Title
Computing With Residue Numbers in High-Dimensional Representation.

Journal
Neural Computation, 37(1)

Authors
Kymn, Christopher
Kleyko, Denis
Frady, E
et al.

Publication Date
2024-12-12

DOI
10.1162/neco_a_01723

Peer reviewed

ARTICLE **Communicated by Tony Plate**

Computing With Residue Numbers in High-Dimensional Representation

Christopher J. Kymn
cjkymn@berkeley.edu
Redwood Center for Theoretical Neuroscience, University of California, Berkeley, CA 94720, U.S.A.

Denis Kleyko
denis.kleyko@oru.se
Centre for Applied Autonomous Sensor Systems, Orebro University, Orebro SE-701 82, Sweden, and Intelligent Systems Lab, Research Institutes of Sweden, 164 40 Kista, Sweden

E. Paxon Frady
e.paxon.frady@intel.com
Neuromorphic Computing Lab, Intel, Santa Clara, CA 95054, U.S.A.

Connor Bybee
bybee@berkeley.edu
Pentti Kanerva
pkanerva@csli.stanford.edu
Redwood Center for Theoretical Neuroscience, University of California, Berkeley, CA 94720, U.S.A.

Friedrich T. Sommer
fsommer@berkeley.edu
Redwood Center for Theoretical Neuroscience, University of California, Berkeley, CA 94720, U.S.A., and Neuromorphic Computing Lab, Intel, Santa Clara, CA 95054, U.S.A.

Bruno A. Olshausen
baolshausen@berkeley.edu
Redwood Center for Theoretical Neuroscience, University of California, Berkeley, CA 94720, U.S.A.

We introduce residue hyperdimensional computing, a computing framework that unifies residue number systems with an algebra defined over random, high-dimensional vectors. We show how residue numbers can

Christopher Kymn and Bruno Olshausen are the corresponding authors.

Neural Computation **37, 1–37** (2025)
https://doi.org/10.1162/neco_a_01723

be represented as high-dimensional vectors in a manner that allows algebraic operations to be performed with component-wise, parallelizable operations on the vector elements. The resulting framework, when combined with an efficient method for factorizing high-dimensional vectors, can represent and operate on numerical values over a large dynamic range using resources that scale only logarithmically with the range, a vast improvement over previous methods. It also exhibits impressive robustness to noise. We demonstrate the potential for this framework to solve computationally difficult problems in visual perception and combinatorial optimization, showing improvement over baseline methods. More broadly, the framework provides a possible account for the computational operations of grid cells in the brain, and it suggests new machine learning architectures for representing and manipulating numerical data.

1 Introduction

The problem of representing and computing on high-dimensional representations of numerical values—such as position, velocity, and color—is central to both machine learning and computational neuroscience. In machine learning, vector representations of numbers are useful for defining position or function encodings in neural networks (Sitzmann et al., 2020; Tancik et al., 2020; Vaswani et al., 2017), improving robustness to adversarial examples (Buckman et al., 2018), and generating efficient classifiers (Diao et al., 2021). In neuroscience, experimentalists seek to understand how populations of neurons in the brain represent and transform perceptual or cognitive variables, and so numerous theorists have constructed models for how these variables could be encoded in and decoded from high-dimensional vector encodings (Bordelon & Pehlevan, 2022; Kriegeskorte et al., 2008; Pouget et al., 2000).

A particularly salient example of high-dimensional representation in neuroscience is the grid cell encoding of spatial position in the medial entorhinal cortex (Hafting et al., 2005). Grid cells have multiple peaks in their firing rates that correlate with spatial positions arranged in a hexagonal lattice. While such a coding scheme may seem somewhat perplexing at first glance, its usefulness becomes apparent from how it functions as a population code. In comparison to a population of neurons with traditional unimodal encoding functions whose coding resolution increases linearly with the number of neurons, a grid cell population possesses a coding resolution that grows exponentially in the number of neurons (Mathis et al., 2012). In particular, Fiete and colleagues have emphasized that this advantage of grid cell encoding uses properties of residue numbers (see section 2.2; Fiete et al., 2008).

Inspired by this observation, we propose a comprehensive algebraic framework for distributed neural computation based on residue number

Table 1: Existing High-Dimensional Vector-Based Schemes for Encoding Numbers (First Five Rows) in Comparison to Our Proposed Framework (Last Row).

Encoding Scheme	Algebra	Expressivity	Efficient Decoding	Robust to Noise
One-hot	×	×	✓	×
Thermometer (Penz, 1987), appendix A.1	×	×	✓	∼
Float (Goltsev, 1996), appendix A.2	×	×	✓	∼
Gaussian population codes (Pouget et al., 2000)				
Scatter (Smith & Stanford, 1990), appendix A.3	×	×	×	✓
Fractional power encoding (Plate, 1994), section 2.1	∼	✓	×	✓
Residue hyperdimensional computing	✓	✓	✓	✓

systems. Our novel framework builds on an existing algebraic framework for computing with high-dimensional random vectors, originally called holographic reduced representation (Plate, 1994) and now commonly referred to, synonymously, as vector symbolic architectures (VSA; Gayler, 2003) or hyperdimensional computing (Kanerva, 2009). We call our new framework residue hyperdimensional computing (RHC) and demonstrate that it inherits the computational advantages of both standard residue number systems and hyperdimensional computing. This enables fault-tolerant computations that can efficiently represent numbers and search over a large dynamic range with greatly reduced memory requirements. Furthermore, as we shall see, the new framework provides a useful formalism for understanding computations in grid cells.

To summarize, we list the four key coding properties we achieve with RHC: (1) algebraic structure: simple operations on vectors perform addition and multiplication on encoded values; (2) expressivity: feasible encoding range scales better than linearly with dimension; (3) efficient decoding: required resources to decode scale logarithmically with encoding range, and (4) robustness to noise. Although a number of previously proposed models achieve some of these properties (see Table 1), RHC is the first, to our knowledge, to achieve all four of these desiderata, as we shall now show.

2 Results

We first define the key concepts on which the RHC framework is based (section 2.1) and then describe the framework fully in section 2.2. We then demonstrate its favorable encoding, decoding, and robustness properties (section 2.3), as well as how it can be extended to multiple dimensions

(section 2.4) and subinteger encodings (section 2.5). Of particular note, we construct a 2D hexagonal residue encoding system, analogous to grid cell coordinates, that provides higher spatial resolution than square lattices. Finally, we describe how the framework can be applied to problems in visual scene analysis and combinatorial optimization (section 3).

2.1 Preliminary Definitions.

Definition 1. *A residue number system (Garner, 1959) encodes an integer $x \in \mathbb{Z}$ by its value modulus $\{m_1, m_2, \ldots, m_K\}$, where the m_k are the* moduli *of the system. For example, relative to moduli $\{3, 5, 7\}$, $x = 20$ would be encoded by the residue $[2, 0, 6]$—that is, $[20 \bmod 3,\ 20 \bmod 5,\ 20 \bmod 7]$. The Chinese remainder theorem states that if the moduli are pairwise co-prime, then for any x such that $0 \leq x < M := \prod_k m_k$, the integer is uniquely encoded by its residue (Goldreich et al., 1999). From here on, we will assume that the pairwise co-prime condition is fulfilled.*

Definition 2. *Fractional power encoding (FPE; Plate, 1994) defines a randomized mapping from an integer x to a high-dimensional vector $\mathbf{z}(x)$. Let D be the dimension of the vector, with D typically in the range $10^2 \leq D \leq 10^4$. The encoding scheme has two steps. First, draw a random base vector, $\mathbf{z}$, defined as follows,*

$$\mathbf{z} = [\exp(i\phi_1), \exp(i\phi_2), \ldots, \exp(i\phi_D)], \tag{2.1}$$

where each element $e^{i\phi_j}$ is a complex number with unit magnitude (a phasor*), and each ϕ_j is a random sample from a specified probability distribution. Second, define a function from x to $\mathbb{C}^D$ via component-wise exponentiation of the base vector,*

$$\mathbf{z}(x) = \mathbf{z}^x = [\exp(i\phi_1 x), \exp(i\phi_2 x), \ldots, \exp(i\phi_D x)]. \tag{2.2}$$

Definition 3. *A kernel, $\mathcal{K}(x_1, x_2)$, is a function $\chi \times \chi \to \mathbb{R}$ that measures the similarity between two objects in a set χ (e.g., vectors in $\mathbb{R}^n$). Notably, FPE implements kernel approximation (Plate, 2003), which is widely used in machine learning (Rahimi & Recht, 2007). More specifically, we can induce a kernel based on the inner products of FPEs,*

$$\mathcal{K}(x_1, x_2) = \frac{1}{D}\, \Re\{\mathbf{z}(x_1)^T\, \overline{\mathbf{z}(x_2)}\}, \tag{2.3}$$

where $\overline{\mathbf{z}(x_2)}$ is the complex conjugate of $\mathbf{z}(x_2)$. This defines a translation-invariant kernel $\mathcal{K}(\Delta x)$ (where $\Delta x = x_1 - x_2$), which converges to a particular $\mathcal{K}^*(\Delta x)$ as $D \to \infty$, where the shape of the kernel is determined by the probability distribution used to draw $\mathbf{z}$ (Frady et al., 2021, 2022).

2.2 Residue Hyperdimensional Computing. We now introduce how FPE can implement a residue number system. As a first step, we explain how FPE can implement congruence (representing a remainder, modulo m).

Definition 4. *Fractional power encoding, modulo m: Let m be a positive integer, and let* $\mathbf{z}_m \in \mathbb{C}^D$ *denote a random vector sampled according to definition 2, in which the support for the probability distribution of the random phases is the mth roots of unity. Then congruent values (i.e., equivalent modulo m) are mapped to the same vector,*

$$\begin{aligned}\mathbf{z}_m(x+m) &= [\exp(i\phi_1(x+m)), \ldots, \exp(i\phi_D(x+m))] \\ &= [\exp(i\phi_1 x)\cdot\exp(i\phi_1 m), \ldots, \exp(i\phi_D x)\cdot\exp(i\phi_D m)] \\ &= [\exp(i\phi_1(x)), \ldots, \exp(i\phi_D(x))] \\ &= \mathbf{z}_m(x),\end{aligned}$$

because $\exp(i\phi_j m) = \exp(2\pi \cdot i \cdot k)$ *for some integer k and* $\exp(2\pi \cdot i \cdot k) = 1$ *for any integer k. Put another way,* $\mathbf{z}_m(x)$ *is a representation (in the abstract algebraic sense) of the additive group of integers modulo m. We emphasize that this result does not depend on the dimension D and that D and m are chosen independent of the other.*

In particular, we focus on the case when all of the mth roots of unity are equally probable. Then the kernel induced by $\mathbf{z}_m(\Delta x)$ is 1 if $\Delta x = 0 \pmod m$, and ≈ 0 otherwise, as shown in Figure 1a. This is highly useful, because it implies that distinct integers behave as quasi-orthogonal vectors, just like symbols in hyperdimensional computing. Unlike symbols, however, we can perform algebraic manipulations transforming one integer into another.

Definition 5. *Residue hyperdimensional computing: Let* $\mathbf{z}_{m_1}, \mathbf{z}_{m_2}, \ldots, \mathbf{z}_{m_K}$ *denote FPE vectors with moduli* $m_1, m_2, \ldots, m_K$*, respectively. Let* $\odot$ *denote component-wise multiplication (also known as Hadamard product). Then we encode an integer x by combining our modulo representations via the Hadamard product:*

$$\mathbf{z}(x) = \bigodot_{k=1}^{K} \mathbf{z}_{m_k}(x). \tag{2.4}$$

The above encoding represents the remainder of x, because each $\mathbf{z}_{m_k}$ represents its value modulo m_k. The code is fully distributed, as every element of the vector $\mathbf{z}$ contains information about each encoding vector $\mathbf{z}_{m_k}(x)$. By contrast, typical implementations of residue number systems compartmentalize information about individual moduli (Omondi & Premkumar, 2007).

The kernel induced by $\mathbf{z}(\Delta x)$ is 1 if $\Delta x = 0 \pmod{\prod_k m_k = M}$ and ≈ 0 for other integer intervals Δx (see Figure 1b). This means that the kernel maps

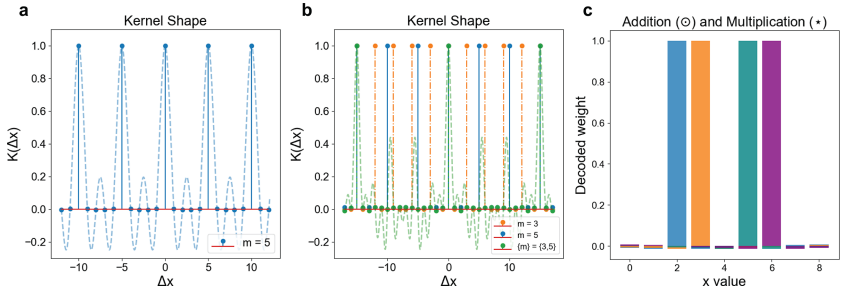

Figure 1: Residue hyperdimensional computing defines a kernel separating different remainder values, and it enables algebraic operations. (a) For fractional power encoding, modulo $m = 5$, inner products between vectors reflect the similarity of points with the same remainder value and are quasi-orthogonal elsewhere. The light blue curve shows the kernel shape when Δx is a real-valued scalar; integers occur approximately at zero crossings. A further derivation is provided in appendix B. (b) The kernel induced by an RHC vector (green) is the product kernel of the moduli used to form it (orange and blue stem plots show, for comparison, the kernel for $m = 3$ and $m = 5$ sampled at integer values of Δx). (c) Demonstration of addition and multiplication. Blue and orange show encodings of 2 and 3, respectively. Teal shows the decoded value of $2 + 3$ (i.e., 5); purple shows decoded value of 2×3 (i.e., 6).

different remainders of our residue number system to quasi-orthogonal directions in high-dimensional vector space and enables computing in superposition over these variables. Examples of possible applications enabled by such a scheme are presented in section 3.

The hallmark of a residue number system is carry-free arithmetic; that is, addition, subtraction, and multiplication can be performed component-wise on the remainders. This enables residue number systems to be highly parallel, avoiding carryover operations required in binary number systems. RHC implements arithmetic with component-wise operations, thus inheriting the premier computational property of residue number systems.

Addition is defined as the Hadamard product between vectors, that is, $\mathbf{z}(x_1 + x_2) = \mathbf{z}(x_1) \odot \mathbf{z}(x_2)$ (see section 5.1.1). This follows from the fact that component-wise multiplication of phasors corresponds to phase addition and the fact that component-wise multiplication is commutative. Subtraction is defined by addition of the additive inverse.

Next, we define a second binding operation that implements multiplication, denoted as $\star$: $\mathbf{z}(x_1 \cdot x_2) = \mathbf{z}(x_1) \star \mathbf{z}(x_2)$. Just as variable addition is implemented by element-wise multiplication, variable multiplication is implemented by another element-wise operation, this one involving exponentiation (see section 5.1.2). We show how this definition for integer multiplication can be generalized to multiplication for vector encodings $\mathbf{z}(x_1)$

and $\mathbf{z}(x_2)$ without invoking the costly step of first decoding the integers back from the vectors.

Here, it is crucial that our encoding function is restricted to integers as its domain because multiplication is commutative and integer powers commute (that is, $(c^{x_1})^{x_2} = c^{x_1 x_2} = (c^{x_2})^{x_1}$ for $c \in \mathbb{C}$ and integers x_1, x_2). In general, this property does not hold for noninteger values. For example, $(-1^2)^{0.5} = 1^{0.5} = \pm 1$ (two possible square roots), but $(-1^{0.5})^2 = (\pm i)^2 = -1$ (only one possible solution). This example illustrates that exponentiation by even a rational power can lead to multiple solutions, which can lead to noncommutativity.

Division is not well defined for residue number systems, because integers are not closed under division. Still, when a modulus m_k is prime, multiplications by nonzero integers are invertible, because each nonzero integer has a unique multiplicative inverse with respect to m_k.

The existence of two distinct binding operators for addition and multiplication is a new contribution to hyperdimensional computing. Previous formulations supported only addition or only multiplication via addition after taking logarithmic transformations (Kleyko, Bybee, et al., 2022). Consequently, we can now formulate a fully distributed residue number system that inherits the benefits of computing with high-dimensional vectors.

Comparing integer values is more difficult for a residue number system than for binary systems, in which one can directly compare values of higher-order bits. Even so, there are multiple good algorithms for performing this comparison (Omondi & Premkumar, 2007), which we can implement with the addition and multiplication operations of RHC (see appendix C). Value comparisons of integers are difficult for high-dimensional distributed representations in general, and we show that our algorithm implements comparisons more efficiently than baseline methods.

2.3 A Resonator Network Enables Efficient Decoding of Residue Numbers. Given the vector representation of a residue number, $\mathbf{z}(x)$, how do we decode it to recover its value, x? One method for decoding commonly used in hyperdimensional computing is codebook decoding (Kleyko, Bybee, et al., 2023), which involves taking the inner product of $\mathbf{z}(x)$ with a set of M codebook vectors with known values of x. However, this procedure requires $\mathcal{O}(M * D)$ storage and M inner product evaluations.

Fortunately, we can improve the situation by utilizing the fact that residue numbers break the dynamic range of a variable into a set of variables, each with lower overall dynamic range. For example, a variable with a dynamic range of 105, when represented modulo $[3, 5, 7]$, requires a set of codebook vectors of only size $3 + 5 + 7 = 15$. This in turn reduces both the memory and computation resources for decoding by a factor of $105/15 = 7$. To make this work, though, requires that we invert equation 2.4—that is, we must factorize $\mathbf{z}(x)$ into the set of constituent vectors

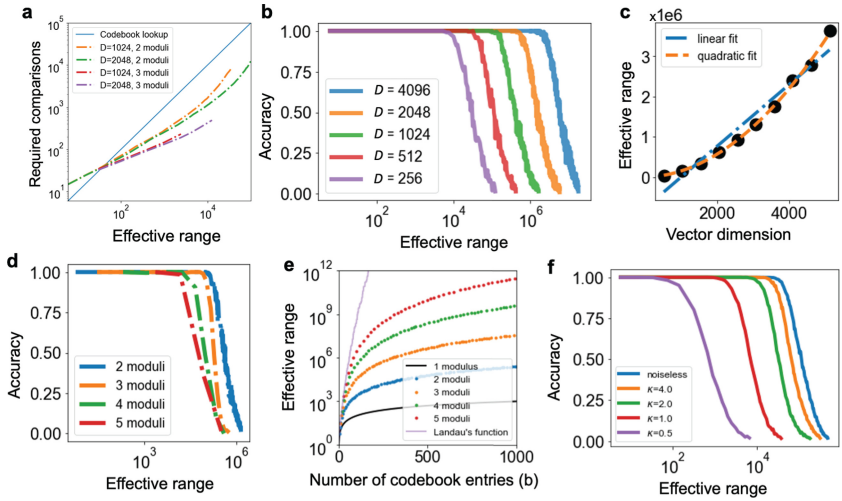

Figure 2: The resonator network can efficiently recover moduli of different numbers. (a) The resonator network outperforms codebook decoding by over an order of magnitude when the effective range is within resonator network capacity. (b) Demonstration of resonator network capacity. For a fixed vector dimension, accuracy remains high up to a given range, before gradually falling off. (c) Scaling of resonator network capacity, C, as a function of dimension, D, is well described by a quadratic fit (orange dashed line; see the linear fit in the blue dash-dotted line). Quadratic fit coefficients are $(1.3 \times 10^{-1})D^2 + (2.4 \times 10^{1})D - 3.7 \times 10^{3}$, and linear fit coefficients are $(7.6 \times 10^{2})D - 7.6 \times 10^{5}$. (d) Resonator network performance is slightly worse for a higher number of moduli, K, but (e) an advantage of more moduli is a much higher effective range given encoding resources. (f) The capacity of the resonator network remains high even in the presence of large amounts of phase noise.

$\{\mathbf{z}_{m_1}(x), \mathbf{z}_{m_2}(x), \ldots, \mathbf{z}_{m_K}(x)\}$ representing x modulo m_k, from which x can be easily recovered. For this, we can use a resonator network (Frady et al., 2020; Kent et al., 2020), a recently discovered method for efficiently factorizing vectors in hyperdimensional computing. Figure 2a shows that for a range of M values, the resonator network can recover vectors over an order of magnitude faster than standard codebook decoding. Two parameters that contribute to this are the vector dimension (D) and number of moduli (K).

To evaluate the dependence of resonator decoding on vector dimension, we fix the number of moduli ($K = 2$) and calculate the empirical accuracy (see Figure 2b) of the resonator network on the effective range M. We find that for a fixed D, the accuracy remains almost perfect up to a certain range of M, after which accuracy rapidly decays. To evaluate scaling with D,

we define the capacity C of a dimension to be the largest M up to which empirical accuracy is at least 95%. We find that the scaling of $C(D)$ is well fit with a quadratic polynomial (see Figure 2c), consistent with previous scaling laws studied for a resonator network with two states per component (Kent et al., 2020). Further tests with higher dimensions would help confirm quadratic scaling, but even the linear scaling has high slope (and note that $C(4096) > 2 \times 10^6$).

To evaluate dependence on the number of moduli, we fix $D = 1024$ and vary K. We find that resonator network capacity decreases as K increases (see Figure 2d), also consistent with prior work (Kent et al., 2020). Still, we emphasize that resonator networks with higher K have two advantages: decreased computation per decoding (see Figure 2a), and decreased memory requirements. The resonator network requires only $\sum_k m_k = b$ codebook vectors, rather than $\prod_k m_k = M$. This means that increasing K can increase the effective range by several orders of magnitude given a fixed codebook budget (see Figure 2e). Remarkably, the maximal M for a given b is given by Landau's function $g(b)$, which scales as $g(b) = e^{(1+o(1))\sqrt{b \ln b}}$ (Landau, 1903). This implies an exponential scaling between storage required and effective range if we proffer sufficient K to achieve it.

Finally, we evaluate the robustness of resonator network decoding to noise. We draw phase noise from a von Mises distribution with mean 0 and concentration κ; higher κ indicates less noise. In Figure 2f, we observe that performance degrades gradually as a function of noise, yet capacity remains remarkably high even at high noise levels.

2.4 Generalization to Multiple Dimensions.

2.4.1 Cartesian Representations of $\mathbf{Z}^n$. Next, we generalize RHC from scalars to multidimensional variables, showing that the core operations and principles still apply. Let $\mathbf{x} \in \mathbb{Z}^n$ be a low-dimensional vector. Let $x_1, x_2, \ldots, x_n$ denote the components of $\mathbf{x}$. To encode $\mathbf{x}$ with a single high-dimensional vector $\mathbf{z} \in \mathbb{V}^D$ ($D >> n$), we form the encoding $\mathbf{z}(x)$ by taking the Hadamard product of the distributed representations of individual components,

$$\mathbf{z}(\mathbf{x}) = \mathbf{z}_1(x_1) \odot \mathbf{z}_2(x_2) \odot \ldots \odot \mathbf{z}_n(x_n), \tag{2.5}$$

where each $\mathbf{z}_i$ is a random vector as generated for the residue representation of a single number in equation 2.4. (Each $\mathbf{z}_i$ is created from binding multiple vectors for each of the moduli, distinct for each dimension i.) Since the binding operation is commutative, we can rearrange our terms to show that the following useful properties hold:

1. The Hadamard product ($\odot$) performs vector addition: $\mathbf{z}(\mathbf{x}) \odot \mathbf{z}(\mathbf{x}') = \mathbf{z}(\mathbf{x} + \mathbf{x}')$.

2. The multiplicative binding operation ($\star$) performs component-wise multiplication of $\mathbf{x}$ and $\mathbf{x}'$: $\mathbf{z}(\mathbf{x}) \star \mathbf{z}(\mathbf{x}') = \mathbf{z}(\mathbf{x} \odot \mathbf{x}')$.
3. The kernel induced by $\mathbf{z}$ is the product of the kernels of the individual components: $K_z(\mathbf{\Delta x}) = \prod_{i=1}^{n} K_{z_i}(\Delta x_i)$, where $\mathbf{\Delta x} = \mathbf{x} - \mathbf{x}'$.

While properties 1 and 3 are general properties of FPE, property 2 is once again unique to RHC. In addition, the savings in decoding resources and computation required (see Figure 2) also scale in higher dimensions, as used in section 3.

2.4.2 Triangular Coordinate Systems. When working in a multi-dimensional space, there are multiple alternatives to a Cartesian coordinate system. For example, grid cells in medial entorhinal cortex encode spatial location with a triangular coordinate system; research in theoretical neuroscience suggests that this is because such a tiling of space has the highest resolution (Fisher information) in 2D space (Mathis et al., 2015). Here we show that residue hyperdimensional computing can also implement triangular coordinate systems and that such a hexagonal lattice retains coding advantages over square lattices. Of particular note, we formulate a simple and efficient RHC encoding of 2D space into nonnegative triangular coordinates.

To encode a two-dimensional position into a three-coordinate frame requires two steps. First, we project the 2D vector $\mathbf{x}$ into a 3D vector $\mathbf{y}$ with unit vectors whose angles each differ by $\frac{2\pi}{3}$ (see section 5.3). This coordinate representation is known as the "Mercedes-Benz" frame in $\mathbb{R}^2$; it is well studied in signal processing (Malozemov & Pevnyi, 2009) and has attracted recent interest in hyperdimensional computing (Dumont & Eliasmith, 2020; Frady et al., 2021; Komer, 2020). For our purposes, this step is necessary but not sufficient, because simple projection results in a redundant representation of space (i.e., the encoding into $\mathbf{y}$ would contain a 1D null-space). This redundancy can be removed by enforcing nonnegativity in $\mathbf{y}$. Therefore, the second step encodes $\mathbf{y}$ as above (see section 2.4.1), but with the additional constraint that $\mathbf{z}([1, 1, 1]) = \mathbf{z}([0, 0, 0])$, ensuring that every state with negative coordinates has an equivalent representation to one with nonnegative coordinates. It turns out that this constraint is easily enforced by ensuring that the phases encoding the three coordinates sum to zero (see section 5.3). This constraint also reflects the fact that equal movement in every direction cancels out, and thus it enforces that different paths to the same 2D position result in the same high-dimensional encoding. The kernels induced by vectors of individual moduli (see Figures 3b and 3d) and by the residue vector (see Figure 3f) exhibit the six-fold symmetry characteristic of hexagonal lattices and grid cells (Hafting et al., 2005).

We can therefore represent the triangular coordinate system with a Voronoi tesselation (see Figure 3e) in which different regions of space are mapped to their nearest integer-valued 3D coordinate. A hexagonal system

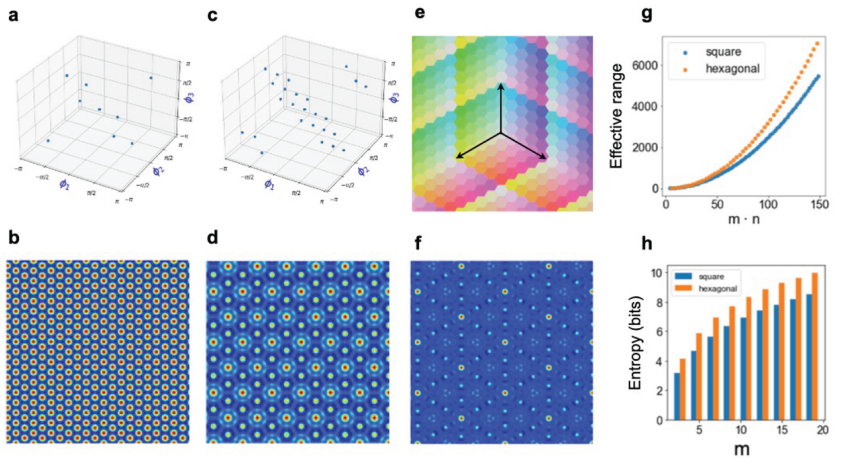

Figure 3: Definition of a residue number system in nonnegative hexagonal coordinates. (a, c) Discrete phase distributions chosen for a 3D coordinate system of a 2D space with moduli 3 and 5, respectively. In addition to requiring that phases are drawn from the mth roots of unity, we enforce that $\phi_1 + \phi_2 + \phi_3 = 0 \pmod{2\pi}$. (b, d) The respective kernels generated by these phase distributions. (e) An example of the Voronoi tesselation of different states composed of a hexagonal coordinate system with modulus 5. Each color corresponds to a different state representation in the vector space of the three integer coefficients of the Mercedes-Benz frame (depicted by black arrows). (f) The kernel induced by a hexagonal residue HD vector (period of $3 \cdot 5 = 15$). (g) Compared to square encodings of space, hexagonal encodings approximately triple the effective range of encodable states with only a 50% increase in required storage space. (h) The Shannon entropy of the hexagonal code is higher than that of the square code with the same modulus, reflecting the benefit of hexagonal packing.

with modulus m has $3m^2 - 3m + 1$ distinct states and requires $3m$ codebook vectors, whereas a square lattice has m^2 distinct states and requires $2m$ codebook vectors (see Figure 3g). Thus, the hexagonal system achieves better spatial resolution (it is a higher-entropy code with regard to space) than a square lattice (see Figure 3h) for the same number of resources.

2.5 Extensions to Subinteger Decoding Resolution. In previous sections, we worked exclusively with integer states and residue number systems implementing them. Intriguingly, however, we can extend our definition of FPE to rational numbers (see section 5.4.1), and the resonator network converges to FPE encodings of nonintegers, even when codebooks contain only encodings of integers (see Figures 4a and 4b). Strictly speaking, such extensions beyond integers are not residue number systems, because

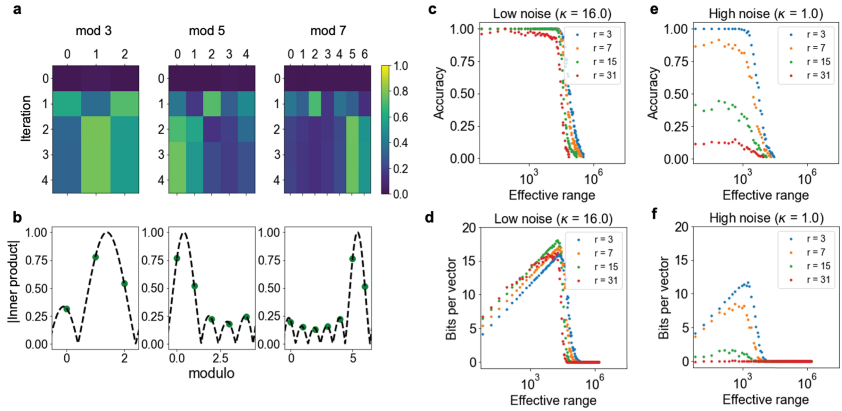

Figure 4: The resonator network enables retrieval of fractional (subinteger) values. (a) Example of the resonator network converging for $\mathbf{z}(x), x = 40.4$. (b) Inner product values decoded by the resonator network are predicted by fractional offsets from a Dirac comb convolved with a sinc function. (c) Subinteger encoding accuracy under a low noise regime for an RHC vector (r denotes number of subinteger partitions). (d) Bits per vector for different fractional decodings. Panels e and f are the same as panels c and d, respectively, but under higher noise conditions.

in these cases, multiplication is not well defined (see also the discussion following definition 5). However, the definition of additive binding can be extended to rational numbers (see section 5), and such extensions toward subinteger resolution have been considered in theoretical analyses of grid cells (e.g., in Fiete et al., 2008, and Srinivasan & Fiete, 2011). We show that the resonator network dynamics achieve this subinteger resolution.

An efficient procedure for decoding with subinteger precision is suggested by Figure 4b. The inner products between codebook states and the final resonator network state are well described by evaluations of a Dirac comb convolved with a sinc function (see appendix B). For integer encodings, this function would evaluate to 1 for the nearest integer state and 0 for all other integer encodings. For noninteger rational numbers, however, the nearest integer state(s) still have the highest inner product, and the deviation of this inner product from 1 indicates the offset from an integer value. Similarly, features near the peak have a nonzero yet predictably small inner product. Therefore, we can find the subinteger offset that best matches the resonator network state in order to decode the subinteger value (see section 5.4.2).

Phase noise is the limiting factor in decoding subinteger states. To quantify this more rigorously, we evaluate a resonator network with varying effective ranges M under different noise regimes ($\kappa = 16$ and 1, respectively).

We then split each unit interval into r partitions, so that there are $M \cdot r$ distinct numbers represented. Figures 4c and 4e show the accuracy of decoding for a different number of partitions with $\kappa = 16$ and $\kappa = 1$, respectively. To account for accuracy and the number of different states distinguished, we also report the bits per vector metric (Frady et al., 2018) in Figures 4d and 4f. This metric validates that with lower noise, we can more reliably decode a higher number of states.

3 Applications

3.1 Efficient Disentangling of Object Shape and Pose from Images. Here, we study the disentangling problem in vision—that is, the task of recovering the underlying components of a scene given only the image pixel values. Such problems abound in visual perception; examples include inferring structure-from-motion or separating spectral reflectance properties from shading due to 3D shape and lighting. How brains disentangle these factors of variation in general is unknown, and it is computationally challenging due to the combinatorial explosion in how factors combine to create any given scene (Olshausen, 2014).

Here, we demonstrate how RHC can efficiently tackle the combinatorial complexity inherent in visual perception by considering a simple case of images containing three factors of variation: object shape, horizontal position, and vertical position. Let O, H, V be finite sets listing the possible vectors for each factor. The goal is to infer the $O_a \in O, H_b \in H, V_c \in V$ provided an image, I. In this setup, the search space is $|O| \cdot |H| \cdot |V|$, and in our example (see Figure 5), $|O| = 10$ and $|H| = |V| = 105$, giving a search space of $\sim 10^5$.

We solve this problem in two stages. First, we form a latent, feature-based representation of the image via convolutional sparse coding (see section 5.5). This step mirrors the neural representation in primary visual cortex, which is hypothesized to describe image content in terms of a small number of active image features (Olshausen & Field, 1996). We observe that this step is useful as it helps to decorrelate image patterns, thus achieving higher accuracy and faster convergence for the resonator network (Kymn et al., 2024).

Second, we encode the latent feature representation into a high-dimensional vector that can be subsequently factorized into its components (O_a, H_b, V_c) via a resonator network. This is accomplished, following Renner et al. (2024), by superimposing the residue number encodings of the position of each image feature into a single scene vector, $\mathbf{s}$ (see section 5.5, equation 5.5). The resulting vector, $\mathbf{s}$, can be expressed equivalently as a product of vectors representing object shape and position, and thus the problem of disentangling these factors essentially amounts to a vector factorization problem.

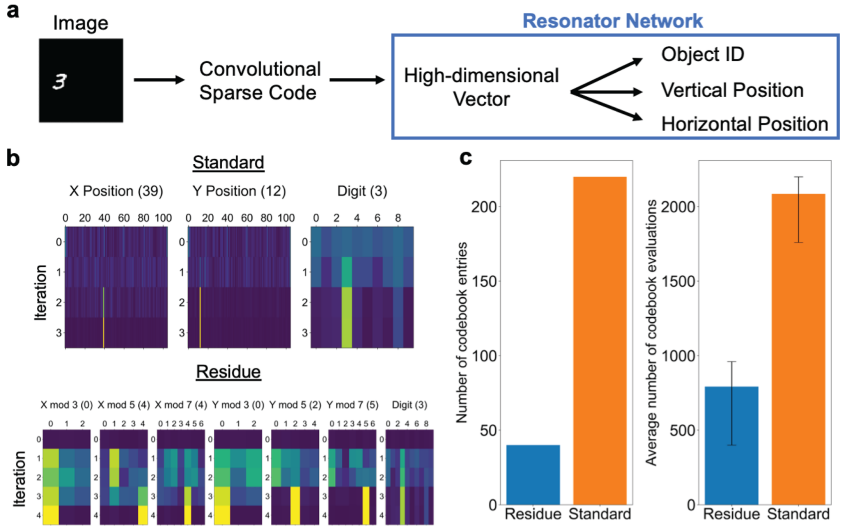

Figure 5: Residue hyperdimensional computing enables efficient disentangling of images. (a) Processing pipeline: an image is first represented in terms of its local shape features via convolutional sparse coding, then converted to a high-dimensional vector $\mathbf{z}$ by superimposing the residue number encodings of the positions of each feature in the image, which is finally factorized into object identity and position via a resonator network. (b) Simulations of the resonator network on visual scenes without the residue number encoding (Standard) and with the residue number encoding (Residue). (c) With a residue number system, the resonator network requires less memory overhead (40 versus 220 codebook vectors) and less total computation to converge to the correct solution (792.4 versus 2085.6 average codebook evaluations). Error bars show the 25th and 75th percentiles of the number of evaluations.

The standard way to factorize the scene vector (e.g., as in Renner et al., 2024) would be to use three codebooks corresponding to shape, horizontal position and vertical position, for a total of $10 + 105 * 2 = 220$ codebook vectors. By contrast, a residue number system with moduli $\{3, 5, 7\}$ uses seven factors but only $10 + (3 + 5 + 7) * 2 = 40$ vectors. Example runs of both problem setups are shown in Figure 5b.

Figure 5c demonstrates the two main advantages of the residue resonator compared to the standard resonator baseline: a reduction in both memory requirements (as just described) and the required number of iterations. Whereas the standard resonator takes over approximately 2,000 codebook evaluations on average in our simulations, the residue resonator averages only approximately 800 codebook evaluations. (Both dramatically improve over the brute force search, which requires 110,250 codebook evaluations.)

The key lesson is that a residue number encoding of position leads to a large multiplicative decrease in the number of computations required to factorize the contents of a scene.

3.2 Generating Exact Solutions to the Subset Sum Problem. Here we apply RHC to the subset sum problem. Formally, the problem asks if a multiset of integers, S, contains a subset S^* that sums to a target integer T. A further demand is to return S^* if it exists. When all integers are positive, the subset-sum problem is NP-complete (Kleinberg & Tardos, 2006). It is a useful case to consider because there are well-known polynomial-time reductions from other NP-complete problems (e.g., 3-SAT) to subset sum (Karp, 1972).

To find solutions to the subset sum problem, we encode T as a vector, $\mathbf{z}(T)$, and use $|S|$ factors. Each factor, F_k, has a codebook of two items: an identity vector $\mathbf{z}(0)$ and $\mathbf{z}(S_k)$—reflecting the binary decision to include that item in the sum or not. Figure 6a demonstrates the resonator network successfully finding the solution when $T = 21$ and $|S| = 6$.

In order to use a residue number system, we need to choose M so that $M > \sum_{s \in S} s$. This means that we need $\lceil \log M \rceil$ bits per vector component. This is an improvement from previous work using resonator networks to solve factorization problems (Kleyko, Bybee, et al., 2022), which requires floating-point precision to perform semi-prime factorization.

To understand the scaling capacity of the residue number system, we evaluate the performance of the resonator network as the set size increases. We observe that the resonator network finds exact solutions to the subset sum problem for large sets and that performance improves with higher vector dimension (see Figure 6b). Figure 6c illustrates that the success probability after up to 10 trials matches what is expected from 10 independent runs of the 1-trial accuracy. This finding suggests that the resonator network constitutes a "Las Vegas" algorithm (Babai, 1979), in which each run has a success probability p, p is independent across runs, and so the algorithm requires $1/p$ iterations on average. Accuracy also depends on the integer range searched over, even for the same set size (see Figure 6d), perhaps because larger integer ranges reduce the probability of multiple subsets matching the target.

Finally, we compare our subset sum algorithm to brute force search and an exponential-time algorithm that solves the decision problem. The average number of iterations required by the resonator network to find a solution is drastically less than the exponentially increasing cost of a brute force search (see Figure 6e) and also improves with higher dimension. We find that on a CPU, the resonator network has faster clock time than the exponential time algorithm for $|S| > 28$ (see Figure 6f and section 5.6); most of the computing time is spent on generating the vector representations rather than the resonator network dynamics itself. More significant, whereas the baseline algorithm required $\mathcal{O}(2^{|S|})$ memory to keep candidate subsets in

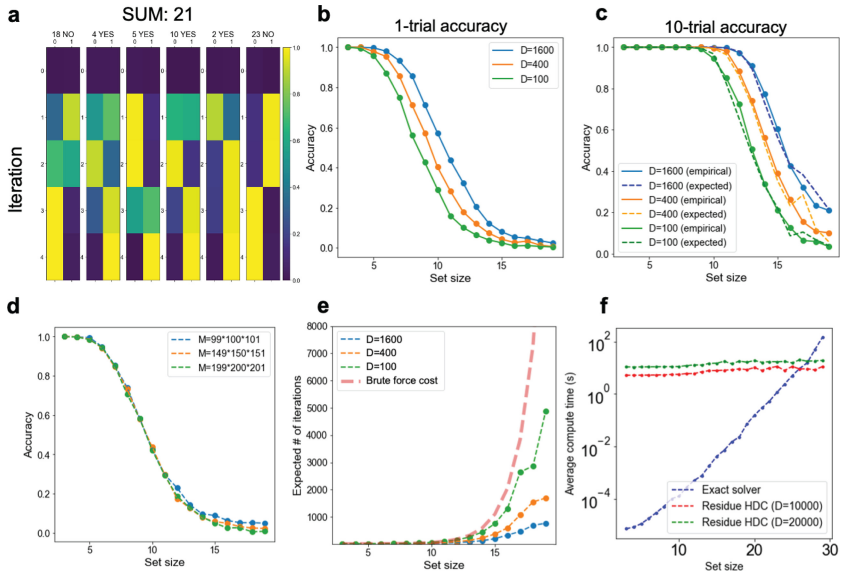

Figure 6: The resonator network, with residue hyperdimensional computing, enables successful searches for solutions to the subset sum problem. (a) Demonstration of the resonator network on the subset sum problem, with $S = \{18, 4, 5, 10, 2, 23\}$. The resonator network converges to the correct solution after a few iterations. (b) Performance of the resonator network on randomly selected subset sum problems with a fixed set size. (c) The probability of success scales as expected with independent trials on each start. (d) Success of the resonator network on the subset sum problem depends on the range of items indexed in the subset sum problem (larger ranges are harder to factorize). (e) Performance of the resonator network in terms of the expected number of iterations compared to chance. The expected number of iterations for the resonator network scales favorably compared to brute force search and improves with higher encoding dimension. (f) Comparison of average compute time of the resonator network versus an exact algorithm. The initialization cost of setting up the resonator network is higher; even so, for large set sizes, the resonator network is faster.

memory, the resonator network only requires $\mathcal{O}(D \cdot |S|)$ memory, since it never needs to explicitly represent every subset. We emphasize that the CPU implementation of the resonator network is primarily intended as a proof of concept and that further performance gains would likely result from implementing the resonator network on emerging computing platforms, as in Langenegger et al. (2023) and Renner et al. (2024).

4 Discussion

Our study provides a definition of residue number systems within the framework of hyperdimensional computing. The new framework inherits the benefits of both systems: the carry-free arithmetic and Chinese remainder theoretic guarantees from residue number systems, along with the robustness and computing-in-superposition properties of hyperdimensional computing (Kleyko, Davies, et al., 2022). The framework provides a favorable way to encode, transform, and decode variables that is robust to noise. Taken together, these properties make residue hyperdimensional computing an appealing framework for efficiently solving difficult computational problems, especially those involving combinatorial optimization. It also has implications for modeling population codes in the brain, in particular grid cells.

Prior work in computational neuroscience (Fiete et al., 2008; Srinivasan & Fiete, 2011) has emphasized that residue number systems endow grid cells with useful computational properties, including high spatial resolution, modular updates, and error correction. We demonstrate that residue hyperdimensional computing successfully achieves each of these coding properties (see sections 2.2 and 2.3) in a working neural implementation. Reciprocally, our algebraic framework makes two contributions to computational neuroscience. First, we show how to extend a residue number system to a self-consistent, nonnegative hexagonal coordinate system. Second, we provide a new algorithm for collectively coupling spatial position to grid cell modules via the resonator network. The core prediction our framework makes for systems neuroscience is that each grid cell module corresponds to a factor estimate in the resonator network. More specifically, each module implements a toroidal attractor network, and multiplicative couplings with hippocampus and other grid cell modules enable error correction. This prediction is consistent with both recent experimental analysis supporting the existence of continuous attractor networks in single grid cell modules (Gardner et al., 2022) and with recent theories of joint attractor dynamics in hippocampus and medial entorhinal cortex (Agmon & Burak, 2020). Note, however, that while the 2D kernel of our model reflects the periodic, hexagonal structure of grid cell response fields, the individual elements of the vector representations are phasors that have "band-like" receptive fields in 2D (Krupic et al., 2012), and so reconciling this mismatch will be an important goal of future research.

Here, we used vectors of complex numbers and algebraic operations to formulate the neural representations. These operations are not easily realized in perceptron-like neurons operating over weighted sums of their inputs. However, there are various alternative mechanistic theories of how such operations could be represented through more biologically realistic spiking neurons—for instance, complex phasors can be represented

through a spike-timing code (Frady & Sommer, 2019). It is a question, then, how the mechanisms we have proposed are implemented by neural populations. Mechanisms such as dendritic nonlinearities could be a potential implementation of the required multiplicative operations in our model.

Our work also draws heavily from prior results from vector symbolic architectures and hyperdimensional computing. Originally developed as a framework in cognitive science for structured symbolic reasoning with distributed representations, our work helps extend this approach to numeric, nonsymbolic, tasks. The heart of the number encoding, fractional power encoding, was introduced by Tony Plate (1992) and further developed in Plate (1995). Though vector operations with modular arithmetic have been previously used (Frady et al., 2021, 2022; Komer, 2020; Snaider & Franklin, 2014; Yu et al., 2022), this study is the first instance leveraging a residue number system with more than the trivial first modulus. The results shown in Figure 2 demonstrate that having multiple moduli is necessary to achieve efficient scaling laws.

The framework also has implications for how neural populations can solve difficult optimization problems, such as disentangling visual scenes. Recent work has emphasized the promise of hyperdimensional computing as an abstraction for neuromorphic computing (Frady & Sommer, 2019; Kleyko, Davies, et al., 2022; Renner et al., 2024). Residue hyperdimensional computing substantially reduces the storage and average number of operations needed for solving decoding problems and combinatorial optimization, contributing a simple yet powerful improvement. In addition, the phasor representations suggested by our framework directly map onto Q-state phasor networks (Noest, 1988), suggesting promising implementations in spiking neural networks (Bybee & Sommer, 2022) and strategies for solving combinatorial optimization problems (Wang & Roychowdhury, 2019).

The performance of our framework on the subset sum problem suggests a new route for solving optimization problems with distributed representations and unconventional hardware. The subset sum problem is a particularly good fit for our framework because it is easily implemented by the Hadamard product operation on high-dimensional vectors. Since other hard problems, such as 3-SAT, can be efficiently mapped to subset sum, our results potentially point the way to a new class of parallel algorithms for efficiently solving NP-hard problems.

Finally, we hope that our work facilitates new connections between existing applications of residue number systems and hyperdimensional computing. Residue number systems have attracted strong interest in their own right for their useful theoretical properties and efficient realizations of fault-tolerant computer hardware (Mohan, 2016; Omondi & Premkumar, 2007) and error-correcting codes (Goldreich et al., 1999) useful in communication systems. Additionally, residue number systems have useful applications in

fields such as signal processing and fully homomorphic encryption. The rich theoretical foundations of residue number systems could suggest new routes to improve the error-correction capabilities of existing hyperdimensional computing algorithms (Kim, 2018); conversely, residue hyperdimensional computing could enable development of new applications using robust distributed codes and ability to compute-in-superposition.

5 Methods

5.1 Definitions of Algebraic Operations.

5.1.1 Definition of Additive Binding Operation. We implement addition within RHC by using the Hadamard product operation (component-wise multiplication, $\odot$): $\mathbf{z}(x_1 + x_2) = \mathbf{z}(x_1) \odot \mathbf{z}(x_2)$. The Hadamard product correctly implements addition because it is commutative. This can be seen as follows:

$$\begin{aligned}
\mathbf{z}(x_1) \odot \mathbf{z}(x_2) &= \Big(\bigodot_k \mathbf{z}_{m_i}(x_1)\Big) \odot \Big(\bigodot_k \mathbf{z}_{m_i}(x_2)\Big) \\
&= \bigodot_k (\mathbf{z}_{m_i}(x_1) \odot \mathbf{z}_{m_i}(x_2)) \\
&= \bigodot_k \mathbf{z}_{m_i}(x_1 + x_2) \\
&= \mathbf{z}(x_1 + x_2).
\end{aligned}$$

5.1.2 Definition of Multiplicative Binding Operation. To implement a second binding operation ($\star$), such that $\mathbf{z}(x_1 \cdot x_2) = \mathbf{z}(x_1) \star \mathbf{z}(x_2)$, every component of the vector $\mathbf{z}(x_1)$ must be multiplied by x_2. If we had the value of x_2 explicitly, then we could directly implement $\mathbf{z}(x_1 \cdot x_2)$ by component-wise exponentiation of $\mathbf{z}(x_1)$ by x_2. However, decoding incurs additional computational costs, and we show here that multiplications can be computed without this intermediate step.

We require a few simplifying assumptions to define our multiplication operation. First, we assume that we have access to the individual base vectors for each modulus (e.g., $\mathbf{z}_{m_1}(x_1)$). If we do not, then we can use the resonator network to recover them. The key observation is that if x is an integer, then each component of $\mathbf{z}_{m_k}(x)$ is itself a m_kth root of unity. More specifically, it equals $\exp(2\pi i r_j/m_k)$, for some integer $r_j = (m_k/2\pi)\phi_j x \pmod{m_k}$.

Therefore, we define an operation, f, that can multiply two discrete phases when they are both drawn from the m_kth roots of unity:

$$f\left(\exp\left(i\frac{2\pi}{m_k}r\right), \exp\left(i\frac{2\pi}{m_k}s\right)\right) = \exp\left(i\frac{2\pi}{m_k}rs\right).$$

When f is applied to two vectors of the same dimension, the multiplication is applied component-wise. Supposing that $r = (m_k/2\pi)\phi x_1$ and $s = (m_k/2\pi)\phi x_2$, we obtain $rs = (m_k/2\pi)\phi^2 x_1 x_2$, which is off from our desired result by a multiplicative factor of ϕ. This motivates a final step of cancelling out the extra factor.

Because each phase ϕ is drawn from the m_kth roots of unity, it can be written as $2\pi u/m_k$, where $u \in \mathbb{Z} \pmod{m_k}$. When m_k is prime, then any nonzero integer u has a unique modular multiplicative inverse $v \in \mathbb{Z} \pmod{m_k}$, such that $u \times v = 1 \pmod{m_k}$. For example, the modular multiplicative inverse of 3 (mod 5) is 2. We therefore assume that whenever multiplicative binding is used, the multiplicative inverse exists. The simplest way to guarantee this in practice is by choosing every modulus m_k to be prime. This assumption allows us to define an "anti-base" vector, $\mathbf{y}_{m_k}$, whose components are defined by the modular multiplicative inverses of $\mathbf{z}_{m_k}$. That is, if the jth component of $\mathbf{z}_{m_k}$ is $\exp(2\pi iu/m_k)$, then the jth component of $\mathbf{y}_{m_k}$ is $\exp(2\pi iv/m_k)$.

These assumptions motivate the following definition of the multiplicative operation, which we show successfully performs the multiplication of the arguments along with necessary cancellations:

$$\begin{aligned}
\mathbf{z}(x_1) \star \mathbf{z}(x_2) &= f(f(\mathbf{z}_{m_1}(x_1), \mathbf{z}_{m_1}(x_2)), \mathbf{y}_{m_1}) \odot \ldots \odot f(f(\mathbf{z}_{m_K}(x_1), \mathbf{z}_{m_K}(x_2)), \mathbf{y}_{m_K}) \\
&= \bigodot_k [\exp(i\phi_{k,1}x_1\phi_{k,1}x_2\phi_{k,1}^{-1}), \ldots, \exp(i\phi_{k,D}x_1\phi_{k,1}x_2\phi_{k,D}^{-1})] \\
&= \bigodot_k [\exp(i\phi_{k,1}x_1x_2), \ldots, \exp(i\phi_{k,D}x_1x_2)] \\
&= \mathbf{z}(x_1 \times x_2).
\end{aligned}$$

We implement f by taking the angle of the phasor $\exp(2\pi is/m_k)$, multiplying the angle by $m_k/(2\pi)$ and exponentiating $\exp(2\pi ir/m_k)$ by the result. We compute modular multiplicative inverses via the built-in *pow* function in Python. However, we note that both functions can also be implemented by lookup tables, and precomputing all input-output pairs may be optimal when many computations are reused and lookups are inexpensive.

5.2 Decoding Methods. In the context of high-dimensional distributed representations, the decoding problem is to recover a variable x from a distributed representation $\mathbf{z}(x)$. In all of our decoding experiments, x is either an integer or rational number. A survey of decoding methods, applied to symbolic hyperdimensional computing models, can be found in Kleyko, Bybee, et al. (2023).

5.2.1 Codebook Decoding. Codebook decoding estimates x by taking inner products between $\mathbf{z}(x)$ and a precomputed set of reference vectors: $\hat{x} = \arg\max_{x_k} \langle \mathbf{z}(x), \mathbf{z}(x_k) \rangle$.

5.2.2 Resonator Network Details. The resonator network (Frady et al., 2020; Kent et al., 2020) is an algorithm for factoring an input vector, $\mathbf{z}$, into the primitives $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K\}$ that compose it via Hadamard product binding: $\mathbf{z} = \mathbf{z}_1 \odot \mathbf{z}_2 \odot \ldots \odot \mathbf{z}_K$. Each $\mathbf{z}_j$ is specified to come from a set of candidate vectors concatenated in a codebook matrix, $\mathbf{Z}_j \in \mathbb{C}^{D \times m_j}$, where each D-dimensional column vector represents a particular value (mod m_j). In other words, the matrix consists of the column vectors of $[\mathbf{z}_{m_j}(0), \mathbf{z}_{m_j}(1), \ldots, \mathbf{z}_{m_j}(m_j - 1)]$, with each vector defined by definition 5.

The resonator network functions as a dynamical system with the following update equations,

$$\hat{\mathbf{z}}_j(t+1) = g\left(\mathbf{Z}_j \mathbf{Z}_j^{\dagger}\left(\mathbf{z} \odot \prod_{i \neq j} \hat{\mathbf{z}}_i^{*}(t)\right)\right), \tag{5.1}$$

where $\mathbf{Z}_j^{\dagger}$ is the Hermitian conjugate operator applied to $\mathbf{Z}_j$, * denotes the complex conjugate of the vector $\mathbf{z}_i$, and g is a nonlinearity, preserving phase and normalizing the amplitudes of each complex-valued component to 1. The two matrix multiplications, carried out by $\mathbf{Z}_j^{\dagger}$ and $\mathbf{Z}_j$, respectively, can be thought of as a cleanup memory, projecting estimates into the low-dimensional subspace spanned by the column vectors of $\mathbf{Z}_j$. In all experiments, we use an asynchronous update rule in which at each time step, only one factor estimate is updated, and every set of time steps, each vector is updated once. The algorithm runs either until convergence or until a maximum number of iterations has been reached. We consider the resonator network to have converged when the normalized cosine similarity between two successive states exceeds a threshold, α (for all experiments, $\alpha = 0.95$).

5.2.3 Evaluation of Resonator Network Decoding Accuracy, Capacity, and Robustness to Noise. We evaluate resonator network accuracy as a function of vector dimension (D), effective range (M), number of moduli (K), and noise level (dependent on κ). We add noise only in experiments shown in Figure 2f, and $K = 2$ unless stated otherwise. $D = 1024$ in Figure 2d, and $D = 512$ in Figure 2f. To compute data points for curves that are a function of M, we generate a list of ascending primes and select K consecutive primes as moduli. The effective range, M, is the product of these moduli. We continue experiments for a fixed D and increasingly large M until empirical accuracy falls below a given threshold (0.95 for Figures 2a and 2c, and 0.05 otherwise). To report the required number of comparisons for Figure 2a, we normalize the average number of inner product iterations by the accuracy and visualize curves only in the high-accuracy regime (above 95%).

5.3 Triangular Residue Encodings. To project a $2D$ vector $\mathbf{x}$ to a $3D$ triangular coordinate $\mathbf{y}$, we multiply it by a matrix Ψ, which is the Mercedes-Benz frame in $2D$:

$$\Psi = \begin{bmatrix} -1/\sqrt{3} & -1/3 \\ 1/\sqrt{3} & -1/3 \\ 0 & 2/3 \end{bmatrix}.$$

The resulting vector $\mathbf{y} = \Psi\,\mathbf{x}$ is encoded as a high-dimensional vector using the generalization to multiple dimensions specified in equation 2.5. As an additional constraint, we require that $\mathbf{z}([1,1,1]) = \mathbf{z}([0,0,0])$. This condition implements the self-cancellation property (i.e., that moving equally along the three equiangular directions cancels out). It also converts possible negative values arising from the projection step to an equivalent nonnegative coordinate encoding. Somewhat fortuitously, this constraint is naturally enforced in RHC by ensuring that for each component of $\mathbf{z}$, the three phases corresponding to the three directions sum to 0 (mod 2π). This is achieved by constraining the joint probability distribution over triplets of phases so that this requirement is met (see Figures 3a and 3c).

5.4 Decoding with Subinteger Precision.

5.4.1 Extension of Encoding Scheme to Rational Numbers. For a rational number $q \in \mathbb{Q}$, we define $\mathbf{z}_m(q)$ based on fractional power encoding, modulo m, as

$$\mathbf{z}_m(q) = [\exp(i\phi_1 q), \exp(i\phi_2 q), \ldots, \exp(i\phi_D q)]. \tag{5.2}$$

We then form our representation of the remainders (modulo m_k) via the same process described in equation 2.4. If q is an integer, then this procedure matches that of definition 4. But in general, $\mathbf{z}_m(q) \neq (\mathbf{z}_m)^q$. This is significant because while we can still evaluate similarity via inner products and perform addition operations, multiplication operations are no longer well defined.

5.4.2 Subinteger Decoding with the Resonator Network. Subinteger decoding with the resonator network proceeds in three steps. First, we let the resonator update its factor estimates until convergence to a fixed point. We emphasize that subinteger encodings are also fixed points of the resonator, even when the codebooks of the resonator contain only integers (see section 2.5). Second, we find the nearest integer codebook for each moduli, and generate the nearest codebooks for the fractional values within range 1 of that decoded integer (r in total). Third, we use codebook decoding over these vectors encoding fractional values to return our result.

5.4.3 Evaluation of Subinteger Decoding with Noise. We fix $D = 512$, and let $\kappa = \{16.0, 1.0\}$. We run the resonator network until a maximum number of iterations or convergence and evaluate if both the nearest integer and nearest fractional state are correct. If so, we regard the solution as correct, reporting accuracy and bits per vector.

5.4.4 Measuring Bits per Vector. To measure the total amount of information decoded, we account for the accuracy of decoding and the number of states distinguished. The amount of information decoded for a single number (denoted as I_{num}) is calculated using the corresponding accuracy (a) and size (P) of the total search space as

$$I_{\text{num}}(a, P) = a \log_2(Pa) + (1 - a) \log_2 \left(\frac{P}{P-1}(1 - a) \right). \tag{5.3}$$

For a detailed derivation of this equation, refer to section 2.2.3 of Frady et al. (2018). According to this metric, the amount of decoded information is 0 when the accuracy is at chance ($1/P$).

5.5 Visual Scene Factorization Experiments. Convolutional sparse coding learns a dictionary of basis functions, $\{\phi_j(x, y)\}$ and infers a set of sparse latent representations, $\{A_j(x, y)\}$, for each image, $I(x, y)$, by minimizing the following energy function, E,

$$E = \frac{1}{2}||I - \sum_{j=1}^{n} \phi_j * A_j||_2^2 + \lambda \sum_{j=1}^{n} ||A_j||_1, \tag{5.4}$$

where $*$ denotes convolution and λ is a hyperparameter weighting the trade-off between reconstruction error and sparsity. We use the SPORCO implementation of convolutional sparse coding introduced by Wohlberg (2017) to learn the $\{\phi_j(x, y)\}$ for an ensemble of MNIST digits and to infer the sparse representation A for each image I.

A useful feature of convolutional sparse coding is its equivariance to 2D translation; that is, 2D translation in the image domain results in 2D translation of the sparse representations, $\{A_j(x, y)\}$. We can thus convert the set of sparse feature maps $\{A_j(x, y)\}$ to a high-dimensional vector as follows:

$$\mathbf{s} = \sum_{j,x,y} \mathbf{h}(x) \odot \mathbf{v}(y) \odot \mathbf{d}_j \cdot A_j(x, y). \tag{5.5}$$

Here, $\mathbf{h}(x)$ and $\mathbf{v}(y)$ denote the RHC encodings of horizontal (x) and vertical (y) position. $\mathbf{d}_j$ is a random vector generated i.i.d. that represents the identity of each basis function ϕ_j. By expectation, most values of each

$A_j(x, y)$ will be zero because the energy function for sparse coding penalizes nonzero coefficients. Thus, the scene vector, $\mathbf{s}$, can be seen as a sparse superposition of position encodings of features (ϕ_i) contained in the image.

Now we can separately define the vector encoding of each object i to be recognized as

$$\mathbf{O}^{(i)} = \sum_{j,x,y} \mathbf{h}(x) \odot \mathbf{v}(y) \odot \mathbf{d}_j \cdot o_j^{(i)}(x, y), \tag{5.6}$$

where $o^{(i)}(x, y)$ is the sparse representation of the image of object i within a canonical reference frame. If we were to place object i at position (x', y') within an image, the resulting scene vector computed according to equation 5.5 will be given as

$$\mathbf{s} = \mathbf{h}(x') \odot \mathbf{v}(y') \odot \mathbf{O}^{(i)}. \tag{5.7}$$

Therefore, our scene analysis problem amounts to one of factorizing $\mathbf{s}$ into its constituent vectors $\mathbf{h}(x')$, $\mathbf{v}(y')$, and $\mathbf{O}^{(i)}$.

We can factorize $\mathbf{s}$ using a resonator network with three codebooks, $\mathbf{O}$, $\mathbf{H}$, and $\mathbf{V}$. Each element $\mathbf{O}^{(i)} \in \mathbf{O}$ consists of an encoding of each object as above, and $\mathbf{H}$ and $\mathbf{V}$ contain RHC encodings of horizontal and vertical position.

For our object examples, we use 10 images from the MNIST data set. Sparse coding dictionary elements are optimized over a subset of the MNIST data set. After inferring a sparse code for each image, we encode it as a high-dimensional vector ($D = 10{,}000$). We use a residue number system with bases $\{3, 5, 7\}$ for both horizontal and vertical dimension and then either enumerate all 105 codebooks for a single factor (Standard) or use three factors with 3, 5, and 7 codebooks, respectively (Residue). In either case, we ran the resonator network until convergence to a vector matching the scene representation (including reinitialization, if it did not converge after a fixed number of iterations or became stuck in a local minima) and record the average number of iterations multiplied by the average number of codebook evaluations (which is smaller for the residue encoding).

5.6 Subset Sum Experiments. We use a residue number system with 3 moduli, $\{m - 1, m, m + 1\}$, where m is an even positive integer, ensuring that our moduli are co-prime. To generate random subset sum problems, we first define a maximum sum range to be $M/2$. For Figures 6b and 6c, $m = 200, M \approx 200^3$. Then we draw random variables from a uniform distribution (scaled between 0 and half of the maximum sum over the largest set size tested). We then select a random subset of the set (all subsets are equally likely) and compute the sum. This sum forms the input to the resonator network, and we treat its solution as correct if it converged to the same

sum. If the resonator network returns the wrong output, we restart it from a different random initialization, up to a maximum number of trials. We vary both the vector dimension (D) and set size ($|S|$), reporting accuracy after multiple simulations. For Figure 6d, $D = 400$.

To compare the number of evaluations relative to brute force (see Figure 6e), we record the average number of evaluations on each set size. We divide the number of inner product comparisons required for brute force evaluation by the number of comparisons per resonator network iteration. Further, we normalize the number of resonator iterations by the accuracy to ensure a fair comparison. In comparing our algorithm to a solver, we implement an exact subset-sum algorithm as a baseline (Nanda, 2005). We let $m = 1000$, $D = \{10{,}000, 20{,}000\}$, and draw integers uniformly from the range [0,5000].

Appendix: Supplemental Material

A.1 A Brief Survey of Distributed Coding Schemes. In order to process vector representations of numbers, such as in machine learning settings (e.g., Kleyko, Osipov, et al., 2018; Kleyko, Rachkovskij, et al., 2023; Rachkovskij, 2007; Rahimi et al., 2019; Räsänen & Saarinen, 2016; and Schindler & Rahimi, 2021), previous work combined hyperdimensional computing with different kinds of locality-preserving encodings for representing numeric data with vectors. The requirement to be locality preserving is that inner products between vectors encode similarity of the underlying data. Here we briefly review some locality-preserving encoding schemes that have been used in the past (see also Kleyko, Rachkovskij, et al., 2022), assessing their kernel properties.

A.1.1 The Thermometer Code. The thermometer code (Buckman et al., 2018; Kleyko, Kheffache et al., 2020; Penz, 1987; Rachkovskij et al., 2005) is a simple and structured way to form a locality-preserving encoding for a range of discrete levels s, $s \in [0, D]$. The first code $\mathbf{z}(0)$ consists of all -1s. For other levels, the components of $\mathbf{z}(s)$ are determined as

$$z_i(s) = \begin{cases} +1, & i \leq s \\ -1, & \text{otherwise} \end{cases} \tag{A.1}$$

Thus, the last code $\mathbf{z}(D)$ consists of all $+1$s, and, in total, the thermometer code can represent $D + 1$ levels. Figure 7 shows how cosine similarity appears for several different levels when $D = 50$. Thermometer codes produce a translation-invariant kernel that is triangular and has a width of $2D + 1$ levels. It is a nonlocal kernel, in the sense that there are no two points in the encoding range that have a similarity of zero. In practice, thermometer

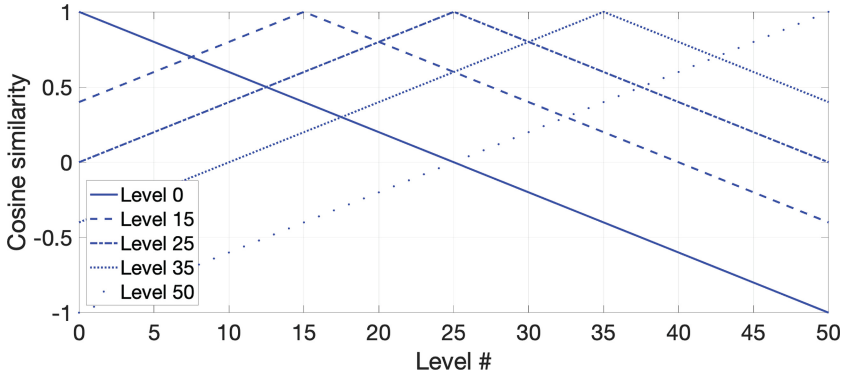

Figure 7: Similarity kernel of the thermometer code shown for several levels; D was set to 50.

codes are used commonly when applying hyperdimensional computing to classification problems.

A.2 The Float Code. The float code, also known as the sliding code, (Goltsev, 1996; Rachkovskij et al., 2005) addresses the issue of the thermometer code—that is, that the similarity decay is not local. This is done by using w consecutive $+1$ components ("float") where the size of w regulates similarity characteristics of the code. For the binary case, the similarity kernel of the float code is the triangular kernel of width $2w + 1$ levels. To encode the lowest value $\mathbf{z}(0)$, the first w components of the vector are set to $+1$s while the rest of the components are 0s. In general, the components of $\mathbf{z}(s)$ are determined as

$$z_i(s) = \begin{cases} +1, & s \leq i < s + w \\ 0, & \text{otherwise.} \end{cases} \tag{A.2}$$

Figure 8 depicts how similarity (inner product normalized by w) decays for several levels in the float code for $D = 60$, $w = 10$. The float code also produces a triangular kernel, but in contrast to the thermometer code, it allows controlling the width of the triangular kernel. The number of levels it could encode is still limited and equals $n - w + 1$.

A.3 The Scatter Code. Scatter codes (Kleyko, Rahimi, et al., 2018; Rachkovskij et al., 2005; Smith & Stanford, 1990) are another alternative to form a locality-preserving encoding where similarity decays nonlinearly. In scatter codes, the code for the first level $\mathbf{z}(0)$ is chosen randomly while each subsequent code is obtained from the previous one by randomly swapping its components with some probability p:

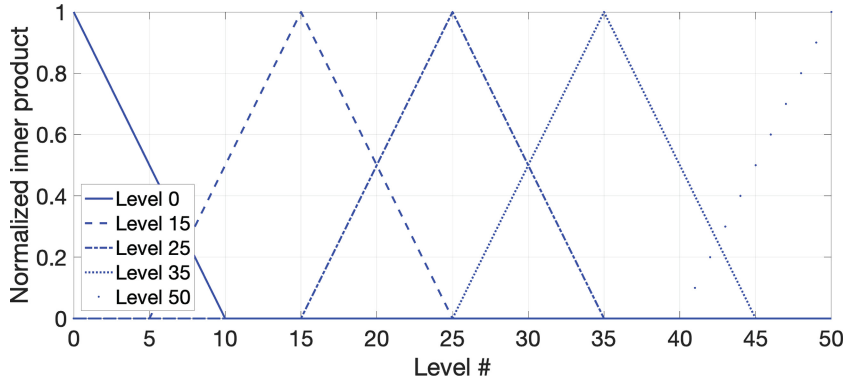

Figure 8: Similarity kernel of the float code shown for several levels; D was set to 60 while w was set to 10.

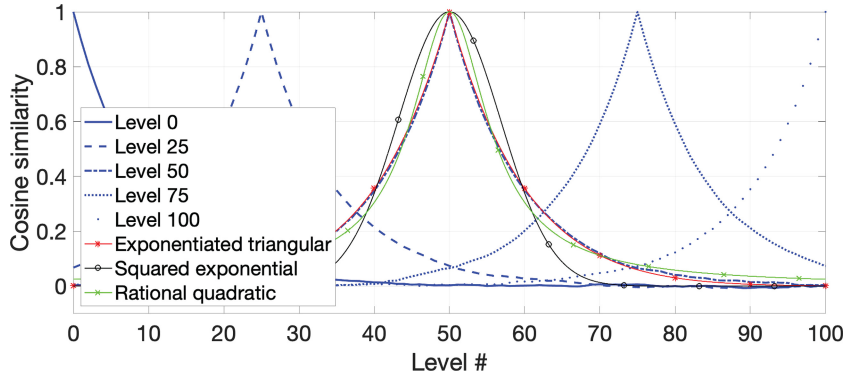

Figure 9: Similarity kernel of a scatter code; D was set to 1000, p was 0.05. The values of similarities were averaged over 50 random initializations of the code.

$$z_i(s) = \begin{cases} -z_i(s-1), & r_i \leq p \\ z_i(s-1), & \text{otherwise} \end{cases} \tag{A.3}$$

where r_i is a random value for the i-th component of $\mathbf{z}(s)$ chosen from the uniform distribution. Note that potentially there is no limitation on how many levels can be created with the scatter codes.

Figure 9 shows how the cosine similarity looks for several different levels formed with the scatter code. Interestingly, the kernels are bell shaped with the exact shape depending on the parameter settings. To better determine which standard kernel will correspond to this similarity, we have empirically fitted three kernels: exponentiated triangular,

$$\mathcal{K}(s_1, s_2) = (1 - \gamma|s_2 - s_1|)^{\alpha}; \tag{A.4}$$

squared exponential,

$$\mathcal{K}(s_1, s_2) = e^{-\frac{(s_2-s_1)^2}{2l^2}}; \tag{A.5}$$

and rational quadratic,

$$\mathcal{K}(s_1, s_2) = \left(1 + \frac{(s_2 - s_1)^2}{2\alpha l^2}\right)^{-\alpha}. \tag{A.6}$$

The parameters of the kernels were chosen using the mean squared error as the fit criterion.

B.1 Kernel Properties of Residue Hyperdimensional Computing

To show that fractional power encoding (modulo m) results in approximation of a particular periodic kernel with period m, we observe that our probability distribution can be written as a Dirac comb function pointwise multiplied by a rect function. This fact becomes useful when we see that our kernel approximates a Fourier integral. Letting $x = x_1 - x_2$, we take the following steps to show convergence in the infinite-dimensional limit:

$$\begin{aligned}
\mathcal{K}_m^*(x_1, x_2) &= \lim_{D\to\infty} \frac{1}{D} \sum_{d=1}^{D} e^{i\phi_d x_1} \overline{e^{i\phi_d x_2}} \\
&= \lim_{D\to\infty} \frac{1}{D} \sum_{d=1}^{D} e^{i\phi_d (x_1 - x_2)} \\
&= \int e^{i\phi(x_1-x_2)} p(\phi) d\phi \\
&= \mathscr{F}^{-1}[p(\phi)](x) \\
&= \mathscr{F}^{-1}\left[\left(\frac{1}{m}\left(\sum_{s\in\mathbb{Z}} \delta\left(\omega - \frac{2\pi}{m}s\right)\right) \cdot \left(\operatorname{rect}\left(\frac{\omega}{2\pi}\right)\right)\right](x) \\
&= \left[\sum_{s\in\mathbb{Z}} \delta(x - ms)\right] \circledast \operatorname{sinc}(x) \\
&= \sum_{s\in\mathbb{Z}} \operatorname{sinc}(x - ms).
\end{aligned}$$

Thus, our kernel is a "sinc comb" function: a sum of sinc functions spaced with a period of m. This result is particularly notable because sinc evaluates

to 0 for integers that are not a multiple of m and means that distinct integers (and remainders) are orthogonal in the high-dimensional space.

To simplify the equation even further, we can derive a sum-less expression for an infinite number of sinc functions. We need to consider two cases: (1) $x = ms$ for some $s \in \mathbb{Z}$ and (2) $x \neq ms$ for all $s \in \mathbb{Z}$.

Case 1 is straightforward. Without loss of generality, let x be the value for which $x - ms = 0$. Then we have:

$$\begin{aligned}\mathcal{K}_m(x) &= \sum_{s\in\mathbb{Z}} \operatorname{sinc}(x - ms) \\ &= 1 + \sum_{n\in\mathbb{N}} \operatorname{sinc}(-mn) + \operatorname{sinc}(mn) \\ &= 1 \text{ (since sinc evaluates to 0 for non-zero integers).}\end{aligned}$$

For case 2, without loss of generality, let $0 < |x| \leq m/2$. The answer differs subtly depending on whether m is even or odd. Let us derive the even case first:

$$\begin{aligned}\mathcal{K}_m(x) &= \sum_{s\in\mathbb{Z}} \operatorname{sinc}(x - ms) \\ &= \sum_{s\in\mathbb{Z}} \frac{\sin(\pi\,(x + ms))}{\pi\,(x + ms)} \\ &= \frac{1}{\pi} \sum_{s\in\mathbb{Z}} \frac{\sin(\pi x)}{x + ms} \\ &= \frac{\sin(\pi x)}{\pi} \sum_{s\in\mathbb{Z}} \frac{1}{x + ms} \\ &= \frac{\sin(\pi x)}{\pi m} \sum_{s\in\mathbb{Z}} \frac{1}{\frac{x}{m} + s} \\ &= \frac{\sin(\pi x)}{\pi m} \pi \cot\left(\frac{\pi x}{m}\right) \\ &= \frac{1}{m} \sin(\pi x) \cot\left(\frac{\pi x}{m}\right),\end{aligned}$$

where the infinite sum is replaced due to an identity via the Herglotz trick: $\pi \cot(\pi x) = \sum_{s\in\mathbf{Z}} \frac{1}{x+s}$. We can also use a second Herglotz identity—that $\pi \csc(\pi x) = \sum_{n\in\mathbf{N}} \frac{(-1)^n}{x+n}$—to solve for the odd case:

$$\mathcal{K}_m(x) = \sum_{s\in\mathbb{Z}} \operatorname{sinc}(x - ms)$$

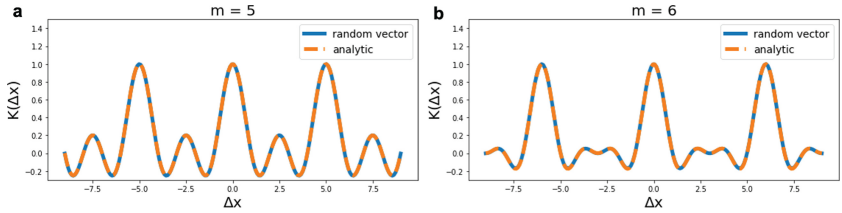

Figure 10: The analytic kernel expected by dashed lines matches the approximate kernel generated by a random vector of sufficiently high dimension ($D = 50{,}000$). (a) Match for an odd modulus ($m = 5$). (b) Match for an even modulus ($m = 6$).

$$
\begin{aligned}
&= \sum_{s\in\mathbb{Z}} \frac{\sin(\pi(x - ms))}{\pi(x - ms)} \\
&= \frac{\sin(\pi x)}{\pi} \sum_{s\in\mathbb{Z}} \frac{(-1)^n}{x - ms} \\
&= \frac{\sin(\pi x)}{\pi} \sum_{s\in\mathbb{Z}} \frac{(-1)^n}{x + ms} \\
&= \frac{\sin(\pi x)}{\pi m} \sum_{s\in\mathbb{Z}} \frac{(-1)^n}{\frac{x}{m} + s} \\
&= \frac{1}{m} \sin(\pi x) \csc\left(\frac{\pi x}{m}\right).
\end{aligned}
$$

We confirm our result by comparing our analytic values for both cases to the kernel induced by a high-dimensional vector (see Figure 10). In the limit of $m \to \infty$, both kernels converge to the sinc function: $\mathcal{K}(x) = \frac{\sin(\pi x)}{\pi x}$.

C.1 Improved Efficiency of Integer Comparison

Finally, we demonstrate how a RNS makes integer comparison within hyperdimensional computing more efficient. In general, it is not possible within hyperdimensional computing to determine which vector encodes a larger number unless one decodes their values to compare them efficiently. Here we demonstrate that RHC significantly reduces the number of comparisons, and for this reason it is more efficient.

Suppose a vector encodes one of M discrete values. A simple baseline method for computing which vector is larger is to decode the values for each via codebook decoding (see section 5.2.1) and then compare the corresponding values. However, this method scales poorly with the range of M, requiring M comparisons per vector.

Instead, we propose a method using a residue number system, where we assume that $M = \prod_{k=1}^{K} m_k$, and the m_k are co-prime. For simplicity, we also assume that the possible integer values lie within the interval $[0, M-1]$. We can then use the sum-of-quotients technique (Dimauro et al., 1993), which remarkably requires only up to $m_1 + m_K + \sum_{m=1}^{K} m_k$ comparisons per vector when m_K is carefully chosen (as described in the next paragraph). This is a meaningful savings in overhead: if the moduli are 99, 100, 101, and 29,999, then $M \approx 3 \times 10^{10}$, whereas our implementation requires $\approx 6 \times 10^4$ comparisons.

For completeness, we introduce the necessary algorithmic steps in this section.

As a preliminary step, we first select moduli $m_1, \ldots, m_{k-1}$, and let $\tilde{M} = \prod_{k=1}^{K-1} m_k$. Then choose the final modulus, m_K, to equal $\sum_{k=1}^{K-1} \frac{\tilde{M}}{m_k}$. The resulting m_k can be shown to be co-prime relative to other moduli, providing a total range of $M = \tilde{M} \cdot m_K$.

Now suppose that we have factorized representations $\{\mathbf{z}_1(a), \mathbf{z}_2(a), \ldots, \mathbf{z}_K(a)\}$ and $\{\mathbf{z}_1(b), \mathbf{z}_2(b), \ldots, \mathbf{z}_K(b)\}$ for two integers, a and b, in the range $[0, M]$, respectively. The algorithm involves the following steps (for simplicity, we describe them for a, but the steps are identical for b).

First, for all moduli except the last, compute $\mathbf{z}_k(\lfloor a/m_K \rfloor)$ and $\mathbf{z}_k(\lfloor b/m_K \rfloor)$. This can be computed entirely with the two binding operations as follows:

$$\mathbf{z}_k(\lfloor a/m_K \rfloor) = (\mathbf{z}_k(a) \odot \mathbf{z}_k(-m_K)) \star \mathbf{z}_k(\xi_k) \tag{C.1}$$

where ξ_k is the modular multiplicative inverse of m_K with respect to m_k.

Second, convert these residue values from a base m_k representation to a base m_K representation. This step can be efficiently performed with codebook lookup because the codebooks are small, requiring $\sum_{k=1}^{K-1} m_k$ comparisons. Then we have representations: $\mathbf{z}_K(\lfloor a/m_K \rfloor), \ldots, \mathbf{z}_K(\lfloor a/m_K \rfloor)$. For shorthand we call these $\zeta_k(a), \ldots, \zeta_{K-1}(a)$, respectively.

Third, we define a new number representation for a and b as follows:

$$\mathbf{v}(a) = \odot_{i=1}^{K-1} (\mathbf{z}_K(s_k) \star \zeta_k(a)) \tag{C.2}$$

The resulting vector $\mathbf{v}(a)$ corresponds to one of m_K integers in the range $[0, m_K - 1]$. It is then tractable to use codebook decoding to determine which value is larger. We'll use the term $\chi(\mathbf{z})$ to denote the integer decoded from $\mathbf{z}$ with codebook decoding. The following steps resolve the comparison:

1. Compare $\chi(\mathbf{v}(a))$ and $\chi(\mathbf{v}(b))$. If $\chi(\mathbf{v}(a)) > \chi(\mathbf{v}(b))$, conclude $a > b$. If $\chi(\mathbf{v}(a)) < \chi(\mathbf{v}(b))$, conclude $a < b$. If the two values are equal, proceed to step (ii).

2. Compare $\chi(\mathbf{z}_1(\lfloor a/m_K \rfloor))$ and $\chi(\mathbf{z}_1(\lfloor b/m_K \rfloor))$. Conclude that $a > b$ if $\chi(\mathbf{z}_k(\lfloor a/m_K \rfloor)) > \chi(\mathbf{z}_k(\lfloor b/m_K \rfloor))$, and that $a < b$ if $\chi(\mathbf{z}_k(\lfloor a/m_K \rfloor)) < \chi(\mathbf{z}_k(\lfloor b/m_K \rfloor))$. If the two values are equal, proceed to step 3.
3. Compare the values of $\chi(\mathbf{z}_K(a))$ and $\chi(\mathbf{z}_K(b))$. If $\chi(\mathbf{z}_K(a)) > \chi(\mathbf{z}_K(b))$, conclude $a > b$. If $\chi(\mathbf{z}_K(a)) < \chi(\mathbf{z}_K(b))$, conclude $a < b$. Otherwise conclude $a = b$.

The proof of correctness can be found in Dimauro et al. (1993); the algorithm proposed here can be thought of as an implementation using RHC. Both additive and multiplicative binding are necessary to implement the approach, demonstrating the value of having both.

Acknowledgments

We thank Anthony Thomas, Eric Weiss, Joshua Cynamon, Amir Khosrowshahi, and Madison Cotteret for insightful discussions and feedback. The work of C.J.K. was supported by the Department of Defense through the National Defense Science and Engineering Graduate Fellowship Program. The work of D.K., C.B., F.T.S., and B.A.O. was supported in part by Intel's THWAI program. The work of C.J.K., C.B., P.K., and B.A.O. was supported by the Center for the Co-Design of Cognitive Systems, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation program sponsored by DARPA. D.K. has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement 839179. The work of F.T.S. was supported in part by NIH under grant R01-EB026955 and in part by NSF under grant IIS-118991.

Code Availability

Code is available at https://github.com/cjkymn/residuehdcomputing.

References

Agmon, H., & Burak, Y. (2020). A theory of joint attractor dynamics in the hippocampus and the entorhinal cortex accounts for artificial remapping and grid cell field-to-field variability. In *Proceedings of the eLife*, *9*, e56894.

Babai, L. (1979). *Monte-Carlo algorithms in graph isomorphism testing*. Technical report, DMS 79-10, Université de Montréal.

Bordelon, B., & Pehlevan, C. (2022). Population codes enable learning from few examples by shaping inductive bias. *eLife*, *11*, e78606. 10.7554/eLife.78606

Buckman, J., Roy, A., Raffel, C., & Goodfellow, I. (2018). Thermometer encoding: One hot way to resist adversarial examples. In *Proceedings of the International Conference on Learning Representations* (pp. 1–22).

Bybee, C., & Sommer, F. (2022). Optimal oscillator memory networks. In *Proceedings of the Neuro-Inspired Computational Elements Conference* (pp. 81–83).

Diao, C., Kleyko, D., Rabaey, J. M., & Olshausen, B. A. (2021). Generalized learning vector quantization for classification in randomized neural networks and hyperdimensional computing. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1–9).

Dimauro, G., Impedovo, S., & Pirlo, G. (1993). A new technique for fast number comparison in the residue number system. *IEEE Transactions on Computers*, *42*(5), 608–612. 10.1109/12.223680

Dumont, N. S.-Y., & Eliasmith, C. (2020). Accurate representation for spatial cognition using grid cells. In *Proceedings of the Annual Meeting of the Cognitive Science Society* (pp. 2367–2373).

Fiete, I. R., Burak, Y., & Brookings, T. (2008). What grid cells convey about rat location. *Journal of Neuroscience*, *28*(27), 6858–6871. 10.1523/JNEUROSCI.5684-07.2008

Frady, E. P., Kent, S. J., Olshausen, B. A., & Sommer, F. T. (2020). Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures. *Neural Computation*, *32*(12), 2311–2331. 10.1162/neco_a_01331

Frady, E. P., Kleyko, D., Kymn, C. J., Olshausen, B. A., & Sommer, F. T. (2021). *Computing on functions using randomized vector representations.* arXiv:2109.03429.

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

Frady, E. P., Kleyko, D., & Sommer, F. T. (2018). A theory of sequence indexing and working memory in recurrent neural networks. *Neural Computation*, *30*(6), 1449–1513. ACM. 10.1162/neco_a_01084

Frady, E. P., & Sommer, F. T. (2019). Robust computation with rhythmic spike patterns. *Proceedings of the National Academy of Sciences*, *116*(36), 18050–8059. 10.1073/pnas.1902653116

Gardner, R. J., Hermansen, E., Pachitariu, M., Burak, Y., Baas, N. A., Dunn, B. A., . . . Moser, E. I. (2022). Toroidal topology of population activity in grid cells. *Nature*, *602*(7895), 123–128. 10.1038/s41586-021-04268-7

Garner, H. L. (1959). The residue number system. In *Proceedings of the Western Joint Computer Conference* (pp. 46–153).

Gayler, R. W. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In *Proceedings of the Joint International Conference on Cognitive Science* (pp. 133–138).

Goldreich, O., Ron, D., & Sudan, M. (1999). Chinese remaindering with errors. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing* (pp. 225–234).

Goltsev, A. D. (1996). An assembly neural network for texture segmentation. *Neural Networks*, *4*(9), 643–653. 10.1016/0893-6080(95)00136-0

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801–806. 10.1038/nature03721

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, *1*, 139–159. 10.1007/s12559-009-9009-8

Karp, R. (1972). Chapter reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer computations*. Plenum Press.

Kent, S. J., Frady, E. P., Sommer, F. T., & Olshausen, B. A. (2020). Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods. *Neural Computation*, *32*(12), 2332–2388. 10.1162/neco_a_01329

Kim, H.-S. (2018). HDM: Hyper-dimensional modulation for robust low-power communications. In *Proceedings of the 2018 IEEE International Conference on Communications* (pp. 1–6).

Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. Pearson Education India.

Kleyko, D., Bybee, C., Huang, P.-C., Kymn, C. J., Olshausen, B. A., Frady, E. P., & Sommer, F. T. (2023). Efficient decoding of compositional structure in holistic representations. *Neural Computation*, *35*(7), 1159–1186. 10.1162/neco_a_01590

Kleyko, D., Bybee, C., Kymn, C. J., Olshausen, B. A., Khosrowshahi, A., Nikonov, D. E., . . . Frady, E. P. (2022). Integer factorization with compositional distributed representations. In *Proceedings of the Neuro-Inspired Computational Elements Conference* (pp. 73–80).

Kleyko, D., Davies, M., Frady, E. P., Kanerva, P., Kent, S. J., Olshausen, B. A., . . . Sommer, F. (2022). Vector symbolic architectures as a computing framework for emerging hardware. *Proceedings of the IEEE*, *110*(10), 1538–1571. 10.1109/JPROC.2022.3209104

Kleyko, D., Kheffache, M., Frady, E. P., Wiklund, U., & Osipov, E. (2020). Density encoding enables resource-efficient randomly connected neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(8), 3777–3783. 10.1109/TNNLS.2020.3015971

Kleyko, D., Osipov, E., Papakonstantinou, N., & Vyatkin, V. (2018). Hyperdimensional computing in industrial systems: The use-case of distributed fault isolation in a power plant. *IEEE Access*, *6*, 30766–30777. 10.1109/ACCESS.2018.2840128

Kleyko, D., Rachkovskij, D. A., Osipov, E., & Rahimi, A. (2022). A survey on hyperdimensional computing aka vector symbolic architectures, Part I: Models and data transformations. *ACM Computing Surveys*, *55*(6), 1–40.

Kleyko, D., Rachkovskij, D. A., Osipov, E., & Rahimi, A. (2023). A survey on hyperdimensional computing aka vector symbolic architectures, Part II: Applications, cognitive models, and challenges. *ACM Computing Surveys*, *55*(9), 1–52.

Kleyko, D., Rahimi, A., Rachkovskij, D. A., Osipov, E., & Rabaey, J. M. (2018). Classification and recall with binary hyperdimensional computing: Trade-offs in choice of density and mapping characteristics. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(12), 5880–5898. 10.1109/TNNLS.2018.2814400

Komer, B. (2020). *Biologically inspired spatial representation*. PhD diss., University of Waterloo.

Kriegeskorte, N., Mur, M., & Bandettini, P. A. (2008). Representational similarity analysis: Connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience*, *2*, 4. 10.3389/neuro.01.016.2008

Krupic, J., Burgess, N., & O'Keefe, J. (2012). Neural representations of location composed of spatially periodic bands. *Science*, *337*(6096), 853–857. 10.1126/science.1222403

Kymn, C. J., Mazelet, S., Ng, A., Kleyko, D., & Olshausen, B. A. (2024). Compositional factorization of visual scenes with convolutional sparse coding and resonator networks. In *Proceedings of the 2024 Neuro-Inspired Computational Elements Conference* (pp. 1–9).

Landau, E. (1903). Über die maximalordnung der permutationen gegebenen grades. *Archiv der Math. und Phys*, *3*, 92–103.

Langenegger, J., Karunaratne, G., Hersche, M., Benini, L., Sebastian, A., & Rahimi, A. (2023). In-memory factorization of holographic perceptual representations. *Nature Nanotechnology*, *18*(5), 479–485. 10.1038/s41565-023-01357-8

Malozemov, V. N., & Pevnyi, A. B. (2009). Equiangular tight frames. *Journal of Mathematical Sciences*, *157*(6), 789–815. 10.1007/s10958-009-9366-6

Mathis, A., Herz, A. V., & Stemmler, M. B. (2012). Resolution of nested neuronal representations can be exponential in the number of neurons. *Physical Review Letters*, *109*(1), 018103. 10.1103/PhysRevLett.109.018103

Mathis, A., Stemmler, M. B., & Herz, A. V. (2015). Probable nature of higher-dimensional symmetries underlying mammalian grid-cell activity patterns. *eLife*, *4*, e05979. 10.7554/eLife.05979

Mohan, P. A. (2016). *Residue number systems*. Springer.

Nanda, S. (2005). *Subset sum problem*. https://cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/nanda-scribe-3.pdf

Noest, A. J. (1988). Discrete-state phasor neural networks. *Physical Review A*, *38*(4), 2196. 10.1103/PhysRevA.38.2196

Olshausen, B. A. (2014). Perception as an inference problem. In M. S. Gazzaniga & G. R. Mangun (Eds.), *The cognitive neurosciences* (5th ed., pp. 295–304). MIT Press.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607–609. 10.1038/381607a0

Omondi, A. R., & Premkumar, A. B. (2007). *Residue number systems: Theory and implementation* (vol. 2). World Scientific. 10.1142/p523

Penz, P. A. (1987). The closeness code: An integer to binary vector transformation suitable for neural network algorithms. In *Proceedings of the IEEE First Annual International Conference on Neural Networks* (pp. 515–522).

Plate, T. A. (1992). Holographic recurrent networks. In S. Hanson, J. Cowan, & C. Giles (Eds.), *Advances in neural information processing systems, 5* (pp. 34–41). Curran.

Plate, T. A. (1994). *Distributed representations and nested compositional structure*. PhD diss., University of Toronto.

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, *6*(3), 623–641. 10.1109/72.377968

Plate, T. A. (2003). *Holographic reduced representation: Distributed representation for cognitive structures* (vol. 150). CSLI Publications Stanford.

Pouget, A., Dayan, P., & Zemel, R. (2000). Information processing with population codes. *Nature Reviews Neuroscience*, *1*(2), 125–132. 10.1038/35039062

Rachkovskij, D. A. (2007). Linear classifiers based on binary distributed representations. *Information Theories and Applications*, *14*(3), 270–274.

Rachkovskij, D. A., Slipchenko, S. V., Kussul, E. M., & Baidyk, T. N. (2005). Sparse binary distributed encoding of scalars. *Journal of Automation and Information Sciences*, *37*(6), 12–23. 10.1615/J Automat Inf Scien.v37.i6.20

Rahimi, A., Kanerva, P., Benini, L., & Rabaey, J. M. (2019). Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals. *Proceedings of the IEEE*, *107*(1), 123–143. 10.1109/JPROC.2018.2871163

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. In J. Platt, D. Koller, Y, Singer, & S. Roweis (Eds.), *Advances in neural information processing systems, 20* (pp. 1–8). Curran.

Räsänen, O., & Saarinen, J. (2016). Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(9), 1878–1889.

Renner, A., Supic, L., Danielescu, A., Indiveri, G., Olshausen, B. A., Sandamirskaya, Y., . . . Frady, E. P. (2024). *Neuromorphic visual scene understanding with resonator networks*. *Nature Machine Intelligence*, *6*(6), 641–652.

Schindler, K. A., & Rahimi, A. (2021). A primer on hyperdimensional computing for iEEG seizure detection. *Frontiers in Neurology*, *12*, 1–12. 10.3389/fneur.2021.701791

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., & Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 7462–7473). Curran.

Smith, D., & Stanford, P. (1990). A random walk in Hamming space. In *Proceedings of the International Joint Conference on Neural Networks*, *2* (pp. 465–470).

Snaider, J., & Franklin, S. (2014). Modular composite representation. *Cognitive Computation*, *6*, 510–527. 10.1007/s12559-013-9243-y

Srinivasan, S., & Fiete, I. (2011). Grid cells generate an analog error-correcting code for singularly precise neural computation. *Nature Neuroscience*, *14*(10), 1330–1337. 10.1038/nn.2901

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Sing- hal, U., . . . Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 7537–7547). Curran.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*, *30*. Curran.

Wang, T., & Roychowdhury, J. (2019). OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems. In *Proceedings of the 18th International Conference on Unconventional Computation and Natural Computation* (pp. 232–256).

Wohlberg, B. (2017). SPORCO: A Python package for standard and convolutional sparse representations. In *Proceedings of the Python in Science Conference*, *15* (pp. 1–8).

Yu, T., Zhang, Y., Zhang, Z., & De Sa, C. M. (2022). Understanding hyperdimensional computing for parallel single-pass learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems, 35* (pp. 1157–1169). Curran.

Received March 27, 2024; accepted August 10, 2024.