Brain-Inspired Hypervector Processing at the Edge of Large Language Models

Alaaddin Goktug Ayar, Sercan Aygun, M. Hassan Najafi, and Martin Margala
University of Louisiana at Lafayette, LA, USA {alaaddin.ayar1, sercan.aygun, najafi, martin.margala}@louisiana.edu

***Abstract*—Brain-inspired hypervector processing, also known as hyperdimensional computing (HDC), has received significant attention in the last-decade computing era. However, only a few studies have explored its application as an intermediary tool to develop efficient language processing machine learning systems, particularly targeting large language models (LLMs). This research introduces the integration of brain-inspired hypervector processing into the `BERT`, `DistilBERT`, and `GPT-2` language model classifier components to alleviate network load and reduce classifier model size. Compared to conventional machine learning classifiers achieving similar accuracy levels, hypervector encoding and processing offer a lightweight and cost-effective alternative for language models. Utilizing the IMDb dataset for sentiment analysis, the proposed architecture achieves up to 83% classification accuracy while maintaining only a 9KB model size, rendering it highly deployable to edge devices compared to fundamental machine learning-based models like support vector machine (SVM), multilayer perceptron (MLP), and random forest.**

I. Introduction

The birth of brain-inspired computing approaches in machine learning systems marks a significant evolution beyond traditional neural networks. Achieving a lightweight and hardware-aware design for neural systems necessitates the transformation of intricate architectures into streamlined versions. Hyperdimensional computing (HDC) systems offer a promising solution by efficiently processing data through the representation of hypervectors instead of fixed or floating point processing. Hypervectors encode scalars or symbols using binary values, where randomly occurring $+1$ (*logic-1*) and -1 (*logic-0*) values are manipulated through logic operations to form holographic representations of the data. Holographic (or holistic) representations encompass multiple data dimensions [1], [2], ideally incorporating various features within a single embedded dimension akin to the brain's structure in living beings.

In the past decade, language models have gained significant attention in machine learning and artificial intelligence, necessitating an integration of two emerging sciences: *brain-inspired computing* and *language processing*. Like HDC, input data in language models is "encoded" to represent various linguistic elements. While encoder models manipulate data using fractional *fixed* or *floating point* values, this paper explores the potential advantages of binary hypervectors instead of scalars.

This study presents a novel approach for representing the `BERT`, `DistilBERT`, and `GPT-2` based encoders in hypervectors, with a distinctive method involving the utilization of adjustable weights tailored for the HDC classifier. A key innovation of this method lies in its integration of on-the-go training for encoder values created by large language models (LLMs) [3], such as the `BERT` encoder, enabling seamless mapping into HDC space. During this process, encoder values (E) are multiplied by specific weights (w), which are trained using *sigmoid activations*. As a result, all trained encoder values and their corresponding weights are mapped into a single-dimensional (of size D) hypervector corresponding to a distinct class. This unique methodology underscores the adaptability and efficiency of hypervector-based representations in language processing tasks.

This paper compares our proposed architecture with conventional machine learning approaches, providing insights into the potential benefits of hypervector-based encoding. After giving the proposed method in Section II, Section III discusses the design and implementation results, while Section IV presents conclusions.

II. Proposed Method

This section presents the proposed method behind HDC architectures that can be used for the language models. The architecture contains three modules for the training phase. We target different language models to align HDC systems with larger language models for future research. `BERT`, `DistilBERT`, and `GPT-2` language models are targeted for this late-breaking work. Each language architecture has core attention mechanisms inside having the encoders [5]. These encoders, while considering positional language-word-linguistic information, serve the purpose of numerical coding representation. Therefore, our research endeavors to present this data in a more efficient manner, leveraging the emerging computing paradigm of HDC.

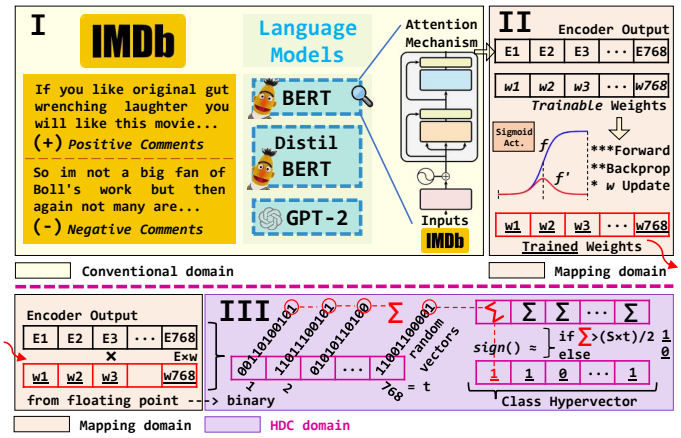

Fig. 1. The overall proposal. Step **I** illustrates the IMDb dataset [4], comprising 40,000 training samples and 10,000 test samples, employed to train various models, including `BERT`, `DistilBERT`, and `GPT-2`. This extensive collection of data provides a robust foundation for analyzing sentiment, where reviews are classified as either *positive* or *negative*. Step **II** is an intermediary step in mapping into the HDC domain. Training weights better transfer encoder values into the hypervector space. Step **III** is the HDC in binary; random hypervectors are accumulated & thresholded for class hypervector generation in holistic form.

In HDC, data is encoded in random binary vectors, which we utilize to represent floating-point values in random hypervectors. The initial step involves obtaining the output of the language model, referred to as the encoder output. Fig. 1 depicts this in Step **I**. While encoder outputs could be directly embedded into hypervectors, we propose a transition step for HDC to adjust trainable weights in conjunction with each separate encoder value. This adjustment is facilitated by a simple learning model employing *sigmoid activations*, thereby creating a simplistic HDC model within the conventional machine learning domain; there will be no need to reiterate training over the binary values for a better accuracy outcome of the model [6].

In Step **II** of the training model, trainable weights are randomly assigned and subsequently trained using conventional neural network steps, including forward path, backpropagation, and weight update. Given the nature of the problem, *sigmoid activations* are employed. At this stage, conventional machine learning and HDC methodologies converge, characterized by random assignments and approximate binarizations through *sigmoid* functions. These steps align with the *multiply*, *add*, and *sign* operation characteristic of conventional HDC [7].

The multiplied weights and encoder values in random hypervectors are utilized in HDC. $E \times w$ are in the unit interval, where 0 scalars are represented by '00...000' binary hypervector, and *1* scalars are represented by '11...111' binary hypervector. Intermediate values are represented proportionally based on the total number of 1s and 0s in a ratio. For instance, a value of 0.5 is represented as '11..11..00..00', containing $D/2$ of *1*s and $D/2$ of *0*s randomly distributed within a D-sized hypervector [8].

Upon obtaining the hypervectors, the summation of each bit position across all encoding hypervectors is conducted. This summation, depicted in Step **III** of Fig. 1, accumulates values in identical positions throughout the vectors. Subsequently, the final HDC step involves determining the *sign* (or binary logic value in memory) [9]. In the binary domain, this approximate sign resembles a threshold function. At each training sample, the contributing binary vector is cumulatively recorded into the vector. This is executed in the framework without floating-point format, wherein a dynamic adder checks for the threshold at each bit addition. If the total accumulation exceeds half the size of $(threshold \times S)$, a binary *1* is recorded in the final class hypervector. Here, S represents the total number of samples for the corresponding class, while the threshold (t) denotes the size of the language model encoder. The maximum achievable accumulation is $(threshold \times S)$, with its half-size serving as the reference point for *1* or *0* binarization.

The training concludes after obtaining the class hypervector for each class, with no further iterative training steps in HDC. This streamlined approach significantly contributes to the classifier's lightweight nature. Ultimately, the binary encoding of the language model encoder using trained weights as mappers into the HDC domain yields binary class models, echoing the brain-inspired holistic representation.

During inference, the process involves following the encoder outputs with the trained weight multiplications in the random vectors. This yields an output test hypervector akin to obtaining the class hypervector. Subsequently, this test hypervector is compared with all class hypervectors in binary. The class with the highest similarity (cosine similarity score [10]) to the test sample's hypervector is assumed to be the label of the test sample. This straightforward inference mechanism effectively leverages the encoded language model representations and trained weights within the HDC domain.

III. Design and Implementation

This section evaluates the implementation of both training and inference processes. The deployed model in an edge device is experimented with for runtime. These experiments encompass three LLMs: `BERT`, `DistilBERT`, and `GPT-2`. During the training phase, alongside the HDC classifier, conventional machine learning models such as support vector machine (SVM), multi-layer perceptron (MLP), and random forest are also employed.

Table I presents the performance of the embedding models, considering iso-accuracy values of HDC versus other machine learning classifiers. Referring back to Fig. 1, Step **II** comprises the encoder outputs inputted directly to traditional machine learning algorithms for conventional models. However, inherently complex and iterative algorithms, such as SVM, bring about a complex structure. The common benchmark among these models is the iso-accuracy derived from HDC, representing the accuracy level achievable by powerful yet resource-intensive machine learning models. Remarkably, for this level of accuracy, the HDC-based model demonstrates a model size of merely 9 kilobytes (KB). HDC's compact model size is a result of its binary vector encoding strategy, which simplifies complex linguistic information into a form that is both easy to handle and requires less storage space. This binary format, unlike the floating-point representations used by more traditional machine learning algorithms such as SVM, MLP, and random forest, is key to HDC's reduced size.

`DistilBERT` emerges as the optimal LLM for HDC, a significant outcome considering the lightweight nature of the "distil" architecture [11]. As anticipated, HDC also surpasses in terms of training time, owing to its lightweight learning approach to holistic representation. In contrast, conventional machine learning algorithms exhibit larger model sizes and training times (up to 4× longer) than HDC for the same level of accuracy.

Furthermore, inference time is also considered for ARM processor-based edge device testing. The HDC model proves to be 6× faster than traditional SVM, 3× faster than MLP, and 5× faster than random forest. These improvements are attributed to HDC's reliance on binary data. Specifically, the inference process involves only adding the binary vectors, then the thresholded accumulation is dynamically compared to the class hypervectors. Overall, this study advocates for the utilization of `DistilBERT` LLM in conjunction with the HDC classifier. Leveraging `DistilBERT`'s lightweight architecture, combined with a lighter classifier, promises a design with the lowest training time, smallest model size, and lowest inference time.

Consolas

TABLE I
HDC vs. ML Algorithms using LLM-based Text Embeddings

Embedding Model	Algorithm	Iso-Accuracy	Training Time (sec.) ❖	Model Size	Inference Time (sec.)★
`BERT`	SVM	81%	946.32	213.51 MB	6.3357
	MLP		300.21	18.05 MB	3.3300
	Random Forest		534.59	63.50 MB	5.7673
	HDC		**272.94**	**9KB**	**1.0180**
`DistilBERT`	SVM	83%	800.45	95.58 MB	4.5522
	MLP		590.75	1.81 MB	2.9476
	Random Forest		519.53	55.76 MB	4.8522
	HDC		**269.41**	**9KB**	**1.0127**
`GPT-2`	SVM	77%	750.45	229.67 MB	6.5548
	MLP		343.52	5.42 MB	3.2187
	Random Forest		458.48	55.44 MB	4.8012
	HDC		**306.41**	**9KB**	**1.0290**

★ Inference conducted on an edge device equipped with a 1.2GHz quad-core ARM Cortex-A53 CPU, 1GB of RAM, and 64-bit architecture.
❖Embeddings for this study were generated by LLMs utilizing an NVIDIA RTX A1000 GPU, while the HDC and machine learning models were trained on a 12th Gen Intel® Core™ i7-12800H CPU with 20 logical cores.

IV. Conclusions

In conclusion, this study demonstrates the significant advantages of HDC as an effective and compact approach to utilize advanced LLMs like `BERT`, `DistilBERT`, and `GPT-2` for processing language tasks. By employing HDC for the task of binary classification on the IMDb dataset, we have effectively transformed complex language data into a binary vector space, achieving high levels of accuracy while maintaining a very small model size. Our method captures the indirect differences in language that distinguish *positive* from *negative* movie reviews, achieving accuracy levels of up to 83% with a model size of just 9KB. This breakthrough in merging HDC with advanced language models not only underscores HDC's potential to make natural language processing applications more accessible and practical for tiny device computations but also outlines a scalable and cost-effective approach. This aligns with cognitive computing goals, thereby expanding the possibilities for tackling language processing challenges on computationally constrained edge devices.

References

[1] P. Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, Jun 2009.

[2] D. Kleyko *et al.* A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *ACM Comput. Surv.*, 55(9), jan 2023.

[3] N. Muennighoff *et al.* MTEB: Massive text embedding benchmark. In A. Vlachos and I. Augenstein, editors, *17th Conference of the European Chapter of the Association for Computational Linguistics*, May 2023.

[4] A. L. Maas *et al.* Learning word vectors for sentiment analysis. In *49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[5] K. Clark *et al.* What does BERT look at? an analysis of BERT's attention. In T. Linzen *et al.*, editors, *ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2019.

[6] M. Imani *et al.* Quanthd: A quantization framework for hyperdimensional computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2268–2278, 2020.

[7] S. Duan *et al.* Lehdc: learning-based hyperdimensional computing classifier. In *DAC'22*, 2022.

[8] L. Ge and K. K. Parhi. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47, 2020.

[9] S. Aygun *et al.* Learning from hypervectors: A survey on hypervector encoding, 2023. arXiv, 2308.00685.

[10] F. Asgarinejad *et al.* Enhanced noise-resilient pressure mat system based on hyperdimensional computing. *Sensors*, 24(3), 2024.

[11] V. Sanh *et al.* Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.