SpamHD: Memory-Efficient Text Spam Detection using Brain-Inspired Hyperdimensional Computing

Rahul Thapa*, Bikal Lamichhane*, Dongning Ma, Xun Jiao
* Equal Contributions
Villanova University

***Abstract*—Brain-inspired hyperdimensional Computing (HDC) leverages the mathematical properties of high-dimensional vectors (hypervectors) which show remarkable agreement with how brain functions. Hypervectors (HVs) are high-dimensional (e.g., 10,000 dimensions), holographic, and (pseudo)random with independent and identically distributed (i.i.d) components. Recently, HDC has demonstrated promising capability in a wide range of applications such as robotics, bio-medical signal processing, and genome sequencing. Text spam detection is a classic natural language processing (NLP) task that is usually solved using machine learning methods associated with data preprocessing techniques such as *tokenization*. In this paper, we develop a memory-efficient text spam detection approach called `SpamHD` based on HDC methods. In addition to the conventional tokenization-based approach, we also develop a tokenization-free HDC approach with N-gram encoding. Experimental results on three real-world spam datasets (Hotel review, SMS text, and YouTube comments) show that `SpamHD` is able achieve similar or even outperform baseline tokenization-based learning methods, but with significantly less storage requirements (30X-115X model size reduction). Further, we perform a design space exploration for `SpamHD` by tuning the number of dimensions of HVs and encoding methods, and evaluate the impact of such design parameters on accuracy and memory requirements.**

I. Introduction

Brain-inspired hyperdimensional computing (HDC) is an emerging computing paradigm that mimics the way that human brain works, i.e., using abstract and complex neural activity patterns of the size of the brain's circuits [10], [9]. HDC relies on deep and high-dimensional abstract patterns to perform "computations", rather than processing actual numbers. Recently, HDC has demonstrated promising capability in a wide range of emerging applications such as anomaly detection [19] and bio-medical signal processing [11]. Compared with deep neural networks (DNNs), HDC classifiers do not require back-propagation for training, drastically reducing the computation cost and time consumption of building HDC models. Further, HDC also uses a memory-centric architecture (e.g., associative memory), which makes it easy for HDC to embrace the emerging in-memory computing methodologies, further enhancing its energy efficiency and acceleration [8].

Text spam detection is a classic NLP task that aims to classify a given text sample (e.g., Email, text message, or Youtube comment) into either *ham* (not spam) or *spam*. While traditional machine learning methods such as random forest (RF), kNN (K-nearest Neighbors), Multi Layer perceptron(MLP) and support vector machines (SVM) all present satisfying accuracy [14], [21], [15], and on this binary classification task, they often require tokenization which extracts keywords in the text. In addition, these methods may pose a considerable amount of storage requirements for their trained models. Considering many of these tasks are being performed increasingly in edge devices such as mobile phones subject to memory and computation resource constraints, it is desirable to reduce the memory requirement of spam detection tasks as much as possible.

In this paper, we design, implement, and evaluate a memory-efficient text spam detection approach based on HDC methods called **`SpamHD`**. We first develop **`SpamHD`** based on tokenization which is referred to as **`SpamHD`**$-Token$, and then we also develop the character-based **`SpamHD`**$-Ngram$, which follows the *N-gram* scheme, free of the effort of *tokenization*, *vocabulary-building* and *out-of-vocabulary handling*.

Specifically, we make the following major contributions:

- We develop **`SpamHD`**, the first HDC-based spam detection approach and present two encoding mechanisms based on tokenization and N-gram for **`SpamHD`**: **`SpamHD`**$-Token$ and **`SpamHD`**$-Ngram$.
- We evaluate the performance of **`SpamHD`** using three real-world spam datasets: hotel reviews, SMS text, and YouTube comments. Experimental results show that **`SpamHD`** can achieve similar or even outperform baseline tokenization-based machine learning methods, but with significantly less storage requirements (30X-115X model size reduction).
- We perform a design space exploration by tuning the design parameters of **`SpamHD`** such as the number of dimensions for HVs and the encoding mechanisms. By evaluating their impact on accuracy and model storage size, we further shed light on HDC designs for NLP tasks.

II. Related Works

Hyperdimensional Computing Existing works on HDC focus mainly on two aspects: the application of HDC and optimization of HDC processing. The research of applying HDC to different problem domains focus on developing novel encoding and decoding mechanism. Recently, HDC achieves 97.8% accuracy on hand gesture recognition [10], which is around 9% higher than the state-of-the-art SVM classifier. HDC also shows success in multimedia applications such as voice recognition [5]. HDC is also applied to achieve higher

energy-efficiency in emerging technologies such as genome sequencing [6] and language recognition [12].

Optimization of HDC processing focuses on improving the efficiency of HDC models. For example, by exploring the data locality of HDC applications, computation reuse schemes are proposed to reduce the computation cost of HDC implemented on FPGAs [13]. Energy-efficiency in memory computing schemes and frameworks of HDC are also developed [8].

Spam Classification Many machine learning algorithms such as K-NN [2], random forest [16], SVM [18], and neural networks [7] are used for spam detection problem in various context such as text message [20], social media comments [22], and Email [4]. K-NN is used to establish a flow of feature extraction, resampling and classification for spam email classification [2]. Hybrid models involving two or more machine learning algorithms working in synergy to increase classification performance, e.g., using random forest for feature selection and neural networks for classification [16]. Neural networks show a strong capability of catching the semantics, achieving high accuracy on classification [7]. However, these classification models typically require a considerable amount of memory size to store their parameters.

Our Work This paper presents the first effort in using HDC for text spam detection. We compare **SpamHD** with existing methods of spam detection and demonstrate the memory efficiency of HDC-based approach. We also present different encoding schemes with or without the need of tokenization.

III. Background

A. Hypervectors

Hypervectors (HVs) are the basic elements of HDC. They are high-dimensional (usually around 10,000 dimensions) and holographic (not micro-coded) vectors composed of i.i.d. (independent and identically distributed) components. In HDC, those HVs are used to represent information of every data or item, imitating the mechanism of human brain to perceive and understand information. Specifically in the spam classification task, HVs can be used to represent different levels of information: from a letter, to a trigram, and to the entire text message.

B. Operations

To represent different levels of information, HDC features the encoding process in which HVs representing lower-level information (such as a letter) are dynamically aggregated into HVs representing higher-level information (such as a text message) by different operations. HVs, as operands, support three types of operations that act as the fundamental mechanisms of HDC: (element-wise) addition (+) , (element-wise) multiplication (*), and permutation (ρ). The three types of operations illustrate different properties of the HV operands.

Addition combines the information from the two operand HVs, therefore the similarity between the output HV of addition and each of the operand HVs is 0.5. Multiplication attaches the information of two HVs together, forming up a new and higher level HV with 0 similarity with the two operand HVs. Permutation performs cyclic rotation over an HV to reflect temporal or spatial changes. For example, in the task of spam detection, **SpamHD** uses addition for aggregating different encoded text HVs to train the model, uses multiplication to embed characters within each trigram to represent the flow of natural language, and uses permutation to reflect position and orders of characters in one trigram.

C. HDC Classification Process

Similar to other machine learning techniques, the flow of applying HDC in classification tasks requires training and inference using the trained models. Additionally, retraining or fine-tuning can be used to update the models for further improving the performance. HDC can address various types of input data, including letters, signals, and images. The process of mapping those input data into hypervectors is called encoding. This process is somewhat parallel to feature extraction.

In HDC, the first step is encoding. Encoding is to encode the original input sample (training or testing sample) into its corresponding HV. Encoding process is highly dependent on application characteristic or problem domain, and need to design in a case-by-case manner. We describe the encoding used in this paper in the next section.

The encoded HVs are then used in training process. The training process will use the operations mentioned above including addition, multiplication, and permutation on encoded HVs to generate class HVs, which represent different classes. For example, in spam detection, one class HV will be generated to represent spam and the other class HV will represent ham. The class HVs are stored in an associative memory (AM), which is the outcome of the training process.

Inference is the process of determining the class of unseen test data. This classification is accomplished by performing a similarity check between the HV representing the unseen test data and each of the class HVs in the associative memory. Higher similarity between HVs indicate overlap of information contained in the HV. Thus, the label of the class HV with the highest similarity to the test data HV will considered as the predicted class by the HDC model.

Retraining is the process of fine-tuning the trained associative memory to improve accuracy. Retraining uses the same process of inference to classify the text from the retraining set. If classification is correct, the text sample will be skipped. However, if misclassification occurs, HDC model will subtract the HV from the class HV that is incorrectly predicted to remove the erroneous information. In addition, HDC model adds the misclassified HV into the correct class HV to improve the model's accuracy.

All these stages uses the same encoding module and same shape (dimension) of the HVs, to ensure consistency on how information is aggregated within a classification task. In the next section, we describe how these processes are developed and implemented in **SpamHD** in detail.

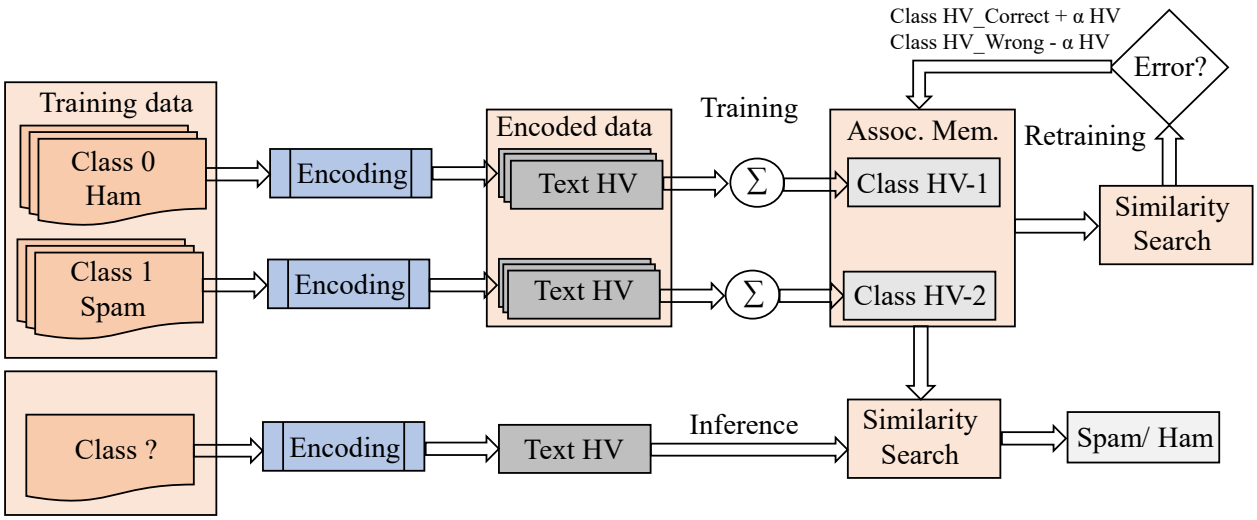


Fig. 1. Overview of **SpamHD** Framework. **SpamHD** features four modules as the main components: Encoding, Training, Retraining and Inference. Further, **SpamHD**$-Ngram$ uses the *N-gram* component inside the encoding module while **SpamHD**$-Token$ uses the *tokenization* component.

IV. SpamHD Framework

In this section, we describe **SpamHD** framework in detail for encoding the textual data into HVs and classifying the text into either **ham** or **spam**. The main components of **SpamHD** are shown in Fig. 1. **SpamHD** takes a text classification dataset and encodes it using two encoding techniques which we describe in following subsections.

A. Encoding

As mentioned above, encoding process is needed to project the data samples into the high-dimensional space, i.e., HVs, which are the basic elements used in HDC operations. In **SpamHD**, we propose two encoding mechanisms, tokenization and N-gram.

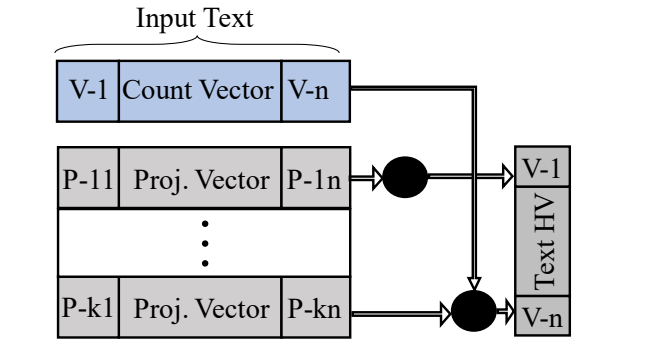
1) Tokenization: In order to encode a textual data into its representing HVs using the *tokenization* encoding module, there are 3 steps. The first step is to tokenize the training dataset, builds a vocabulary of known words, and encodes the dataset into count vector. The size of the each count vector is controlled by a parameter, $max\_features$, which builds a vocabulary that only considers the top $max\_features$ ordered by term frequency across the corpus. For example, if the $max\_features$ value is 700, each text row in the dataset is encoded into an array of length 700. For this paper, we generate features using character *N-grams* instead of word *N-grams*. During the inference phase, the same vocabulary is used to encode the testing dataset into corresponding count vector. The second step is to randomly generate i.i.d. binary HV called projection HV which has equal number of randomly placed 0s and 1s. This projection HV is used to transform our encoded training count vector into high-dimensional text HVs. The size of these projection HVs is (D, $max\_features$), where D is the desired dimension of HV and $max\_features$ is the length of each encoded count vector. Taking dot product (HDC multiplication) of each encoded count vector with this projection HV gives the text HV. The third step is to establish the HV representing the entire training data. After encoding all the count vectors into text HVs, the text HVs are summed up into their corresponding class HV in the associative memory.

2) N-gram: N-gram encoding module takes in a stream of letters in order from an input text, and creates a HV to represent the text. For each letter, the module assigns a unique i.i.d binary HV which has equal number of randomly placed 0s and 1s. In total, we considered 38 unique representations - 26 alphabets in lower case, 10 numerals (0 - 9), white space, and all the other unique characters such as symbols and dashes assigned as a single digit. The input text is first converted to unique digits from 0-37. Thereafter, for each unique digit, a unique HV is correspondingly assigned. For the *N-gram* method, we compute blocks of N consecutive letters with a sliding window of n-letters.

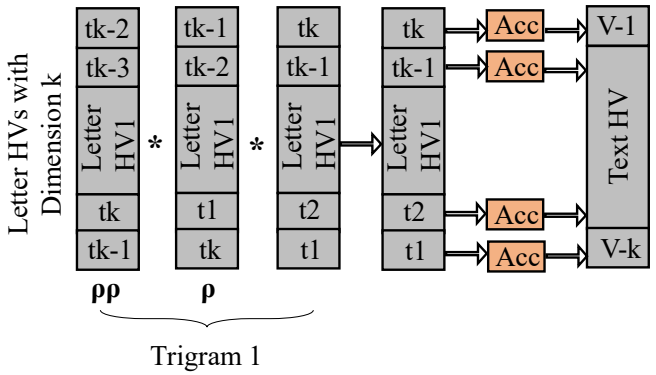
For each block of N consecutive letters, a HV is computed by using HV operations such as element-wise addition, element-wise multiplication and permutation. For example,in a trigram HV X-Y-Z, X is permuted twice ρ ρ X, Y is permuted once ρ Y and Z remains without any permutation. Thereafter, a component-wise multiplication is applied to the permuted HVs. Finally, a HV for a text is computed via component-wise addition of all the *N-gram* HVs by sliding a window of length n across the text.

B. Training

In this module, the encoded text HVs from the encoding module are used to construct an associate memory that contains the class HVs. Initially, the elements inside class HVs are initialized to zeros. Thereafter, all of the training samples, after converted to text HVs by the encoding module, are accumulated via the addition operation of HDC to their respective class HVs.



(a) Tokenization Encoding



(b) Ngram Encoding

Fig. 2. Two different encoding mechanism used to encode raw text into text HVs.

For example, all the encoded HVs representing samples belong to the class spam will be accumulated together to generate a spam class HV, and all the encoded HVs representing samples belong to the class ham will also be added together to generate ham class HV. These class HVs are typically stored in the associate memory that will be used in the inference stage and the retraining stage described as follows.

C. Inference

During the inference phase, the testing sample is encoded to a query text HV using the same encoding module that was used to train the HDC classifiers. The resultant query text HVs are then compared to class HVs in the associate memory using a similarity check module to classify the text as ham or spam. The class with highest similarity among class HVs and text HVs will be predicted as the label. For example, if the test HV is more similar to the spam class HV, then the test HV is classified as spam.

Similarity Check Determining the class of an unknown text is done by comparing its query HV to all the class HVs. This comparison is effectively performed in a distributed fashion using an associative memory. First, every data point in the testing dataset is encoded into its representative HVs, using the identical encoding mechanism, generated position, and value memory in the training phase. After that, we calculate the similarity between the query HV and every class HVs inside the associative memory. In **SpamHD** , similarity is measured using cosine similarity in Eq. 1:

$$Sim = CoSim(qHV, AM[i]) = \frac{qHV \cdot AM[i]}{||qHV|| ||AM[i]||} \quad (1)$$

where qHV refers to the query text HV and $AM[i]$ refers to the i-th class HV inside associative memory. The class with the maximum similarity with the query text HV subsequently becomes the prediction result of this data point. We then compare the true label with the predicted label for this text data point to determine if the prediction is correct. To evaluate the accuracy of the HDC classifer, we iterate this process through the entire testing dataset.

D. Retraining

It is possible that the HDC model will need fine-tuning after initially trained. This is realized using retraining module. In this module, we update the associate memory for a given number of epochs, each time correcting the misclassified text sample in the training dataset.

As an improvement from the traditional retraining schemes of HDC as described in Section III-C, we introduce the retraining learning rate to facilitate the retraining process. The retraining process in **SpamHD** is as follows: for each misclassified text sample in a given epoch, we first multiply text HV by a learning rate parameter alpha (α). We use a dynamic learning rate in this context as opposed to a static learning rate to mitigate potential overfitting. Empirically, we set the initial value of α as the number of epochs, and then after each epoch α is decreased by 1.

Note that just like learning rate in DNN training, the learning rate here is a hyperparameter that can be tuned based on the specific needs. The text HV is then subtracted from the wrong class HV and added to the correct class HV in the associate memory. In this paper, we typically perform 10 epochs for retraining as we observe that after 10 epochs the accuracy starts to saturate.

V. Experimental Results

A. Experimental Setup

We use three real-world datasets: Hotel reviews [17], SMS text [3] and YouTube comments [1]. The Hotel review, SMS spam, and YouTube comments dataset contains 1600, 5574, and 1956 instances respectively. These datasets are fairly balanced with each class (ham or spam) occupying approximately 50% of the dataset. We use 80% of dataset for training the classifiers and the rest 20% for testing. We compare **SpamHD** with various baseline classifiers. We evaluate the classification accuracy and model size for all classifiers. For each classifier, we measure the trained model sizes. For **SpamHD**, the model size is just the size of the trained associative memory, which contains class HVs for ham and spam class. Note that the model size of **SpamHD** does not include the projection vector that was used to encode the input text because projection vectors can be generated during runtime using the same random seed used in training.

TABLE I
SpamHD PERFORMANCE ON HOTEL REVIEW DATASET

Classifier	Test Acc	Model Size (kB)
kNN	70.43%	51,991
SVM	87.56%	9889
Random Forest	89.68%	9261
MLP	89.62%	492
SpamHD-$Ngram$	92.75%	80
SpamHD-$Token$	91.9%	44

TABLE II
SpamHD PERFORMANCE ON SMS SPAM DATASET

Classifier	Test Acc	Model Size (kB)
kNN	94.8%	182,169
SVM	98.5%	11,078
Random Forest	91.0%	859
MLP	98.7%	2412
SpamHD-$Ngram$	97.8%	80
SpamHD-$Token$	96.2%	44

We used four widely-used ML algorithms in NLP problems - Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbor (kNN), and Multi-layer Perceptron (MLP) - as our baseline methods. All of these classifiers were trained using the same count vector that was used for *tokenization* encoding mechanism that we used to train **SpamHD**-$Token$. Note that we performed hyperparameter tuning for each of these baseline ML classifiers using grid search. For example, we use $K = 3$ for KNN, $depth = 10$ for random forest. We also experiment with various N values of *N-grams* and use $N = 3$ for reporting the following results.

For **SpamHD**, we performed a grid search to tune hyper-parameters such as dimension of HV, and n-gram value. We find that trigram gives the best performance for both the encoding mechanisms. Similarly, the optimal value for HV-dimension for **SpamHD**-$Ngram$ is 10000 and for **SpamHD**-$Token$ was 5500. We use these optimal hyperparameters for all the reported results for **SpamHD**.

B. *SpamHD Performance and Model Size*

We compare the performance of **SpamHD** with 4 baseline ML classifiers: SVM, kNN, RF. and MLP on all three spam datasets, where we observe several important facts. Note that for each classifier, we use 5-fold cross-validation to measure the classification accuracy and report the average. First, for all three datasets, **SpamHD** is able to achieve similar or even outperform existing baseline methods. For hotel review dataset, **SpamHD** is able to achieve the highest accuracy at 92.75%. For SMS spam dataset and Youtube comment dataset, **SpamHD** is able to achieve 97.8% and 92.84% accuracy respectively, where the highest accuracy among baseline ML classifiers is 98.7% (random forest) and 93.6% (SVM) respectively.

TABLE III
SpamHD PERFORMANCE ON YOUTUBE DATASET

Classifier	Test Acc	Model Size (kB)
kNN	88.8%	65122
SVM	93.6%	5681
Random Forest	89.8%	1617
MLP	92.8%	246
SpamHD-$Ngram$	92.3%	80
SpamHD-$Token$	92.84%	44

Second, while **SpamHD** accuracy is similar to baseline classifiers, the model size of **SpamHD** is significantly less than all the baseline methods. For hotel review dataset, while being the most accurate classifier, **SpamHD** is also the most memory-efficient one with 115X less model size than the best performing baseline classifier (RF). The model size of **SpamHD** is 80kB, calculated using the formula 1000(HV-dimension) * 2(num-classes) * 4(element size) = 8 kB. (Note that each element is of 4 bytes). For SMS text and Youtube comments dataset, while **SpamHD** is 1-2% less accurate than the SVM and random forest methods, **SpamHD** is able to achieve 30X and 130X less model size than the best performing baseline classifiers (MLP and SVM respectively).

C. Design Space Exploration

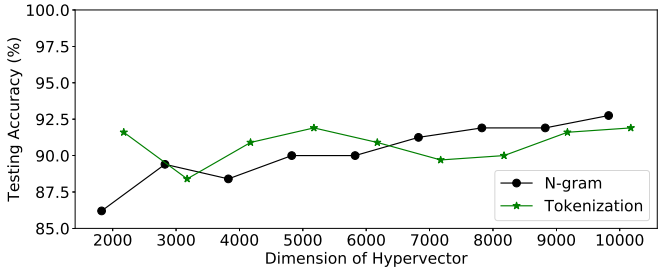
In this subsection, we perform a design space exploration by varying the configurations of **SpamHD**. In particular, we vary the encoding mechanism as well as the number of dimensions to assess their impact on **SpamHD** accuracy and model size.

Fig. 3 (a) shows the accuracy of **SpamHD** on Hotel review dataset. For **SpamHD**$-Ngram$, the accuracy is generally increasing with the increase in dimension of HV, attaining maximum accuracy (92.75%) at 10000 dimension. For **SpamHD**$-Token$, the accuracy is comparatively more stable across various HV dimensions, with the highest accuracy (91.9%) obtained with around 5500 dimension HV.

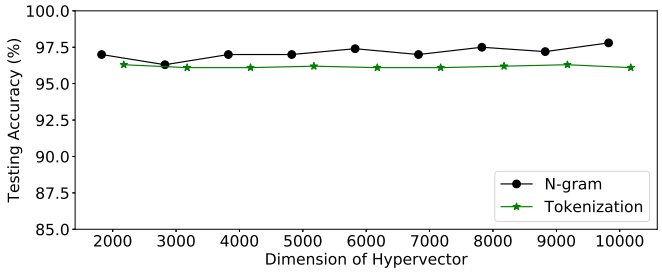
Fig. 3 (b) shows the accuracy of **SpamHD** on SMS spam dataset. We see that the accuracy of **SpamHD**$-Ngram$ classifier is generally increasing with the increase in dimension of HVs. The maximum accuracy (97.8%) is achieved when the dimension is 10,000. However, the accuracy of **SpamHD**$-Token$ is relatively constant (96.2%) for different HV dimensions. Therefore, even though the *N-gram* encoding method seems to perform better than *tokenization*, *tokenization* is able to gain a relatively high accuracy (96.2%) at low dimensions (e.g., 5500).

Interestingly, Fig. 3 (c) shows the accuracy of **SpamHD**$-Token$ is actually higher than **SpamHD**$-Ngram$ on YouTube dataset. For **SpamHD**$-Ngram$, the maximum accuracy (92.3%) is attained when the dimension of HVs are 10,000. However, for the **SpamHD**$-Token$, the curve is relatively flat, attaining maximum accuracy of 92.84% from 5000-7000 HV dimension.

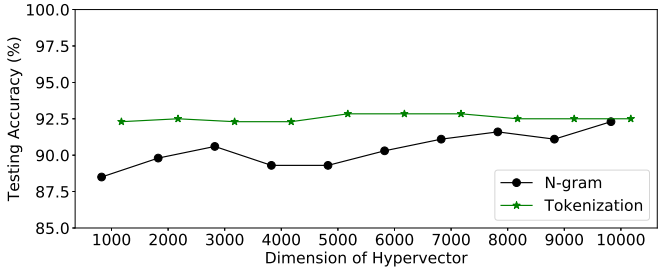
The model size of **SpamHD** is proportionally increasing as the HV dimensions increases. For example, the model size at dimension $D = 10000$ will be twice of that at dimension $D = 5500$.



(a) Hotel Review Dataset



(b) SMS Spam Dataset



(c) YouTube Dataset

Fig. 3. Testing accuracy and model size of **SpamHD**$-Ngram$ and **SpamHD**$-Token$ classifiers with different HV dimensions. The bar plot shows the model sizes and dot plot shows the testing accuracies.

VI. Conclusion

This paper presents **SpamHD**, a memory-efficient HDC-based approach for text spam classification. **SpamHD** encodes textual data into high dimensional vectors based on two encoding schemes - *N-gram* and *tokenization*. Using SMS spam, Youtube spam, and Hotel review datasets, we evaluate the accuracy of **SpamHD** against various baseline machine learning classifiers. Results show that **SpamHD** is able achieve similar or even outperform tokenization-based baseline learning methods, but with significantly less storage requirements. This confirms the potential of HDC on memory-constrained low-cost computing platforms. Further, we explore different design parameters for **SpamHD** including dimensionalities and encoding mechanisms. Our future work will develop HDC-based methods for more complicated NLP tasks such as sentiment analysis and topic segmentation. We will also develop efficient hardware accelerator for HDC-based NLP algorithms.

References

[1] Túlio C Alberto, Johannes V Lochter, and Tiago A Almeida. Tubespam: Comment spam filtering on youtube. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*, pages 138–143. IEEE, 2015.

[2] Loredana Firte, Camelia Lemnaru, and Rodica Potolea. Spam detection filter using knn algorithm and resampling. In *Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*, pages 27–33. IEEE, 2010.

[3] José María Gómez Hidalgo, Tiago A Almeida, and Akebo Yamakami. On the validity of a new sms spam collection. In *2012 11th International Conference on Machine Learning and Applications*, volume 2, pages 240–245. IEEE, 2012.

[4] Ismaila Idris and Ali Selamat. Improved email spam detection model with negative selection algorithm and particle swarm optimization. *Applied Soft Computing*, 22:11–27, 2014.

[5] Mohsen Imani et al. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017.

[6] Mohsen Imani, Tarek Nassar, Abbas Rahimi, and Tajana Rosing. Hdna: Energy-efficient dna sequencing using hyperdimensional computing. In *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 271–274. IEEE, 2018.

[7] Gauri Jain, Manisha Sharma, and Basant Agarwal. Optimizing semantic lstm for spam detection. *International Journal of Information Technology*, 11(2):239–250, 2019.

[8] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. In-memory hyperdimensional computing. *Nature Electronics*, 3(6):327–337, 2020.

[9] Dongning Ma, Jianmin Guo, Yu Jiang, and Xun Jiao. Hdtest: Differential fuzz testing of brain-inspired hyperdimensional computing. *arXiv preprint arXiv:2103.08668*, 2021.

[10] Abbas Rahimi, Simone Benatti, Pentti Kanerva, Luca Benini, and Jan M Rabaey. Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[11] Abbas Rahimi et al. Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials. *Mobile Networks and Applications*, 2017.

[12] Abbas Rahimi, Pentti Kanerva, and Jan 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*, pages 64–69, 2016.

[13] Sahand Salamat, Mohsen Imani, and Tajana Rosing. Accelerating hyperdimensional computing on fpgas by exploiting computational reuse. *IEEE Transactions on Computers*, 69(8):1159–1171, 2020.

[14] Rushdi Shams and Robert E Mercer. Classifying spam emails using text and readability features. In *2013 IEEE 13th international conference on data mining*, pages 657–666. IEEE, 2013.

[15] Sriram Srinivasan, Vinayakumar Ravi, V Sowmya, Moez Krichen, Dhouha Ben Noureddine, Shashank Anivilla, and Soman Kp. Deep convolutional neural network based image spam classification. In *2020 6th conference on data science and machine learning applications (CDMA)*, pages 112–117. IEEE, 2020.

[16] S Sumathi and Ganesh Kumar Pugalendhi. Cognition based spam mail text analysis using combined approach of deep neural network classifier and random forest. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–11, 2020.

[17] Rachael Tatman. Deceptive Opinion Spam Corpus. https://www.kaggle.com/rtatman/deceptive-opinion-spam-corpus. [Online; accessed 01-April-2021].

[18] Chi-Yao Tseng and Ming-Syan Chen. Incremental svm model for spam detection on dynamic email social networks. In *International Conference on Computational Science and Engineering*. IEEE, 2009.

[19] Ruixuan Wang, Fanxin Kong, Hasshi Sudler, and Xun Jiao. Hdad: Hyperdimensional computing-based anomaly detection for automotive sensor attacks. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[20] Qian Xu, Evan Wei Xiang, Qiang Yang, Jiachun Du, and Jieping Zhong. Sms spam detection using noncontent features. *IEEE Intelligent Systems*, 27(6):44–51, 2012.

[21] Bo Yu and Zong-ben Xu. A comparative study for content-based dynamic spam classification using four machine learning algorithms. *Knowledge-Based Systems*, 21(4):355–362, 2008.

[22] Wuxain Zhang and Hung-Min Sun. Instagram spam detection. In *Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2017.