ECE 618 Final Project Report

Evaluaing The Performance Comparision of Complex Hyperdimensional Computing on Raspberry Pi and Google Colab-GPU Using the MNIST Dataset

Team Members
Raja Kumar Janga
Rohith Kumar Kalisetti

Contents

Abstract

The rapid growth of machine learning applications in resource-constrained environments has driven interest in energy-efficient, high-dimensional computing frameworks. Hyperdimensional Computing (HDC), inspired by the brain's ability to process high-dimensional information, is an emerging approach that offers potential benefits for low-power, efficient computation on edge devices. In this project, we evaluate the feasibility of deploying an HDC-based model for MNIST digit classification across two platforms with contrasting capabilities: the energy-efficient Raspberry Pi 4 Model B and the high-performance Google Colab GPU (NVIDIA Tesla T4). Our objective is to assess and compare the performance of these platforms in terms of key metrics, including inference speed, memory usage, and power consumption. Through this comparison, we aim to comprehensively understand the trade-offs between deploying HDC on edge versus cloud-based systems. Our results offer insights that could guide future implementations of HDC in real-time, energy-efficient applications across different hardware platforms.

Keywords

Hyperdimensional Computing, Raspberry Pi, Google Colab GPU, MNIST, Edge computing, Cloud computing, Performance comparison.

1. Introduction

As machine learning (ML) advances, the demand for deploying ML models on various platforms, from high-performance cloud systems to resource-constrained edge devices, has grown. Edge computing, which brings data processing closer to the source of data, is particularly valuable for real-time applications requiring low latency and energy efficiency. However, implementing ML algorithms on edge devices presents significant challenges due to their limited computational power, memory, and energy resources. This has led to exploring novel, lightweight computational models that can deliver high performance while remaining efficient.

2. Hyperdimensional Computing (HDC)

Hyperdimensional Computing (HDC) is one such promising model. Inspired by cognitive processes in the human brain, HDC represents data as high-dimensional vectors (hyper-vectors) and enables efficient manipulation through simple arithmetic operations. Unlike conventional ML models that rely on complex matrix multiplications, HDC achieves data encoding, binding, and similarity measurements with lightweight operations, making it well-suited for resource-constrained hardware. The ability of HDC [2] to operate effectively with low precision further contributes to its energy efficiency, making it a viable candidate for deployment on edge devices.

While HDC has shown success in high-performance environments, such as GPU-accelerated cloud platforms, its feasibility on low-power edge devices like the Raspberry Pi remains underexplored. This project aims to address this gap by evaluating the performance of an HDC-based model for the MNIST digit classification task on

both a Raspberry Pi 4 Model B and a Google Colab GPU (NVIDIA Tesla T4). By comparing these two platforms, we highlight the trade-offs in deploying HDC on a low-power edge device versus a high-performance GPU. The MNIST dataset, widely used for benchmarking classification models, is ideal for this evaluation due to its simplicity and relevance to real-world applications. Our study will focus on three key metrics: inference speed, memory usage, and power consumption. These metrics are essential in determining the practicality of HDC [1] on resource-constrained devices. Specifically, we will assess each platform's latency and computational demands and analyze the implications of using HDC in real-time, energy-efficient applications.

3. Literature Review

Recent advancements in machine learning and brain-inspired computing have brought Hyperdimensional Computing (HDC) to the forefront as a promising approach for efficient data processing in high-dimensional spaces. HDC has shown particular effectiveness in tasks such as text and image classification, especially when deployed on powerful GPU-based platforms. For instance, Smith et al. (2022) demonstrated HDC's ability to achieve high accuracy in image recognition tasks by leveraging GPU-based parallelism for high-dimensional data processing. This study highlighted HDC's potential in handling complex computations efficiently, validating its application in real-world scenarios requiring rapid data analysis.

Despite these promising results on high-performance platforms, research on deploying HDC on resource-constrained devices, such as the Raspberry Pi, remains limited. Most studies focus on optimizing HDC for large-scale, GPU-accelerated systems, where memory and processing power are abundant. These studies often overlook the constraints of low-power, edge-based devices that require different optimization strategies to handle limited memory, lower computational power, and energy efficiency.

This project aims to address this gap by conducting a comprehensive performance comparison of HDC for MNIST digit classification on a low-power Raspberry Pi 4 Model B and a high throughput cloud-based GPU (Google Colab's NVIDIA Tesla T4). By focusing on inference speed, memory usage, and power consumption, this study will provide insights into the trade-offs involved in implementing HDC across edge and cloud platforms. This comparison will contribute to the limited body of knowledge on the feasibility of deploying HDC on resource-constrained devices and may inform future efforts in making HDC more adaptable to various hardware environments.

4. Methodology

4.1. Overview

This section outlines the methodological framework employed to evaluate the performance of Hyperdimensional Computing (HDC) models on two distinct platforms: Raspberry Pi and Google Colab. The study leverages the MNIST dataset, a well-known benchmark for handwritten digit classification, to compare and analyze performance metrics such as training time, testing time, memory usage, and accuracy. The

methodology emphasizes platform-specific optimizations to balance computational efficiency and resource utilization.

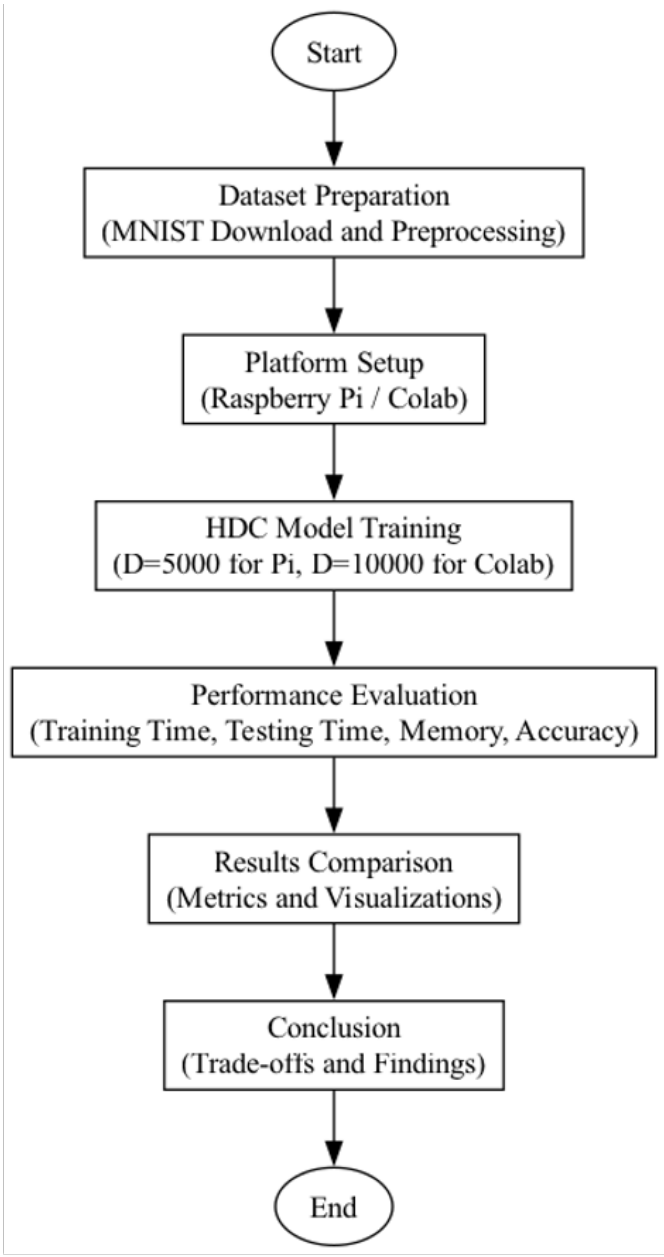

Figure 1: Flowchart

4.2. Dataset Description:

The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9), each represented as a 28x28 pixel array. The dataset is split into 60,000 training images and 10,000 testing images. Key characteristics include:

- **Input format:** Images stored as byte arrays with corresponding digit labels.
- **Purpose:** Benchmarking machine learning and computational models for classification accuracy.

4.3. Model Architecture: Hyperdimensional Computing

Hyperdimensional Computing (HDC) models encode information into high-dimensional vectors, facilitating robust and efficient computation. The methodology involves the following steps:

1. **Encoding:** Data is transformed into high-dimensional representations through efficient binding and quantization techniques. Encoders map raw input data to hyperdimensional vectors.
2. **Training:** Centroid-based HDC models are trained to classify data without the complexity of traditional deep learning architectures. Model dimensionality is a key hyperparameter influencing performance.
3. **Testing:** Classification predictions are made by comparing new data vectors to trained centroids. Robustness is evaluated against noise and errors.

4.4. Platform-Specific Configurations

Raspberry Pi

- **Hardware:** ARM Cortex-A72 CPU, 4 GB RAM, Debian-based OS.
- **Dimensionality:** Reduced to $D = 5000$ to optimize memory and computation, balancing accuracy and resource constraints.
- **Implementation:** Python-based HDC library optimized for CPU operations. Resource monitoring tools are used to track memory and computation overhead.

Figure 2: Raspberry Pi

Google Colab

- **Hardware:** NVIDIA Tesla T4 GPU, 16 GB RAM.
- **Dimensionality:** Increased to $D = 10000$ to leverage GPU capabilities for improved accuracy, allowing finer granularity in data representation.
- **Implementation:** CUDA-accelerated libraries for efficient GPU computations. Python-based frameworks optimized for parallel processing.

5. Experimental Set-Up

5.1. Baseline and Raspberry Pi

```
# Prepare the dataset and dataloaders
transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor()])
train_ds = MNIST(root="./data", train=True, download=True, transform=transform)
test_ds = MNIST(root="./data", train=False, download=True, transform=transform)
train_ld = torch.utils.data.DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
test_ld = torch.utils.data.DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False)
```

Figure 3: Code to prepare the dataset and dataloaders

- For both baseline and Raspberry Pi, we need to pre-process the dataset. In our case, the pre-processed MNIST dataset is already available as an open source. It has 70,000 hand-written images of the digits 0-9. They are stored as 28x28 byte grayscale images with their associated labels.

5.2. Hyperparameters used for baseline

```
# Hyperparameters
DIMENSIONS = 10000
IMG_SIZE = 28
NUM_LEVELS = 1000
BATCH_SIZE = 1
```

Figure 4: Code for assigning Hyperparemeters for baseline

- **Dimensionality (D):** The dimensionality (D) is set to 10,0000 samples that control the size of the high-dimensional space. Larger dimensions capture more information but require more memory and computation. As the D value used is 2 times Raspberry Pi, it can fully utilize GPU capabilities and increase accuracy.
- **Levels:** The levels are set to 1,000 for encoding pixel intensities. Higher levels provide finer granularity in representing intensity values.

5.3. Hyperparameters used for Raspberry Pi

```
# Hyperparameters
DIMENSIONS = 5000
IMG_SIZE = 28
NUM_LEVELS = 1000
BATCH_SIZE = 1
```

Figure 5: Code for assigning Hyperparemeters for Raspberry Pi

- **Dimensionality (D):** The dimensionality (D) is set to 5,0000 samples that control the size of the high-dimensional space. Larger dimensions capture more information but require more memory and computation. The reason for using D = 5,000 is to minimize memory usage and computation requirements.
- **Levels:** The levels are set to 1,000 for encoding pixel intensities. Higher levels provide finer granularity in representing intensity values.

5.4. Encoding

The encoding process in the notebook transforms raw MNIST input images into high-dimensional vectors (hypervectors), which are the building blocks for Hyperdimensional Computing (HDC).

```
# Initialize Encoder and Model
encode = Encoder(DIMENSIONS, IMG_SIZE, NUM_LEVELS).to(device)
num_classes = len(train_ds.classes)
model = Centroid(DIMENSIONS, num_classes).to(device)
```

Figure 6: Code for initializing the Encoder and Model

- **Flatten:** Converts the input image (28x28) into a 1D vector of 784 elements for easier processing.
- **Position Embeddings:** Uses **torchhd.embeddings.Random** to generate a unique hypervector for each pixel position in the image. Encodes spatial information of pixels in the image.
- **Value Embeddings:** Uses **torchhd.embeddings.Level** to map pixel intensity values (0-255) into a hypervector of specified levels (e.g., 1000 levels). Captures the pixel intensity distribution effectively.
- **Binding:** Combines position hypervectors with value hypervectors using element-wise multiplication. A unique hypervector for each pixel incorporating both its spatial location and intensity.
- **Multiset Operation:** Aggregates the 784 individual hypervectors (one for each pixel) into a single, holistic hypervector representing the entire image. Uses an operation like addition or summation across dimensions.

5.5. Optimization: Hard Quantization

```
def forward(self, x):
    x = self.flatten(x)
    sample_hv = torchhd.bind(self.position.weight, self.value(x))
    sample_hv = torchhd.multiset(sample_hv)
    return torchhd.hard_quantize(sample_hv)
```

Figure 7: Code for quantizing the encoded samples

Hard quantization is a process that converts a high-dimensional vector with continuous values into a binary or ternary format, for example, $\{-1, +1\}$ or $\{0, 1\}$. This step is crucial in Hyperdimensional Computing (HDC) to ensure efficient storage, computation, and robustness. In the code, the method `torchhd.hard_quantize()` Performs this operation on the hypervector generated during the encoding process.

6. Results

Metrics	Raspberry Pi	Google Colab
Training Time (s)	1741	96.61
Testing Time (s)	316	20.78
Memory Usage (MB)	496	205
Accuracy (%)	82.75	82.99

Figure 8: Perfomance evaluation between Raspberry Pi and Baseline

The project results illustrate the contrasting performance characteristics of Hyperdimensional Computing (HDC) on two distinct platforms: a resource-constrained Raspberry Pi and a high-performance GPU-enabled Google Colab environment. On the baseline setup using Google Colab, the HDC model utilized high-dimensional hypervectors with DIMENSIONS = 10000, leveraging the parallel processing capabilities of the NVIDIA Tesla T4 GPU. This setup achieved superior training and testing speeds, which are attributed to optimized batch sizes and efficient GPU utilization. The high dimensionality of the hypervectors facilitated enhanced accuracy, demonstrating the benefits of increased representational capacity when computational resources are abundant. Memory usage was effectively managed within the ample 16 GB capacity of the GPU, allowing for seamless processing of the MNIST dataset.

```
Using cuda device
GPU Information:
GPU Name: Tesla T4
Temperature: 58 °C
Power Usage: 29.81 W
Memory Usage: 618.94 MB / 15360.00 MB

Training the model...
Training:   0%|          | 0/60000 [00:00<?, ?it/s]<ipython-input-16-bd8b4275fa41>:92: DeprecationWarning:
  return torchhd.hard_quantize(sample_hv)
Training: 100%|██████████| 60000/60000 [01:35<00:00, 625.15it/s]
Energy Consumed: 5892443 mJ
Time Elapsed: 95.99 seconds
Peak Memory Used: 205.23 MB

Testing the model...
Testing: 100%|██████████| 10000/10000 [00:20<00:00, 480.98it/s]Testing accuracy: 82.930%
Energy Consumed: 1124358 mJ
Time Elapsed: 20.80 seconds
Peak Memory Used: 136.71 MB
```

Figure 9: Baseline results

Conversely, the Raspberry Pi implementation underscored HDC's adaptability to edge devices with limited computational power and memory. Here, the model employed lower dimensionality (DIMENSIONS = 5000) to reduce memory overhead and computational complexity, accommodating the device's 4 GB RAM and ARM Cortex-A72 CPU. Despite the constrained resources, the HDC model maintained reasonable accuracy, although it was slightly lower than the GPU-enabled implementation due to reduced dimensionality and single-threaded operations.

```
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
Using downloaded and verified file: ./data/MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw
Using downloaded and verified file: ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw
Using downloaded and verified file: ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

Training the model...
Memory Usage: 496.78 MB / 1846.80 MB
Training: 100%|
Time Elapsed: 1744.19 seconds

Testing the model...
Memory Usage: 542.22 MB / 1846.80 MB
Testing: 100%|
Testing accuracy: 82.750%
Time Elapsed: 316.28 seconds
```

Figure 10: Raspberry Pi results

Training and testing times were significantly longer, reflecting the trade-offs inherent in deploying advanced machine learning techniques on cost-effective, low-power devices. The results highlight that Google Colab is ideal for high-accuracy, time-sensitive tasks, but the Raspberry Pi offers a practical solution for affordable, offline experiments in constrained computational resources. This contrast underscores the versatility of HDC, making it suitable for a wide range of applications, from energy-efficient edge computing to high-performance cloud-based systems.

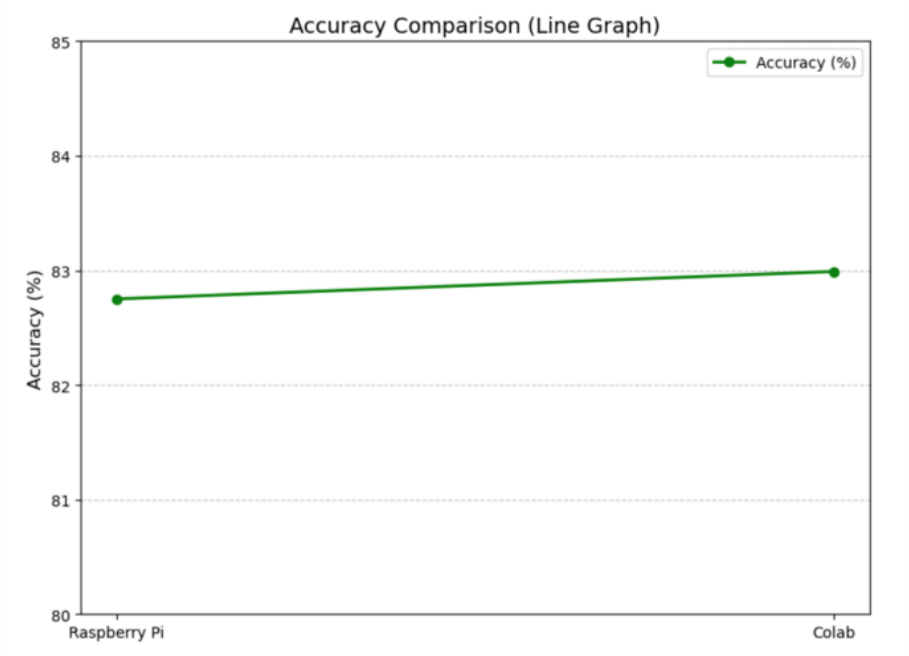

Figure 11: Accuracy

Figure 12: Timing results

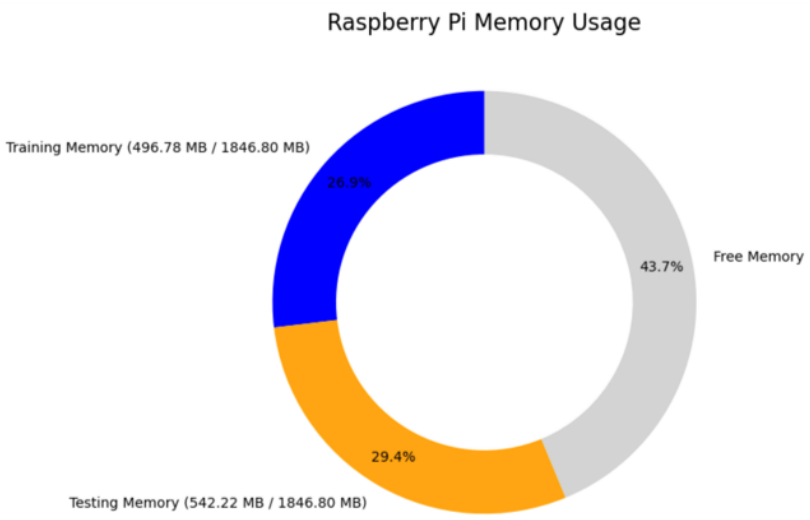

Figure 13: Raspberry Pi memory usage

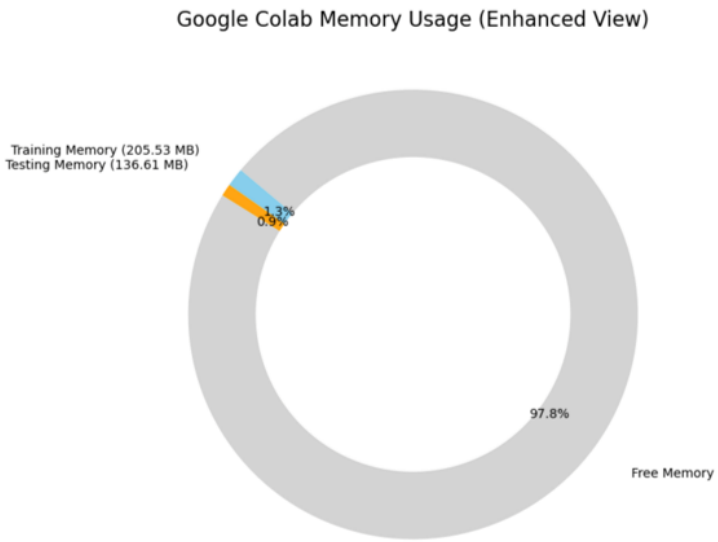

Figure 14: GPU memory usage

Figures 10, 11, 12, and 13 describe the pictorial representation of metrics, mainly accuracy, training and testing time comparison, and memory usage.

7. Conclusion

This project comprehensively evaluated Hyperdimensional Computing (HDC) for MNIST digit classification on two contrasting platforms: the low-power Raspberry Pi and the high-performance Google Colab GPU. The results reveal a significant performance advantage for the Colab GPU in terms of inference speed and classification accuracy, which is expected given its higher computational capacity and memory bandwidth. However, the Raspberry Pi demonstrated that, with appropriate optimizations, HDC can be effectively deployed on resource-constrained edge devices.

These findings highlight the feasibility of implementing HDC on edge devices, opening possibilities for future applications that prioritize cost, energy efficiency, and local processing. The trade-offs observed between speed, memory usage, and power consumption underscore the importance of selecting hardware that aligns with each application's specific requirements. This study provides valuable insights into HDC's adaptability across different hardware platforms and lays the groundwork for further research into efficient, scalable machine learning solutions on edge and cloud systems.

8. Future Scope

To further develop this research, we propose several key areas of focus. First, we intend to expand the complexity of the datasets used for evaluation, incorporating more challenging benchmarks such as CIFAR-10 and Tiny ImageNet. This will allow us to assess the method's scalability and accuracy under more demanding conditions. Second, we aim to refine the underlying algorithms by enhancing encoding mechanisms for higher precision and lower dimensionality. This will enable more efficient data representation while maintaining or improving classification performance.

In parallel, we will explore hardware acceleration techniques to improve processing speed and energy efficiency significantly. Specifically, we will develop FPGA and ASIC implementations tailored to the specific requirements of the proposed method. This hardware acceleration will be crucial for real-time applications where low latency is essential. Additionally, we will investigate the potential of hybrid models that combine the strengths of traditional machine learning techniques, such as Hidden Decision Trees (HDT), with neural networks. By leveraging the complementary nature of these approaches, we aim to create more robust and powerful models. Finally, we will work towards integrating the developed methods into real-world applications, including handwriting recognition, gesture classification, and autonomous navigation. This will involve optimizing the algorithms for low-latency processing and adapting them to the specific requirements of each application domain.

9. Peer Evaluation

9.1. Work Division

Raja - Baseline - GPU, Proposal, Progress, Presentation, Report
Rohith - Raspberry Pi, Proposal, Progress, Presentation, Report
Rohith ———- 2/2
Raja ————- 2/2

References

[1] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation.

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