Supplementary Material for HDQMF: Holographic Feature Decomposition Using Quantum Algorithms

Prathyush Poduval
University of California, Irvine
Irvine, CA 92617
ppoduval@uci.edu

Zhuowen Zou
University of California, Irvine
Irvine, CA 92617
zhuowez1@uci.edu

Mohsen Imani
University of California, Irvine
Irvine, CA 92617
m.imani@uci.edu

1. More on Hyperdimensional Computing

This section provides a more detailed discussion of HDC schemes, semantics, and decomposition challenges.

Scheme and data structures: The schema design for a certain data structure is not unique and is up to the utility of the resulting representation. For a few typical examples:

1. Dictionaries can be considered as a set of key-value pairs. In this case, keys and values are atoms, and are bound together to make pairs: $h_{(x,y)} = h_x \odot h_y$, and the dictionaries is a bundling of such pairs. This is a special case of the "bind-then-bundle" scheme
2. Sequences considers entries as atoms and represents order via permutation: $h_{[x_0,\cdots,x_n]} = \bigodot_{i=0}^{n} \rho^i(h_{x_i})$.
3. n-grams (short sequences of alphabets of length n) considers alphabets as atoms and also uses permutation to represent order. However, the way the indexed atoms are combined is through binding: $h_{[x_0,\cdots,x_n]} = \bigodot_{i=0}^{n} \rho^i(h_{x_i})$.

The design of the schema needs to ensure that the composite hypervector "works as intended". For example, a dictionary needs to support the *get* operation, retrieving a value by its key. For a simple dictionary $D = \{1 : a, 2 : b\}$, we have the dictionary hypervector $h_D = h_1 \odot h_a \oplus h_2 \odot h_b$, and one may retrieve the value for key 1 by (un)binding the h_D with the key hypervector h_1 (note that bipolar vectors are their own inverses):

$$h_D \odot h_1 = (h_1 \odot h_a \oplus h_2 \odot h_b) \odot h_1 = h_a \oplus h_2 \odot h_b \odot h_1 \quad (1)$$

Because the term $h_2 \odot h_b \odot h_1$ in Eq. (1) is a binding of three hypervectors, it will be dissimilar to any of its components and other entries in the value codebook (as well as those in the key codebook). We thus have a "noisy" version of the h_a, which we can decode using the value codebook.

The data structure and the scheme are only loosely coupled: the data structure offers a useful analogy for understanding the functions of a scheme. This analogy assists in two key ways: firstly, it provides a framework for interpreting the scheme's utility from the perspective of data structure; secondly, it facilitates the scheme's design specifically for a given data structure. This is best demonstrated by the difference between the schema for sequences and n-grams. For sequence encoding, one would want to perform tasks such as sequence manipulation, data retrieval, and sequence comparison. This is more suitable for bundling since it preserves the similarity of the components: the resulting sequence hypervector will have high similarity with its entries (permuted by the right amount) and with a hypervector representing a similar sequence. As for n-grams, it may be more desirable to treat each n-gram as different (e.g. in the case of natural language processing). N-grams are also typically treated as "tokens" and therefore do not require decoding back to their component. These reasons make binding a better candidate.

1.1. Establishing HDC Semantic for Interpretability:

With the codebooks and basic HDC operators, one may construct composite hypervectors analogous to more complex data structures. Therefore, one of the advantages of the HDC framework is that the manipulation of the hypervectors is fully interpretable: every transform by the HDC operator can be interpreted as a certain type of data structure manipulation. Consequently, it becomes feasible to extract meaningful insights or interpret outcomes from HDC representations down to the elementary level. To this end, we compare bundling and binding from the perspective of se-

mantics to establish interpretability over HDC algorithms. Most critically, we highlight the effect they have on the semantic domain of HDC and the semantic qualities of the representation. We omit permutation as it is similar to binding and is out of the scope of this paper.

The **semantics** of the HDC representation refers to how much information a hypervector maintains about the relations and the identities of the atomic hypervectors it is built upon. We define the **semantic domain** to be the domain in which the similarity function δ is meaningful. We define **semantic quality** as the accuracy of retrieving said content. The semantic quality is directly influenced by semantic ambiguity and semantic complexity, on which we elaborate below.

First and foremost, without the HDC operators, the semantic domain is the codebook (or union of all the codebooks). With sufficiently large hyperdimension, the codebook entries are nearly orthogonal and hence distinguishable by the similarity function. The higher the hyperdimension, the less noise between the normalized similarity between dissimilar hypervectors. We describe how bundling and binding extend this semantic domain.

Bundling and semantic ambiguity: Treating the bundling of hypervectors as the representation of sets is reasonable because the membership of an element in a set can be approximated via the similarity function. The domain of δ where it is semantically meaningful is lifted from the set of codebook entries to the collection of subsets of the codebook entries. The notion of set cardinality can also be conceived and is usually estimated from the length of the hypervector. We can generalize the approach to multiset by allowing weighted bundling, extending the domain to the additive group generated by the codebook entries.

This "semantic lift" is not infinite, however. Since δ approximates the discrete metric, the error of every atomic evaluation produces a noise term to the final result. In such set membership evaluation, this noise aggregates proportional to the size of the sets. We refer to this noise as the **semantic ambiguity** of the representation, also referred to as crosstalk noise [8], as it directly affects the ability to decode and hence interpret the representation. With enough deviation from the expected value, a semantic error occurs, where a nonmember has a similarity with a set beyond the threshold value or a member below. This implies that there is an intrinsic bound to this semantic lift, and this bound is "soft" in the sense that there is a trade-off between the height of the bound and the quality of the semantics. Many work has proposed estimation to this semantic lift in the form of "memory capacity". Some are characterized asymptotically [1] while others are more exact [8].

Binding and semantic complexity: The binding of hypervectors is considered a tuple because the resulting hypervector is dissimilar to any of its components and "unbinding" one of the components from the hypervector reveals the others. Consequently, the semantic domain extends to any compositions of codebook entries.

This semantic quality of bound hypervectors faces somewhat of a different challenge compared to addition: while multiplication does not increase the length of the hypervector when binding atom hypervectors, the decoding of hypervectors is much more challenging because a similarity check will indicate a match if and only if all components match the given tuple. On the one hand, binding (at the atomic level) does not contribute to the increase of semantic ambiguity: because bound hypervectors are still bipolar and dissimilar to other hypervectors, they contribute a similar amount of noise as an atomic hypervector. On the other hand, since decoding the bound hypervector relies on finding the correct combination of atomic hypervectors and verifying by similarity, the latter increases the search space size. We refer to the size of the search space as the **semantic complexity** of the representation. The semantic complexity also directly affects the interpretability of the model. Even though it does not directly contribute to the magnitude of the noise in similarity as bundling does, it significantly enlarges the search space. Due to the randomizing nature of the hypervectors, it is then more likely that, in the presence of noise, some combination of hypervectors becomes more similar to the original.

1.2. Decomposition for Interpretable HDC

As a computing paradigm inspired by the principles of high-dimensional mathematics and cognitive science, HDC framework has been extended to perform various computational tasks, most prominently in machine learning and cognitive computation. For learning tasks, the design of encoding, data structure, and schema serves as the foundation for data and model representation. In the context of classification, an HDC model trained a class hypervector for each class such that the similarity between a query and class hypervectors determines mode prediction: the class hypervector most similar to the query is the predicted class. For cognitive processing, the HDC framework relies on HDC operators and similarity functions to transform and retrieve information about the complex objects represented by the composite hypervectors. These practical applications include memory storage and retrieval, signal processing, and data analysis, as well as pattern recognition, natural language processing, and robotics [3, 5].

The semantic quality provides us with some insight into how interpretable HDC can be. Although the decoding accuracy of the compound hypervector is affected by both semantic ambiguity and complexity, there are many solutions to mitigate this issue. For example, increasing the hyperdimension can reduce the noise between elementary hypervectors, subsequently improving the decoding accuracy. Al-

ternatively, one may use intermediate codebooks to provide hierarchical decoding.

Practical applications of the HDC model face two challenges in decoding, one in accuracy and one in efficiency. The accuracy challenge relates to the potential loss of precision or fidelity during the encoding process. More specifically:

1. At the implementation level, during the encoding of information into high-dimensional vectors, noise or distortions can be introduced due to imperfect measurement or encoding processes, limitations of the hardware, or other sources of interference. While HDC are well-known for robust representation, decoding from distorted hypervectors nonetheless affects the expected accuracy of the model, if only to a small degree.
2. At the algorithmic level, the distributivity of binding over bundling leads to potential ambiguity in representations; more complex data structures that leverage both binding and bundling may cause confusion if not designed carefully. This has been discussed in detail and partly addressed in [6].
3. At the computational level, HDC representation inherently introduces small noises to the representation of atomic concepts to effectively leverage the high-dimensional space, resulting in a signal-to-noise ratio determined directly by the dimensionality of the hypervectors. The noise is carried over when performing HDC computations and may aggregate over bundling. These scalability issues are typically resolved by adding "intermediate codebooks" that store composite hypervectors. This allows the decoding to be hierarchical, effectively enabling noise reduction in the process.

The decoding efficiency challenge, which we aim to tackle, happens in the case of binding. Bundling allows the hypervector to retain high similarity with its constituents, so recovering individual components of a bundled hypervector is simply set membership detection (it can also be extended to a weighted set naturally). Permutation has an exhaustive search algorithm that is linear with respect to the number of maximum permutations allowed (or possible; for the circular shift it is the hyperdimension), so one may recover both the position and identity of the hypervector by repeatedly permuting and checking it against the codebook. Binding does not preserve similarity and therefore requires a search over all possible combinations of the atomic hypervectors. Because binding creates dissimilar hypervectors, a bound hypervector cannot be decoded directly by similarity check against the codebook. The exhaustive way of decoding is to compare it with all possible binding results. Consequently, the search space scales exponentially with the level of binding and polynomially with the size of the codebook. Therefore, it is much harder to find all the components of bound hypervectors. We call this the HDC factorization problem.

This hard combinatorial search problem that resides at the core of HDC decoding has brought significant limitations to HDC models. As succinctly put by [2], previous uses of VSAs have mostly avoided this issue by either restricting the complexity of data structures or employing a straightforward method of examining all conceivable combinations when required [7]. However, this has limited the practical use of VSAs since there hasn't been an effective solution for accessing elements within compound data structures that contain multiple components. In the next section, we formalize the HDC factorization problem and discuss existing classical approaches and their limitations.

2. Resonator network

The state-of-the-art approach to solving the high-dimensional factorization problem is the Resonator network [2]. This method utilizes the principle of superposition to iteratively and approximately find the factorization. Initially, each factor's prediction is formed as the superposition of all entries in the codebook: $\hat{c}_i^{(0)} = g(\bigoplus_{j=1}^{N} \mathcal{C}_{i,j})$ where the superscript indicates the iteration step and g represents the sign function, ensuring the hypervector remains bipolar. At each iteration, the prediction for each factor is updated based on the composite vector and the predictions of other factors:

$$\hat{c}_i^{(t+1)} = g(\mathcal{C}_i \mathcal{C}_i^T (c \odot (\bigodot_{j \neq i} \hat{c}_j^{(t)})) \tag{2}$$

Here, the hypervectors are column vectors and each codebook C is hence a $D \times N$ matrix. The algorithm terminates when the solution stabilizes, or when it has run the maximum number of iterations allowed.

Intuition: Due to the properties of the high dimensional space, a thresholded superposition retains a relatively high similarity to its components. The initial guess approximately contains all possible candidates for a factor. When we bind all the initial guesses together, the resulting hypervector contains all possible combinations of the factors, thus enumerating all candidates for the factorization, in a much lower resolution as constrained by the capacity of the hypervector. Since bipolar hypervectors are binding inverses to themselves, the term $c \odot (\bigodot_{j \neq i} \hat{c}_j^{(t)})$ in Eq. (2) can be interpreted as: for each factor i, we unbind the all other factors, each a superposition of the candidates, such that the result is the correct solution to the codebook i and a collection of noise terms. Finally, the resonator network performs a noise reduction operation $\mathcal{C}_i \mathcal{C}_i^T$ for the rest of Eq. (2). Multiplying with $\mathcal{C}_i^T$ produces a vector of length N with the j^{th} entry measuring the similarity with the j^{th} hypervector of the codebook. Subsequently multiplying with $\mathcal{C}_i$ produces a hypervector that is a weighted bundling of the codebook entries. As a result, after the sign function, we again receive a bipolar hypervector that is the superposition of the

codebook entries, only that the coefficients for each entry changed in the direction of the approximated similarity.

Advantages and Limitation: [4] demonstrated several pros and cons of the resonator network methods compared to optimization-based methods. Firstly, while optimization methods assure convergence (typically to local minima) under any initial condition, the resonator network does not offer such a guarantee, particularly when the problem size exceeds the model's parameter limits. Secondly, in terms of handling problem sizes within its operational capacity, the resonator network can, with a high degree of probability, address problems significantly larger (by two orders of magnitude) than those manageable by optimization-based methods. This capability is formally quantified and empirically validated as "operational capacity". Regarding computational efficiency, both methodologies necessitate a comparable number of iterations for convergence when the problem size falls within their respective operational capacity. However, due to its simpler and more parallelizable computational processes, the resonator network boasts greater speed. In essence, unlike optimization-based methods that make no assumption of the properties of the codebooks, the resonator network recognizes that they are generated from i.i.d. distribution and hence leveraged HDC principles, as suggested in the previous section.

Despite its empirical successes, the theoretical understanding of the resonator network's capacity and behavior remains incomplete. In fact, practical HDC factorization still faces the fundamental limit of the problem size, and this limit comes in two levels. First, the problem space scaled poorly w.r.t. the codebook size and the number of factors - a direct result of combinatorial complexity. Secondly, the inherent and implicit requirement of "high-dimensionality" of the vectors imposes further demand on the dimension of the hyperspace. This is due to a demand for high signal-to-noise ratios and is partly what gives HDC its nice properties such as robustness. When it comes to decoding, however, even in the absence of noise, the enforced high dimensionality limits the scalability of HDC. To overcome the problems of capacity and convergence and to improve upon efficiency, we move to quantum computing, which both allows a quadratic speed up over the linear search and bypasses the high signal-to-noise ratio, as the quantum basis states are strictly orthogonal.

References

[1] E Paxon Frady, Denis Kleyko, and Friedrich T Sommer. A theory of sequence indexing and working memory in recurrent neural networks. *Neural Computation*, 30(6):1449–1513, 2018. 2

[2] E Paxon Frady, Spencer J Kent, Bruno A Olshausen, and Friedrich T Sommer. Resonator networks, 1: an efficient solution for factoring high-dimensional, distributed representations of data structures. *Neural computation*, 32(12):2311–2331, 2020. 3

[3] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. Voicehd: Hyperdimensional computing for efficient speech recognition. In *International Conference on Rebooting Computing (ICRC)*, pages 1–6. IEEE, 2017. 2

[4] Spencer J Kent, E Paxon Frady, Friedrich T Sommer, and Bruno A Olshausen. Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods. *Neural computation*, 32(12):2332–2388, 2020. 4

[5] Denis Kleyko, Evgeny Osipov, Daswin De Silva, Urban Wiklund, Valeriy Vyatkin, and Damminda Alahakoon. Distributed representation of n-gram statistics for boosting self-organizing maps with hyperdimensional computing. In *Perspectives of System Informatics: 12th International Andrei P. Ershov Informatics Conference, PSI 2019, Novosibirsk, Russia, July 2–5, 2019, Revised Selected Papers 12*, pages 64–79. Springer, 2019. 2

[6] Tony A Plate. *Distributed representations and nested compositional structure*. Citeseer, 1994. 3

[7] Tony A Plate. Analogy retrieval and processing with distributed vector representations. *Expert systems*, 17(1):29–40, 2000. 3

[8] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. A theoretical perspective on hyperdimensional computing. *Journal of Artificial Intelligence Research*, 72:215–249, 2021. 2