1. Appendix

1.1. Encoding examples

Exclusive encoding example: suppose $SUN = \langle round, yellow, hot\rangle$ and $BALL = \langle round, yellow, cold\rangle$ are two objects belong to $\mathcal{U} = \langle Shape, Color, Temp\rangle$. by assigning a random vector to each feature, the exclusive encoding maps these two objects into an orthogonal space, because of a single feature difference between the objects. In other words:

$$\begin{aligned}\phi(SUN) &= \mathbf{C}_{round} \odot \mathbf{C}_{yellow} \odot \mathbf{C}_{hot} \\ \phi(BALL) &= \mathbf{C}_{round} \odot \mathbf{C}_{yellow} \odot \mathbf{C}_{cold}\end{aligned} \tag{1}$$

$$\delta(\phi(SUN), \phi(BALL)) \approx 0 \tag{2}$$

Inclusive encoding example: suppose $SUN = \langle round, yellow, hot\rangle$ and $BALL = \langle round, yellow, cold\rangle$ are two objects belong to $\mathcal{U} = \langle Shape, Color, Temp\rangle$. The include encoding maps these two objects such that it preserves the similarity of the objects:

$$\begin{aligned}\phi(SUN) = \mathbf{K}_{shape} \odot \mathbf{C}_{round} \oplus \mathbf{K}_{color} \odot \mathbf{C}_{yellow} \\ \oplus \mathbf{K}_{temp} \odot \mathbf{C}_{hot} \\ \phi(BALL) = \mathbf{K}_{shape} \odot \mathbf{C}_{round} \oplus \mathbf{K}_{color} \odot \mathbf{C}_{yellow} \\ \oplus \mathbf{K}_{temp} \odot \mathbf{C}_{cold}\end{aligned} \tag{3}$$

$$\delta(\phi(SUN), \phi(BALL)) \approx \frac{2}{3} \tag{4}$$

Similarly, the inclusive encoding of SUN will have a similarity of around 1/3 with $\langle yellow\rangle$, 2/3 with $\langle yellow, hot\rangle$ and 2/3 with $\langle square, yellow, hot\rangle$.

1.2. Sequence matching

DNA Sequence matching is one of the key problems in identifying and analyzing genome data (Fig. 1a). HDlm transforms inherent sequential processes of genome pattern matching and alignment to highly parallelizable computation tasks. Our platform exploits HDC memorization to encode and represent the genome sequences using high-dimensional vectors. Then, it imitates the essential functionalities of human memory with hypervector operations.

DNA encoding: Given the base DNA (`A`, `C`, `G`, `T`), HDlm assigns a random hypervector to each alphabet: $\{\vec{A}, \vec{C}, \vec{G}, \vec{T}\} \in \{-1, +1\}^D$. Here, we explain how one can use the base hypervectors to encode a DNA sequence. Let us consider a short query string, '`GCAT`'. The exclusive encoding is performed by binding the corresponding base hypervectors while utilizing the permutation to preserve the position of each base in the sequence: $\mathbf{h} = \vec{G} * \rho^1\vec{C} * \rho^2\vec{A} * \rho^3\vec{T}$. The inclusive encoding maps the same sequence using: $\mathbf{h} = \mathbf{K}_1 * \vec{A} + \mathbf{K}_2 * \vec{C} + \mathbf{K}_3 * \vec{G} + \mathbf{K}_4 * \vec{T}$, where $\mathbf{K}$s are randomly generated position hypervectors. Finally, correlative association encodes a sequence using the same equation as inclusive encoding, but it preserves their similarity by lifting a pre-defined portion of the dimensions for each base hypervector: $\mathbf{h} = \phi_1(\vec{A}) * \phi_2(\rho^1\vec{C}) * \phi_3(\rho^2\vec{G}) * \phi_4(\rho^3\vec{T})$. The ϕ functions are non-overlapped and uniformly split D dimensions into equal sub-dimensions.

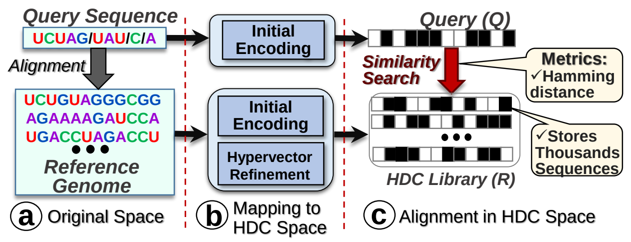

Figure 1. HDlm application in DNA sequence matching

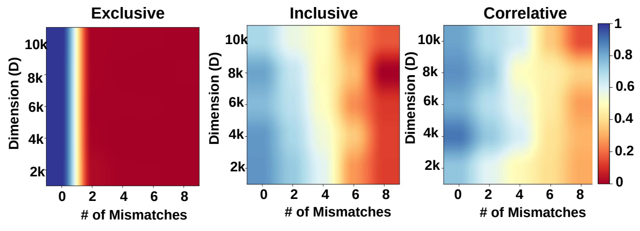

Figure 2. HDlm Heatmap performing exact and approximate match in different dimensionality

Reference generation: HDlm aggregates all encoded sequences to generate a reference genome, called *HDC Library*. An HDC library consists of several reference hypervectors, where each exploits HDC mathematics to memorize thousands of encoded genome sequences (Fig. 1c). Similar to the human brain, it is difficult for HDlm to remember the information of all accumulated data into each reference hypervector using single-pass memorization. In HDlm, the pattern of the most common sequence dominates the reference hypervector and results in the vanishing of the less frequent patterns. To better memorize the information, HDlm looks at the same object multiple times. During this iterative process, HDlm boosts the HDC library's accuracy by discarding the mispredicted queries from the corresponding model hypervector and adding them to the right one.

Inference: HDlm performs pattern matching by checking the similarity of an encoded protein sequence with the HDC library. Depending on the encoding, HDlm searches for an exact or approximate match of a query with each reference hypervector.

Fig. 2 shows the capability of each encoder to identify exact and approximate matches. Our evaluation shows that the exclusive encoding can only support exact pattern matching. The inclusive encoding is strong on approximate matching while having low hypervector capacity enabling exact search. Another weakness of inclusive encoding is its lack of support in identifying which feature or DNA bases have been mismatched. Finally, our correlative encoding supports both exact and approximate matching. In addition, our approximate match not only reports the number of mismatches between the query and stored reference hypervector but also identifies which features have been mismatched. This enables an opportunity to support alignment, which is a more complex and demanding bioinformatic task.

1.3. More discussion and future work

Implementation Efficiency. Unlike conventional deep learning solutions, HDC's operations (binding, bundling, permutation) and the high-dimensional vector space enable efficient data analysis and learning directly from encoded data, leveraging hardware acceleration and parallel processing capabilities. Therefore, HDC models are typically much more computationally efficient compared to DNN models while neural networks have better feature extraction capabilities that may be suited for more complex tasks; many works hence opt for a small neural network that preprocesses the data before using HDC model [1, 2]. Within the HDC context, Inclusive and exclusive decoding share similar computational complexity; HDlm requires additional memory and computation for listing and masking, although both operations can be very efficient and parallelizable.

To establish a concise mathematical analysis of the behavior of the model, we have focused our attention on the relatively more abstract specification of the encoding. We hope to explore the efficient implementation of HDlm for future work, as there are subtle choices to be made given different constraints. For example, prioritizing computational efficiency over memory efficiency will lead to the design decision of generating and storing the hypervectors under the universal indexing set D, effectively costing $|D|$ dimension per vector. In comparison, a more memory-efficient approach may store the hypervectors under its indexing set costing at most $|D|$ features, and additional acceleration techniques can be applied to allow faster correlative composition, as the indexing set of two operand hypervectors may differ.

References

[1] Michael Hersche, Mustafa Zeqiri, Luca Benini, Abu Sebastian, and Abbas Rahimi. A neuro-vector-symbolic architecture for solving raven's progressive matrices. *Nature Machine Intelligence*, 5(4):363–375, 2023. 2

[2] Igor Nunes, Mike Heddes, Tony Givargis, Alexandru Nicolau, and Alex Veidenbaum. Graphhd: Efficient graph classification using hyperdimensional computing. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1485–1490. IEEE, 2022. 2