Efficient Open Modification Spectral Library Searching in High-Dimensional Space with Multi-Level-Cell Memory

Keming Fan[1*], Wei-Chen Chen[2], Sumukh Pinge[1], H.-S. Philip Wong[2], Tajana Rosing[1*]
[1]University of California, San Diego, [2]Stanford University
*{k4fan,tajana}@ucsd.edu

ABSTRACT

Open Modification Search (OMS) is a promising algorithm for mass spectrometry analysis that enables the discovery of modified peptides. However, OMS encounters challenges as it exponentially extends the search scope. Existing OMS accelerators either have limited parallelism or struggle to scale effectively with growing data volumes. In this work, we introduce an OMS accelerator utilizing multi-level-cell (MLC) RRAM memory to enhance storage capacity by 3x. Through in-memory computing, we achieve up to 77x faster data processing with two to three orders of magnitude better energy efficiency. Testing was done on a fabricated MLC RRAM chip. We leverage hyperdimensional computing to tolerate up to 10% memory errors while delivering massive parallelism in hardware.

CCS CONCEPTS

• **Computing methodologies** → **Symbolic and algebraic manipulation**; • **Hardware** → **Memory and dense storage**.

KEYWORDS

In-memory computing, Hyperdimensional computing

1 INTRODUCTION

Mass spectrometry (MS) analysis is a critical technique for studying proteins, which serve as the fundamental components of modern medicines. In a typical MS experiment, numerous spectra, known as query spectra, are generated and compared with a reference database containing known peptides. Proteins in MS experiments often undergo post-translational modifications (PTM), altering mass and properties. However, the reference database only includes spectra for unmodified peptides. This complicates the search, as many peptides may not find a match.

Open modification search (OMS) offers a promising solution to circumvent this issue by allowing the identification of modified spectra. In the traditional standard search, comparisons are limited to query spectra and reference spectra that share similar precursor mass. In contrast, OMS extends the matching scope to a broader range. It adopts a wide precursor mass window on reference spectra, which accounts for the mass shifts induced by PTMs and other protein modifications. This approach enables the comparison of spectra from modified proteins with their unmodified counterparts, thereby facilitating a more comprehensive analysis.

However, OMS encounters challenges as it vastly expands the search scope, necessitating a more refined design. Specifically, we require (1) dense memory solutions given the exponentially growing size of data, and (2) algorithms that are easily parallelizable in hardware for faster data processing. Several works have accelerated the OMS algorithm. The ANN-SoLo tool [1] uses nearest neighbor indexing to select candidates and employs the shifted dot product to compute scores on those candidates. Nevertheless, ANN-SoLo demonstrates limited data parallelism as it uses complicated high-precision floating-point arithmetic. HyperOMS [12] encodeds input data into high dimensional vectors and performs simple integer operations. This approach results in significant increase in parallelism on GPU architectures. However, both of them do not scale well with the growing data volumes, necessitating high-density memory solutions. In this work, we use dense multi-level-cell (MLC) RRAM to increase the storage capacity. In addition, we employ an in-memory computing approach to reduce data movement, leading to faster data processing. Since MLC RRAM and in-memory computing are usually error prone, we leverage the robustness of hyperdimensional computing (HD) to tolerate these errors. The main contributions of this work are as follows.

- An OMS accelerator using HD and MLC RRAM is proposed. The proposed design achieves 3x better storage capacity per area with comparable accuracy to state-of-the-art, allowing for up to 10% memory error tolerance.
- We accelerate the main stages of the algorithm by processing in memory. The functionality is tested through experiments on a fabricated MLC RRAM chip.
- We propose several hardware-software co-design strategies, including a multi-bit hypervector scheme and an efficient mapping scheme to enhance computational efficiency.

2 RELATED WORK AND MOTIVATION

2.1 Open Modification Search

Mass Spectrometry (MS) is crucial in proteomic research, enabling the analysis of complex biological samples. Open Modification Search (OMS) marks a significant evolution in MS technology, as it allows for the discovery of modified peptides. However, the implementation of OMS is challenging as it significantly enlarges search space. This expanded scope includes both unmodified and modified peptide variants, leading to increased computational demands. HyperOMS, the fastest existing OMS accelerator that operates on GPUs, still faces a challenge with a large memory footprint [12]. This challenge arises from OMS's inherent memory-intensive nature, leading to efficiency concerns and data transfer bottlenecks. Processing in memory enables direct computations within the memory space, offering a better solution for OMS acceleration.

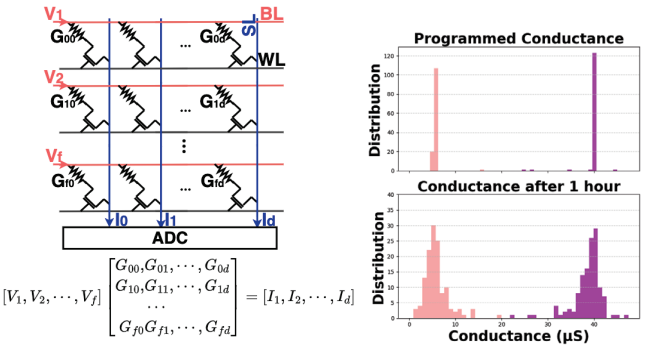
2.2 RRAM and In-memory Computing

Another challenge arises from the escalating data volumes. The public data for mass spectrometry analysis is experiencing exponential

growth [14]; however, existing memory solutions face difficulties in scaling to meet the expanding demands.

Resistive random access memory (RRAM) is an emerging non-volatile memory that stores data by changing its resistance. In TSMC 22nm technology, a single-level-cell (SLC) RRAM provides 3x higher storage capacity per area than high-density SRAM (Static Random Access Memory) [8], positioning it as an optimal solution for extensive data storage requirements.

Despite its density advantage, RRAM can effectively contribute to parallel in-memory acceleration. Prior research has explored the computational power of RRAM, particularly for the MAC (multiply-accumulate) operation that is critical in modern neural networks. Figure 1a shows a typical configuration [18] of 1T1R RRAM array designed for matrix vector multiplication (MVM). Input data is mapped to analog voltages, and RRAM-stored weights are represented by their conductance. Currents are generated across multiple rows and accumulated along the columns, following Kirchhoff's law. The ADC (Analog-to-Digital Converter) then converts currents to digital values, representing the computation outputs.



(a) Crossbar array for MVM **(b) Conductance relaxation**

Figure 1: RRAM

Previous works [6, 20, 21] have primarily utilized SLC RRAM for storage or computation in various applications. In SLC memory, each RRAM device is programmable to low resistance to represent '1' or to high resistance to represent '0'. However, the potential of RRAM extends beyond binary storage, as it can achieve arbitrary analog resistance states by applying different voltages, enabling the storage of multi-bit data. This configuration is commonly referred to as multi-level-cell (MLC) RRAM. However, adopting MLC RRAM, while promising in increasing storage capacity, poses challenges due to device non-idealities. RRAM suffers from conductance relaxation and a relatively low on-off ratio. Figure 1b illustrates the conductance distribution in RRAM collected from a fabricated chip [18] after 60 minutes of programming, displaying a shift in conductance that hinders accurate storage and computing. Some works attempt to harness the potential of MLC RRAM. The authors of [4] use RRAM as a passive array for storage only, without the computing ability. In [13], they proposed an RRAM-based in-memory computing macro, but with only three levels per cell, which doesn't fully use the potential of MLC RRAM. Motivated by these challenges and recognizing the limitations of existing approaches, we propose a robust algorithm using hyperdimensional computing to tolerate errors associated with MLC RRAM.

3 ALGORITHM OVERVIEW

Hyperdimensional computing (HD) is a brain-inspired computing method that emulates neuronal activity. HD encodes information to binary high-dimensional (long) vectors called hypervectors, typically with a dimension of 1k-10k [11]. In this work, we leverage HD for OMS acceleration, benefiting from its high degree of parallelism in hardware implementations and robustness to errors.

Figure 2 illustrates the overall diagram, which includes data preprocessing that turns raw data into spectra vectors. This is followed by HD encoding and hamming search in high-dimensional space (hyperspace). Finally, an FDR filter outputs identified peptides.

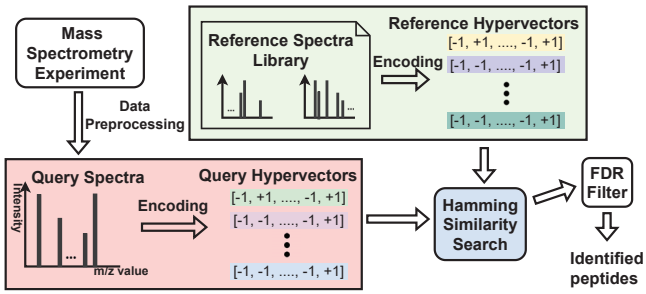


Figure 2: Overall flow

3.1 Data Preprocessing

Preprocessing is the initial step in MS analysis, beginning with the extraction of prominent peak features from raw data. This involves identifying and retaining peaks above a predefined intensity threshold, typically set at 1% of the greatest peak intensity. The goal is to eliminate background noise, resulting in a refined set of 50 to 150 peaks representing the most useful features in each spectrum. Next, spectra are transformed into vectors by categorizing mass-to-charge (m/z) ratios into bins. The resulting vectors contain floating-point values reflecting peak intensities. In cases where multiple peaks fall within a bin, their intensities are summed.

3.2 Encoding

HD encodes spectra vectors into orthogonal binary hypervectors that represent unique features within the spectra. Previous research explored various encoding methods, such as permutation-based [15] and random projection encoding [3]. However, these methods may not effectively capture key features, such as m/z values and peak intensities in the spectra. In this work, we employ an ID-Level encoding method [9] to address this limitation.

During encoding, each peak position (m/z value) is mapped to a hypervector with D dimensions called ID. We quantize the intensity value for each peak to Q levels and assign each a base hypervector denoted as l_j. To generate the Q-level base hypervector, we first create a random binary base-hypervector l_0 with D dimensions, where $l_0 \in \{-1, 1\}^D$. The remaining base hypervectors l_j are generated by flipping $D/(2Q)$ bits of the preceding hypervector l_{j-1}. This approach ensures that neighboring l_j and l_{j+1} pairs maintain more similarity than pairs that are far apart. Previous studies have shown that the number of quantized levels (commonly selected in the range of Q=16~32) does not significantly impact the results for this application [12]. Given two sets of hypervectors, a spectrum

vector is encoded into a hypervector h (see Figure 3) using the following equation.

$$h = Sign\left(\sum_{i \in S} ID_i \otimes LV_i\right) \quad (1)$$

For each peak in spectrum S, we perform element-wise multiplication between the position hypervector ID_i and its corresponding level hypervector LV_i ($LV_i \in \{l_0, \cdots, l_{Q-1}\}$). We then sum the results and apply quantization using the $Sign()$ function to obtain the final binary hypervector h.

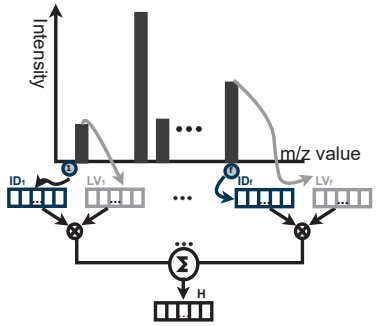


Figure 3: ID-Level Encoding

3.3 Hamming Similarity Search

After encoding, we find the most similar reference hypervector to the query hypervector by calculating their similarity. As all hypervectors are binary, HD replaces the cosine similarity with a simpler Hamming similarity. The Hamming similarity, calculated as the number of equal components in vector pairings, is measured by dot product.

3.4 FDR Filter

The final step includes applying a false discovery rate (FDR) filter, a widely used method for MS analysis. It introduces non-existing decoy spectra into the spectral library. The filter then sifts through decoy spectra selected by the search tool. The performance of different search tools can be compared under a fixed FDR threshold.

4 HARDWARE ACCELERATION

While data preprocessing is typically done offline, encoding and search dominate the algorithm's runtime (over 90%) and they are memory-intensive tasks that best done in memory.

In the data flow, encoding is the first step conducted in memory, followed by storing encoded hypervectors in memory. Subsequently, a similarity search is performed. However, in this section, we introduce our optimization methods in the reverse order, focusing on the acceleration of the search before delving into the encoding. This decision is due to the encoding's dependence on knowledge introduced during the search acceleration.

4.1 In-memory Hamming Similarity Search

The basic operation for hamming similarity search is the MVM between the query hypervector and the reference hypervectors, which can be efficiently accelerated in memory. During the search process, we store each reference hypervector (weight) vertically. In each cycle, a query vector (input) is fed into the array as analog voltages (see Figure 4a).

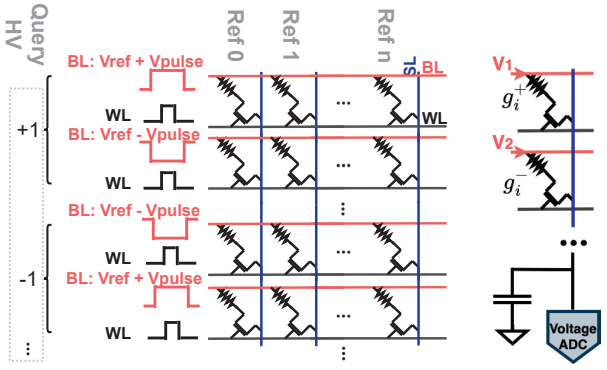
4.1.1 Weight Mapping. We use a differential weight mapping scheme, where two cells in a pair together store one number. Compared to the non-differential version, it offers a better solution to challenges arising from non-linearities, such as residual current, RRAM resistance mismatches, and the on-resistance of peripheral switches.

The stored weights in RRAM are encoded reference hypervectors. A differential pair, comprising two cells in adjacent rows within the same column, together stores one dimension value W. Their conductance g_i^+ and g_i^- are programmed to the opposite value with respect to the middle level $g_{max}/2$ as follows.

$$g_i^+ = \frac{1}{2}(1 + W/W_{max})g_{max} \quad (2)$$

$$g_i^- = \frac{1}{2}(1 - W/W_{max})g_{max} \quad (3)$$

4.1.2 Sensing Scheme. Conventional current sensing [6, 21] suffers from limited throughput due to constraints in concurrent row driving and increased energy consumption resulting from static current during sensing. To mitigate these effects, we apply open-circuit voltage sensing [18] with a differential scheme.



(a) In-memory Search **(b) Sensing**

Figure 4: RRAM for MVM

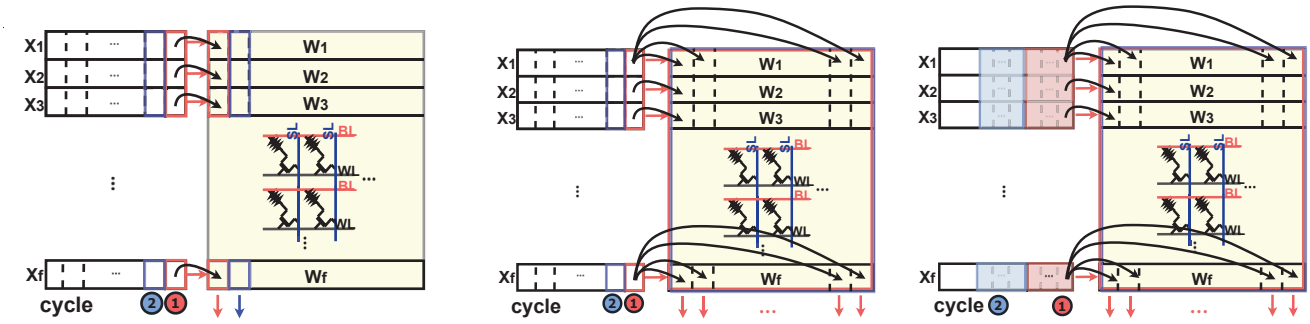
During MVM, the input (query) hypervector is simultaneously transmitted through differential BL voltages to represent signed inputs (Figure 4a). All activated rows contributing to the MAC output, numbering N, receive a high signal from the WL. The resulting currents are collected by the capacitor, generating a voltage on the SL according to the following equation.

$$C\frac{dV_{SL}}{dt} = \sum_{i=0}^{N}(V_{ref} + V_{pulse} \cdot X_i - V_{SL}) \cdot g_i^+ + \sum_{i=0}^{N}(V_{ref} - V_{pulse} \cdot X_i - V_{SL}) \cdot g_i^- \quad (4)$$

Once the SL voltage reaches the steady state,

$$V_{SL} = V_{ref} + \frac{\sum_{i=0}^{N} X_i \cdot (g_i^+ - g_i^-)}{N g_{max}} \cdot V_{pulse} \quad (5)$$

the ADC then converts the voltage into a digital output. According to Equation 5, the resulting output voltage demonstrates a linear relation with the expected MAC output.



(a) Element-wise MAC in RRAM (b) MVM in RRAM (c) Proposed encoding in RRAM

Figure 5: Encoding in RRAM

4.1.3 Errors from Memory. HD exhibits robustness to errors as a result of the the distributed and associative nature of hypervectors. Matched patterns have a significantly higher similarity than unmatched pattern pairs. This substantial difference in similarity acts as a form of tolerance to memory errors.

4.2 Encoding In Memory

Encoding maps input data into hypervectors. The key operation in encoding (see Equation 1) is a MAC, which can be accelerated in memory. However, encoding involves element-wise operations, which are challenging to accelerate as effectively as MVM in memory. Based on these observations, optimizing the encoding processes within the memory domain is important.

We use the same weight mapping as in search to convert hypervectors to RRAM conductance, as well as the same differential sensing scheme. RRAM can conduct MAC operations, which involve multiplication of the input and weight on the same row and addition vertically inside a column. Due to constraints on the orientation of these operations, each position hypervector ID (weight W) is stored horizontally, while multiple level hypervectors LV (inputs X) are simultaneously fed into the array bit by bit.

4.2.1 Efficient Encoding. However, the memory array is less proficient in element-wise operations as compared to MVM. In MVM, during each cycle when inputs are applied, multiple columns are activated, and each of them generates one MAC output, resulting in multiple MACs per cycle (see Figure 4a and 5b). On the other hand, in element-wise operations, only one corresponding output from the array is valid in each cycle (see Figure 5a). Consequently, we need more cycles to finish the same amount of MAC computation. Another challenge arises from the fact that, even though only one column of output is valid, it is not possible to selectively activate cells in that specific column. In the memory array, BL and WL are shared by all cells on the same row. When input comes, all cells on that row are driven, consuming power, even though some of them are generating unnecessary results. This makes it more power-hungry compared to MVM.

Based on this observation, we aim to transform the element-wise operation into an MVM-style to enhance throughput and energy efficiency. This is achieved by modifying the way we generate inputs (level hypervectors). Given that the number of base LV hypervectors (Q=16~32) is much less than the hyperspace size, the choice of LV hypervectors has minimal impact on final results. Instead of randomly generating a base LV hypervector, the D-bit hypervector is divided into several chunks, where all bit values within each chunk are identical. Instead of feeding the entire D-bit LV hypervectors into the array bit by bit, we now do it chunk by chunk since all values inside one chunk are the same. The number of chunks should be selected based on algorithm-related factors such as HD dimension, the number of base LV hypervectors, as well as hardware-related factors including array size, and the column-sharing arrangement for ADCs. Overall, this modification allows all element-wise MAC outputs within one chunk to be obtained in a single cycle, resembling the MVM fashion.

4.2.2 Multi-bit Hypervector. In prior HD research [10, 12, 16, 20], input data is represented by binary hypervectors for further use. However, inspired by the fact that MLC hardware has the capacity to store multi-bits per cell, and considering that synapses in the brain have 4.7-bit precision [2], there arises a potential that using a multi-bit hypervector scheme could produce better performance.

Instead of generating a binary hypervector $ID \in \{-1, 1\}^D$, we apply a multi-bit approach. Each dimension of the ID hypervector could be up to 3 bits, for example, $ID \in \{-4, -3, -2, -1, 1, 2, 3, 4\}^D$. Additionally, this adaptation introduces no additional hardware cost, while raising the bit precision in input level hypervectors LV would increase the overall number of cycles for processing, since inputs are fed into the array in a bit-serial fashion.

4.2.3 Errors from Memory. During encoding, final outputs are quantized to binary using the $Sign()$ function, requiring only a low-resolution MAC output. This tolerates errors from memory cells, for example, a single bit flipping would not significantly affect the output. It also reduces ADC design requirements, minimizing computing errors introduced by the ADC.

4.3 Hypervector Storage

After encoding, we stored the encoded hypervectors in memory. In MLC RRAM, each cell can exhibit 2^n levels of conductance, allowing it to store n-bit data per cell ($n = 1, 2, 3$). Reference hypervectors used for computation in later stages (Hamming search) are stored in a differential manner, as shown in the previous section. However, to maximize storage capacity, we store all query hypervectors using the following non-differential method. We reshape the D-bit hypervector into segments of n-bits each, forming a new D/n-bit vector denoted as h', and map them to unsigned integer values accordingly. These h' are then further mapped onto the RRAM

conductances g. The following example illustrates how to store a hypervector within 4-level-cell RRAM (n=2 bits per cell).

Binary value to store $h \in \Leftrightarrow$	Integer $h' \in \Leftrightarrow$	Conductance g
$\{(-1,-1),\cdots,(1,1)\}^{D/2}$	$\{0,\cdots,3\}^{D/2}$	$g = h'/h'_{max} * g_{max}$

5 EVALUATION

5.1 Experiment Setup

5.1.1 MLC RRAM Chip Measurement. We tested our algorithm on a fabricated MLC RRAM chip [18] in 130nm technology with a total of 3 million RRAM cells. The Xilinx FPGA integrated on an Opal Kelly XEM6310 module serves as the communication bridge between host computer and the chip (Figure 6). The chip loads/writes the input/output hypervectors from/to off-chip text files.

5.1.2 OMS Workload and Benchmark. We evaluate the design using two real-world datasets. The first dataset uses iPRG2012 (16k spectra in total) [5] as query and human HCD yeast library (1M spectra in total) [17] as reference. The second dataset uses HEK293 b1906 (47k spectra in total) [7] as query and human spectral library (3M spectra in total) [19] as reference. We compare our result against two state-of-the-art OMS work, ANN-SoLo [1] on CPU/GPU and HyperOMS [12] on GPU. Baseline benchmarking is conducted on the NVIDIA GeForce RTX 4090 GPU and the Intel Core i7-11700K CPU, respectively. Parameters for data preprocessing and FDR filtering are presented in Table 1.

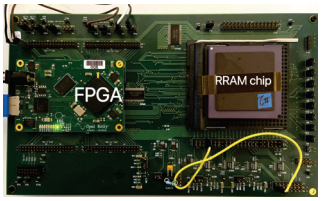


Figure 6: MLC RRAM chip setup

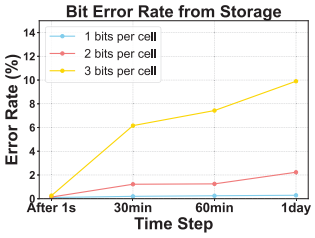


Figure 7: Storage errors

5.2 MLC RRAM Measurement

5.2.1 RRAM for hypervector storage. RRAM relaxation predominantly occurs at the initial stages, so collecting data after 1 or 2 days does not significantly matter. Therefore, we collect data right after programming, after 30 minutes, 60 minutes, and 1 day, respectively. Figure 8 illustrates the histogram of the conductance, from which we compute the bit error rate data in Figure 7 based on the mapping method described in Section 4.3. Our design can store up to 3 bits per cell, leading to a 3x improvement in storage capacity.

In the subsequent sections, all data are collected at least 2 hours after programming to account for RRAM relaxation effects.

5.2.2 RRAM for computing. For encoding, we compare the binary outputs from RRAM with the corresponding ground truth binary values to calculate bit error rate. In the case of in-memory hamming search, as the output consists of integer numbers rather than binary, we report the normalized mean square error. Figure 9 illustrates the error rates with 1/2/3 bits storage per cell, corresponding to 2/4/8 level MLC cells, respectively. With an increasing number of activated rows, we achieve higher throughput but experience more computation errors.

Table 1: OMS workload settings

Dataset	iPRG2012	HEK293
number of query spectra	16k	47k
number of reference spectra	1M	3M

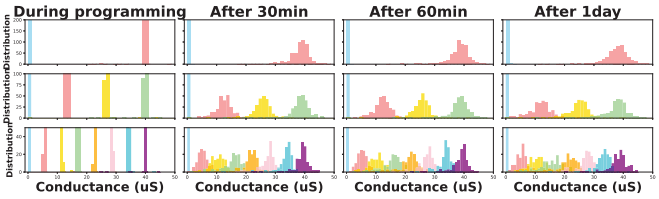
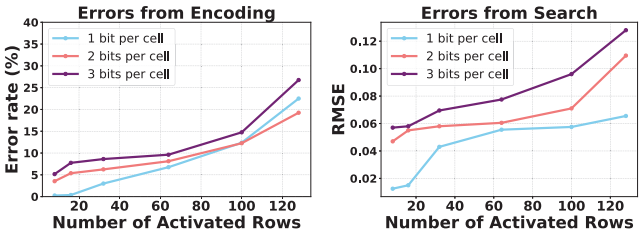


Figure 8: Conductance relaxation effect of 1/4/8-level RRAM



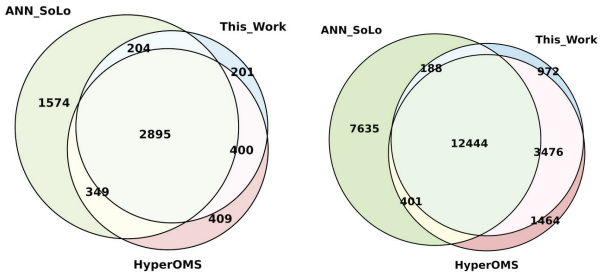
(a) Encoding errors **(b) Search errors**

Figure 9: Computation errors

Compared with the state-of-the-art MLC RRAM design for in-memory computing [13], which can only drive a maximum of 4 rows with 3-level RRAM, our design can activate up to 64 rows (use this setting in following section) with 8-level RRAM, indicating an 16x increase in throughput along with greater storage capacity. The performance gain is due to improvements in RRAM device, design strategy, and benefits from robust HD.

5.3 OMS results

5.3.1 Search quality. Biological data analysis is complicated, and there is no ground truth data for the search results. Consequently, we compare our results with existing tools. We set the dimension to be 8k with an *ID* hypervector precision of 3 bits. Figure 10 shows the comparisons of the identified peptides. It indicates that the majority of the identified peptides from our work align with those identified by other tools, indicating the validity of our results.



(a) iPRG2012 dataset **(b) HEK293 dataset**

Figure 10: Venn Diagram of identified peptides

5.3.2 HD robustness. With HD dimension being 8k, we introduce varying levels of bit error rates for encoding and search into the HD algorithm. Figure 11 shows that our design can tolerate up to 10% errors. Another notable finding is the enhanced performance achieved through the utilization of multi-bit hypervector scheme.

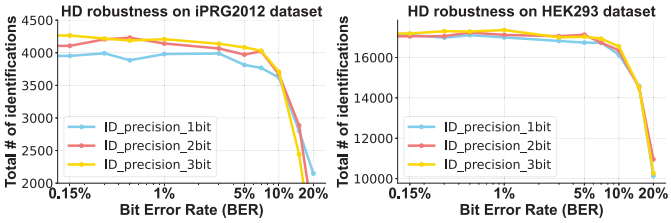


Figure 11: HD robustness

5.3.3 Speedup and Energy Improvement. We simulated the speedup and energy efficiency improvement on iPRG2012 dataset. Our work exhibits 1.7x faster than HyperOMS on GPU, 24.8x/76.7x than ANN-SoLo on GPU/CPU, with 500x-3000x more energy efficiency than state-of-the-art tools (see Figure 12). And we expect our performance to scale with more advanced CMOS technology.

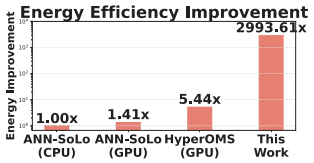


Figure 12: Energy Efficiency

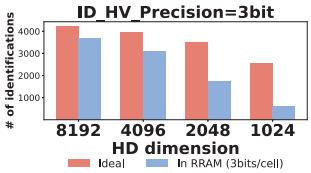


Figure 13: HD Dimensions

5.3.4 HD dimension. The HD dimension is a key factor that impacts final results. Lower dimension is more sensitive to noise and exhibits limited separability (see Figure 13). However, an excessively high dimension introduces more computation resource requirements, emphasizing the need for a balanced consideration for target applications.

6 CONCLUSION

In this paper, we propose an accelerator for open modification library searching. We use multi-level-cell RRAM to increase storage capacity by 3x, along with a robust hyperdimensional computing algorithm that can tolerate up to 10% errors from memory. We accelerate the main stages by computing in memory, leading to 1.7x-76.7x faster processing and 500x-3000x energy efficiency improvement. The functionality of our accelerator has been successfully verified on a fabricated RRAM chip. Although our accelerator focuses on mass spectrometry applications, the ideas of robust HD and hardware acceleration techniques have the potential for broader applications beyond the realm of mass spectrometry.

ACKNOWLEDGMENTS

This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA, TSMC, and NSF grants 2003279, 1826967, 1911095, 2052809, 2112665, 2112167, and 2100237.

REFERENCES

[1] Issar Arab et al. 2023. Semisupervised Machine Learning for Sensitive Open Modification Spectral Library Searching. *Journal of Proteome Research* (2023).
[2] Thomas M Bartol Jr et al. 2015. Nanoconnectomic upper bound on the variability of synaptic plasticity. *elife* 4 (2015), e10778.
[3] Timothy I Cannings et al. 2017. Random-projection ensemble classification. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 79, 4 (2017), 959–1035.
[4] Alex Carsello et al. 2022. Amber: A 367 GOPS, 538 GOPS/W 16nm SoC with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 70–71.
[5] Robert J Chalkley et al. 2014. Proteome informatics research group (iPRG) _2012: a study on detecting modified peptides in a complex mixture. *Molecular & Cellular Proteomics* 13, 1 (2014), 360–371.
[6] Wei-Hao Chen et al. 2018. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 494–496.
[7] Joel M Chick et al. 2015. A mass-tolerant database search identifies a large proportion of unassigned spectra in shotgun proteomics as modified peptides. *Nature biotechnology* 33, 7 (2015), 743–749.
[8] Chung-Cheng Chou et al. 2020. A 22nm 96KX144 RRAM macro with a self-tracking reference and a low ripple charge pump to achieve a configurable read window and a wide operating voltage range. In *2020 IEEE Symposium on VLSI Circuits*. IEEE, 1–2.
[9] Mohsen Imani et al. 2017. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE international conference on rebooting computing (ICRC)*. IEEE, 1–8.
[10] Mohsen Imani et al. 2019. Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *Proceedings of the 56th Annual Design Automation Conference 2019*.
[11] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* (2009).
[12] Jaeyoung Kang et al. 2022. Massively Parallel Open Modification Spectral Library Searching with Hyperdimensional Computing. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*.
[13] Wantong Li et al. 2022. A 40-nm mlc-rram compute-in-memory macro with sparsity control, on-chip write-verify, and temperature-independent adc references. *IEEE Journal of Solid-State Circuits* 57, 9 (2022), 2868–2877.
[14] Perez-Riverol et al. 2022. The PRIDE database resources in 2022: a hub for mass spectrometry-based proteomics evidences. *Nucleic acids research* 50, D1 (2022), D543–D552.
[15] Sahand Salamat et al. 2019. F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.
[16] Sahand Salamat et al. 2020. Accelerating hyperdimensional computing on fpgas by exploiting computational reuse. *IEEE Trans. Comput.* 69, 8 (2020), 1159–1171.
[17] Nathalie Selevsek et al. 2015. Reproducible and consistent quantification of the Saccharomyces cerevisiae proteome by SWATH-mass spectrometry. *Molecular & Cellular Proteomics* 14, 3 (2015), 739–749.
[18] Weier Wan et al. 2022. A compute-in-memory chip based on resistive random-access memory. *Nature* 608, 7923 (2022), 504–512.
[19] Mingxun Wang et al. 2018. Assembling the community-scale discoverable human proteome. *Cell systems* 7, 4 (2018), 412–421.
[20] Weihong Xu et al. 2023. FSL-HD: Accelerating Few-Shot Learning on ReRAM using Hyperdimensional Computing. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.
[21] Cheng-Xin Xue et al. 2019. 24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 388–390.