

MeshHD: Near-Linear Encoding for Hyperdimensional Computing via Multi-Scale Bases and Kronecker Factorization

Woong Jae Han¹, Jiseung Kim¹, Hyukjun Kwon¹, Hojeong Kim¹,
Selim An¹, Shinhyoung Jang¹, and Yeseong Kim¹

¹DGIST

{jerry1101, js980408, durwjsdnehd3523, cabil1124, phantom06, shinhyoung.jang, yeseongkim}@dgist.ac.kr

Abstract—Hyperdimensional (HD) computing is attractive for low-power platforms, but common encoders flatten inputs and treat neighboring features as independent, discarding spatial structure and inflating the cost of a dense $F \times D$ apply. We present MESHHD, a spatially aware, *relative* and *multi-scale* base that maps 2D coordinates with random Fourier features to approximate a distance kernel; nearby locations receive similar hypervectors regardless of absolute position. We further introduce a compact Kronecker-structured apply that realizes the bundled base with three small GEMMs, reducing arithmetic and weight movement from $O(FD)$ toward a near-linear form while preserving encoder semantics. Our experimental results show that MESHHD consistently improves accuracy over the state-of-the-art nonlinear HD encoders, especially at smaller D , and reduces per-batch encoding time by $\sim 3\times$, with up to $10\times$ savings in encoder MACs/state at $D=10,000$.

Index Terms—Hyperdimensional Computing, HDC encoding, Spatial-aware, Kronecker Factorization

I. INTRODUCTION

Hyperdimensional computing (HDC) offers a lightweight learning pipeline for embedded and edge systems because training and inference are performed with simple operations in a high-dimensional space [1]. The pipeline begins with an encoder that maps each input feature to a high-dimensional vector. The model then mixes feature vectors elementwise (binding) and aggregates them into a single representation (bundling). In common practice, encoders flatten inputs and assign quasi-orthogonal hypervectors to features, which ignores the spatial or temporal relations inherent to structured data. This issue is not confined to images. Many two-dimensional signals, such as depth maps and spectrograms [2], and one-dimensional signals such as multichannel sensor or biomedical time series [3], [4], exhibit coordinate-like dependencies that current bases do not model. As a result, the classifier must infer locality and neighborhood structure from labels alone, which slows convergence and limits accuracy [5].

The source of this limitation is the independence assumption built into per-feature standard bases. In most state-of-the-art encoding methods, random or pseudo-orthogonal hypervectors are designed to be unrelated across features, which means adjacent pixels in a grid or adjacent samples in a sequence are mapped to distant directions in the representation space [6]. When geometry is absent at the encoder, the learner has to reconstruct it indirectly from supervision. This increases data requirements and makes predictions sensitive to changes that alter coordinates but not semantics, such as small shifts [7].

A few prior works have attempted to introduce structure in the encoding method [8]–[11]. For example, StemHD incorporates absolute coordinates via interpolation and linear

distance heuristics and reports gains over random encoders [8]. However, absolute indexing ties the base to the coordinate frame. A translation changes all indices, resampling compresses or expands them, and padding or cropping repositions identical motifs. Under these commonplace transformations, absolute-coordinate bases change even when the underlying pattern is the same, which degrades generalization unless the model relearns invariances for each perturbation [12], [13]. These observations motivate a base that encodes *relative* relations, so that similarity depends on pairwise distance rather than absolute position.

In addition to the quality of hypervector encoding, the cost of encoding also poses a significant challenge for deploying HDC in real-world computing platforms. In state-of-the-art encoders, a dense basis of size $F \times D$ is applied to F input features (e.g., $F = H \times W$ for a 2D grid or $F = T$ for a sequence), which costs $O(FD)$ multiply-accumulate operations and moves $O(FD)$ parameters per sample, dominating time and energy.

These findings raise a central question: *can a relative, coordinate-invariant, and multi-scale base for HDC, paired with a structured encoder that avoids the dense $O(FD)$ apply, improve accuracy on 1D/2D structured data while substantially reducing encoding time?* Addressing this question entails three coupled challenges: (i) represent inter-feature relations explicitly so that similarity reflects relative proximity across both fine and coarse neighborhoods in both modalities; (ii) replace absolute-coordinate dependencies with a coordinate-invariant representation that remains stable under shifts, resampling, and mild deformations; and (iii) cut encoder cost of the dense $O(FD)$ apply to lower arithmetic and parameter movement without harming accuracy.

In this paper, we introduce *MeshHD*, an HDC encoder that enables relative, multi-scale coordinate encoding and an effective factorization for fast end-to-end training/inference across 2D and 1D inputs. First, MeshHD constructs a *relative* base using random Fourier features (RFF) over coordinates. RFF approximates a distance kernel [14], so two locations receive similar embeddings when their separation is small, independent of absolute indices. Aggregating multiple length scales captures both fine and medium-range neighborhoods. The same construction applies to two-dimensional grids by embedding pixel coordinates and to one-dimensional sequences by embedding sample indices, which yields robustness to small shifts, resampling, and mild warps while preserving locality. Unlike absolute-position bases such as StemHD, this design encodes *relative* proximity directly and operates uniformly across grids and sequences.

Second, MeshHD makes the new encoding scheme with this base efficient to apply by replacing a single dense projection with a compact Kronecker-structured chain. The transformation retains the semantics of the RFF base and evaluates the encoder with three small matrix multiplies. Concretely, the factorization reduces per-sample arithmetic from FD MACs to $Fb_1 + d_1b_1b_2 + b_1D$ (and to $Fb_1 + b_1b_2 + b_1D$ in 1D), lowering compute and parameter movement without compromising the prediction accuracy. In practice $b_1, b_2, d_1 \ll \min(F, D)$ and much smaller than F and D , so the encoder cost scales nearly linearly in F and D rather than multiplicatively, yielding substantial savings over the dense baseline. Since the optimization technique uses ubiquitous dense-matrix primitives, it can be mapped effectively on various accelerator platforms, e.g., in practice to *batched GEMM* on commodity GPUs with state-of-the-art GEMM kernel libraries like cuBLAS [15].

- **MeshHD framework (encoding-based HDC).** We propose *MeshHD*, an encoding-based HDC framework that is *relative*, *coordinate-invariant*, and *multi-scale*, which can be plugged into standard HD classifiers without architectural changes and offer improved prediction quality.
- **Relative base encoding.** We devise a Random Fourier Feature (**RFF**) construction that maps coordinates to hypervectors by *pairwise distance*, aggregates *multiple length scales*, and yields translation-stable representations under shifts, resampling, and mild warps in both 1D sequence and 2D grid modalities.
- **Kronecker-optimized encoder.** We develop a **Kronecker-structured** factorization of the dense basis that evaluates as three batched matrix multiplies, reducing per-sample compute and parameter movement significantly with minimal accuracy loss of $\leq 0.3\%$ compared to the dense encoder.

In our evaluation conducted on the RTX 5090 and RTX 2080Ti for real-world datasets, we show that MeshHD improves accuracy by 3-4 percentage points over the classic non-linear HDC baselines, including StemHD and reduces *encoding time* by $3\times$. Also, our method reduces the memory required to store the base hypervector by $10\times$ for $D=10,000$.

II. RELATED WORK AND BACKGROUND

A. HDC Primitives

Hyperdimensional computing (HDC) represents data and learned associations using *hypervectors*, i.e., high-dimensional vectors with thousands of components [16]. An *encoding* maps an input x to a hypervector $\mathbf{h}(x) \in \mathbb{R}^D$, where similarity is typically evaluated by cosine similarity. A high-quality encoder places semantically similar inputs close in the hyperspace and unrelated inputs nearly orthogonal.

HDC composes and aggregates information with two primitive operations. *Binding* (denoted \otimes) combines two hypervectors elementwise to produce a representation that is approximately orthogonal to each operand, thereby associating key-value pairs or feature-position pairs. Depending on the hypervector type, binding is realized by elementwise multiplication (real-valued) or XOR (bipolar/binary). *Bundling* (denoted \oplus) aggregates a set $\{\mathbf{h}_i\}$ by elementwise addition (and optional normalization or sign), yielding a composite hypervector that

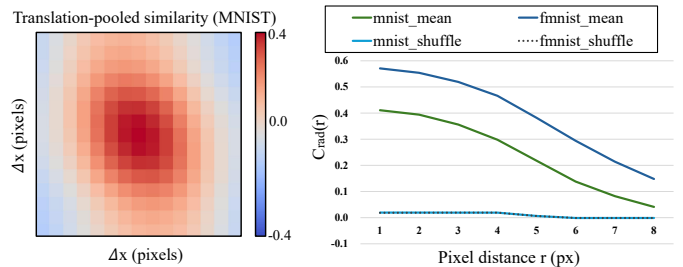


Fig. 1: Inter-feature relationships: heatmap, radial curve.

preserves similarities to its constituents while suppressing unrelated directions. A standard HDC classifier maintains one prototype hypervector per class by iterative bundling during training, and predicts by maximum cosine similarity at inference time.

B. HDC Encoding Methods

Classical HDC encoders construct a basis of quasi-orthogonal hypervectors and map each input feature independently. Common approaches include ID-level encoding (one basis vector per discrete feature), random projection of real-valued features, and nonlinear encodings using Gaussian or bipolar bases [1]–[3]. These encoders are robust and enable fast similarity computation, but they impose an *independence* assumption: adjacent pixels in an image grid or adjacent samples in a sequence are projected to unrelated directions. As a result, locality in the input space is not preserved in the hypervector space, which increases data requirements and limits robustness to small geometric perturbations. To incorporate structure, several encoders exploit spatial or spatiotemporal context [17]–[19]. A representative work is STEMHD [8], which assigns random corner hypervectors to a $k \times k$ partition of the grid, then interpolates interior positions from the corners to maintain local continuity. While this absolute-index construction improves accuracy on seen layouts, it ties the representation to the coordinate frame, i.e., shifts or mild warps change absolute indices and can produce noticeably different hypervectors for the same pattern.

Unlike prior encoding methods, either assuming an independent relationship or utilizing absolute indexing, we construct encodings that reflect *relative* geometry so that hypervector similarity follows kernel distance rather than absolute position. This yields translation-stable similarity and improved robustness to small shifts and resampling, while remaining compatible with the standard HDC training pipeline.

Real-world data, e.g., natural images, exhibit local dependence, e.g., nearby pixels co-vary, and this dependence decays with distance. To better understand inter-feature relationships in real-world data, we first quantify how much spatial locality exists and whether it depends primarily on absolute or pairwise distance. To this end, we conducted a study that, for each image $X \in [0, 1]^{28 \times 28}$, we compute a per-image z -score $Z = (X - \mu)/(\sigma + \epsilon)$ with $\epsilon = 10^{-6}$ to suppress bias from uniform backgrounds. To remove centering effects, we apply translation pooling: for K small integer jitters $(\delta_x, \delta_y) \in \{-3, \dots, 3\}^2$, we shift Z by (δ_x, δ_y) (crop, no wrap) and average statistics over the valid overlap, then over jitters and images. The translation-pooled similarity map $C(\Delta x, \Delta y)$ is defined as the mean of the elementwise product between Z

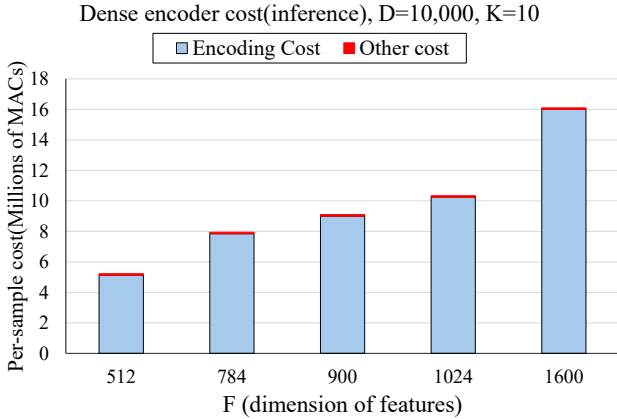


Fig. 2: Encoding cost per feature F

and a copy shifted by $(\Delta x, \Delta y)$ on the overlapping region, $C(\Delta x, \Delta y) = \mathbb{E}[\langle Z, \text{shift}(Z, \Delta x, \Delta y) \rangle_{\text{valid}}]$, where the expectation averages over jitters and the dataset. Fig. 1(a) visualizes $C(\Delta x, \Delta y)$ for MNIST. The map has a bright, near-circular peak at zero offset that decays smoothly with radius, indicating approximately isotropic local dependence and rapid attenuation with distance.

To summarize the dependence on distance, we collapse $C(\Delta x, \Delta y)$ to a radial curve by binning offsets with the same Euclidean radius $r = \sqrt{\Delta x^2 + \Delta y^2}$, $C_{\text{rad}}(r) = \frac{1}{|\mathcal{B}(r)|} \sum_{(\Delta x, \Delta y) \in \mathcal{B}(r)} C(\Delta x, \Delta y)$, and report mean and bootstrap 95% confidence intervals across images in Fig. 1(b). We overlay $C_{\text{rad}}(r)$ for MNIST and Fashion-MNIST and include a pixel-shuffle control that permutes pixels uniformly at random before the same computation. Both datasets exhibit a clear, monotone distance-decay, with Fashion-MNIST showing stronger short-range correlation. The shuffled control is nearly flat, confirming that the effect arises from spatial arrangement rather than marginal intensity statistics. These results support using relative proximity as the similarity prior that the encoder should preserve.

C. Encoding Cost

We evaluate the computational burden of the encoder to motivate the need for more efficient constructions. In a standard HDC pipeline, the classifier and prototype updates require only $O(DK)$ operations for K classes, which is lightweight even for $D \approx 10^4$ and $K = 10$. By contrast, the encoder applies a dense basis $B \in \mathbb{R}^{F \times D}$ to each input $x \in \mathbb{R}^F$, $\text{HV}(x) = x^\top B$, which entails $O(FD)$ multiply-accumulate operations (MACs) and streams $F \times D$ parameters from memory for every sample.

For feature counts typical of image and sequence domains, $F \in \{512, 784, 900, 1024, 1600\}$ with $D \approx 10^4$, the encoder requires millions to hundreds of millions of MACs per sample. Figure 2 illustrates how this cost grows linearly with F . It shows that as the feature dimension increases, the encoder quickly dominates the inference budget and accounts for nearly all execution time. This imbalance motivates our structured encoder design, which reduces both arithmetic and memory traffic while preserving accuracy.

III. MESHHD DESIGN

A. Overview

Figure 3 illustrates the MeshHD framework, which enables fast end-to-end HDC training/inference across 2D and 1D

inputs based on relative, multi-scale coordinate encoding and an effective factorization. MeshHD keeps the standard HDC workflow of encoding, training class prototype hypervectors representing each class, and inference procedure based on similarity measures. The main goal of MeshHD is to make hypervector similarity reflect relative proximity among features on the input grid rather than absolute feature indices, while keeping the downstream HDC machinery intact. Given F input features (for example, a flattened image) and their 2D coordinates $P = \{p_i \in \mathbb{R}^2\}_{i=1}^F$, MeshHD constructs a coordinate-driven base using random Fourier features so that two locations that are close on the grid map to hypervectors with high inner product. To capture structure at multiple neighborhood sizes, the encoder builds several bases at different bandwidths and bundles them into a single representation.

To make this base practical at scale, MeshHD applies a structured encoder that realizes the bundled basis through a Kronecker-factorized apply implemented as three small GEMMs. This reduces arithmetic and parameter movement compared to a dense $F \times D$ matrix while preserving the geometry of the multi-scale mapping. The rest of the HDC pipeline is unchanged, i.e., class prototypes are learned on the resulting encoded hypervectors, and inference uses cosine similarity. In summary, MeshHD introduces a relative, multi-scale coordinate encoding and a compact apply that maintains encoder semantics and lowers the $O(FD)$ cost of dense encoding.

B. Mesh base generator

Classical HDC encoders assign each feature an independent, quasi-orthogonal hypervector. This design treats nearby pixels or samples as unrelated, so the classifier must rediscover geometric relations from labels. It slows convergence and reduces robustness under small shifts, uniform resampling, or mild warps. Our aim is an encoder whose similarity reflects *relative proximity* on the input grid across multiple neighborhood sizes, while remaining lightweight and compatible with standard HDC training and inference.

Random Fourier Features (RFF) provide a simple way to make similarity depend on distance rather than absolute indices. For 2D coordinates $p \in \mathbb{R}^2$, we map

$$z_\sigma(p) = \sqrt{\frac{2}{D}} [\cos(w_1^\top p + b_1), \sin(w_1^\top p + b_1), \dots, \cos(w_{D/2}^\top p + b_{D/2}), \sin(w_{D/2}^\top p + b_{D/2})], \quad (1)$$

with $w_j \sim \mathcal{N}(0, \sigma^{-2} I_2)$ and $b_j \sim \text{Unif}[0, 2\pi)$. Inner products of these features approximate a radial basis function, $\langle z_\sigma(p_i), z_\sigma(p_j) \rangle \approx \exp(-\|p_i - p_j\|^2 / (2\sigma^2))$, so two locations that are close on the grid map to highly similar hypervectors and similarity decays smoothly with distance. Because the kernel depends on pairwise distances, the representation is stable under small shifts and uniform resampling once coordinates are placed on a consistent scale.

Figure 4 illustrates our proposed procedure for the base hypervector generation. We construct a single-scale basis $B(\sigma) \in \mathbb{R}^{F \times D}$ by evaluating $z_\sigma(p)$ at each grid location and stacking the results by row. To capture both fine and coarse neighborhoods, we generate several bases at different bandwidths $\{\sigma_m\}$ and combine them into a multi-scale base (concatenation or a weighted sum; the choice and weights

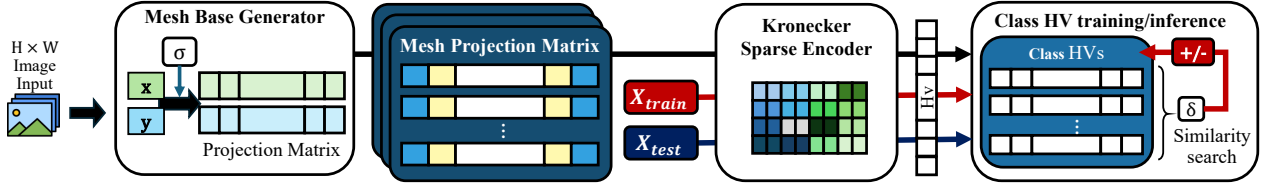


Fig. 3: Overview of MeshHD

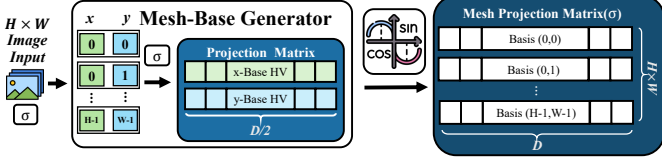


Fig. 4: Illustration of Mesh Base Hypervector Generation

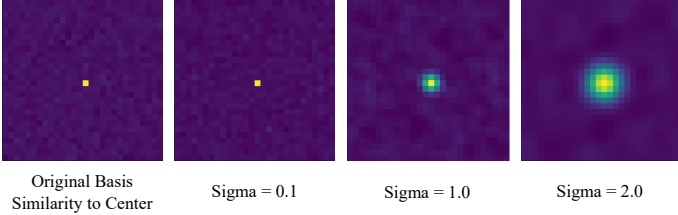


Fig. 5: Visualization of Similarity between Basis with Proposed Mesh Hypervector Bases

are described later). Figure 5 visualizes the induced similarity around a reference pixel. It shows that classical encoders remain near-orthogonal off the diagonal, while our RFF basis produces a bull’s-eye pattern. Small σ yields a tightly localized footprint that emphasizes local neighborhoods; larger σ broadens the footprint to reflect longer-range relationships. This relative, multi-scale mapping preserves the usual HDC pipeline, i.e., after encoding, class prototypes are learned as in standard HDC, and inference uses cosine similarity.

Algorithm 1 summarizes the single-scale construction. The input dimensionality D must be even because we form cosine pairs per frequency. We build the coordinate grid p_i for the $H \times W$ layout (we normalize coordinates to a fixed range for stability), sample frequencies and phases, and fill the $F \times D$ matrix with alternating \cos and \sin columns. Multi-scale bundling then applies this routine for each σ_m and combines the resulting bases as specified above. The structured approach that realizes the bundled base efficiently is developed in the next subsection.

C. Multi-scale bundling and relative similarity

A single bandwidth σ forces a trade-off. Small σ creates sharply localized hypervector neighborhoods that preserve fine structure but miss medium-range relations; large σ captures broader context but blurs detail. We therefore combine a few distance scales so the encoder responds to both fine and medium-range neighborhoods in one base.

Concretely, we precompute a small pool of single-scale bases $\{B^{(\sigma_\ell)}\}_{\ell=1}^L$ by evaluating the RFF map at coordinates p_i with bandwidths $\{\sigma_\ell\}$ that span a geometric range on the normalized grid (e.g., from a few pixels up to a fraction of the image size).

Algorithm 1 Generate MESH basis($D, H, W, \sigma, seed$)

Require: $D \in 2\mathbb{N}$ (even), $H, W \in \mathbb{N}$, $\sigma > 0$, optional *seed*

Ensure: $B \in \mathbb{R}^{F \times D}$ with $F = H \cdot W$, $D_2 = D/2$

- 1: $F \leftarrow H \cdot W$, $D_2 \leftarrow D/2$
- 2: **if** *seed* is provided **then**
- 3: set RNG seed
- 4: **end if**
- 5: Build coordinate grid $p_i \in \mathbb{R}^2$ for $i = 1, \dots, F$
- 6: Sample $w_j \sim \mathcal{N}(0, I_2)/\sigma$ and $b_j \sim \text{Unif}[0, 2\pi]$
- 7: **for** $i = 1$ to F **do**
- 8: **for** $j = 1$ to D_2 **do**
- 9: $\theta_{ij} \leftarrow w_j^\top p_i + b_j$
- 10: $B_{i,2j-1} \leftarrow \cos(\theta_{ij})$, $B_{i,2j} \leftarrow \sin(\theta_{ij})$
- 11: **end for**
- 12: **end for**
- 13: **return** B

The bundled base is a convex mixture

$$\bar{B} = \sum_{\ell=1}^L \alpha_\ell B^{(\sigma_\ell)}, \quad \alpha_\ell \geq 0, \quad \sum_{\ell} \alpha_\ell = 1,$$

which preserves the kernel geometry at each constituent scale while letting the data choose how much of each neighborhood size to emphasize. Using a small mixture (typically $M = 36$ active scales) balances expressiveness and encoder cost, and the simplex constraint regularizes the selection.

We choose the mixture weights with a lightweight evolution-strategy routine that optimizes the metric we actually deploy using validation accuracy after a short HDC retrain. The routine operates in the log-scale domain $\theta_\ell = \log \sigma_\ell$ for numerical stability. At each iteration, it perturbs the current parameters with zero-mean Gaussian noise, converts the perturbed parameters to mixture weights via softmax, forms the bundled base as the weighted sum of *precomputed* single-scale bases, and scores the candidate by running a few quick epochs of prototype retraining on a small train/validation split. After a fixed budget, we retain the top- k scales by average weight and renormalize. In practice, this procedure converges quickly and yields mixtures that preserve sharp locality while adding enough longer-range context to improve robustness.

D. Training and inference pipeline

Given an input image, we form a coordinate mesh over height and width, normalize it to a fixed range, and evaluate the RFF map at several bandwidths to build the pool $\{B^{(\sigma_\ell)}\}$. The multi-scale bundling step produces the convex combination \bar{B} , which remains fixed for the dataset. Each flattened image $x \in \mathbb{R}^F$ is encoded once as a hypervector $\vec{H} = x^\top \bar{B}$ (or via the fast structured apply introduced next). The downstream classifier follows standard HDC.

Let $\{\vec{C}_k \in \mathbb{R}^D\}_{k=1}^K$ denote the class prototypes (rows of a class hypermatrix). We initialize \vec{C}_k with a single pass of zeros. Retraining then fine-tunes the prototypes over multiple epochs.

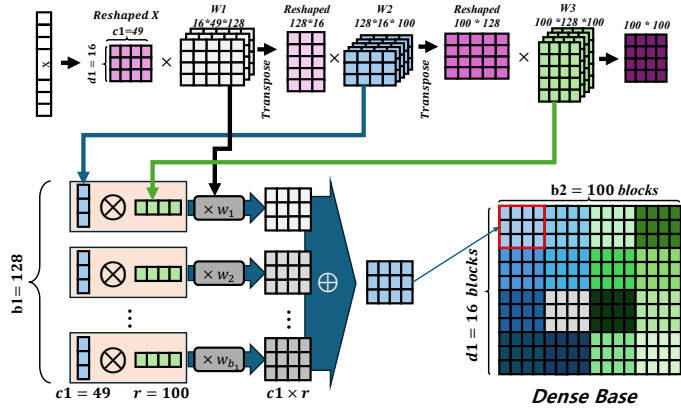


Fig. 6: Kronecker Factorization

For each sample hypervector \vec{H} with label y we compute the most similar class

$$\hat{k} = \arg \max_{k \in \{1, \dots, K\}} \delta(\vec{C}_k, \vec{H}),$$

where δ is cosine similarity for real-valued hypervectors. If $\hat{k} \neq y$, we update

$$\vec{C}_y \leftarrow \vec{C}_y + \eta \vec{H}, \quad \vec{C}_{\hat{k}} \leftarrow \vec{C}_{\hat{k}} - \eta \vec{H},$$

with a small step size η (often $\eta=1$ suffices). We optionally apply per-prototype normalization or clipping to keep magnitudes comparable and to stabilize cosine similarities. This additive update accumulates the hyperdimensional pattern of \vec{H} into the correct class while reducing it in the strongest competitor, and repeating over several epochs yields a compact, well-separated set of prototypes. At inference, the bundled base \vec{B} is fixed; each query is encoded once and classified by the same similarity rule.

This pipeline makes the computational bottleneck explicit. Prototype updates and similarity evaluations are $O(DK)$ and lightweight; both training and inference are dominated by the encoding step, i.e., applying \vec{B} to each input. A dense apply costs $O(FD)$ MACs and streams $F \times D$ parameters per sample. The next section introduces a Kronecker factorization that realizes the same multi-scale base with three small GEMMs, substantially reducing arithmetic and memory traffic while preserving encoder semantics.

IV. KRONECKER PRODUCT-BASED OPTIMIZATION

A. Kronecker-structured factorization of the RFF basis

The dense RFF encoder $B \in \mathbb{R}^{F \times D}$ preserves the desired geometry but requires FD multiply-accumulate operations and streams FD parameters per sample, which quickly dominates runtime. Our goal is to realize the same mapping with far lower arithmetic and memory cost.

Figure 6 illustrates how we decompose the dense mesh base hypervectors into a lightweight mapping based on factorization. The construction of our proposed method exploits separability. Each RFF basis column has the form $\cos(w_x x + w_y y + b)$ or $\sin(\cdot)$, which can be decomposed into products of one-dimensional functions using angle-addition identities. This means each basis vector is well-approximated as a low-rank Kronecker product of row-only and column-only components [20]. By extension, the bundled multi-scale basis

lies close to a low-rank Kronecker subspace. We therefore approximate B by a structured chain

$$\hat{B} \approx (K_2 \otimes K_1) K_3,$$

where $K_1 \in \mathbb{R}^{H \times b_2}$ mixes along the vertical axis, $K_2 \in \mathbb{R}^{W \times b_1}$ mixes along the horizontal axis, and $K_3 \in \mathbb{R}^{(b_1 b_2) \times D}$ recombines intermediate channels into the target dimension. Encoding then reduces to

$$y = K_3^\top \text{vec}(K_1^\top X K_2),$$

where $X \in \mathbb{R}^{H \times W}$ is the input grid and $\text{vec}(\cdot)$ flattens the intermediate channels. This form uses three small matrix contractions separated by lightweight reshapes, rather than a single dense $F \times D$ multiply.

For a given mesh base hypervectors, B , we obtain the factors (K_1, K_2, K_3) by minimizing the Frobenius reconstruction error $\|(K_2 \otimes K_1) K_3 - B\|_F^2$, which can be solved by fitting a three-layer neural network model. Because the number of parameters is only $Fb_1 + d_1 b_1 b_2 + b_1 D$, training is lightweight and converges quickly. Note that this procedure runs offline and is data-independent. Thus, once we obtain the three matrices offline, the factors remain fixed and can be reused across all inputs during inference and training without incurring the extra cost.

B. Complexity and runtime realization

The factorized encoder reduces both arithmetic and parameter movement. Whereas a dense apply requires FD MACs and streams FD weights, the factorized form costs

$$Fb_1 + d_1 b_1 b_2 + b_1 D,$$

with corresponding parameter storage. For $b_1, b_2 \ll \min(F, D)$ this scales nearly linearly in F and D , rather than multiplicatively. In typical image settings ($F = 784$, $D \approx 10^4$), this yields about an order-of-magnitude reduction in both MACs and weight bytes.

At runtime, the mapping is executed as three strided-batched GEMMs separated by reshapes. A minibatch $X \in \mathbb{R}^{B \times F}$ is first viewed as $[B, H, W]$ and contracted with K_2 along the horizontal axis, yielding $[B, H, b_1]$. A reshape folds to $[B \cdot b_1, H]$ for contraction with K_1 , producing $[B, b_1, b_2]$. Finally, contraction with K_3 yields $[B, D]$. This layout maintains batch-first, contiguous memory access so that each GEMM operates on coalesced blocks without explicit transposes. Because the factor sizes are small, weights typically reside in on-chip caches, reducing external bandwidth pressure.

The schedule maps directly to commodity BLAS libraries over various platforms as three strided-batched GEMMs, including common GPU platforms supporting the batched GEMM. The inference cost remains fixed at $Fb_1 + d_1 b_1 b_2 + b_1 D$, independent of the number of scales. This preserves the geometry of the multi-scale RFF mapping while making the encoder, i.e., the bottleneck in both training and inference, fast and bandwidth-efficient.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

We implement the proposed MeshHD framework in PyTorch. The proposed Kronecker product-based optimization is implemented with cuBlas batched GEMM operations. We compare

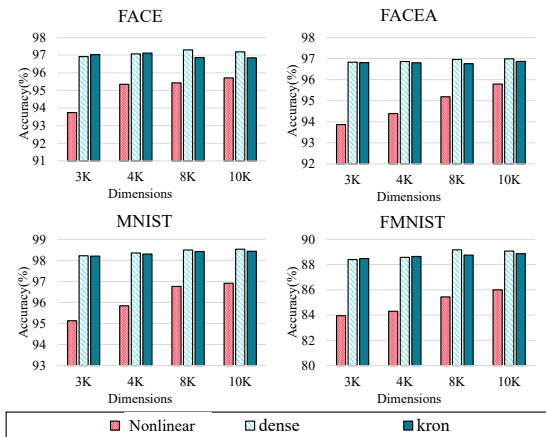


Fig. 7: Evaluation on accuracy for different datasets

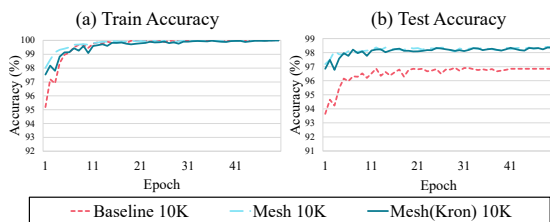


Fig. 8: Training and test accuracy across epochs

the accuracy of our proposed method with classical HD nonlinear encoding and evaluate their efficiency on RTX 2080Ti. We set the hyperdimensionality $D=3K$ to $10K$. In MeshHD, we bundled 7 σ -scales to enhance accuracy. The classification accuracy is measured from the average of the top-3 results from 10 trials with random seeds. We use 4 datasets including MNIST [21], Fashion MNIST [22], FACE, and FACEA [23] to evaluate our model.

B. Classification Accuracy

Figure 7 shows accuracy across different datasets and dimensions. The results present that MESHHD (both *dense* and *kron*) consistently outperforms the classical HD *nonlinear* baseline. The improvements are most visible at smaller D where the multi-scale, locality-preserving bases translate into better class separation. Notably, the accuracy margin is largest at smaller dimensions, i.e., ($D=3K-4K$) and narrows modestly as D increases to $8K-10K$. This pattern indicates that MESHHD uses capacity more efficiently at low dimensions by injecting spatial priors, while the baseline partially catches up only when dimensionality is increased. The *kron* variant matches the *dense* apply within plot resolution across all settings, confirming that factorization does not degrade accuracy. Overall trends are monotone with D in that accuracy rises for all methods as dimensionality increases, but MESHHD maintains a consistent margin over the baseline on every dataset.

Figure 8 plots training and test accuracy over epochs at $D=10K$ on MNIST dataset. MESHHD achieves higher accuracy earlier in training and stabilizes quickly, presenting the benefit of encoding spatial proximity directly in the base hypervectors. The Kronecker-structured version (*Mesh(Kron)*) overlays the *Mesh* curve throughout training and testing, reinforcing that the efficiency-oriented factorization leaves the

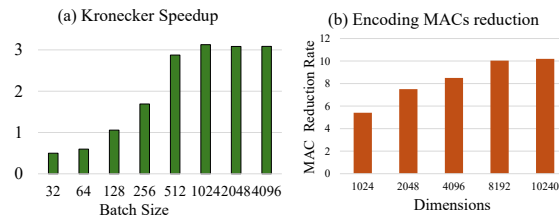


Fig. 9: Performance Evaluation

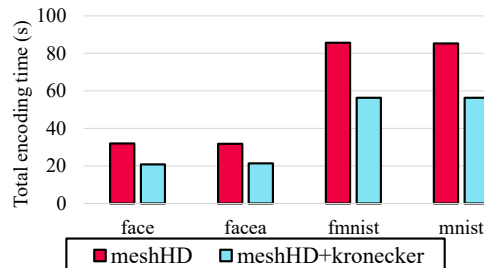


Fig. 10: MeshHD Encoding Time

training trend intact. The classical nonlinear baseline converges more slowly and plateaus at a lower level, consistent with the hypothesis that feature independence in conventional HD encoders forces the classifier to infer locality from labels alone.

C. Computational Efficiency

We evaluate the efficiency of MESHHD with a dense apply vs. a Kronecker apply, producing identical hypervectors (seven σ -scales fused offline). As batch size grows ($32 \rightarrow 4096$), the Kronecker path reaches $\sim 3\times$ speedup (Fig. 9a). Arithmetic cost also drops as D increases because the dense path scales as FD while the factorized path scales as $Fb_1 + d_1b_1b_2 + b_1D$ (Fig. 9b), resulting in lower latency and fewer MACs.

MESHHD+Kronecker consistently reduces full-dataset encoding time relative to the dense apply, with the largest absolute savings on the bigger datasets (MNIST/FMNIST) (Fig. 10). In sum, Kronecker factorization achieves practical wall-clock gains, making MESHHD practical for resource- and latency-constrained deployments.

VI. CONCLUSION

We presented **MeshHD**, a relative, coordinate-invariant, multi-scale encoder for hyperdimensional computing that preserves locality by approximating the distance kernel with 2D random Fourier features and evaluates efficiently via a compact Kronecker-structured apply. MeshHD drops the dense $O(FD)$ encoder bottleneck to a near-linear form, integrates without architectural changes into the standard HDC pipeline, and improves prediction quality and efficiency across 2D data. In our experiments, MeshHD achieves a higher accuracy of about 3pp on average, 3x lower per-batch encoding latency, and 10x reductions in MACs.

ACKNOWLEDGEMENTS

This work was supported by the NRF grant funded by the Korea government (MSIT) (RS-2025-24803164) and the Technology Innovation Program “Development of Navigation Technology Utilizing Visual Information Based on Vision-Language Models for Understanding Dynamic Environments in Non-Learned Spaces” (RS-2024-00445759).

REFERENCES

- [1] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1:139–159, 2009.
- [2] Mohsen Imani, et al. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE international conference on rebooting computing (ICRC)*, pages 1–8. IEEE, 2017.
- [3] Abbas Rahimi, et al. Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.
- [4] Yeseong Kim, et al. Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 115–120, 2020.
- [5] Hyunsei Lee, et al. Efficient forward-only training for brain-inspired hyperdimensional computing. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, pages 707–714, 2024.
- [6] Hyunsei Lee, et al. Hyperdimensional computing-based federated learning in mobile robots through synthetic oversampling. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13406–13412, 2025.
- [7] Jiseung Kim, et al. Advancing hyperdimensional computing based on trainable encoding and adaptive training for efficient and accurate learning. *ACM Trans. Des. Autom. Electron. Syst.*, 29(5), September 2024.
- [8] Jiseung Kim, et al. Efficient brain-inspired hyperdimensional learning with spatiotemporal structured data. In *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8, 2021.
- [9] Igor Nunes, et al. Graphhd: Efficient graph classification using hyperdimensional computing. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1485–1490, 2022.
- [10] Saeid Pourmand, et al. Laplace-hdc: Understanding the geometry of binary hyperdimensional computing. *Journal of Artificial Intelligence Research*, 82:1293–1323, 2025.
- [11] Alejandro Hernández-Cano, et al. Hyperdimensional computing with holographic and adaptive encoder. *Frontiers in Artificial Intelligence*, 7:1371988, 2024.
- [12] Osman Semih Kayhan et al. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14274–14285, 2020.
- [13] Ruilong Li, et al. Cameras as relative positional encoding. *arXiv preprint arXiv:2507.10496*, 2025.
- [14] Ali Rahimi et al. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.
- [15] NVIDIA Corporation. *cuBLAS Library API Reference Guide, Version 13.0*. NVIDIA Corporation, 2025.
- [16] Denis Kleyko, et al. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *arXiv preprint arXiv:2112.15424*, 2021.
- [17] Zhuowen Zou, et al. Eventhd: Robust and efficient hyperdimensional learning with neuromorphic sensor. *Frontiers in Neuroscience*, Volume 16 - 2022, 2022.
- [18] Laura Smets, et al. An encoding framework for binarized images using hyperdimensional computing. *Frontiers in big data*, 7:1371518, 2024.
- [19] Sanggeon Yun, et al. Spatial-aware image retrieval: A hyperdimensional computing approach for efficient similarity hashing. *CoRR*, 2024.
- [20] Mohra Zayed et al. Kronecker product bases and their applications in approximation theory. *Electronic Research Archive*, 33(2), 2025.
- [21] Yann LeCun, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [22] Han Xiao, et al. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [23] Yeseong Kim, et al. Orchard: Visual object recognition accelerator based on approximate in-memory processing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 25–32, 2017.