Evolvable Hyperdimensional Computing: Unsupervised Regeneration of Associative Memory to Recover Faulty Components

Author(s):
Hersche, Michael; Sangalli, Sara; Benini, Luca; Rahimi, Abbas

Publication date:
2020

Permanent link:
https://doi.org/10.3929/ethz-b-000387115

Originally published in:
https://doi.org/10.1109/AICAS48895.2020.9073871

Funding acknowledgement:
780215 - Computation-in-memory architecture based on resistive devices (EC)

Evolvable Hyperdimensional Computing: Unsupervised Regeneration of Associative Memory to Recover Faulty Components

Michael Hersche, Sara Sangalli, Luca Benini, Abbas Rahimi
Integrated Systems Laboratory, ETH Zurich, Switzerland
Emails: sansara@student.ethz.ch, {hersche, benini, abbas}@iis.ee.ethz.ch

***Abstract*—This paper proposes evolvable hyperdimensional (HD) computing to maintain high classification accuracy as permanent faults occur in emerging non-volatile memory fabrics. Our proposed HD architecture can detect, localize, and isolate faulty PCM blocks in discriminative classifiers, followed by unsupervised regeneration of new blocks to compensate accuracy loss. We demonstrate its application on a language recognition task: it is able to quickly relearn and fully recover the accuracy from 90.48% to 96.86% at fault rates as high as 42% by using solely 4.2 MB of text for regeneration. The new evolved model is still 285× more compact than state-of-the-art fastText.**

***Index Terms*—Evolvable hardware, HD computing, PCM.**

I. Introduction

Emerging memory technologies such as phase-change memory (PCM), resistive RAM (RRAM) with monolithic 3D integration can provide efficient computing fabrics for AI by naturally implementing neurons, or synapses, and by eliminating the von Neumann memory-wall bottleneck [1]–[3]. These computing fabrics however face challenges such as permanent failures, defects, variations, and noise. Novel computational paradigms that are inherently robust, distributed, and modular with fast learning capabilities could come to rescue.

One viable option is to exploit hyperdimensional (HD) computing [4] that is inspired by very size of the brain's circuits to model *neural activity patterns* with points of an HD space, that is, with high-dimensional vectors. These vectors can be manipulated using well-defined vector space operations inside an *encoder*, and can be compared by an associative memory (AM) using Hamming distance to solve cognitive [5]–[7] and classification [8]–[11] tasks. When the dimensionality is in the thousands, operations on these vectors create a computational behavior with unique features in terms of robustness and simplicity of operations paving the way for efficient realization in low SNR nanoscalable fabrics [12]. For instance, in the EU language recognition task [8], HD computing degrades very gracefully in the presence of faults in memory compared to a k-nearest neighbors classifier: by injecting the intermittent errors in both classifiers, HD computing tolerates 8.8× higher probability of failure per individual memory cells [9]; considering the permanent hard errors in RRAMs, HD computing tolerates 60× higher probability of failures [13]. Further, using 760,000 PCM devices in large cross-bar arrays to implement the same recognition task, HD computing tolerates the spatial and temporal variabilities of PCM devices reaching to comparable accuracies to software [3].

This graceful degradation under variability is due to inherent robustness of vector representation and related operations. Representation with vectors begins with independent and identically distributed (i.i.d.) components and when combined with the vector space operations, the resulting vectors also appear as identically distributed random vectors, and the independence of the individual components is mostly preserved [9], [12]. Hence, a faulty component of a vector is not *contagious*, and the rest of error-free components can provide a useful representation below a certain fault rate. For instance, when the number of faulty RRAM components in AM goes above 10%, the language accuracy starts to decline [13]. In this paper, we aim to go beyond this robustness, by isolating those components and regenerating new components without any supervision: *enabling HD architecture to regrow its lost parts.*

Accordingly, we propose an evolvable HD computing architecture to detect, localize, isolate, and recover (by means of regeneration) from faulty PCM hardware blocks—all together in an unsupervised manner: 1) The presence of faults is *detected* non-intrusively in a discriminative HD classifier by analyzing the standard deviation of the Hamming distances between different classes. 2) Partial Hamming distances are computed on a block that help *localizing* and *isolating* erroneous blocks via K-means clustering (K=2) of relative sum of Hamming distances. 3) Finally, new HD blocks are *regenerated* by replicating encoder blocks, and by retraining the contents of AM to compensate for isolated faulty blocks. We demonstrate its application on the 21 EU language recognition task using a simulated PCM architecture [3]. Experimental results show that the classification accuracy can be fully recovered from 90.48% to 96.86% at fault rates of up to 42% in AM. Moreover, at highest fault rate of 48.5% the accuracy of the faulty hardware is recovered from 75.50% to 95.05%. This overall maintains HD as a compact model (285× fewer trainable parameters than fastText [14]) with ability to quickly relearn and recover from faults without any supervision.

II. HD Computing for Learning and Classification

Here, we briefly describe how HD computing can be used in learning and classification tasks. We focus on HD architecture for widely-used language recognition task [2], [3], [8], [9],

This project was supported in part by EU's H2020 under grant no. 780215 and by ETH Research Grant 09 18-2.

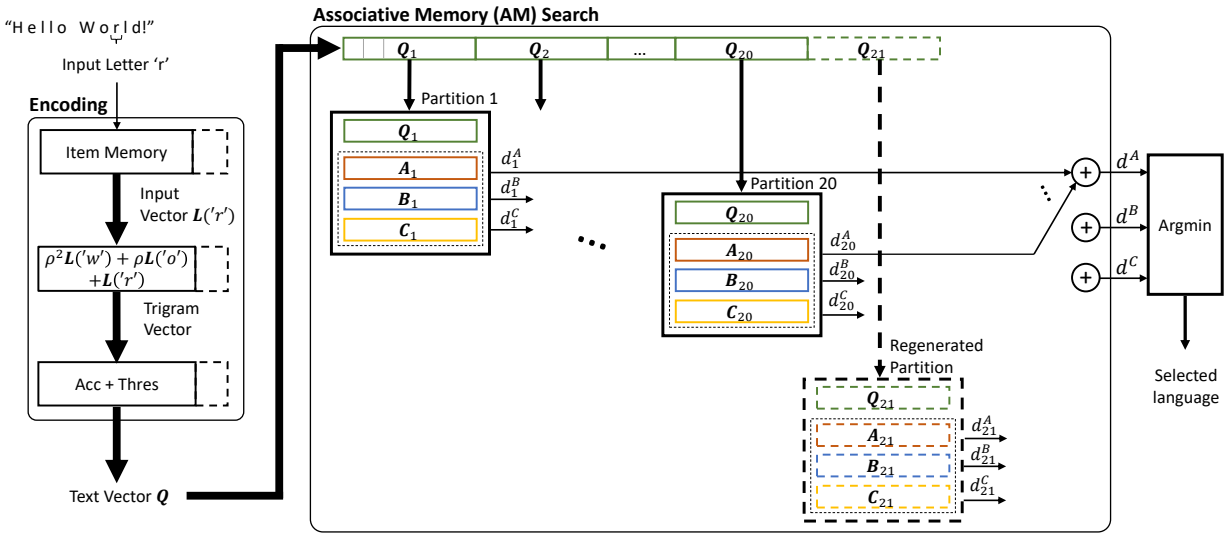

Fig. 1: Architecture for three class language classification consisting of encoding of text letter into binary text vectors and AM [9]. Query vector $\mathbf{Q}$ is divided into 20 sub-query vectors $\mathbf{Q}_1, \mathbf{Q}_2, ..., \mathbf{Q}_{20}$, which are fed to their corresponding partition block. Encoder and AM can be extended with additional partition blocks (dashed) in presence of faults.

[13]; this architecture with our method can be applied to other applications [12] such as news [10], or embedded biosignal [11] classifications.

Encoding and AM are the two main modules. The encoding module embeds an input text, composed of a stream of letters, to a HD binary vector (10,000-bit). This module includes an item memory that holds a random, binary vector for each letter of the Latin alphabet and the space. The encoder computes a vector for every three consecutive letters (i.e., trigram) as the text streams in. A trigram vector is created by successively permuting the letter vectors based on their order and binding them together, which creates a unique representation for each trigram. The trigram vectors are superposed (i.e., added and then thresholded) over the input text to generate a binary text vector as the output of encoder (see Fig. 1). This text vector is sent to AM for training, or testing. During training, AM uses the text vector to store a set of language vectors based on provided labels. For instance, AM includes 21 language vectors, each stored in its own row in AM, when trained with the 21 EU languages dataset. During testing, the language of an unknown text is determined by comparing its text vector, called *query vector*, to all language vectors. The comparison between prototypes and query vector is done using AM search with Hamming distance. Finally, AM returns the label of language that has minimum Hamming distance to query [9].

Overall, the encoder embeds the input sentence into a HD vector, followed by the AM to integrate and store these vectors during learning, and make comparison during inference. This architecture is somewhat similar to fastText [14] where the input sentence is hashed to a large feature vector (600,000-d), followed by a fully connected hidden layer (100-d) and the final classification layer (21-d). fastText is an efficient baseline for text classification, on par with deep learning in accuracy, but many orders of magnitude faster for training [14].

III. Evolvable HD Computing

This section presents the main contribution of this paper by proposing an evolvable algorithm, exploiting the features of HD computing, to detect, localize, isolate, and recover faulty components without any supervision. Fig. 1 shows the overall architecture composed of the encoding and AM for classifying a text. For sake of simplicity, we show an example of only three languages (A,B,C); the extension to the 21 languages is straight forward. As discussed, the encoder encodes a text to a text vector, or query ($\mathbf{Q}$) by superposing all trigram vectors of the text. For classification in AM, the query vector is divided into 20 tiles and fed to the corresponding partition block, which stores a tile of every language prototype vector. Every partition block i computes the Hamming distance between the tile of the query vector and all prototypes, giving d_i^A,d_i^B, and d_i^C. Finally, the partial Hamming distances from every language are summed up to d^A,d^B, and d^C; the selected language is the one with smallest Hamming distance. These partial Hamming distances are the only observations that our algorithm requires which is aligned with the nature of computing Hamming distance in a distributed fashion among multiple HD processors [15].

We only consider faults occurred during learning or inference in the AM because it is the common module in HD architecture across all different applications [12], [15], and grows rapidly with increasing number of items for classification; further, it is updated incrementally while the encoder is a fixed entity. Our algorithm can be deployed anytime after HD training by using the following three main steps. Given a faulty AM: 1) Our algorithm non-intrusively senses the presence of faults through the analysis of the distribution of Hamming distances; 2) It then localizes the erroneous blocks of AM via K-means clustering and isolates them from execution; 3) Finally, it reproduces the architecture in a completely unsupervised way to compensate for isolated faulty blocks.

This overall recovers the accuracy loss, and gets closer to the non-faulty model.

A. Detection

The presence of faults in AM is sensed non-intrusively by analyzing the standard deviation between the Hamming distances

$$\sigma = \text{std}\left(d^A, d^B, d^C\right). \quad (1)$$

Intuitively, a discriminative HD classifier separates different classes well, i.e., the distance to the "true" class is significantly lower than the distances to all remaining "wrong" classes. As a result, the standard deviation σ of the discriminative classifier is high. If the discriminability of the classifiers decreases, e.g., due to errors, the distance of the "true" class gets closer to the remaining distances resulting in a lower standard deviation. In our experiments, σ varied between 6.5 and 0.09 when injecting errors with fault rates in the range of 0% and 49.5%. Therefore, σ between Hamming distances is a useful measure to estimate the reliability of HD architectures. Before making a decision about the presence of faults, standard deviations of multiple test texts are averaged to $\overline{\sigma}$. The architecture is considered faulty if $\overline{\sigma} < 4$, corresponding to a fault rate higher than 20%.

B. Localization of Faults

The next step is to localize and isolate faulty partitions in an unsupervised way. Basically, we analyze the sum of Hamming distance of every partition block i:

$$s_i = d_i^A + d_i^B + d_i^C. \quad (2)$$

This sum represents the fraction of Hamming distance every block contributes, independent of the class. The i.i.d. and holographic representation of query and prototype vectors ensure that all partitions contribute *equally* to the total Hamming distance. Formally speaking, in a non-faulty architecture all relative sum of distances are similar:

$$r_i = \frac{s_i}{\sum_{k=1}^{20} s_k} \approx \frac{1}{20} \quad i = 1, 2, ..., 20. \quad (3)$$

As soon as one ore multiple partition blocks become faulty, their relative distance changes. In order to discriminate faulty partition blocks from non-faulty ones, the set of relative sum of distances $\{r_1, r_2, ..., r_{20}\}$ are clustered using simple K-means with K=2. Again, the relative distances are first averaged over multiple test text samples to $\overline{r}_i$. Once localized, a faulty block i is isolated by ignoring its Hamming distances d_i^A, d_i^B, d_i^C.

C. Regeneration

The localized faulty partition blocks are replaced by provided new blocks. Regeneration in the encoder is simply done by replicating new blocks analogous to crossover operator in the genetic algorithms: the item memory is randomly initialized for the new block and the operations used in encoder (permutation, binding, and superposition) are extended to work on the new block. However, the regeneration of new blocks in AM requires new contents to be written in those blocks: we trained them unsupervised by only relying on current labels estimated by the classifier. In this state, the faulty partition blocks are already isolated. The encoder is then extended by the dimension corresponding to isolated partition blocks. A new partition block is trained on the extended dimension by accumulating the query vector per class in accordance to the estimated label from the remaining blocks. At the end of regeneration, a threshold is applied on the accumulated vectors to obtain binary prototype vectors. Finally, the "old" kept blocks and the "new born" blocks of AM are assembled together to form a crossbar AM with the same dimension as the original one.

IV. Experimental Results

A. Experimental Setup

Language recognition dataset. We evaluate our method on the 21 EU languages using the Wortschatz Corpora [16] for training, detection, localization, regeneration and validation and the Europarl Parallel Corpus [17] for testing. Text samples from the Wortschatz Corpora containing a total of 21 MB are split up into initial training set (4.2 MB), detection and localization set (1.05 MB), and regeneration task (4.2 MB). The remaining text samples of the Wortschatz Corpora are used as validation set for tuning hyperparameters of the model. The Europarl Parallel Corpus consists of 21 MB texts which are used for testing.

Architecture. Our architecture is based on an in-memory HD computing architecture using PCM devices [3]. The $10,000$-bit vectors inside AM are mapped to a PCM crossbar for Hamming distance computation. As the vector components are binary, a PCM device is programmed to the maximum/minimum conductance level to represent a vector component with logic-1/0. The AM crossbar is split into 20 partition blocks with vector dimension 500 each.

Faults simulation. The PCM devices are programmed in a single-shot, i.e., with a single reset/set pulse aiming for minimum/maximum conductance levels (without iterative program-and-verify) [3]. Further, these conductance values drift due to spatial and temporal variabilities. Specifically, the variability in cells programmed to logic-1 is higher making a 1-to-0 bit flip more probable than a 0-to-1 bit flip. A 1-to-0 bit flip is occurred when the conductance level of the cell falls under a certain threshold. We injected such errors into the AM based on the measured spatial conductance variability of $21 \times 10,000$ PCM devices in a crossbar array [3]. In our fault simulations, these conductance variabilities result in a fault rate (FR) from 8% to 50%.

B. Impact of Faults in HD and fastText

Fig. 3 shows the degradation in accuracy when introducing faulty elements into both HD and fastText. In HD, we inject the faults in AM, and in fastText in the fully connected layers by setting weights to zero; we exclude HD encoder and fastText hashing due to their fixed immutable nature. Before fault simulations, both classifiers are either trained with initial training set of 4.2 MB or the full Wortschatz Corpora with 21 MB text size.

We first compare the accuracy without injecting faults, i.e., at fault rate of 0%: fastText achieves 97.62% accuracy when

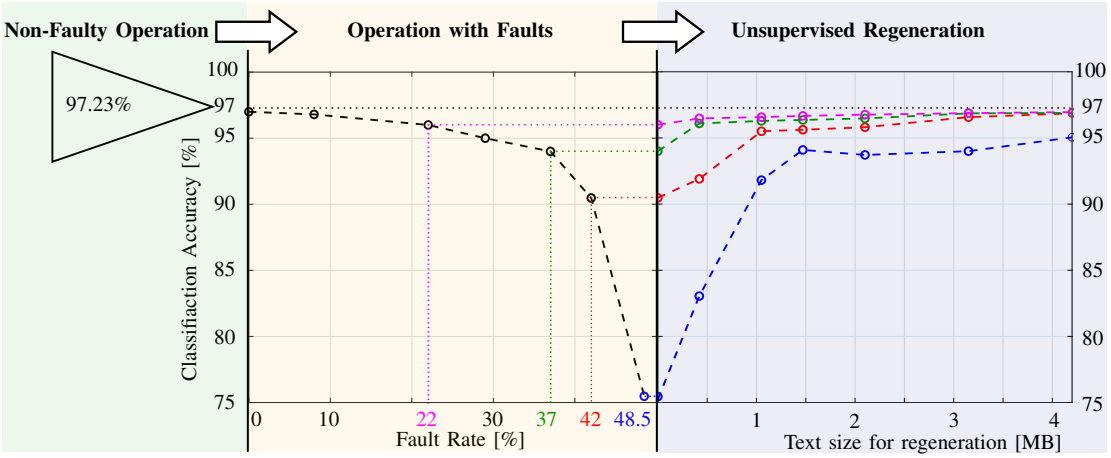

Fig. 2: Summary of classification accuracy of HD classifier in initial non-faulty operation (97.23%) trained with 4.2 MB, operation with faults in AM, and during unsupervised regeneration. Regeneration is shown for four fixed fault rates (22%, 37%, 42%, and 48.5%).

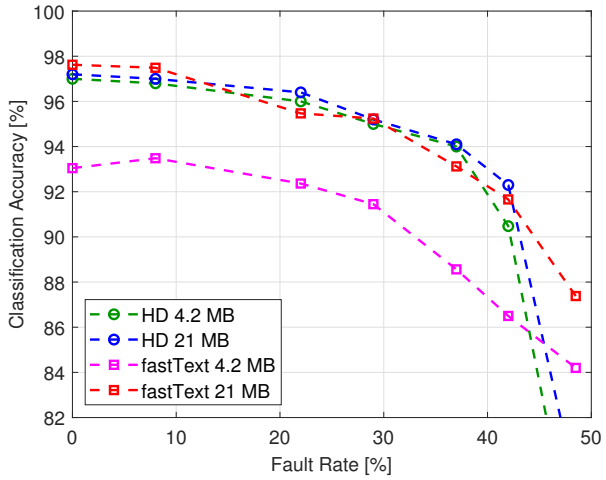

Fig. 3: Classification accuracy of HD classifier and fastText [14] depending on the fault rates. Classifiers are either trained on full training set (21 MB) or on initial training set (4.2 MB). At highest fault rate of 48.5%, HD accuracy is 75.50% (4.2 MB) and 79.00% (21 MB).

trained with the full training set, which is 0.39% higher than HD with 97.23%. When reducing the training set size to 4.2 MB, HD shows its fast learning capabilities and remains the accuracy at 97.00%, whereas fastText decreases by 4.58% due to the large number of weights to be learned.

Both HD and fastText degrade similarly at fault rates up to 42% with a loss of ≈6%. However, at highest fault rate of 48.5%, the accuracy of HD decreases more rapidly with 18.20% loss compared to fastText with 10.24% loss when using the full training set, and by 21.50% compared to 9.04% loss using the reduced training set. This is due to massive redundancy in fastText: its fully connected layers have 285× larger number of trainable parameters than AM.

C. Regeneration of Associative Memory (AM)

HD provides a very compact model compared to fastText but still exhibits graceful degradation under faults. Here, we evaluate the impact of unsupervised AM regeneration that enables HD to recover from high fault rates. Fig. 2 summarizes the accuracy of all three main stages: 1) Accuracy of the original non-faulty architecture trained on initial 4.2 MB training set; 2) Accuracy under each faulty setup for fault rates up to 48.5%; 3) The remaining 4.2 MB of training set is used to regenerate new blocks for AM where we report the accuracy of newly assembled architecture with different amount of text used during regeneration. We have not tried to retrain fastText because of its poor performance with small size of training dataset (see Fig. 3), and the fact that its training algorithm requires iterations that render it unsuitable for online adaptation (time and memory required to pass over training examples) as opposed to HD learning that is done by a single-pass over data.

As mentioned earlier, HD accuracy decreases by 21.50% at a fault rate of 48.5%. When regenerated with the full 4.2 MB, the accuracy is recovered by 16.02% to 95.05%, which is only 2.18% lower than in initial non-faulty architecture. At lower fault rates of 22%, 37%, and 42%, accuracy is fully recovered to 96.86%.

V. Conclusions

We propose evolvable HD computing that can detect and localize faults in PCM crossbar by analyzing statistics of Hamming distances during operation. As soon as the faulty parts are localized and isolated, new parts are regenerated in an unsupervised way. Experimental results on classification task of 21 EU languages show that HD classifier exhibits similar graceful degradation as fastText while requiring 285× lower number of parameters. Moreover, up to fault rates of 42% (or, 48.5%), new HD hardware blocks can be regenerated to recover the accuracy loss reaching to the same (or, 2% lower) accuracy than the non-faulty architecture at the same size.

REFERENCES

[1] N. K. Upadhyay, H. Jiang, Z. Wang, S. Asapu, Q. Xia, and J. Joshua Yang, "Emerging memory devices for neuromorphic computing," *Advanced Materials Technologies*, vol. 4, no. 4, p. 1800589, 2019.

[2] T. F. Wu, H. Li, P. C. Huang, A. Rahimi, J. M. Rabaey, H. S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 492–494.

[3] G. Karunaratne, M. L. Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *arXiv preprint arXiv:1906.01548*, 2019.

[4] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[5] B. Emruli, R. W. Gayler, and F. Sandin, "Analogical mapping and inference with binary spatter codes and sparse distributed memory," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug 2013, pp. 1–8.

[6] P. Kanerva, "What we mean when we say "what's the dollar of mexico?": Prototypes and mapping in concept space," in *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*, 2010, pp. 2–6.

[7] S. Slipchenko and D. Rachkovskij, "Analogical mapping using similarity of binary distributed representations," *J. Information Theories and Applications*, vol. 16, pp. 269–290, 01 2009.

[8] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers*, J. A. de Barros, B. Coecke, and E. Pothos, Eds. Cham: Springer International Publishing, 2017, pp. 265–274.

[9] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISPLED '16, New York, NY, USA: ACM, 2016, pp. 64–69.

[10] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, 2016.

[11] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, Jan 2019.

[12] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, Sept 2017.

[13] H. Li, T. F. Wu, A. Rahimi, K. S. Li, M. Rusch, C. H. Lin, J. L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn *et al.*, "Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 16.1.1–16.1.4.

[14] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *CoRR*, vol. abs/1607.01759, 2016.

[15] S. Datta, R. A. G. Antonio, A. R. S. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, Sep. 2019.

[16] M. R. U. Quasthoff and C. Biemann, "Corpus portal for search in monolingual corpora,," in *Proc. of the International Conference on Language Resources and Evaluation,*, 2006.

[17] P. Koehn. (2005) Europarl: A parallel corpus for statistical machine translation. http://www.statmt.org/europarl/.