PerfHD: Efficient ViT Architecture Performance Ranking using Hyperdimensional Computing

Dongning Ma
Villanova University
Villanova, PA 19085
dma2@villanova.edu

Pengfei Zhao
Beijing Xiaochuan Technology Co., Ltd.
Haidian, Beijing 100191, China
zhaopengfei2014@xiaochuankeji.cn

Xun Jiao
Villanova University
Villanova, PA 19085
xun.jiao@villanova.edu

Abstract

Neural Architecture Search (NAS) aims at identifying the optimal network architecture for a specific need in an automated manner, which serves as an alternative to the manual process of model development, selection, evaluation and performance estimation. However, evaluating performance of candidate architectures in the search space during NAS, which often requires training and ranking a mass amount of architectures, is often prohibitively computation-demanding. To reduce this cost, recent works propose to estimate and rank the architecture performance without actual training or inference. In this paper, we present ***PerfHD****, an efficient-while-accurate architecture performance ranking approach using hyperdimensional computing for the emerging vision transformer (ViT), which has demonstrated state-of-the-art (SOTA) performance in vision tasks. Given a set of ViT models,* ***PerfHD*** *can accurately and quickly rank their performance solely based on their hyper-parameters without training. We develop two encoding schemes for* ***PerfHD****, Gram-based and Record-based, to encode the features from candidate ViT architecture parameters. Using the VIMER-UFO benchmark dataset of eight tasks from a diverse range of domains, we compare* ***PerfHD*** *with four SOTA methods. Experimental results show that* ***PerfHD*** *can rank nearly* $100K$ *ViT models in about just 1 minute, which is up to* $10X$ *faster than SOTA methods, while achieving comparable or even superior ranking accuracy. We open-source* ***PerfHD*** *in PyTorch implementation at* https://github.com/VU-DETAIL/PerfHD.

1. Introduction

As deep learning models broaden their applications and enhance their capability to various domains, there is a growing demand of developing and optimizing architectures for higher model performance. However, with the architectures evolving increasingly deep, the architecture engineering usually requires enormous effort due to the expansion of design space, which is hardly possible using manual effort. To address this issue, neural architecture search (NAS) has been proposed and adopted to identify the optimal neural network architecture in an automated manner [12, 15]. Recently, vision transformer (ViT) emerges as a promising algorithm and receives wide attention as it achieves SOTA performance in vision tasks [5]. NAS is particularly useful and necessary for ViT because of the heterogeneity of components and blocks, which serves as the potential knobs in the search process.

Generally speaking, NAS can be summarized as shown in Fig. 1: First, system designers set custom constraints which defines the search space. Then, within this search space, candidate architectures are sampled using search strategies iteratively. Lastly, the performance of candidate architecture will get evaluated/estimated to obtain a performance ranking, which determines the optimal architecture for this NAS task. NAS process itself can be a resource-demanding process. For example, NAS has spent days or even weeks using clusters of hundreds of GPUs to identify architectures that can achieve comparable results on challenging computer vision tasks [16]. One major computation-expensive process in NAS is the process of evaluating the performance of neural architectures sampled from the predefined search space. A canonical training and validation flow can be applied to evaluate the performance of an architecture. However, this is prohibitively slow, thus many recent works develop various efficient performance evaluation techniques such as sharing weights from a supernet to avoid training models afresh [3, 20, 21]. However, this still cannot avoid the process of evaluating the performance of sampled architectures by running inference on each one of them. To address this issue, multiple works have been proposed to use machine learning for predicting the performance of a given architecture without training or inference [11, 22].

In this work, we depart from the conventional learning

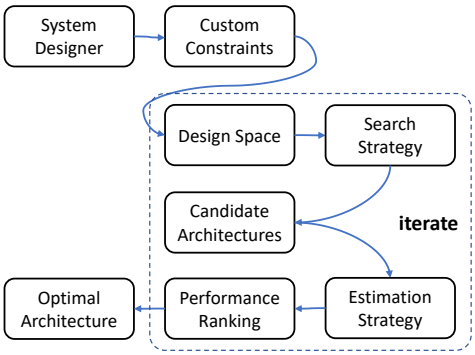

Figure 1. A typical NAS flow.

algorithms and embrace the hyperdimensional computing (HDC), to develop an efficient-yet-accurate performance ranking approach for ViT architectures. HDC is an emerging non von Neumann computation paradigm, leveraging the learning and representation capabilities of extremely high dimensional vectors inspired from the abstract brain activity functionalities [18]. We identify two highlighted characteristics of HDC which is suitable to address the performance ranking problem during NAS. First, HDC is known for its efficiency over existing machine learning algorithms such as neural networks, including smaller model size, faster model convergence and less computational intensity [6, 10, 13, 17]. This can reduce the overhead of NAS performance ranking. Second, HDC is better at learning with limited data such as one-shot learning tasks and are less likely to over-fit [1, 14]. This can enable HDC models to learn how to rank the performance of a large set of unseen architectures with only a handful of training samples, which is usually the realistic case [2, 7]. The main contributions of this paper are as follows:

- In this paper, we propose **PerfHD**, a supervised ViT performance ranking algorithm using HDC. **PerfHD** can efficiently rank the performance of architectures given the configurations of ViTs. To the best of our knowledge, this is the first work to leverage HDC for NAS.
- We propose two different HDC encoding schemes (Gram and Record) based on the architecture of ViT and evaluate and compare their performance. We also propose retraining methods based on weight-update to enhance model performance after initial training.
- We evaluate **PerfHD** performance on the architectures curated from VIMER-UFO benchmark of eight computer vision applications from different domains and compare with four SOTA baselines. Experimental results show that **PerfHD** can rank around 100K architectures in about 1 minute, which is up to $10X$ faster than SOTA methods.

2. Related Works

Different machine learning algorithms are proposed to estimate and rank performance of sampled architectures during NAS, particularly given there is a limited training dataset, e.g., just a few architectures with their performance. For example, widely used gradient boosting algorithms for ranking such as LightGBM and Catboost have been applied to rank the ViT architecture performance where the hyper-parameters of ViT models are used as categorical features [7]. **TF-TAS** leverages two theoretical perspectives of synaptic diversity and synaptic saliency to evaluate and rank ViT architectures which speeds up the architecture search by up to 48X [22]. On the other hand, **GP-NAS** leverage the capability of Gaussian process and mutual information to accurately model the performance correlations using a small amount of samples [11]. Ensemble learning is also used to enhance the **GP-NAS** performance, however as ensemble learning usually incorporates a large set of models and require extensive hyper-parameter fine-tuning [2]. This drastically increases the overhead of the algorithm such as run-time by orders of magnitude, which can potentially offset the benefit of efficiency. Therefore, it is of great significance to explore methods that are able to rank the performance of ViT models with desirable accuracy and efficiency.

3. Hyperdimensional Computing

3.1. Notions and Operations

3.1.1 Hypervector

Hypervectors (HV) are the fundamental elements of an HDC model. HVs are numerical vectors with several specific characteristics. 1). **high-dimensional**: the dimension (the amount of elements) of HVs is extremely high, usually reaching 10,000 and above; 2). **holographic**: each HV is recognized as the minimal unit in an HDC model, individual number inside HV does not own unique representation; 3). **(pseudo-)random**: for the initial generated HVs, the numbers are i.i.d., thus two randomly generated HVs are approximately orthogonal to each other due to the extremely high dimensionality. We use $\vec{V} = (v_1, v_2, ..., v_D)$ as the notion of a D-dimensional HV in which v_i is the i-th number inside the HV. For differentiation purposes, HVs are marked with arrows in this paper, while other vectors such as features are bold instead.

3.1.2 HV Operations

In reality, the information can originate from diverse modalities, and different sources of information can correlate. To aggregate information of the same modality or to combine information from different modalities to also provide

hierarchy of information, we use HDC operations. The three mostly used HDC operations are addition, multiplication and permutation. The addition and multiplication operations take two HVs as inputs and perform linear and element-wise vector operations correspondingly. The permutation operation takes only one HV as input and perform cyclic shift of a specific amount n. Note that the dimensions of input HVs and output HVs are the same for all the three operations.

Bipolarization (or binarization) is another important operation for HV beyond the three basic operations. Bipolarization takes the sign bit of the HV elements that any number larger than 0 is bipolarized into 1, while any number smaller than 0 is bipolarized into -1. Binarization is similar, but numbers are binarized into 1 and 0 instead. Bipolarization and binarization introduces additional non-linearity into HDC models, and also to limit the range of elements inside each HV to prevent overflow issues during aggregation. In **PerfHD**, we slightly modify the bipolarization behavior that any number larger than 1 or smaller than -1 will be capped into 1 and -1 while the numbers within $(-1, 1)$ are kept as-is.

3.1.3 Similarity

With HVs representing information and HDC operations aggregating and combining information, there is then a need of metric that can quantitatively measure the similarity between information that different HVs accommodate. Cosine similarity is one of the most frequently used metrics while other similarities such as Euclidean distance and Hamming distance can also be used. A higher cosine similarity indicates that the two HVs compared share more similar information, thus are more alike.

3.2. HDC Memories

HDC leverages memories to host information. HDC memories are specialized clusters of HVs with different objectives during a learning task. There are two major categories of HDC memories: the item memory and the associative memory. Item memories are related with the input realistic features: each item memory hosts item HVs at the same amount of possible feature values K of the corresponding features as shown by $\mathbb{I} = \{\vec{I_1}, \vec{I_2}, ...\vec{I_K}\}$. If the feature is a continuous variable, quantization can be applied beforehand to avoid item memories of infinite size. Associative memory, on the other hand, is related to the output of the model. For a classification task, the associative memory hosts class HVs, each of which represents a class of the task, as shown by $\mathbb{A} = \{\vec{A_1}, \vec{A_2}, ...\vec{A_T}\}$. For this paper, since we are ranking the performance of various ViT architectures, the associative memory of **PerfHD** hosts an HV aiming to represent high performance architectures. Details of how the associative memory can help with ranking architecture performance are present in Sec. 4.

4. PerfHD Methodology

4.1. Problem Formulation and Motivation

In NAS, architectures are usually sampled from a supernet. Instead of training every sampled architecture to obtain the performance, designers usually have a small set of architectures trained with their performance ranked, and want to rank a larger set without actually training them. Additionally the algorithm or method to rank the larger set of architecture should be efficient and fast. We found that two highlighted characteristics of HDC match the requirements and can potentially enable a new direction of addressing this problem: 1. HDC is known for its efficiency over existing machine learning algorithms such as neural networks, including smaller model size, faster model convergence and less computational intensity [10, 13, 17]. 2. HDC is better at learning with limited data such as one-shot or few-shot learning tasks and are less likely to over-fit [1, 14].

4.2. Overview of PerfHD

An overview of **PerfHD** is present in Fig. 2. **PerfHD** first iterates through the training set, takes parameters of all the architectures and encodes them into the corresponding HVs referred to as the architecture HVs. **PerfHD** also converts the task rankings into weights of each architecture in the training set. **PerfHD** uses the weights to perform weighted sum to establish the associative memory which accommodates the task HVs. In brief, architectures with higher rankings are assigned with larger weights thus having more information impact on the task HVs after the weighted sum. When predicting rankings of architectures from the test set, **PerfHD** performs the same architecture encoding to obtain the architecture HV. Then, **PerfHD** computes and checks the similarity between the HV and each task HV in the associative memory. Once the similarity metrics of all the architectures from the test set are obtained, **PerfHD** can obtain the ranking of the similarity metrics as the predicted rankings. To enhance **PerfHD** performance, we also propose "retraining by weight-difference" to update the associative memory accordingly.

4.3. Feature Description

The features used in **PerfHD** are the architectural parameters of the ViT model. The available training data consists of three architectural parameters including the depth of encoders ($depth$), number of attention heads ($\#head$) and the dilation ratio of the MLP (mlp_ratio) of each layer. Each parameter has three possible values: $depth$: $\{10, 11, 12\}$; $\#head$: $\{10, 11, 12\}$; mlp_ratio: $\{3.0, 3.5, 4.0\}$. Therefore, for each encoder, there are 3×3 possible parameter

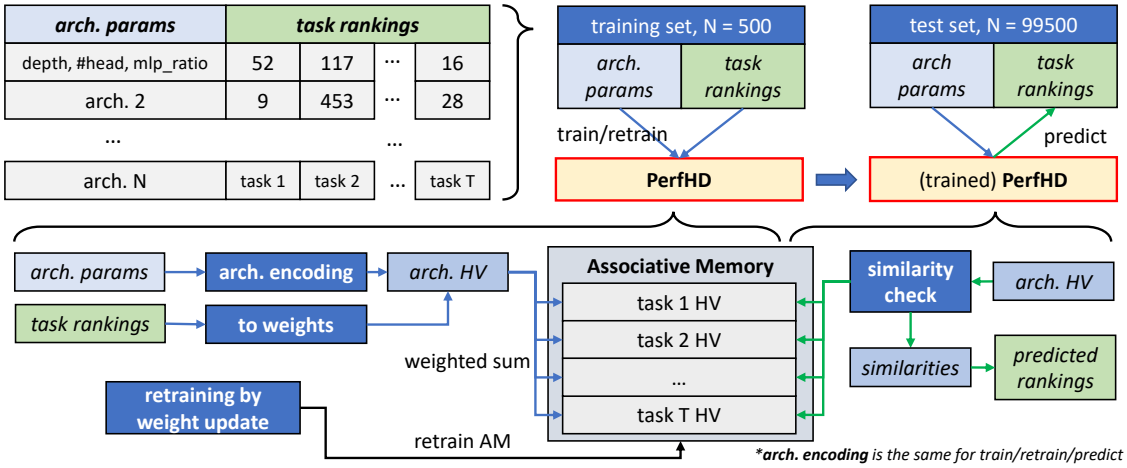

Figure 2. Training **PerfHD** to rank ViT performance based on architecture parameters.

combinations, so the number of possible architectures in the search space are $(3 \times 3)^{10} + (3 \times 3)^{11} + (3 \times 3)^{12}$. For convenience in data processing, the parameters are (label-)encoded into numbers of $\{1, 2, 3\}$. The labels are the performance rankings of the architecture on different tasks. Smaller number refers to higher performance, e.g., a ranking of 0 means this architecture is the best-performing architecture of this task.

4.4. ViT Architecture Encoding

In this subsection, we introduce the architecture encoding in detail. Specifically, we propose two schemes of encoding: Gram-based encoding and Record-based encoding, as indicated by Fig. 3. Note that the architecture HVs $\vec{V}$ are initialized with 0.

4.4.1 Gram-based Encoding

Gram-based encoding recognizes each block inside a ViT as a "gram" and groups parameters by tuples based on which block they belong to. It features two item memories: the $\#head$ memory and the mlp_ratio memory. Since each parameter has three possible values as introduced in Sec. 4.3, each item memory hosts three item HVs, each representing a possible value in the high-dimensional space.

Gram-based encoding first obtains the corresponding item HVs from the item memory based on the parameter tuples of $\#head$ and mlp_ratio. The two indexed HVs are then aggregated by HV multiplication, forming the gram HV. Then, the gram HVs are permuted where the shift amount is based on the depth index of the encoder, i.e., encoders closer to the output of the ViT model are shifted for more dimensions, or vice versa. The permuted gram HVs are summed up into the architecture HV $\vec{V}$. Encoded HVs are bipolarized (or binarized).

Note that the Gram-based encoding can be paralleled, since the encoding within each gram is independent from each other, and the shift amount of permutation is solely dependent on the depth index. A major disadvantage of Gram-based encoding, however, is that the permutation operation is not very straightforward for implementation of acceleration, since the cyclic rotation of high dimensional vectors are not linear vector operations like addition and multiplication. Therefore, Record-based encoding is an alternative to the Gram-based encoding which eliminates the use of permutation, but only keeps addition and multiplication of HVs.

4.4.2 Record-based Encoding

Record-based encoding uses one additional item memory, the depth memory as an alternative to the gram-based permutation. The depth memory hosts item HVs at the number of depth index, therefore, there are 12 item HVs in the depth memory given the max possible depth is 12 for this dataset. Within each encoder, the HV aggregation is the same as Gram-based encoding. However, instead of permutation by the depth index, the HV of each encoder is multiplied by the item HV of the corresponding depth, indexed from the depth memory. Then the HVs are summed up to obtain the architecture HV, which is also bipolarized (or binarized). In realistic implementation, based on the commutative properties of multiplication, the item HVs in the item memories can encode first and stored as a unified item memory to avoid repetitive index and HV multiplications.

4.5. PerfHD Training

Training is to establish the associative memory that accommodates the task HVs which are initialized with zeros. In **PerfHD**, training is the weighted sum of the HVs

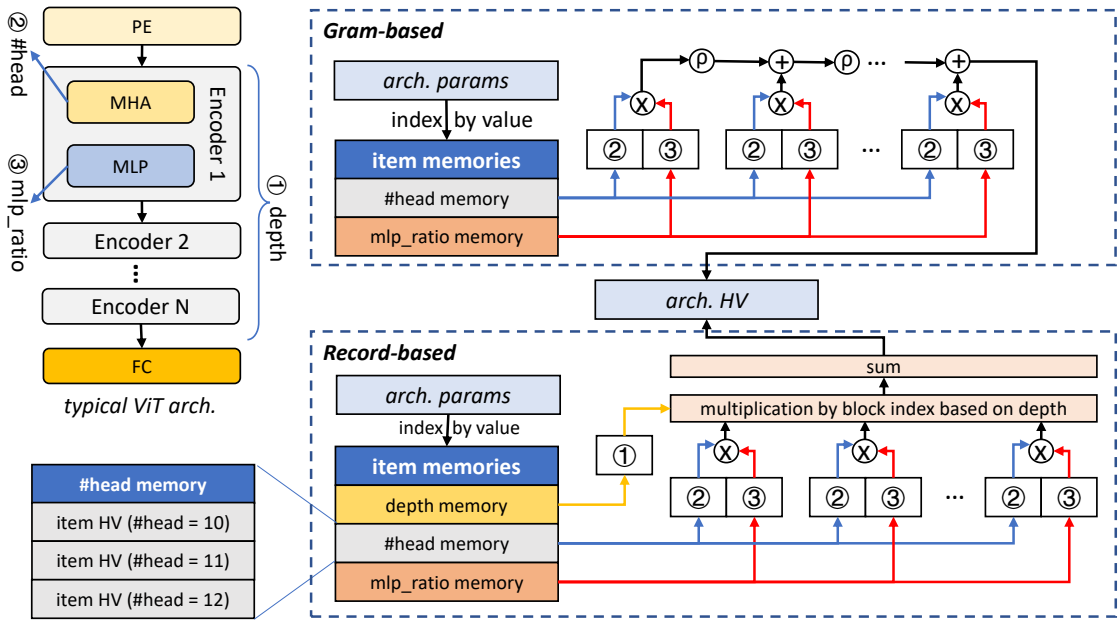

Figure 3. **PerfHD** encoding schemes to encode the ViT parameters into an HV. Two schemes are introduced: Gram-based and Record-based encoding.

obtained from the encoding phase as described in Eq. (1), where $\vec{A}_t$ is the HV of task t in the associative memory, $w_n^{(t)}$ is the task t weight of the n-th architecture in the training set, and $\vec{V}_n$ is the encoded HV of the n-th architecture as well. Architectures with higher ranks of a specific task are assigned with higher weights, while architectures with lower ranks are assigned with lower weights. A custom learning rate γ can be specified, however, during training we use the constant 1. The objective of training is to incorporate more information about high performing architectures into the task HV, while still trying to consider information from others HVs as much as possible to avoid over-fitting, since there are only 500 architectures in the training set.

$$\vec{A}_t = \vec{A}_t + \gamma \times w_n^{(t)} \times \vec{V}_n \quad (n = 1, 2, \cdots, N) \tag{1}$$

Intuitively, we convert the ranking into weights using a straightforward inverse number method as Eq. (2). $\boldsymbol{w_n} = \{w_{n1}, w_{n2}, \cdots, w_{nT}\}$ is the weight vector of the n-th architecture, in which w_{it} refers to the weight of t-th task. $\boldsymbol{r_n} = \{r_{n1}, r_{n2}, \cdots, r_{nT}\}$ is the ranking vector from the training set, with rankings of each task correspondingly, as shown in Fig. 2. μ is a constant scaling factor and we use 1.0 during the experiments. Therefore, worse-than-average performing architectures are assigned with negative weights and better-than-average ones are assigned with positive weights.

$$\boldsymbol{w_n} = \mu(\mathbf{1} - \frac{\mathbf{2}}{\boldsymbol{r_n} + \mathbf{1}}) \tag{2}$$

4.6. PerfHD Prediction

Instead of directly predicting the actual ranking, **PerfHD** uses the similarity metrics to determine the ranking of the test set. For an architecture with unknown ranking, **PerfHD** first encodes its parameters into the architecture HV using the same item memories and encoding scheme for training. Then, **PerfHD** checks the similarity between the architecture HV and the task HVs in the associative memory. For each task, the similarity of all the architectures are recorded and then used to rank the task performance. Specifically, as the task HV in the associative memory is the aggregation of architectures with high performance, therefore, a higher similarity will then be predicted with higher ranking by **PerfHD**.

4.7. PerfHD Retraining

In HDC, training usually takes one epoch (a full iteration of the entire training set). Additional epochs of retraining (still using the training set) can be optionally performed to enhance the performance of the model. HDC models for classification tasks leverages prediction labels to update the associative memory when a mis-classification is identified [17]. However, **PerfHD** targets at a ranking task, therefore we propose a novel retraining methodology referred to as **retraining by weight update** to increase the consistency of predicted rankings and the ground-truth.

PerfHD retraining is based on updating the associative memory by weight-difference. The main concept is to make sure the similarity ranking of each architecture is consis-

tent with the ground-truth ranking to minimize the Kendall tau [9] score difference between the prediction and the ground-truth. First after the training epoch, **PerfHD** calculates the similarity metrics of all the N architectures in the training set which is referred to as $\boldsymbol{\delta} = \{\delta_1, \delta_2, \cdots, \delta_N\}$. Based on the similarity metrics we are able to obtain the predicted rankings $\hat{\boldsymbol{r}} = \{\hat{r}_1, \hat{r}_2, \cdots, \hat{r}_N\}$, which can be subsequently converted into the speculated weights $\hat{\boldsymbol{w}} = \{\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_N\}$ according to Eq. (2).

We use the difference between the speculated weights and the ground-truth weights $\boldsymbol{\Delta w} = \{w_1 - \hat{w}_1, w_2 - \hat{w}_2, \cdots, w_N - \hat{w}_N\}$ to perform training again to update the associative memory based on Eq. (1). A negative weight difference means the architecture is "under-ranked" and its HV should be added into the task HV during retraining, on the other hand, a positive difference means the architecture is "over-ranked" and its HV should be subtracted instead. Retraining can iterate for multiple epochs and can be stopped when the (average) difference is smaller than a threshold to prevent over-fitting. We use the starting learning rate of 1 for consistency with training, however, for additional epochs in retraining we apply a decay of 0.8.

5. Experimental Results

5.1. Experimental Setup

The dataset of architecture performance is curated from the multi-task VIMER-UFO benchmark, with eight different computer vision tasks: CPLFW, Market1501, DukeMTMC, MSMT-17, Veri-776, VehicleId, VeriWild, and SOP [19], provided by the Second Lightweight NAS Challenge[1]. There are 500 architectures in the training set with performance ranked on each of the task, and 99,500 for ranking prediction as the test set. We implement **PerfHD** using PyTorch and evaluate **PerfHD** with one NVIDIA P100 GPU. We also compare **PerfHD** with four baseline methods:

- GP-NAS [11], is Gaussian Process Neural Architecture Search where the correlation between performances and architectures and the correlation between different architectures are explicitly modeled. An efficient sampling method is also proposed which enables GP-NAS learning on a small set of samples.
- LightGBM [7, 8], which is a widely adopted gradient boosting decision tree.
- CatBoost [4], is another gradient boosting baseline but with categorical features support.
- GP-NAS Ensemble [2], which is an ensemble version of GP-NAS [11], it applies enhancements such as additional feature engineering, label transformation and weighted ensemble kernels.

[1] https://aistudio.baidu.com/aistudio/datasetdetail/134077

$$\tau = \frac{n_c - n_d}{n(n-1)/2} \tag{3}$$

Kendall tau score is used to describe the consistency between the model prediction and the ground-truth, a higher scores means the ranking predictions are more accurate. Kendall tau τ between two vectors of ranking can be calculated by Eq. (3), where n_c and n_d are the number of concordant and discordant pairs, respectively and n is the total number of pairs [9].

5.2. Comparison on Accuracy

We present the comparison between **PerfHD** and the baselines on the VIMER-UFO benchmark in Tab. 1. Based on this table, we observe several important facts. First, we want to mention that the average performance across all models can drastically vary amongst different datasets. For CPLFW, all the models are having significantly lower score than the other 7 tasks, making this face related dataset the most challenging task in this benchmark. MSMT, on the contrary, is the easiest task amongst all that all the methods evaluated can achieve over 0.75 in score.

Moreover, GP-NAS and LightGBM are the two subperforming baselines as their average scores are less than 0.7. On the other hand, CatBoost and GP-NAS Ensemble can achieve higher accuracy with scores achieving over 0.785. As to HDC-based methods, **PerfHD**-Record outperforms **PerfHD**-Gram on all the tasks and in general shows a superior score by around 0.03. We discuss that such advantage comes from the additional depth memory in **PerfHD**-Record, which provide an additional hyperdimensional space to represent different encoder blocks compared with **PerfHD**-Gram which only uses permutation in fixed amount of vector shift.

Comparing **PerfHD** with other baselines, we can notice that across all the methods evaluated, **PerfHD**-Record is able to achieve the second highest score on average, and is only 0.0079 lower than the GP-NAS Ensemble and around 0.0050 higher than the third place Catboost. Specifically, **PerfHD**-Record achieves 0.6634 score on VehicleId, which is the highest score of this task across all the compared models. For VeriWild, **PerfHD**-Record also achieves near-top performance of around 0.92. **PerfHD**-Gram on the other hand, has relatively inferior performance compared with Catboost and GP-NAS Ensemble by 0.0243 and 0.0370 respectively, however still performs better than GP-NAS and LightGBM for over 0.08.

5.3. Comparison on Efficiency

We also compare the execution time of **PerfHD** and baselines to show the efficiency of **PerfHD**. The execu-

Table 1. Comparison of Models on Kendall Tau Scores of the VIMER-UFO Benchmark

Model	Average	CPLFW	Market	MTMC	MSMT	Veri	VehicleId	VeriWild	SOP
GP-NAS	0.6196	0.2350	0.7391	0.7052	0.8063	0.6319	0.4012	0.6731	0.7654
LightGBM	0.6810	0.2755	0.7742	0.7723	0.7599	0.7469	0.5900	0.7860	0.7433
Catboost	0.7851	**0.3220**	0.8687	0.8883	0.9430	0.8930	0.6576	0.9099	0.7980
GP-NAS Ensemble	**0.7978**	0.3188	**0.8864**	**0.9045**	**0.9678**	**0.9106**	0.6624	**0.9199**	**0.8119**
PerfHD-Gram	0.7608	0.2927	0.8489	0.8580	0.9004	0.8571	0.6598	0.8819	0.7874
PerfHD-Record	0.7899	0.3074	0.8754	0.8945	0.9555	0.8974	**0.6634**	0.9195	0.8060

Table 2. Comparison of Execution Time of the VIMER-UFO Benchmark

Model	GP-NAS	LightGBM	CatBoost	GP-NAS Ensemble	**PerfHD**-Gram	**PerfHD**-Record
Execution Time (second)	92	69	80	>600	425	**62**

tion time for training the model and making predictions are listed in Tab. 2. Less sophisticated models such as GP-NAS and LightGBM are faster to learn and predict, however their performance is relatively sub-par. CatBoost, based on the time reported, uses relatively short execution time. However, to achieve such enhanced performance, it requires extensive parameter tuning which also takes much longer time than just training and prediction. This parameter optimization is required for each task, which also limits the flexibility of this algorithm. GP-NAS Ensemble, the more complicated model, achieves the highest performance overall however during our evaluation, it spends way more than 10 minutes for training and prediction since each of the ensemble models require intensive training effort. We mark the execution time with >600 since it is already much time-consuming than most of the baselines.

The time reported for both of the **PerfHD** implementations include the entire process of all the tasks, i.e., no additional pre-training or fine-tuning is required. **PerfHD**-Gram requires considerably long processing time compared to most of the baselines, as the permutation operations of HVs occupy most of the time spent. As **PerfHD**-Gram also shows sub-par ranking scores, so we conclude that **PerfHD**-Record is preferred for this benchmark. Specifically, **PerfHD**-Record can finish training and prediction together with around 1 minute, which is the fastest method across all the compared baselines and is more than $10X$ faster than GP-NAS Ensemble which is the baseline with highest score.

5.4. Impact of HV Dimension

We also present a case study on the Record-based encoding to analyze the impact of using different HV dimensions on **PerfHD** performance. In related HDC literature, dimensions of 10,000 – 20,000 is usually used as the dimension range and higher dimensions may not guarantee a higher performance. Therefore, we choose 10,000 as the starting point of HV dimension analysis on **PerfHD**. However, we observed that the dimension of HVs can be increased to over 100,000 without performance saturation. Experiments show that a dimension around 100,000 achieves the best performance across all the dimensions evaluated, as shown in Fig. 4.

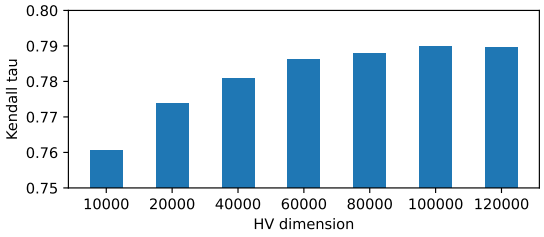

Figure 4. **PerfHD**-Record task-average scores under different HV dimensions.

5.5. Discussion

The efficiency of **PerfHD** comes from the simplicity of HDC training and retraining, which follows a concise linear operations with matrices (record-based encoding). This can hugely benefit from the high parallelism of GPU processing that significantly accelerates the **PerfHD**. **PerfHD** also does not need back-propagation like neural networks, which requires maintaining a complicated computation graph. Instead, **PerfHD** retraining is via the update to the associative memory which is also a weighted sum with much less computational overhead. Another advantage is that when evaluating different tasks, **PerfHD** can share the same item memory and encoding process. This grants a tremendous advantage over other baselines that different tasks are required to train individual models or apply relatively more complicated transfer processing.

6. Conclusion

Efficiency of neural architecture search (NAS) algorithms has been one major bottleneck for this automated model engineering process. Specifically, the performance estimation of searched architectures often requires significant effort. How to efficiently and accurately rank a large set of ViT model performance given a small amount of training set is a realistic and important problem. In this paper we leverage the hyperdimensional computing (HDC) computing and propose **PerfHD** for ViT performance ranking. On the VIMER-UFO benchmark with eight different tasks, **PerfHD** is able to achieve top-level results yet also has tremendous acceleration compared with four baselines. The future work will focus on exploring more encoding schemes and hardware-specific designs for further acceleration and/or more energy-efficiency.

References

[1] Alessio Burrello, Kaspar Schindler, Luca Benini, and Abbas Rahimi. One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing. In *BioCAS*. IEEE, 2018. 2, 3

[2] Kunlong Chen, Liu Yang, Yitian Chen, Kunjin Chen, Yidan Xu, and Lujun Li. Gp-nas-ensemble: a model for the nas performance prediction. *Third workshop on Neural Architecture Search*, 2022. 2, 6

[3] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv:2102.11535*, 2021. 1

[4] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv:1810.11363*, 2018. 6

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv:2010.11929*, 2020. 1

[6] Eman Hassan, Yasmin Halawani, Baker Mohammad, and Hani Saleh. Hyper-dimensional computing challenges and opportunities for ai applications. *IEEE Access*, 2021. 2

[7] Di He. Skt-nas: Soft kendall's tau based neural architecture search. *Third workshop on Neural Architecture Search*, 2022. 2, 6

[8] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *NeuIPS*, 30, 2017. 6

[9] Maurice George Kendall. Rank correlation methods. 1948. 6

[10] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *DATE*, pages 115–120. IEEE, 2020. 2, 3

[11] Zhihang Li, Teng Xi, Jiankang Deng, Gang Zhang, Shengzhao Wen, and Ran He. Gp-nas: Gaussian process based neural architecture search. In *CVPR*, pages 11933–11942, 2020. 1, 2, 6

[12] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *TNNLS*, 2021. 1

[13] Dongning Ma, Rahul Thapa, and Xun Jiao. Molehd: Drug discovery using brain-inspired hyperdimensional computing. *arXiv preprint arXiv:2106.02894*, 2021. 2, 3

[14] Abbas Rahimi, Pentti Kanerva, Luca Benini, and Jan M Rabaey. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals. *Proceedings of the IEEE*, 107(1):123–143, 2018. 2, 3

[15] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *CSUR*, 54(4):1–34, 2021. 1

[16] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pages 2820–2828, 2019. 1

[17] Rahul Thapa, Bikal Lamichhane, Dongning Ma, and Xun Jiao. Spamhd: Memory-efficient text spam detection using brain-inspired hyperdimensional computing. In *ISVLSI*, pages 84–89. IEEE, 2021. 2, 3, 5

[18] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. Theoretical foundations of hyperdimensional computing. *JAIR*, 72:215–249, 2021. 2

[19] Teng Xi, Yifan Sun, Deli Yu, Bi Li, Nan Peng, and Gang Zhang. Ufo:unified feature optimization. In *ECCV*, 2022. 6

[20] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *CSUR*, 54(9):1–37, 2021. 1

[21] Yibo Yang, Shan You, Hongyang Li, Fei Wang, Chen Qian, and Zhouchen Lin. Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search. In *CVPR*, pages 6667–6676, 2021. 1

[22] Qinqin Zhou, Kekai Sheng, Xiawu Zheng, Ke Li, Xing Sun, Yonghong Tian, Jie Chen, and Rongrong Ji. Training-free transformer architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10894–10903, 2022. 1, 2