlike optical flow, it is not affected by the aperture problem [94], as it only captures motion along the image gradient direction. Because normal flow is a projection of optical flow, it satisfies the following constraint, which will be used extensively in this paper:

$$\mathbf{n} \cdot (\mathbf{u} - \mathbf{n}) = 0 \tag{5.2}$$

Normal flow is typically estimated by fitting planes to very small event cloud neighborhoods [95, 98, 99, 129], a simple and reliable method across various datasets. However, the approach encounters difficulties when the edge is curved, or the local region features a corner.

Normal flow has been applied to motion and structure estimation [95, 96, 97]. Traditionally, 3D motion has been computed from normal flow via classification approaches [130, 131] using the so-called depth positivity constraint, relating the 2D to the 3D measurements. Researchers recently developed ways to incorporate the depth positivity constraint into optimization frameworks [3], and neural networks [132] to estimate 3D motion.

The egomotion estimation algorithm proposed in this paper builds on the approach of [3], which also relies on the depth positivity constraint and optimization. However, we reformulate the optimization problem as training a support vector classifier, enhancing stability and accuracy.

5.3 Methodology

5.3.1 Problem Definition and Overview

Point-Based Normal Flow Prediction. The input to our point-based normal flow estimator is a sequence of events $(\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_N)$, where $\mathbf{e}_k = (t_k, x_k, y_k)$[1]. We assume access to GT per-event optical flow $(\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_N)$ for supervision (see Appendix 5.8.2 for how to obtain it). The output of the estimator is a sequence of normal flow predictions $(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, \cdots, \hat{\mathbf{n}}_N)$, where $\hat{\mathbf{n}}_k \in \mathbb{R}^2$. Each prediction $\hat{\mathbf{n}}_k$ is determined by the centered neighboring events of $\mathbf{e}_k$:

$$\begin{aligned} &\hat{\mathbf{n}}_k = f\big(\mathcal{N}(\mathbf{e}_k)\big), \text{ where } \mathcal{N}(\mathbf{e}_k) := \\ &\Big\{\mathbf{e}_j - \mathbf{e}_k : \Big|\Big|\Big(\frac{t_j - t_k}{\delta t}, \frac{x_j - x_k}{\delta x}, \frac{y_j - y_k}{\delta y}\Big)\Big|\Big|_2 < 1\Big\} \end{aligned} \tag{5.3}$$

where δt, δx, δy are the hyper-parameters controlling the neighborhood size. Note we allow the number of neighboring events to be different, i.e., $|\mathcal{N}(\mathbf{e}_k)|$ can be different for different events $\mathbf{e}_k$. As discussed in Section 5.2.3, the neighboring events contain sufficient information to estimate normal flow because it is determined by intensity gradients, which are inherently local.

Uncertainty Quantification (UQ). After predicting the per-event normal flows $(\hat{\mathbf{n}}_k)_{k=1}^N$, we estimate uncertainty scores $(\sigma_k)_{k=1}^N$ of the predictions, where $\sigma_k \geq 0$. UQ is crucial in per-event normal flow estimation because, as shown in Figure 5.3, there are neighborhoods where normal flow can not be estimated reliably. UQ identifies and removes these noisy

[1] We omit polarity here because we found the polarity does not improve prediction accuracies.

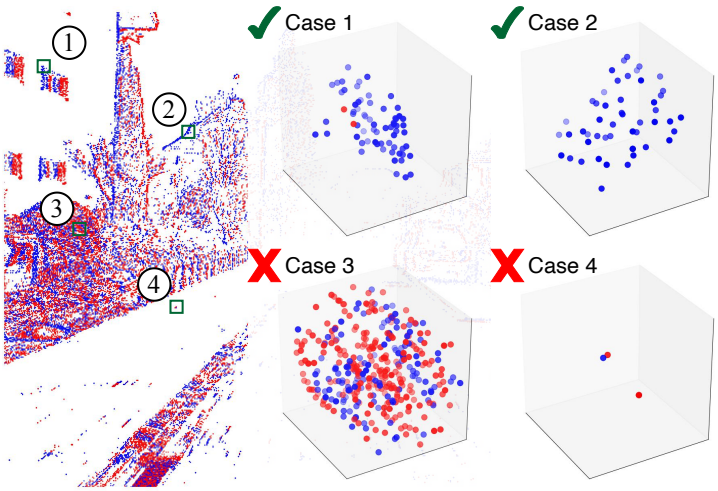


Figure 5.3: Uncertainty quantification (UQ) is important for per-event normal flow estimation, as it helps filter out less reliable predictions. For example, the normal flow predictions in Cases 1 and 2 are more reliable compared to those in Cases 3 and 4.

predictions before passing them to downstream tasks.

Overview. Our solution to the problem is outlined in Figure 5.1. Each component of the pipeline is detailed as followed. Section 5.3.2 introduces the local events encoder, which efficiently transforms the local events $\mathcal{N}(\mathbf{e}_k)$ into a representative vector. Section 5.3.3 introduces a novel loss function that guides the network to predict normal flows, supervised by optical flow ground-truth. Section 5.3.4 introduces the data augmentation tricks used during training, which is a unique advantage of our point-based method. Section 5.3.5 introduces how to compute uncertainty scores during inference time. Section 5.3.6 details an egomotion estimation algorithm that uses normal flow inputs, where we enhance the robustness and stability of the algorithm from [3].

5.3.2 Local Events Encoder

We introduce how to encode the neighboring events $\mathcal{N}(\mathbf{e}_k)$ into a vector representation for each event $\mathbf{e}_k$. Given the high volume of input events($\sim$ 80k events every 20 ms), the

local events encoder must be scalable and efficient.

We use a recently developed a local geometry encoder named VecKM [100], which is designed to process large point clouds efficiently. VecKM models local events as samples from a kernel mixture and uses random Fourier features to transform the kernel mixture into a vector representation. Such formulation allows using all neighboring points to compute the local geometric encoding without down-sampling the neighborhood. Besides, it eliminates the need for explicitly grouping and sampling the event neighborhoods. These merits make VecKM well-suited for handling high-volume event data and dense local event regions.

VecKM Encoding of Local Events. Given the normalized events $X_{N\times 3} = \{(\frac{t_k}{\delta t}, \frac{x_k}{\delta x}, \frac{y_k}{\delta y})\}_{k=1}^{N}$ in Eqn. (5.3), the VecKM local events encoding $G_{N\times d}$ is computed by:

$$\begin{aligned} J_{N\times N} &= adjacency_matrix(X_{N\times 3}) \\ \mathcal{A}_{N\times d} &= \exp(iX_{N\times 3}A_{3\times d}) \\ G_{N\times d} &= normalize\big((J_{N\times N}\mathcal{A}_{N\times d})./\mathcal{A}_{N\times d}\big) \end{aligned} \tag{5.4}$$

J is a sparse adjacency matrix where the (j,k)-entry is 1 if $\mathbf{e}_j$ and $\mathbf{e}_k$ are close, and 0 otherwise. $A_{3\times d}$ is a randomized fixed matrix with entries drawn from a normal distribution with a mean of 0 and variance of 25. The function $\exp(i\cdot)$ is the element-wise Euler formula, and ./ denotes the element-wise complex number division.

The neighbor information is captured in the sparse adjacency matrix J and incorporated into the local event encoding G through matrix multiplication. The centralizing step is implicitly achieved by the element-wise division. This formulation eliminates the need for

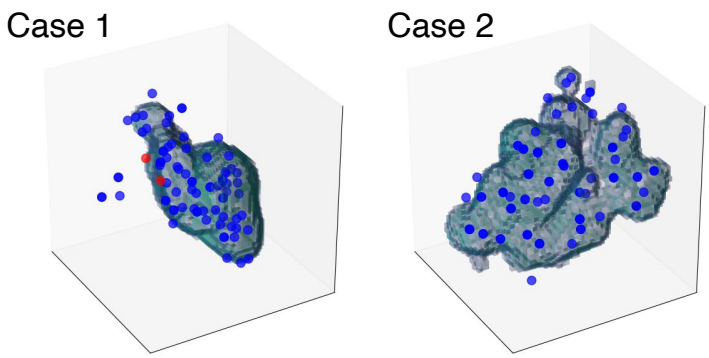


Figure 5.4: Reconstruction of a density distribution (shown in gray) from VecKM's local events encoding. The reconstructed 3D distribution closely aligns with the original (blue and red) events, demonstrating that VecKM's encoding effectively represents the event data. The examples shown are identical to those in Figure 5.3.

explicitly grouping and sampling event neighborhoods, making it descriptive and scalable. Kindly refer to [100] for detailed derivation.

Qualitative Evaluation of VecKM Encoding. VecKM [100] proves that $G[k,:] \in \mathbb{C}^d$ in Eqn. (5.4) effectively represents local events $\mathcal{N}(\mathbf{e}_k)$. Specifically, the local events distribution can be reconstructed from the encoding $G[k,:]$. Figure 5.4 shows examples of local events and the distribution reconstruction from the local events encoding. The strong alignment between the events and the reconstructed distributions suggests that VecKM generates a representative encoding.

Normal Flow Prediction. After computing the local events encoding $G_{N\times d}$ from Eqn. (5.4), we transform each event's encoding $G[k,:]$ to normal flow prediction $\hat{\mathbf{n}}_k$ using a multi-layer perceptron (MLP) [55]. The MLP is trained by a novel loss function introduced in the next section.

5.3.3 Normal Flow Learning with a Two-Term Loss

In frame-based vision, *normal flow* has been defined as the flow component along the image gradient (or perpendicular to the local edge). In event space, however, no obvious definition exists for an image gradient that can be computed per-event and so obtaining ground truth (GT) normal flow for an event camera is challenging. Instead, we use GT optical flow as supervision, and guide the network to predict normal flow (the component of optical flow along some direction) by training with a novel motion field loss function. This loss function is composed of two components: a radial and an angular component, as illustrated in Figure 5.5.

The **radial component** guides the predicted flow to satisfy the normal flow magnitude constraint (5.2). Geometrically, this means the predicted flow lies on a circle where the GT optical flow represents the diameter. When the normal flow constraint is satisfied, the radial loss becomes zero.

Note that predicting zero flow is a trivial solution for minimizing the radial component. To avoid such prediction, we introduce an **angular component**, which encourages the predicted flow to have the same direction as the optical flow.

Heuristically, training on the sum of these two components allows the network to predict the full optical flow when possible. If the texture information is insufficient to predict the full optical flow, the network resorts to predicting the normal flow. We show that our intention is achieved in Figure 5.7 in Appendix 5.5.1.

Mathematically, given the GT optical flow, the normal flow is predicted using the

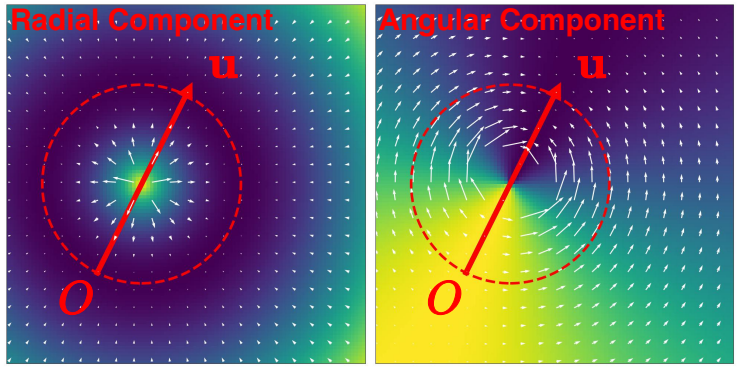


Figure 5.5: Loss maps and gradient fields of the motion field loss function. Our motion field loss function consists of radial and angular components. Given the GT optical flow $\mathbf{u}$, the radial component guides the predicted flow to lie on the circle with $\mathbf{u}$ as the diameter. The angular component guides the predicted flow to align with $\mathbf{u}$, which prevents the trivial prediction of zero flow.

following loss function:

$$Radial(\mathbf{u}, \hat{\mathbf{n}}) = \log\left(\frac{\epsilon + ||\hat{\mathbf{n}} - \mathbf{u}/2||}{\epsilon + ||\mathbf{u}/2||}\right)^2 \tag{5.5}$$

$$Angular(\mathbf{u}, \hat{\mathbf{n}}) = -\frac{(\hat{\mathbf{n}} - \mathbf{u}/2) \cdot \mathbf{u}}{||\hat{\mathbf{n}} - \mathbf{u}/2|| \cdot ||\mathbf{u}||} \tag{5.6}$$

Figure 5.5 shows the resulting loss map and gradient field. Note, that the two components gradients are orthogonal everywhere, so the optimization of one component does not interfere with the other.

5.3.4 Improve Transferability by Data Augmentation

Diverse data augmentation is a unique advantage of our point-based network, which improves the transferability of the estimator. Below, we enumerate the data augmentation tricks applied during training. We denote the input events as $X_{N\times3}$ and their corresponding GT optical flow as $U_{N\times2}$.

Random Rotation. Motion field estimation is an equivariant task, meaning that rotating the events on the image plane causes the motion field to rotate by the same angle. Thanks to this, we apply a uniformly sampled random rotation angle $\theta \in [0, 2\pi)$ to the event cloud and GT flow. The augmented inputs and targets are obtained by:

$$\begin{aligned} X_{N\times 3} &\leftarrow X_{N\times 3} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \\ U_{N\times 2} &\leftarrow U_{N\times 2} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \end{aligned} \tag{5.7}$$

This forces the network to estimate direction without bias.

Random Scaling. We scale the event streams by a random scalar $\alpha \in (0.75, 1.25)$. The augmented inputs and targets are obtained by: $X \leftarrow X * \alpha$, $U \leftarrow U$. This improves the estimator's performance on small and large objects.

Random Sampling. We sample $50\% \sim 100\%$ percentage of the events and their corresponding flows. This makes the estimator robust to density variation.

5.3.5 Inference with Uncertainty Quantification

As explained in Section 5.3.4 **Random Rotation**, motion field estimation is an equivariant task, which we exploit to estimate an uncertainty score based on how well the predicted flow remains equivariant to rotation. Specifically, we sample K rotation angles and infer the normal flow with the rotated events. Then we de-rotate the predicted normal

flow with the corresponding rotation angle:

$$\hat{U} = estimator\Big(X_{N\times 3}\begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & R(\theta) \end{bmatrix}\Big)R(\theta)^{-1} \tag{5.8}$$

By doing so, we obtain an ensemble of K predicted flows for each event, and can test their consensus to compute an uncertainty score. Specifically, we use the circular standard deviation [133] of the ensemble as the uncertainty score. The final prediction is given by the average of the ensemble in polar coordinates. Predictions corresponding with uncertainty above a threshold are discarded.

5.3.6 Egomotion Estimation from Normal Flow

We demonstrate the usefulness of our normal flow for egomotion estimation. We assume within a short time interval, we have normalized the event coordinates, per-event normal flow predictions, and we have a rotational velocity estimate from the IMU sensor. We solve for the translation direction.

Our egomotion solver uses the depth positivity constraint, which states that all world points are in front of the camera, i.e., have positive depth. Existing methods [3, 131, 132] utilize this constraint by maximizing the negative depths. However, the solution to this problem is not unique with respect to normal flow because depth positivity only considers the sign (or the direction) of the normal flow and is thus a weak constraint. So, when estimated normal flow is used, errors in the estimated normal flow direction determine the output. Thus we reformulate the problem as maximum margin problem solved by training

a support vector classifier, which results in a robust estimate.

Given an estimate of the angular velocity Ω from the IMU, we can calculate the rotational component $\mathbf{u}_{\mathbf{x}}^{rot} = B_{\mathbf{x}}\Omega$ of the optical flow at pixel $\mathbf{x} = (x, y)$ where $B_{\mathbf{x}}$ is:

$$B_{\mathbf{x}} = \begin{bmatrix} xy & -(x^2+1) & y \\ (y^2+1) & -xy & -x \end{bmatrix} \tag{5.9}$$

Then the magnitude of the derotated normal flow at pixel $\mathbf{x}$, called $n_{\mathbf{x}}$, can be calculated through the relation $n_{\mathbf{x}} = \|\hat{\mathbf{n}}_{\mathbf{x}}\| - \mathbf{g}_{\mathbf{x}}^T \mathbf{u}_{\mathbf{x}}^{rot}$ where $\mathbf{g}_{\mathbf{x}} \in \mathbb{R}^{2\times 1}$ is the direction of normal flow with unit norm. Then $n_{\mathbf{x}}$ is related to translation by:

$$n_{\mathbf{x}} = \frac{1}{Z_{\mathbf{x}}}(\mathbf{g}_{\mathbf{x}}^T A_{\mathbf{x}})V \tag{5.10}$$

Where $V \in \mathbb{R}^{3\times 1}$ denotes the direction of translation, $Z_{\mathbf{x}} \in \mathbb{R}$ denotes depth, and $A_{\mathbf{x}}$ is:

$$A_{\mathbf{x}} = \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix} \tag{5.11}$$

Since the depth $Z_{\mathbf{x}}$ is positive, the following product, denoted as $\rho_{\mathbf{x}}(V)$, should be positive for all $\mathbf{x}$:

$$\rho_{\mathbf{x}}(V) = n_{\mathbf{x}}(\mathbf{g}_{\mathbf{x}}^T A_{\mathbf{x}} V) > 0 \tag{5.12}$$

We impose this constraint by formulating Eqn. (5.12) into a linear support vector classification without intercept, namely solving V such that $(\mathbf{g}_{\mathbf{x}}^T A_{\mathbf{x}})V$ has a target sign decided by $n_{\mathbf{x}}$. The pseudo-code is given below. p is the number of events. A, B are obtained through

Eqn. (5.9) and (5.11), N are the norms of the predicted normal flows, G are the directions of the predicted normal flows with unit lengths, and Ω_0 is the rotational estimate from the IMU sensor.

Algorithm 4 Egomotion Solver

1: **Input:** $\mathrm{A}_{p\times2\times3}$, $\mathrm{B}_{p\times2\times3}$, $\mathrm{N}_{p\times1}$, $\mathrm{G}_{p\times2}$, $\Omega_0 \in \mathbb{R}^{3\times1}$.
2: **Output:** $V \in \mathbb{R}^{3\times1}$.
3: $Q_{p\times3} =$ `batch_mat_mul`(G, A)
4: $R_{p\times1} = \mathrm{N} -$ `batch_mat_mul`$(\mathrm{G}, \mathrm{B}) \times \Omega_0$
5: $Q_{2p\times3} =$ `concat(`Q`,`$-Q$`)`
6: $R_{2p\times1} =$ `sign(concat(`R`,`$-R$`))`
7: `svm = LinearSVM(fit_intercept=False)`
8: $V_{3\times1} =$ `svm.fit(`Q`,`R`).coef_`
9: $V = V \,/\, ||V||$

5.4 Experiments on Normal Flow Estimation

Datasets. We use MVSEC [134], EVIMO2 [135], and DSEC [136] to evaluate the accuracy and transferability of our normal estimator. We undistort and transform the events to normalized camera coordinates (focal length one). We train on each dataset and evaluate the model on all three datasets, resulting in nine combinations. Additionally, we train our model on the union of the three (and potentially more) datasets. We evaluate it and release it for public use.

Note that MVSEC has different camera resolution and distortion compared to EVIMO2 and DSEC. Therefore, there is a substantial domain gap among the cameras used in the three datasets.

Evaluation Metrics. Quantitatively, we use projection endpoint error (PEE) and percentage of sign correctness ($\%Pos$), following the convention in [132], to evaluate the accuracy of

normal flow predictions.

$$PEE(\mathbf{u}, \hat{\mathbf{n}}) = \left|\left| \frac{\mathbf{u} \cdot \hat{\mathbf{n}}}{||\hat{\mathbf{n}}||} - ||\hat{\mathbf{n}}|| \right|\right| \tag{5.13}$$

$$\%Pos(\mathbf{u}, \hat{\mathbf{n}}) = percentage(\mathbf{u} \cdot \hat{\mathbf{n}} > 0) \tag{5.14}$$

PEE measures how well the normal flow magnitude constraint (5.2) is satisfied. It measures the error in the length of the normal flow. $\%Pos$ measures how well the predicted normal flow has correct orientation as determined by the GT optical flow. Note that since the direction of normal flow is independent of motion, optical flow also satisfies the constraint equation (5.2) and thus also minimizes PEE.

Qualitative Evaluation. In this drive, we include flow prediction videos for every evaluation. Qualitative evaluation is crucial for assessing aspects like prediction sharpness and handling independently moving objects, which are not fully captured by quantitative metrics. We strongly encourage readers to focus on the qualitative results for a comprehensive assessment of model performance. We show visualizations in Figure 5.6 and upload flow prediction videos to this drive. See Appendix 5.8.1 for detailed video descriptions.

Compared Models. We compare our point-based learning-based normal flow estimator with state-of-the-art event-based optical flow estimators. We compare against MultiCM [2], E-RAFT [18], and Taming Contrast Maximization (TCM) [1], which are all frame-based estimators using model-based, supervised, and self-supervised learning approaches. Additionally, we compare against two point-based normal flow estimators: PCA [98] and PointNet [4].

Implementation details and hyper-parameter setting are presented in Appendix

5.8.3.

5.4.1 Evaluation on MVSEC

As shown by the quantitative results in Table 5.1, **our point-based method is on par with state-of-the-art frame-based methods despite using only local information.** While TCM achieves lower PEE than our method on MVSEC, this is because MVSEC mainly features slow-moving, static scenes without independently moving objects. In these scenarios, TCM and E-RAFT using CNNs, can leverage larger receptive fields (through stacked convolution and pooling layers) to smooth the flow predictions. As we will show in Section 5.4.2, TCM and E-RAFT's performance deteriorates when scenes include independently moving objects, whereas our method remains robust.

Our method outperforms the point-based estimators, PCA (MB) and PointNet (SL) by achieving lower PEE and higher $\%Pos$, highlighting the benefits of supervised training. Additionally, PointNet requires explicit grouping of events into neighborhoods and sampling, while our method does not. This allows us to use information from more neighboring events to predict the flows and yield better performance. In cross-dataset evaluations, our estimator outperforms PCA and PointNet when trained on DSEC and evaluated on MVSEC even though PointNet was trained on MVSEC. When our method is trained on EVIMO-`imo`, the performance remains comparable though the EVIMO-`imo` train set containing only 2.74 minutes of data.

Table 5.1: Quantitative results on MVSEC. The estimators are classified into model-based (MB), supervised learning (SL), self-supervised learning (SSL), frame-based (F), point-based (P). They are trained on MVSEC (M), DSEC (D), EVIMO-`imo` (E).

		Input	Training Set	Indoor Flying 1		Indoor Flying 2		Indoor Flying 3		Outdoor Day 1		Average	
				PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑
MultiCM	MB	F	-	0.993	97.8%	1.378	98.1%	1.191	98.2%	1.422	93.2%	1.246	96.8%
PCA	MB	P	-	1.460	76.2%	1.586	76.3%	1.552	75.0%	1.548	79.9%	1.537	76.9%
E-RAFT	SL	F	M	1.836	76.1%	2.867	74.7%	1.951	75.4%	**0.677**	97.8%	1.833	81.0%
			D	0.797	92.5%	1.163	91.6%	0.948	92.9%	0.846	97.0%	0.939	93.5%
TCM	SSL	F	M	0.319	95.1%	0.638	91.3%	0.490	94.4%	0.948	97.3%	0.599	94.5%
			D	**0.303**	93.9%	**0.546**	92.0%	**0.437**	93.6%	0.778	97.1%	**0.516**	94.1%
PointNet	SL	P	M	0.973	96.7%	1.428	95.6%	1.224	97.0%	1.032	97.1%	1.164	96.6%
Ours	SL	P	M	0.970	98.4%	1.090	99.4%	1.040	99.6%	0.880	**99.2%**	0.995	**99.2%**
			D	0.922	99.3%	1.216	99.2%	1.282	99.6%	1.004	97.8%	1.106	99.0%
			E	1.669	99.0%	1.483	**99.6%**	1.671	99.7%	0.854	98.6%	1.419	**99.2%**
			M+D+E	0.968	**99.5%**	1.057	99.5%	1.065	**99.8%**	0.879	97.3%	0.992	99.0%

5.4.2 Evaluation on EVIMO2

EVIMO2-`imo` sequences feature fast independently moving objects. Table 5.2 presents quantitative results on the samsung_mono camera and Figure 5.6 visualizes the flow prediction colored in HSV. As shown on the Table 5.2, **our method, significantly outperforms all compared estimators in the presence of independently moving objects even when trained on other datasets.** In addition, as shown in Figure 5.6, our method effectively preserves the boundaries of the IMOs, closely matching the ground truth.

Our estimator shows highly consistent performance and strong generalizability across different scenes, motion types and datasets as shown in Table 5.2. This is due to the model's use of geometric features from local event neighborhoods, which capture domain-invariant patterns essential for normal flow estimation. Besides, our approach employs normalized camera coordinates, utilizes extensive data augmentation, and prevents global information from affecting estimates in regions with independently moving objects (IMOs), which all lead to the strong performance.

Table 5.2: Quantitative results on the first five scenes of EVIMO2-`imo`'s evaluation set. The complete table is in Appendix 5.8.4.

		Input	Training Set	Scene_13_00		Scene_13_05		Scene_14_03		Scene_14_04		Scene_14_05		Average (8 scenes)	
				PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑
MultiCM	MB	F	-	1.509	53.2%	4.315	75.7%	1.611	79.2%	1.800	73.2%	2.768	72.9%	1.800	68.5%
PCA	MB	P	-	1.573	88.2%	2.035	87.5%	1.580	91.9%	1.784	90.3%	1.823	89.4%	1.712	87.8%
E-RAFT	SL	F	M	1.370	71.9%	2.406	90.6%	1.356	69.5%	1.458	64.6%	2.186	67.1%	1.470	70.2%
			D	0.843	88.9%	1.185	97.5%	0.517	88.1%	0.538	85.9%	0.908	86.3%	0.705	87.9%
TCM	SSL	F	M	0.823	85.6%	3.201	95.3%	1.111	86.3%	1.532	86.0%	2.445	82.2%	1.383	84.6%
			D	0.774	87.3%	2.541	95.1%	0.872	87.8%	1.090	86.5%	1.640	84.1%	1.105	85.3%
PointNet	SL	P	E	1.047	88.1%	0.924	97.7%	0.848	98.3%	0.892	96.2%	1.053	96.6%	0.933	95.0%
Ours	SL	P	M	0.713	95.6%	0.269	99.3%	0.676	98.8%	0.651	98.1%	0.806	98.2%	0.551	97.1%
			D	0.590	96.6%	**0.230**	**99.8%**	0.575	**99.8%**	0.625	**99.5%**	**0.567**	**99.4%**	0.463	**97.9%**
			E	0.497	**96.7%**	0.399	99.2%	**0.478**	99.2%	0.515	98.8%	0.584	98.6%	0.423	97.9%
			M+D+E	**0.465**	96.2%	0.308	99.2%	0.544	99.3%	**0.467**	98.8%	0.568	98.5%	**0.396**	97.8%

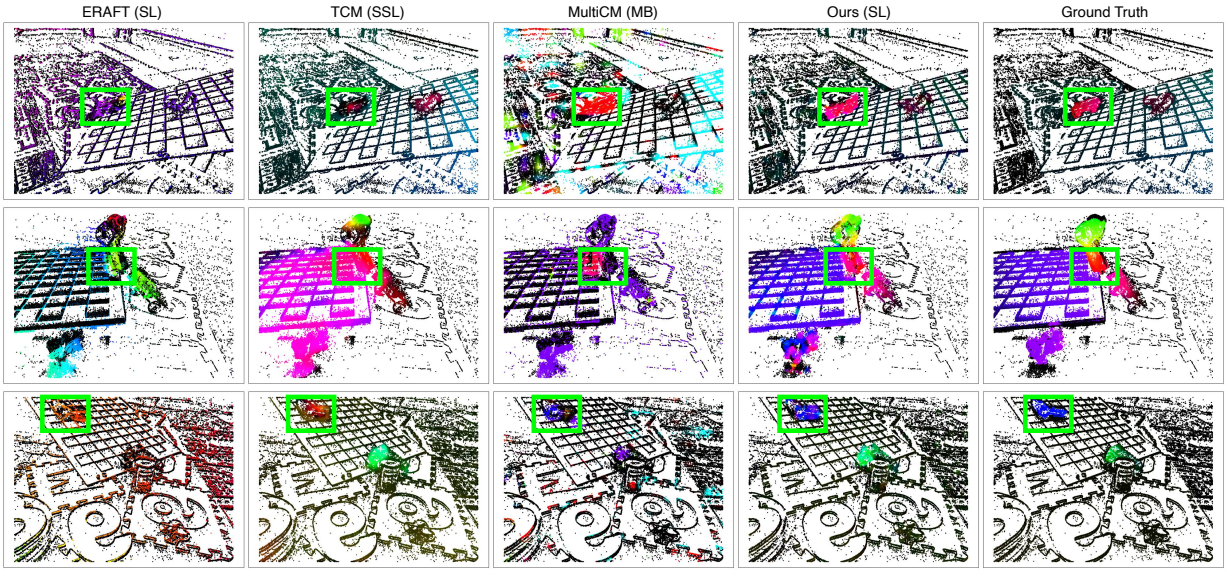


Figure 5.6: Visualization of flow prediction on EVIMO2-`imo`. The flows are displayed in HSV color space. Our method effectively preserves the boundary of the IMOs, while other methods fail.

5.4.3 Evaluation on DSEC

Since DSEC does not provide GT optical flow, we cannot compute error metrics for normal flow evaluation. We only provide a qualitative evaluation. The flow prediction videos in this drive show **our method performs equally well even when trained on different datasets**. Our method produces consistently accurate predictions though the event density of MVSEC and the motion type of EVIMO2 are very different from those in

DSEC,

5.5 Ablation Studies

We use the EVIMO2-`imo` dataset for our ablation studies because it presents challenging scenarios with independently moving objects. The models are trained on EVIMO2-`imo` training set to better capture the impact of the ablated factors.

5.5.1 Effect of Motion Field Loss

We show that our estimator benefits from being trained on the novel motion field loss by comparing to an optical flow loss. While variations on average end-point error (AEE) are typically used for supervised training of optical flow estimators, our method, which is designed to estimate normal flow, does not converge when trained with such losses. Thus, we designed the following norm + direction loss to train our estimator to estimate optical flow, defined as follows:

$$\mathcal{L}_1 = \log\left(\frac{\epsilon + ||\mathbf{u}||}{\epsilon + ||\hat{\mathbf{u}}||}\right)^2$$

$$\mathcal{L}_2 = -\frac{\mathbf{u} \cdot \hat{\mathbf{u}}}{||\mathbf{u}|| \cdot ||\hat{\mathbf{u}}||}$$

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$$

Where $\hat{\mathbf{u}}$ is the output of our method when being trained with this optical flow loss.

As shown in Table 5.3, our motion field loss function significantly enhances the estimator's performance in terms of PEE and $\%Pos$.

In addition, we analyze the behavior of our estimator qualitatively in Figure 5.7. After the model is trained using our motion field loss function, the model can choose between estimating full optical flow or normal flow depending on the texture of the local regions. This further justifies the effect of our motion field loss function.

Table 5.3: Comparison between the estimator trained with our motion field loss function and the one trained with the standard norm-plus-direction loss function. Using our motion field function significantly improves the model's performance.

	13_00		13_05		14_03		14_04		14_05		15_01		15_02		15_05		Average	
	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos	PEE	%Pos
Norm + Direction Loss	0.972	90.0%	0.912	98.1%	0.749	98.9%	0.762	96.9%	1.150	97.3%	0.559	96.9%	0.610	94.5%	1.274	91.5%	0.87	95.5%
Ours	0.497	96.7%	0.399	99.2%	0.478	99.2%	0.515	98.8%	0.584	98.6%	0.286	98.1%	0.274	96.8%	0.354	95.5%	0.42	97.9%
Difference	↓ 0.475	↑ 6.7%	↓ 0.513	↑ 1.1%	↓ 0.271	↑ 0.3%	↓ 0.247	↑ 1.9%	↓ 0.566	↑ 1.3%	↓ 0.273	↑ 1.2%	↓ 0.336	↑ 2.3%	↓ 0.92	↑ 4.0%	↓ 0.45	↑ 2.4%

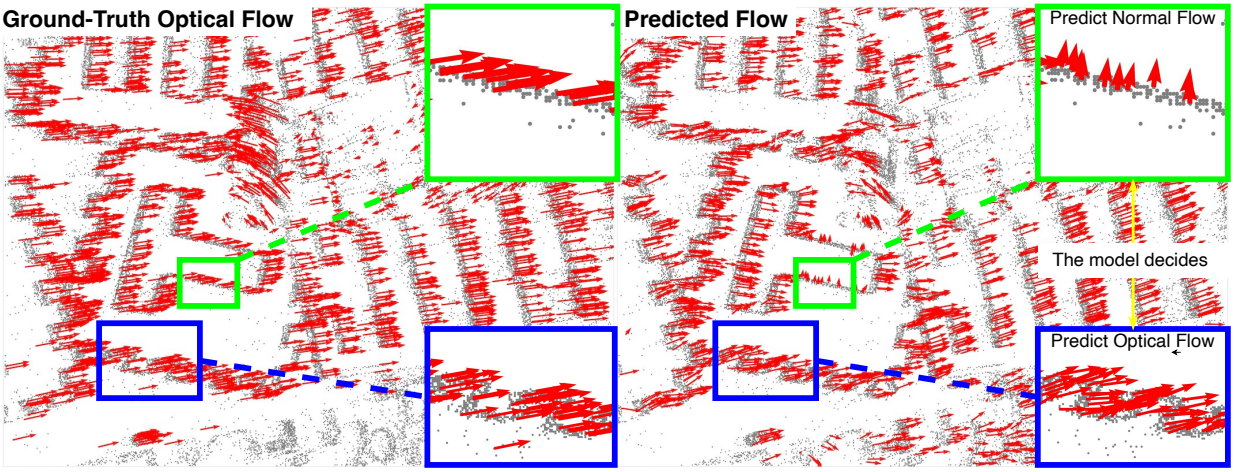


Figure 5.7: The model can choose between estimating the full optical flow or normal flow depending on the texture of the local region. If the local texture is rich enough (e.g. a corner), the model will estimate full optical flow. If the local texture only contains strong edges, the model will estimate normal flow.

5.5.2 Effect of Uncertainty Quantification

Table 5.4 provides a comprehensive analysis of the estimator's performance, showing prediction errors alongside the percentage of confident predictions across various ensemble sizes and uncertainty thresholds. The positive correlation observed between prediction

errors and uncertainty scores underscores the effectiveness of the uncertainty quantification. Our results indicate that an uncertainty threshold between 0.3 and 0.6 achieves an optimal balance between valid prediction rates and accuracy. Additionally, the table reveals that 3 to 4 ensemble predictions are sufficient for consistent uncertainty estimation, though larger ensembles generally yield improved performance. For scenarios where runtime is not a constraint, employing larger ensembles can enhance prediction accuracy.

Table 5.4: Performance of the estimator under varying ensemble sizes and uncertainty thresholds. Increasing the ensemble size generally enhances results, with 4 to 6 ensembles already providing sufficiently stable outcomes.

% Pos	num_ensemble=2	num_ensemble=4	num_ensemble=6	num_ensemble=10
conf_thres=0.1	97.2%	97.7%	98.4%	98.2%
conf_thres=0.2	96.9%	98.0%	98.3%	98.4%
conf_thres=0.3	96.5%	97.7%	98.0%	98.1%
conf_thres=0.4	96.2%	97.4%	97.7%	97.9%
conf_thres=0.5	95.9%	97.2%	97.5%	97.7%
conf_thres=0.6	95.7%	96.9%	97.3%	97.5%
conf_thres=0.7	95.5%	96.7%	97.1%	97.2%
conf_thres=nfty	92.3%	92.6%	92.8%	92.8%

PEE	num_ensemble=2	num_ensemble=4	num_ensemble=6	num_ensemble=10
conf_thres=0.1	0.467	0.529	0.580	0.686
conf_thres=0.2	0.461	0.442	0.436	0.436
conf_thres=0.3	0.454	0.436	0.423	0.423
conf_thres=0.4	0.454	0.436	0.423	0.423
conf_thres=0.5	0.454	0.436	0.423	0.423
conf_thres=0.6	0.454	0.436	0.423	0.429
conf_thres=0.7	0.454	0.436	0.423	0.423
conf_thres=nfty	0.442	0.423	0.411	0.411

Valid Pct.	num_ensemble=2	num_ensemble=4	num_ensemble=6	num_ensemble=10
conf_thres=0.1	45.0%	17.4%	4.3%	1.6%
conf_thres=0.2	66.8%	53.3%	49.6%	47.9%
conf_thres=0.3	74.9%	66.6%	65.0%	63.3%
conf_thres=0.4	78.8%	72.2%	70.6%	68.8%
conf_thres=0.5	81.1%	75.1%	73.4%	71.5%
conf_thres=0.6	82.7%	77.0%	75.3%	73.6%
conf_thres=0.7	83.9%	78.5%	76.9%	76.8%
conf_thres=nfty	100%	100%	100%	100%

5.5.3 Runtime and Memory Usage

Table 5.5 reports the computational cost of our estimator. Table 5.6 reports the statistics about the event density of some selected scenes from each dataset.

Table 5.5: Computational cost of our normal flow estimator.

	Inference Time (5 ensembles)	Max GPU Memory Allocation
num_events = 10k	0.111 s	0.70 GB
num_events = 20k	0.287 s	1.36 GB
num_events = 40k	0.910 s	2.71 GB
num_events = 80k	3.138 s	5.39 GB

Table 5.6: Statistics of event density of each dataset.

	#events every 20 ms – quantile				
	min	25%	50%	75%	max
MVSEC – indoor_flying1	85	2396.25	3720.0	5376.0	16177
MVSEC – indoor_flying2	78	3157.0	5297.5	7987.5	23890
MVSEC – indoor_flying3	78	2746.0	4850.0	6845.0	17476
MVSEC – outdoor_day1	58	4412.0	6903.0	10646.75	96327
EVIMO – IMO_13_00	7954	23564.5	35535.0	46824.5	68946
EVIMO – IMO_13_05	10806	55951.0	78963.0	87891.0	120730
EVIMO – SFM_03_00	2962	15766.0	76979.0	88460.5	105177
EVIMO – SFM_03_01	15315	41049.0	80223.0	94096.0	118441
DSEC – interlaken_00_a	123449	133520.0	146397.0	157920.0	165356
DSEC – interlaken_00_b	183286	187122.75	189133.5	196445.75	209048
DSEC – thun_01_a	66476	83569.25	101858.5	117016.75	121001
DSEC – zurich_city_12_a	116746	138822.0	167154.0	196738.5	228827

5.6 Experiments on Egomotion Estimation

We evaluate our egomotion solver on the EVIMO2 `sfm` split, which includes scenes with fast camera motions and a focus of expansion far from the camera frame. The normal flow is estimated using our method trained on the EVIMO2-`imo` training split. Table 5.7 and Figure 5.8 compares our solver using SVM, to a method based on [3] that estimates translation by maximizing negative depth using rotation estimates from the IMU. The results demonstrate that our normal flow predictions and egomotion solver yield more accurate egomotion estimations. In Figure 5.8 the estimated translation direction is scaled to m/s using ground truth. Plots of the predictions for all scenes are provided in Appendix 5.8.5.

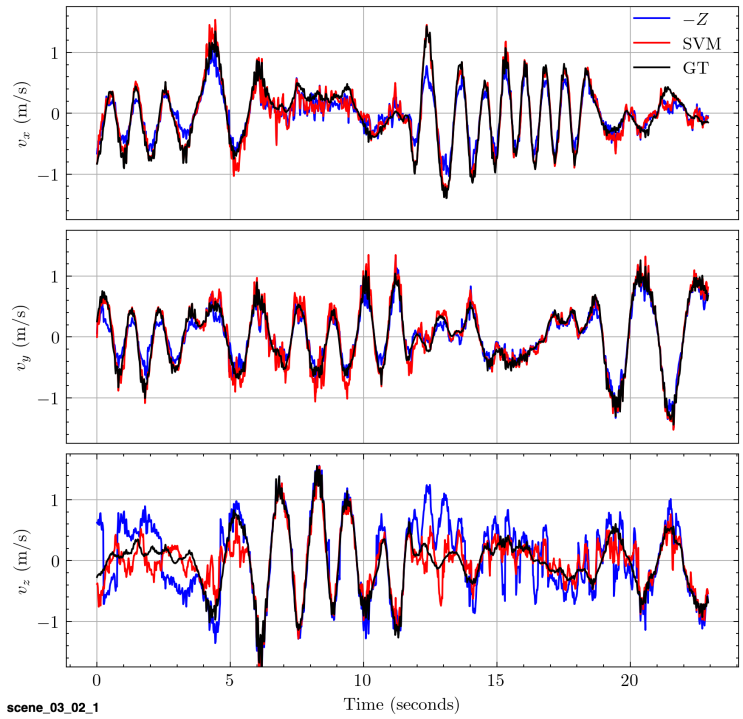


Figure 5.8: Comparison of estimated translation velocity produced by maximizing negative depth [3] and our SVM-based method. Our approach yields more accurate predictions.

5.7 Conclusion and Future Work

We introduce a point-based method for normal flow estimation that overcomes limitations in existing model and learning based methods by using local information without explicit grouping. This allows improved cross domain transfer and good performance in the presence of independently moving objects. While efficient, the method may face computational challenges with higher camera resolutions. Thus, future work can consider optimizing the encoding and transformation to normal flow. Additionally, our method currently depends on ground-truth flow for training. A self-supervised approach could further improve its usability. Finally, while our method benefits from using local information, future work can consider careful incorporation of global information that may be beneficial for tasks such

Table 5.7: Egomotion estimation error on EVIMO2-`sfm`.

Seq.	sfm 03_04_0	sfm 03_02_0	sfm 03_02_1	sfm 03_00_0	sfm 03_03_0
Method			RMS V (m/s)		
Depth Pos.	0.217	0.500	0.243	0.237	0.135
SVM	**0.136**	**0.227**	**0.156**	**0.141**	**0.116**
Seq.	**sfm 03_01_0**	**sfm 03_02_2**	**sfm 03_03_2**	**sfm 03_02_3**	**sfm 03_03_1**
Method			RMS V (m/s)		
Depth Pos.	0.369	**0.091**	0.137	**0.213**	0.265
SVM	**0.219**	0.127	**0.108**	0.279	**0.136**

as action recognition and object detection.

5.8 Supplementary Materials

5.8.1 List of Flow Prediction Videos

We provide flow prediction videos for each evaluated dataset—MVSEC, EVIMO2, and DSEC. These visualizations showcase predictions from models trained on each of the three datasets. Figure 5.9 shows some screenshots of the video. The videos are in this link.

5.8.2 Dataset Preprocessing

In this section, we detail how to preprocess the data to obtain undistorted normalized per-event optical flow on MVSEC, EVIMO2, and DSEC.

MVSEC & EVIMO2 both provide frame-based forward optical flows in the distorted camera coordinates. We first interpolate the flow in the time domain. If an event (t, x, y) lies between t_0 and t_1, the optical flow at this event is computed as:

$$\mathbf{u}(t, x, y) = \frac{t - t_0}{t_1 - t_0} flow(t_1, x, y) + \frac{t_1 - t}{t_1 - t_0} flow(t_0, x, y)$$

After this, we convert the per-event distorted flow in the raw pixel coordinates into undistorted flow in the normalized pixel coordinates using `cv2.undistortPoints`.

$$\begin{aligned}
\text{start} &= \texttt{cv2.undistortPoints}(x, y, K, D) \\
\text{end} &= \texttt{cv2.undistortPoints}(x + \mathbf{u}_x, y + \mathbf{u}_y, K, D) \\
\text{out} &= (\text{end} - \text{start})/(t_1 - t_0)
\end{aligned}$$

This will transform the flow into undistorted normalized camera coordinates, with unit normalized pixel per second.

DSEC, different from the previous two datasets, provides frame-based forward optical flow and backward optical flow, which can be used to obtain more accurate per-event optical flow. Specifically, we let

$$
\begin{aligned}
flow(t_1, x, y) = \frac{1}{2}(\\
\quad flow_forward(t_1, x, y) - flow_backward(t_1, x, y) \\
)
\end{aligned}
$$

The following procedures are the same as the previous two datasets.

5.8.3 Implementation Details of Our Model

We transform the event pixels and flows into undistorted, normalized camera coordinates as explained in Appendix 5.8.2. The resulting flows are then scaled such that their unit is in pixels per second. After this scaling, the flow norms fall within a range of 0 to 3.

During training, we randomly sample an event, using a uniform distribution over the logarithm of the flow norm, within the range of 0.01 to 3. We then slice the event stream around the sampled event to create the training samples. We apply the data augmentation techniques described in Section 5.3.4. The pixel radius parameters (δx, δy in Eqn. (5.3)) are set to 0.02, which correspond to 4.5, 10.4, and 11.1 pixels for the MVSEC, EVIMO2, and DSEC datasets, respectively, measured in terms of raw pixels. The time radius (δt in Eqn. (5.3)) is 20 ms. The parameter ϵ in Eqn. (5.5) is set to 0.1. The dimension of the

local event encoding is 384. We remove the predictions with circular standard deviation larger than 0.3 (Section 5.3.5). If the events size within 20 ms is larger than 80,000, we randomly sample 80,000 events from the 20 ms interval.

5.8.4 Per-Scene Normal Flow Evaluation on EVIMO2

We present the per-scene normal flow evaluation on EVIMO2-`imo`, as shown in Table 5.8.

5.8.5 Per-Scene Egomotion Evaluation on EVIMO2

We present the per-scene egomotion evaluation on EVIMO2, as shown in Figure 5.10.

Table 5.8: Per-scene normal flow evaluation on EVIMO2-`imo` split.

		Input	Training Set	Scene_13_00		Scene_13_05		Scene_14_03		Scene_14_04	
				PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑
MultiCM	MB	F	-	1.509	53.2%	4.315	75.7%	1.611	79.2%	1.800	73.2%
PCA	MB	P	-	1.573	88.2%	2.035	87.5%	1.580	91.9%	1.784	90.3%
E-RAFT	SL	F	M	1.370	71.9%	2.406	90.6%	1.356	69.5%	1.458	64.6%
			D	0.843	88.9%	1.185	97.5%	0.517	88.1%	0.538	85.9%
TCM	SSL	F	M	0.823	85.6%	3.201	95.3%	1.111	86.3%	1.532	86.0%
			D	0.774	87.3%	2.541	95.1%	0.872	87.8%	1.090	86.5%
PointNet	SL	P	E	1.047	88.1%	0.924	97.7%	0.848	98.3%	0.892	96.2%
Ours	SL	P	M	0.713	95.6%	0.269	99.3%	0.676	98.8%	0.651	98.1%
			D	0.590	96.6%	**0.230**	**99.8%**	0.575	**99.8%**	0.625	**99.5%**
			E	0.497	**96.7%**	0.399	99.2%	**0.478**	99.2%	0.515	98.8%
			M+D+E	**0.465**	96.2%	0.308	99.2%	0.544	99.3%	**0.467**	98.8%

			Training Set	scene_14_05		scene_15_01		scene_15_02		scene_15_05	
				PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑	PEE ↓	% Pos ↑
MultiCM	MB	F	-	2.768	72.9%	0.852	68.0%	0.802	66.2%	0.744	59.8%
PCA	MB	P	-	1.823	89.4%	1.467	92.1%	1.612	78.2%	1.821	84.7%
ERAFT	SL	F	M	2.186	67.1%	0.899	72.7%	0.980	67.1%	1.100	57.9%
			D	0.908	86.3%	0.432	91.3%	0.541	90.9%	0.674	73.9%
TCM	SSL	F	M	2.445	82.2%	0.588	85.4%	0.556	87.7%	0.811	68.0%
			D	1.640	84.1%	0.523	85.5%	0.528	87.9%	0.871	68.1%
PointNet	SL	P	E	1.053	96.6%	0.765	96.1%	0.752	95.1%	1.185	91.5%
Ours	SL	P	M	0.806	98.2%	0.470	96.6%	0.433	95.8%	0.392	94.3%
			D	**0.567**	**99.4%**	0.391	**98.1%**	0.298	**97.1%**	0.424	93.2%
			E	0.584	98.6%	**0.286**	**98.1%**	**0.274**	96.8%	0.354	95.5%
			M+D+E	0.568	98.5%	0.319	97.8%	0.300	97.0%	**0.201**	**95.7%**

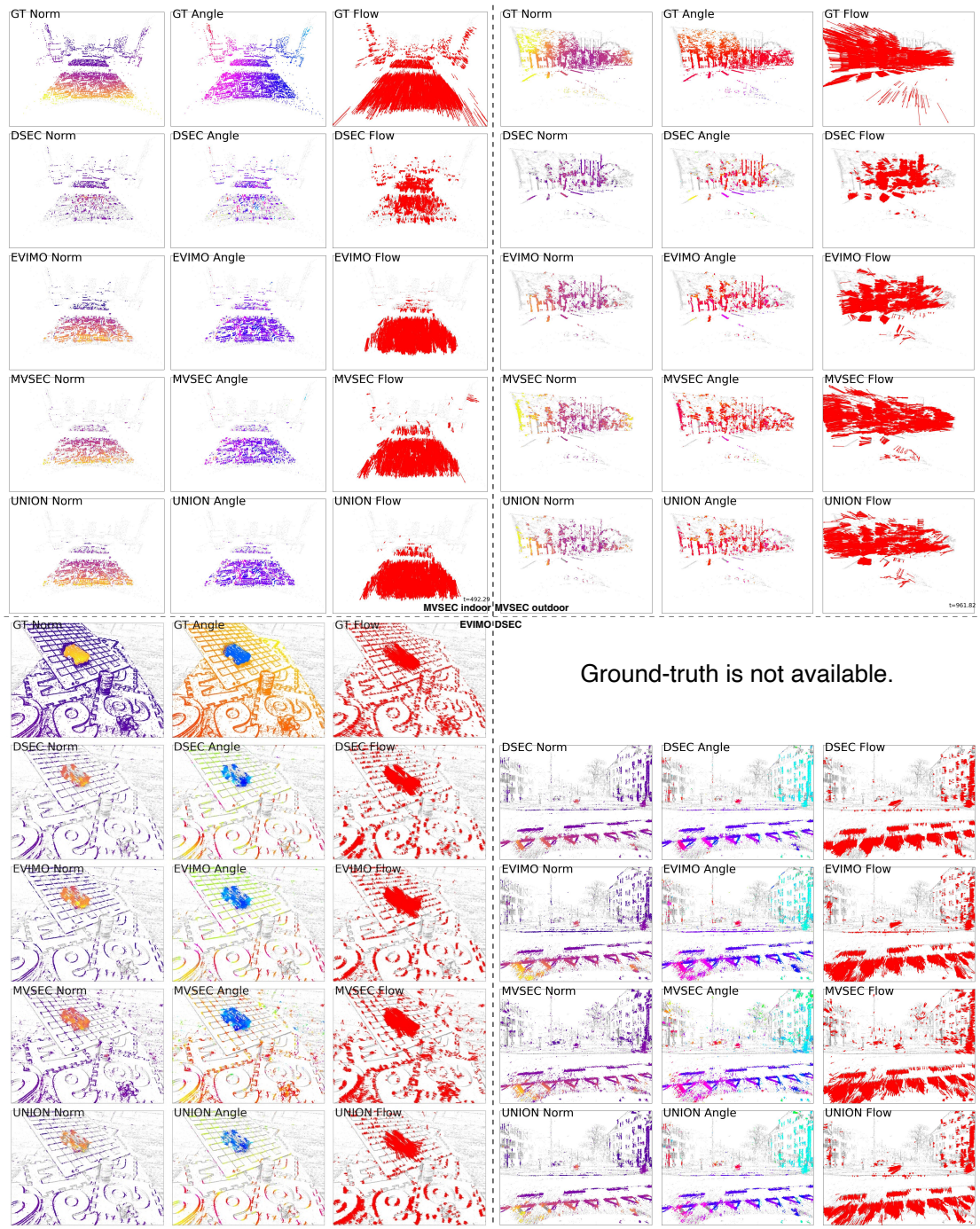


Figure 5.9: Screenshot of the flow prediction videos. Each row displays the norm, angle, and flow fields of both ground-truth and predicted flows. The first row visualizes the ground-truth optical flow, while subsequent rows show model predictions trained on each dataset. To illustrate the flow field, we sample 5,000 flow points for visualization. If a pixel is gray, it means the flow prediction has a high uncertainty.

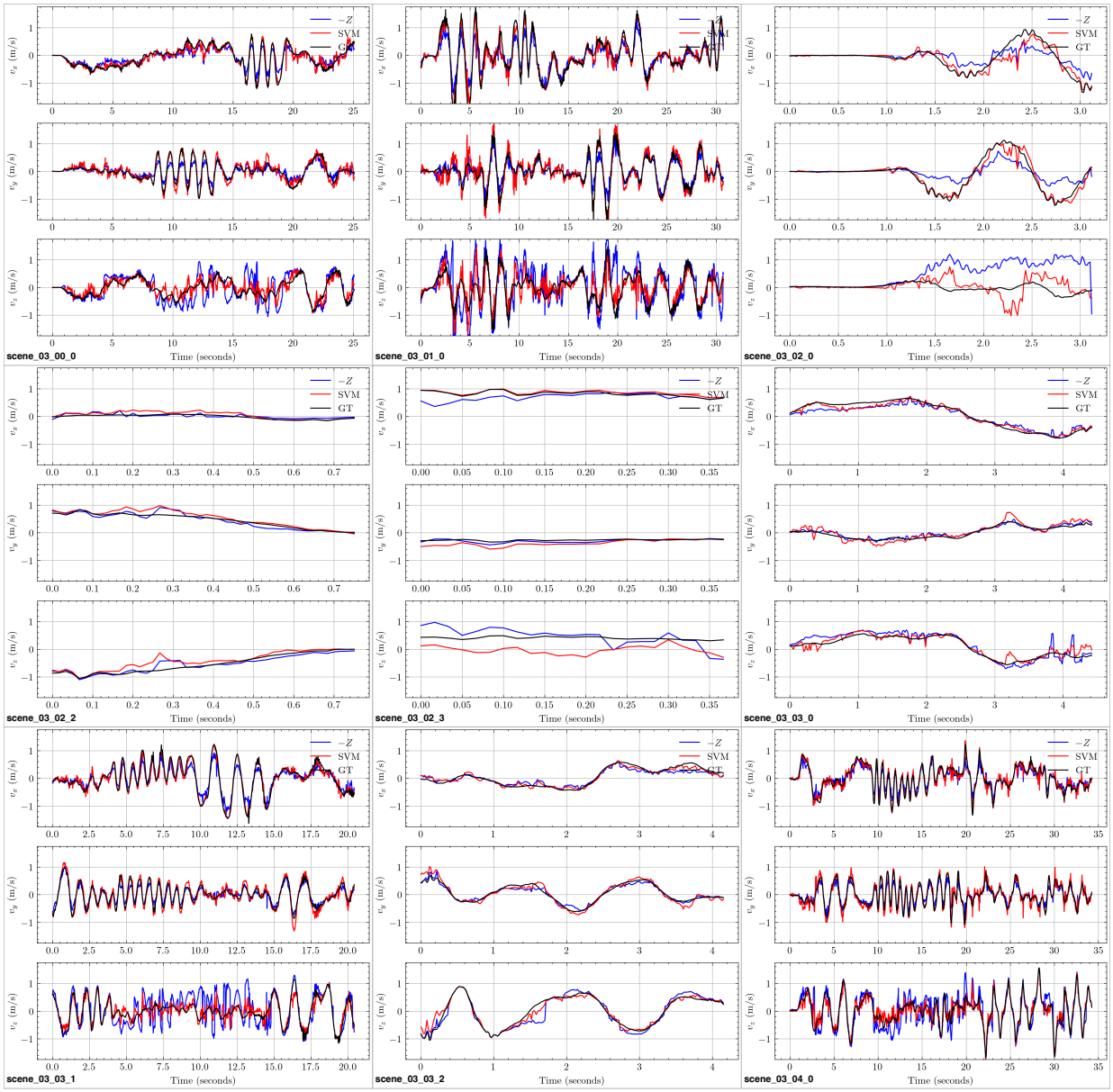


Figure 5.10: Per-scene egomotion evaluation on EVIMO2 `sfm` split.

Chapter 6: Conclusion and Future Work

This thesis introduces and thoroughly investigates Vector Kernel Representation (VKR), a novel approach to point set representation. The work begins by establishing the Vector Kernel Representation Theorem, which demonstrates that a Gaussian kernel mixture can be represented as a fixed-length complex vector through a predefined operation. Building on this theoretical foundation, we explore several applications, showing that VKR consistently delivers superior performance and greater computational efficiency compared to existing point set representation methods.

In Chapter 3, we formally establish VKR and initially demonstrate its effectiveness in function-to-function mapping tasks and point cloud processing. Chapter 4 leverages VKR's inherent equivariance properties to enhance both the performance and efficiency of local point cloud encoding. In Chapter 5, we extend VKR's application to event-based normal flow estimation, showing great potential of point-based methods for event stream processing. We find VKR's advantages not only in accuracy but also in generalizability across diverse domains, outperforming traditional voxel-based methods.

Building on its theoretical foundation and promising applications, I outline several potential future directions where VKR can offer benefits and advancements.

1. Time Series Data Modeling: Time series data can be seen as samples taken from an

underlying function, which makes VKR a good fit for representing them. VKR can easily handle data of different temporal resolutions and is naturally resistant to noise and missing values. This makes it a strong option for time series tasks, especially in cases where traditional models have trouble dealing with incomplete or noisy data.

2. VKR + RNN Architectures for Event-Based Tasks: In Chapter 5, VKR was applied to estimate event-based normal flow using data from short time intervals, achieving superior generalizability compared to voxel-based methods. A natural extension is to integrate VKR with temporal models such as recurrent neural networks (RNNs) or state-space models (SSMs) to capture longer-term temporal dependencies. This hybrid architecture could enhance performance in tasks like event-based object detection and tracking, where understanding motion over time is critical.

3. Moving Object Segmentation: Our work in Chapter 5 showed that event-based normal flow estimation is reliable even across different domains, and egomotion estimation in static scenes is highly accurate using VKR-derived normal flow. This opens new possibilities for moving object segmentation, where an end-to-end pipeline combining both segmentation and egomotion estimation could be developed. By leveraging VKR's robustness and efficiency, such a system could significantly improve performance in dynamic scene understanding.

This thesis not only establishes VKR as a powerful tool for point set representation but also lays the groundwork for future research in time series analysis, event-based vision, and point cloud processing. The potential for VKR to drive advancements in these fields is both exciting and substantial.

Bibliography

[1] Federico Paredes-Vallés, Kirk YW Scheper, Christophe De Wagter, and Guido CHE De Croon. Taming contrast maximization for learning sequential, low-latency, event-based optical flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9695–9705, 2023.

[2] Shintaro Shiba, Yoshimitsu Aoki, and Guillermo Gallego. Secrets of event-based optical flow. In *European Conference on Computer Vision*, pages 628–645. Springer, 2022.

[3] Francisco Barranco, Cornelia Fermüller, Yiannis Aloimonos, and Eduardo Ros. Joint direct estimation of 3d geometry and 3d motion using spatio temporal gradients. *Pattern Recognition*, 113:107759, 2021.

[4] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas 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*, pages 652–660, 2017.

[5] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the cloud: Learning curves for point clouds shape analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 915–924, 2021.

[6] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *arXiv preprint arXiv:2202.07123*, 2022.

[7] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019.

[8] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[9] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019.

[10] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European conference on computer vision (ECCV)*, pages 87–102, 2018.

[11] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[12] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

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

[14] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[15] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

[16] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

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

[18] Mathias Gehrig, Mario Millhäusler, Daniel Gehrig, and Davide Scaramuzza. E-raft: Dense optical flow from event cameras. In *2021 International Conference on 3D Vision (3DV)*, pages 197–206. IEEE, 2021.

[19] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*, 2018.

[20] Zhexiong Wan, Yuchao Dai, and Yuxin Mao. Learning dense and continuous optical flow from an event camera. *IEEE Transactions on Image Processing*, 31:7237–7251, 2022.

[21] Zhexiong Wan, Yuxin Mao, Jing Zhang, and Yuchao Dai. Rpeflow: Multimodal fusion of rgb-pointcloud-event for joint optical flow and scene flow estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10030–10040, 2023.

[22] Chi Zhang, Chenxu Jiang, and Lei Yu. Cross-modal learning for optical flow estimation with events. *Signal Processing*, page 109580, 2024.

[23] Haotian Liu, Guang Chen, Sanqing Qu, Yanping Zhang, Zhijun Li, Alois Knoll, and Changjun Jiang. Tma: Temporal motion aggregation for event-based optical flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9685–9694, 2023.

[24] Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Dense continuous-time optical flow from event cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[25] Yaozu Ye, Hao Shi, Kailun Yang, Ze Wang, Xiaoting Yin, Yining Lin, Mao Liu, Yaonan Wang, and Kaiwei Wang. Towards anytime optical flow estimation with event cameras. *arXiv preprint arXiv:2307.05033*, 2023.

[26] E Paxon Frady, Denis Kleyko, Christopher J Kymn, Bruno A Olshausen, and Friedrich T Sommer. Computing on functions using randomized vector representations. *arXiv preprint arXiv:2109.03429*, 2021.

[27] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020.

[28] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020.

[29] Xingcheng Lu, Dehao Yuan, Yiang Chen, and Jimmy CH Fung. Impacts of urbanization and long-term meteorological variations on global pm2. 5 and its associated health burden. *Environmental Pollution*, 270:116003, 2021.

[30] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[31] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404. IEEE, 2017.

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

[33] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. 2008.

[34] Anil K Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989.

[35] Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. Bcr-net: A neural network based on the nonstandard wavelet form. *Journal of Computational Physics*, 384:1–15, 2019.

[36] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[37] Gege Wen, Zongyi Li, Qirui Long, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. Real-time high-resolution co 2 geological storage prediction using nested fourier neural operators. *Energy & Environmental Science*, 16(4):1732–1741, 2023.

[38] Peter I Renn, Cong Wang, Sahin Lale, Zongyi Li, Anima Anandkumar, and Morteza Gharib. Forecasting subcritical cylinder wakes with fourier neural operators. *arXiv preprint arXiv:2301.08290*, 2023.

[39] Vignesh Gopakumar, Stanislas Pamela, Lorenzo Zanisi, Zongyi Li, Anima Anandkumar, and MAST Team. Fourier neural operator for plasma modelling. *arXiv preprint arXiv:2302.06542*, 2023.

[40] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[41] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[42] Mor Joseph-Rivlin, Alon Zvirin, and Ron Kimmel. Momen (e) t: Flavor the moments in learning to classify shapes. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.

[43] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3323–3332, 2019.

[44] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5565–5573, 2019.

[45] Yueqi Duan, Yu Zheng, Jiwen Lu, Jie Zhou, and Qi Tian. Structural relational reasoning of point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 949–958, 2019.

[46] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5589–5598, 2020.

[47] Hongxin Lin, Zelin Xiao, Yang Tan, Hongyang Chao, and Shengyong Ding. Justlookup: One millisecond deep feature extraction for point clouds by lookup tables. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 326–331. IEEE, 2019.

[48] Yanni Ma, Yulan Guo, Hao Liu, Yinjie Lei, and Gongjian Wen. Global context reasoning for semantic segmentation of 3d point clouds. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2931–2940, 2020.

[49] Zihao Li, Shu Zhang, Junge Zhang, Kaiqi Huang, Yizhou Wang, and Yizhou Yu. Mvp-net: multi-view fpn with position-aware attention for deep universal lesion detection. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part VI 22*, pages 13–21. Springer, 2019.

[50] Charles R Qi, Xinlei Chen, Or Litany, and Leonidas J Guibas. Imvotenet: Boosting 3d object detection in point clouds with image votes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4404–4413, 2020.

[51] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11040–11048, 2020.

[52] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[53] Denis Kleyko, Dmitri Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *ACM Computing Surveys*, 55(9):1–52, 2023.

[54] Brent Komer and Chris Eliasmith. Efficient navigation using a scalable, biologically inspired spatial representation. In *CogSci*, 2020.

[55] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[56] Chau Hsing Su and Clifford S Gardner. Korteweg-de vries equation and generalizations. iii. derivation of the korteweg-de vries equation and burgers equation. *Journal of Mathematical Physics*, 10(3):536–539, 1969.

[57] MR Tek. Development of a generalized darcy equation. *Journal of Petroleum Technology*, 9(06):45–47, 1957.

[58] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[59] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. Pcpnet learning local shape properties from raw point clouds. In *Computer graphics forum*, volume 37, pages 75–85. Wiley Online Library, 2018.

[60] Qing Li, Yu-Shen Liu, Jin-San Cheng, Cheng Wang, Yi Fang, and Zhizhong Han. Hsurf-net: Normal estimation for 3d point clouds by learning hyper surfaces. *Advances in Neural Information Processing Systems*, 35:4218–4230, 2022.

[61] Qing Li, Huifang Feng, Kanle Shi, Yue Gao, Yi Fang, Yu-Shen Liu, and Zhizhong 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*, pages 13591–13600, 2023.

[62] Xian-Feng Han, Zhi-Ao Feng, Shi-Jie Sun, and Guo-Qiang Xiao. 3d point cloud descriptors: state-of-the-art. *Artificial Intelligence Review*, pages 1–51, 2023.

[63] E Paxon Frady, Denis Kleyko, Christopher J Kymn, Bruno A Olshausen, and Friedrich T Sommer. Computing on functions using randomized vector representations (in brief). In *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, pages 115–122, 2022.

[64] Dehao Yuan, Furong Huang, Cornelia Fermüller, and Yiannis Aloimonos. Decodable and sample invariant continuous object encoder. *arXiv preprint arXiv:2311.00187*, 2023.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[66] Sai Manoj Prakhya, Jie Lin, Vijay Chandrasekhar, Weisi Lin, and Bingbing Liu. 3dhopd: A fast low-dimensional 3-d descriptor. *IEEE Robotics and Automation Letters*, 2(3):1472–1479, 2017.

[67] Xuming Ge. Non-rigid registration of 3d point clouds under isometric deformation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 121:192–202, 2016.

[68] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, page 2. Citeseer, 2010.

[69] Nicolas Vandapel, Daniel F Huber, Anuj Kapuria, and Martial Hebert. Natural terrain classification using 3-d ladar data. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 5117–5122. IEEE, 2004.

[70] Duc Fehr, Anoop Cherian, Ravishankar Sivalingam, Sam Nickolay, Vassilios Morellas, and Nikolaos Papanikolopoulos. Compact covariance descriptors in 3d point clouds for object recognition. In *2012 IEEE international conference on robotics and automation*, pages 1793–1798. IEEE, 2012.

[71] Rudolph Triebel, Kristian Kersting, and Wolfram Burgard. Robust 3d scan point classification using associative markov networks. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2603–2608. IEEE, 2006.

[72] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.

[73] Haoxi Ran, Jun Liu, and Chengjie Wang. Surface representation for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18942–18952, 2022.

[74] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021.

[75] Xian-Feng Han, Zhang-Yue He, Jia Chen, and Guo-Qiang Xiao. 3crossnet: Cross-level cross-scale cross-attention network for point cloud representation. *IEEE Robotics and Automation Letters*, 7(2):3718–3725, 2022.

[76] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. 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*, pages 19313–19322, 2022.

[77] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[78] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7463–7472, 2021.

[79] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8500–8509, 2022.

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

[81] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[82] Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

[83] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

[84] Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J. Pal. Deep complex networks. *CoRR*, abs/1705.09792, 2017. URL `http://arxiv.org/abs/1705.09792`.

[85] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[86] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[87] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[88] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017.

[89] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 989–997, 2019.

[90] Jingxi Chen, Brandon Y Feng, Haoming Cai, Mingyang Xie, Christopher Metzler, Cornelia Fermuller, and Yiannis Aloimonos. Timerewind: Rewinding time with image-and-events video diffusion. *arXiv preprint arXiv:2403.13800*, 2024.

[91] Jingxi Chen, Brandon Y Feng, Haoming Cai, Tianfu Wang, Levi Burner, Dehao Yuan, Cornelia Fermuller, Christopher A Metzler, and Yiannis Aloimonos. Repurposing pre-trained video diffusion models for event-based video interpolation. *arXiv preprint arXiv:2412.07761*, 2024.

[92] Tianyi Xiong, Jiayi Wu, Botao He, Cornelia Fermuller, Yiannis Aloimonos, Heng Huang, and Christopher A Metzler. Event3dgs: Event-based 3d gaussian splatting for fast egomotion. *arXiv preprint arXiv:2406.02972*, 2024.

[93] Yijin Li, Zhaoyang Huang, Shuo Chen, Xiaoyu Shi, Hongsheng Li, Hujun Bao, Zhaopeng Cui, and Guofeng Zhang. Blinkflow: A dataset to push the limits of event-based optical flow estimation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3881–3888. IEEE, 2023.

[94] The aperture problem. `https://elvers.us/perception/aperture`. Accessed: 2024-09-24.

[95] Zhongyang Ren, Bangyan Liao, Delei Kong, Jinghang Li, Peidong Liu, Laurent Kneip, Guillermo Gallego, and Yi Zhou. Motion and structure from event-based normal flow. *arXiv preprint arXiv:2407.12239*, 2024.

[96] Xiuyuan Lu, Yi Zhou, and Shaojie Shen. Event-based visual inertial velometer. *arXiv preprint arXiv:2311.18189*, 2023.

[97] Jinghang Li, Bangyan Liao, Xiuyuan Lu, Peidong Liu, Shaojie Shen, and Yi Zhou. Event-aided time-to-collision estimation for autonomous driving. *arXiv preprint arXiv:2407.07324*, 2024.

[98] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. Event-based visual flow. *IEEE transactions on neural networks and learning systems*, 25(2):407–417, 2013.

[99] Elias Mueggler, Christian Forster, Nathan Baumli, Guillermo Gallego, and Davide Scaramuzza. Lifetime estimation of events from dynamic vision sensors. In *2015 IEEE international conference on Robotics and Automation (ICRA)*, pages 4874–4881. IEEE, 2015.

[100] Dehao Yuan, Cornelia Fermuller, Tahseen Rabbani, Furong Huang, and Yiannis Aloimonos. A linear time and space local point cloud geometry encoder via vectorized kernel mixture (VecKM). In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 57871–57886. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/yuan24b.html`.

[101] Ryad Benosman, Sio-Hoi Ieng, Charles Clercq, Chiara Bartolozzi, and Mandyam Srinivasan. Asynchronous frameless event-based optical flow. *Neural Networks*, 27: 32–37, 2012.

[102] Francisco Barranco, Cornelia Fermüller, and Yiannis Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proceedings of the IEEE*, 102(10): 1537–1556, 2014.

[103] Min Liu and Tobi Delbruck. Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. BMVC, 2018.

[104] Min Liu and Tobi Delbruck. Edflow: Event driven optical flow camera with keypoint detection and adaptive block matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(9):5776–5789, 2022.

[105] Shintaro Shiba, Yoshimitsu Aoki, and Guillermo Gallego. Fast event-based optical flow estimation by triplet matching. *IEEE Signal Processing Letters*, 29:2712–2716, 2022.

[106] Hongzhi You, Yijun Cao, Wei Yuan, Fanjun Wang, Ning Qiao, and Yongjie Li. Vector-symbolic architecture for event-based optical flow. *arXiv preprint arXiv:2405.08300*, 2024.

[107] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3867–3876, 2018.

[108] Himanshu Akolkar, Sio-Hoi Ieng, and Ryad Benosman. Real-time high speed motion prediction using fast aperture-robust event-driven visual flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):361–372, 2020.

[109] Tobias Brosch, Stephan Tschechne, and Heiko Neumann. On event-based optical flow detection. *Frontiers in neuroscience*, 9:137, 2015.

[110] Francisco Barranco, Cornelia Fermüller, and Yiannis Aloimonos. Bio-inspired motion estimation with event-driven sensors. In *Advances in Computational Intelligence: 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Part I 13*, pages 309–321. Springer, 2015.

[111] Mohammed Almatrafi, Raymond Baldwin, Kiyoharu Aizawa, and Keigo Hirakawa. Distance surface for event-based optical flow. *IEEE transactions on pattern analysis and machine intelligence*, 42(7):1547–1556, 2020.

[112] Vincent Brebion, Julien Moreau, and Franck Davoine. Real-time optical flow for vehicular perception with low-and high-resolution event cameras. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):15066–15078, 2021.

[113] Zengyu Wan, Yang Wang, Zhai Wei, Ganchao Tan, Yang Cao, and Zheng-Jun Zha. Event-based optical flow via transforming into motion-dependent view. *IEEE Transactions on Image Processing*, 2024.

[114] Yilun Wu, Federico Paredes-Vallés, and Guido CHE De Croon. Lightweight event-based optical flow estimation via iterative deblurring. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14708–14715. IEEE, 2024.

[115] Xinglong Luo, Kunming Luo, Ao Luo, Zhengning Wang, Ping Tan, and Shuaicheng Liu. Learning optical flow from event camera with rendered dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9847–9857, 2023.

[116] Xinglong Luo, Ao Luo, Zhengning Wang, Chunyu Lin, Bing Zeng, and Shuaicheng Liu. Efficient meshflow and optical flow estimation from event cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19198–19207, 2024.

[117] Jesse Hagenaars, Federico Paredes-Valles, and Guido de Croon. Self-supervised learning of event-based optical flow with spiking neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 7167–7179. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/39d4b545fb02556829aab1db805021c3-Paper.pdf`.

[118] Javier Cuadrado, Ulysse Rançon, Benoit R Cottereau, Francisco Barranco, and Timothée Masquelier. Optical flow estimation from event-based cameras and spiking neural networks. *Frontiers in Neuroscience*, 17:1160034, 2023.

[119] Rui Zhao, Ruiqin Xiong, Jing Zhao, Zhaofei Yu, Xiaopeng Fan, and Tiejun Huang. Learning optical flow from continuous spike streams. *Advances in Neural Information Processing Systems*, 35:7905–7920, 2022.

[120] Yisa Zhang, Hengyi Lv, Yuchen Zhao, Yang Feng, Hailong Liu, and Guoling Bi. Event-based optical flow estimation with spatio-temporal backpropagation trained spiking neural network. *Micromachines*, 14(1):203, 2023.

[121] Wachirawit Ponghiran, Chamika Mihiranga Liyanagedera, and Kaushik Roy. Event-based temporally dense optical flow estimation with sequential learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9827–9836, 2023.

[122] Yingfu Xu, Guangzhi Tang, Amirreza Yousefzadeh, Guido de Croon, and Manolis Sifalakis. Event-based optical flow on neuromorphic processor: Ann vs. snn comparison based on activation sparsification. *arXiv preprint arXiv:2407.20421*, 2024.

[123] Renrui Zhang, Liuhui Wang, Yali Wang, Peng Gao, Hongsheng Li, and Jianbo Shi. Starting from non-parametric networks for 3d point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5344–5353, 2023.

[124] Yue Wu, Jiaming Liu, Maoguo Gong, Zhixiao Liu, Qiguang Miao, and Wenping Ma. Mpct: Multiscale point cloud transformer with a residual network. *IEEE Transactions on Multimedia*, 2023.

[125] Hongwei Ren, Yue Zhou, Jiadong Zhu, Haotian Fu, Yulong Huang, Xiaopeng Lin, Yuetong Fang, Fei Ma, Hao Yu, and Bojun Cheng. Rethinking efficient and effective point-based networks for event camera classification and regression: Eventmamba. *arXiv preprint arXiv:2405.06116*, 2024.

[126] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. Eventnet: Asynchronous recursive event processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3887–3896, 2019.

[127] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. Space-time event clouds for gesture recognition: From rgb cameras to event cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1826–1835. IEEE, 2019.

[128] Hongwei Ren, Yue Zhou, Yulong Huang, Haotian Fu, Xiaopeng Lin, Jie Song, and Bojun Cheng. Spikepoint: An efficient point-based spiking neural network for event cameras action recognition. *arXiv preprint arXiv:2310.07189*, 2023.

[129] Bas J Pijnacker Hordijk, Kirk YW Scheper, and Guido CHE De Croon. Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, 35(1):69–90, 2018.

[130] Tomáš Brodskỳ, Cornelia Fermüller, and Yiannis Aloimonos. Structure from motion: Beyond the epipolar constraint. *International Journal of Computer Vision*, 37:231–258, 2000.

[131] Cornelia Fermüller. Passive navigation as a pattern recognition problem. *International journal of computer vision*, 14(2):147–158, 1995.

[132] Chethan M Parameshwara, Gokul Hari, Cornelia Fermüller, Nitin J Sanket, and Yiannis Aloimonos. Diffposenet: Direct differentiable camera pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6845–6854, 2022.

[133] scipy.stats.circstd. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.circstd.html`. Accessed: 2024-11-07.

[134] Alex Zihao Zhu, Dinesh Thakur, Tolga Özaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

[135] Levi Burner, Anton Mitrokhin, Cornelia Fermüller, and Yiannis Aloimonos. Evimo2: an event camera dataset for motion segmentation, optical flow, structure from motion,

and visual inertial odometry in indoor scenes with monocular or stereo algorithms. *arXiv preprint arXiv:2205.03467*, 2022.

[136] Mathias Gehrig, Mario Millhäusler, Daniel Gehrig, and Davide Scaramuzza. E-raft: Dense optical flow from event cameras. In *International Conference on 3D Vision (3DV)*, 2021.