MetaLDC: Meta Learning of Low-Dimensional Computing Classifiers for Fast On-Device Adaption

ABSTRACT

Fast model updates for unseen tasks on intelligent edge devices are crucial but also challenging due to the limited computational power. In this paper, we propose MetaLDC, which meta trains brain-inspired ultra-efficient low-dimensional computing classifiers to enable fast adaptation on tiny devices with minimal computational costs. Concretely, during the meta-training stage, MetaLDC meta trains a representation offline by explicitly taking into account that the final (binary) class layer will be fine-tuned for fast adaptation for unseen tasks on tiny devices; during the meta-testing stage, MetaLDC uses closed-form gradients of the loss function to enable fast adaptation of the class layer. Unlike traditional neural networks, MetaLDC is designed based on the emerging LDC framework to enable ultra-efficient inference. Our experiments have demonstrated that compared to SOTA baselines, MetaLDC achieves higher accuracy, robustness against random bit errors, as well as cost-efficient hardware computation.

KEYWORDS

High-dimensional computing, low-dimensional computing, meta learning, fast adaption, scalability

ACM Reference Format:
. 2022. MetaLDC: Meta Learning of Low-Dimensional Computing Classifiers for Fast On-Device Adaption. In *Proceedings of tinyML Research Symposium (tinyML Research Symposium'22).* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

Deep neural networks (DNNs) have become the backbone of intelligent applications in a wide range of domains from computer vision to natural language processing [1, 5, 23]. Meanwhile, compared to cloud-based inference, on-device inference has numerous advantages, including better privacy preservation and anytime inference without relying on network connections. Nonetheless, despite the recent progress [25, 33], directly running DNN inference and adapting the model to unseen tasks on the edge are still challenging due to the conflict between high computational demand of DNNs and the low resource availability of edge devices, especially tiny devices such as microcontrollers and Internet-of-Things (IoT) devices.

In response to the excessive resource demand of DNNs, hyperdimensional computing (HDC) has emerged as an alternative towards efficient on-device inference [18]. The key idea of HDC is to encode data into (binary) hypervectors each with dimensions of thousands or even more, and then perform cosine/Hamming distance similarity for inference using bit-wise binary operations in parallel. Owning to its hardware friendliness and efficiency, HDC classifiers have been adopted to an increasingly broader range of inference tasks for resource-constrained devices [6, 10].

Nonetheless, there are still fundamental limitations that prohibit the applicability of HDC for tiny devices with extremely limited resources. First, the orders of megabyte of memory required by HDC to support its hyperdimensional data representation can be too costly for tiny devices [11, 17]. Second, compared to today's DNNs, the HDC training process is extremely rudimentary (e.g., simply taking average of the input data, plus some semi-blind heuristic adjustments, without loss functions), resulting in low inference accuracy. Last but not least, each HDC training process can only fit into one data distribution, which means that HDC training does not scale to a large number of tiny devices each having potentially different distributions.

While the recent brain-inspired low-dimensional computing (LDC) classifiers outperform HDC by utilizing ultra-low dimensional vectors and principled training based on an equivalent neural network to improve the inference efficiency and accuracy [11], it cannot support fast model adaptation to unseen tasks on tiny devices. More concretely, LDC trains an individual model for each device, and hence the total training cost can be labor-intensive when there are many tiny devices deployed in heterogeneous environments each with a different data distribution [22]. Additionally, fast adaption to unseen tasks with only a handful of data points presents substantial challenges for tiny devices, due to their constrained computing resources that prohibit traditional model updates based on gradients and backpropagation. While some studies have proposed to train a collaborative (DNN) model in a distributed manner to facilitate knowledge transfer between edge devices [22], the communication latency among edge nodes and complicated local neural network computation make this approach still too expensive for a tiny device.

In the presence of heterogeneous tiny devices each with a different data distribution, we propose MetaLDC, a new LDC-based meta learning approach that achieves fast model adaptation to an unseen task and ultra-efficient inference on tiny devices. Specifically, MetaLDC meta trains an LDC-equivalent neural network, in which the first few layers are non-binary values specifically for LDC data encoding and the last layer has binary weights (denoted by ϕ) for classification. Crucially, given the difficulty in calculating gradients and performing backpropagation over the entire LDC-equivalent neural network on resource-constrained tiny devices, we only treat the last layer ϕ as task-specific: the LDC data encoding part is explicitly learned to fit heterogeneous distributions, while only the classification layer ϕ is adapted using a simple closed-form gradient expression to each tiny device for fast and inexpensive model update with minimal on-device computational costs. Thus,

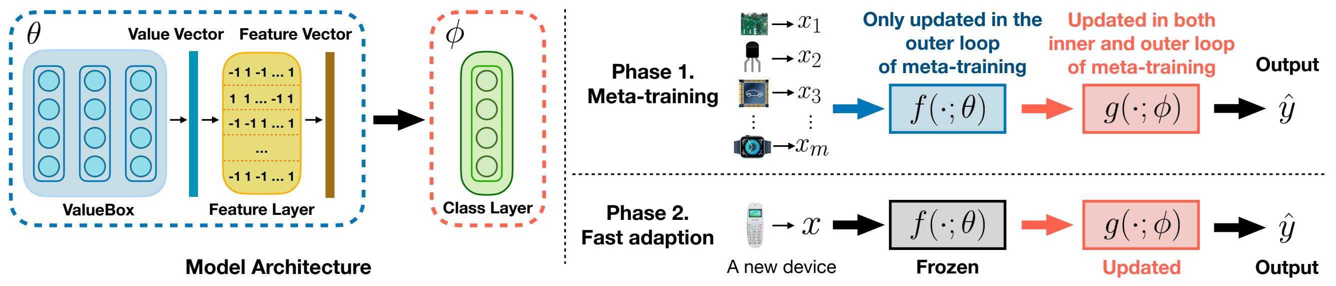

Figure 1: Overview of MetaLDC. Our MetaLDC is based on the ultra-lightweight LDC network and composed of two stages: meta-training and fast adaption. The meta parameter θ keeps learned knowledge from heterogeneous data distributions, while the task-specific parameter ϕ is used to quickly adapt to unseen tasks from a new tiny device.

unlike the standard meta learning technique (e.g., MAML [12]), MetaLDC meta trains the first few layers in the LDC-equivalent neural network by explicitly considering that these layers will be fixed without updates on tiny devices.

Our experimental results have empirically shown that MetaLDC can significantly outperform state-of-the-art (SOTA) baselines including SOTA HDC [19] and pretrained LDC model on both vision and non-vision datasets in terms of accuracy, robustness against the random bit errors on hardware, as well as cost efficiency including energy consumption, latency and model size.

2 BACKGROUND

Hyper-dimensional computing (HDC): The HDC is a brain-inspired, cognitive computing architecture built on a unique data type, referred to as the hypervectors. The dimensionality of the hypervectors can be from a thousand to tens of thousands. By manipulating these hypervectors, the HDC aims to perform cognitive tasks via hardware-efficient operations like element-wise additions and dot products.

In the general supervised classification based on HDC, the features of the input sample would be encoded into hypervectors $\mathcal{F}_i$s, together with their corresponding value hypervectors $\mathcal{V}_{f_i}$s, pre-stored in the item memory. By binding $\mathcal{F}$ and $\mathcal{V}$ via:

$$sgn(\sum_i \mathcal{F}_i \times \mathcal{V}_{f_i}),$$

we can obtain the encoded input sample $\mathcal{H}$. In training, all $\mathcal{H}$s belonging to the same class would be summed and averaged to obtain the class hypervectors, stored in the associative memory. In the inference stage, the testing data would be transformed into query hypervectors using the same encoder. Then a similarity checker like the Hamming distance would be applied in the associative memory between each trained class hypervectors and the query hypervector. The class label with the closest distance would be finally returned. Due to the simplicity of bitwise operations, the HDC has achieved success on platforms like FPGA and ASIC [6].

However, the large model size resulted from the ultra-high dimensions of the data representation in the HDC compromises its wide adoption on tiny devices, which are usually under severe resource consumption constraints. On the other hand, although numerous endeavors have been put to improve the accuracy of the HDC classifier [19], there is still a large accuracy gap between the HDC models and a simple modern neural network model like the Multi-Layer Perceptron (MLP).

Low-Dimensional Computing (LDC): To overcome the fundamental limitations of low accuracy and inference efficiency in HDC, the low-dimensional computing (LDC) classifiers are proposed as a brain-inspired substitute of HDC classifiers with higher accuracy and order-of-magnitude better on-device inference efficiency, especially for tiny devices with intelligent needs. Unlike HDC, LDC classifiers utilize a rigorous and systematic training procedure, where the value vectors $\mathcal{V}$s and feature vectors $\mathcal{F}$s are explicitly optimized rather than being randomly generated. On the other hand, the required order of magnitude of the involved vectors dimension size in the LDC is only a few to tens to achieve a higher accuracy compared to the state-of-the-art HDC, e.g. 87.38% w/ $D = 8,000$ vs. 91.22% w/ $D = 4/64$ on the MNIST dataset [11].

The blue and orange boxes with dashed lines in Figure 1 provides an overview of the LDC model architecture. The ValueBox is an encoding network. It maps the feature values $\mathcal{F}_i$s of an input sample into a bipolar value vector $\mathcal{V}_{f_i}$. Followed is the feature layer, which is essentially a sparse binary neural network to bind the bipolar feature values $\mathcal{V}_{f_i}$ with the corresponding feature vectors $\mathcal{F}_i$ through the Hadamard product. The last class layer equivalently performs similarity checking, where the weights of the layer are collections of all class vectors. This layer outputs the score product for each class, and the class label with the highest score is taken as the classification result $\hat{y}$. It is worth noting that the inference in an LDC classifier is fully binary as the non-binary weights of ValueBox is not needed after training. Compared to HDC, LDC classifiers have been demonstrated as a more promising alternative due to its lightweight model and high inference accuracy to support intelligent agents in the tiny devices.

3 PROBLEM SETUP

We focus on the few-shot supervised learning in this work. Supervised learning learns a model that maps input data points $x \in \mathcal{X}$ which have a true label $y \in \mathcal{Y}$ to predictions $\tilde{y}$. A task $\mathcal{T}_i$ is composed of $(\mathcal{X}, \mathcal{Y}, L, q)$, where L is the task-specific loss function and q is the data distribution of $\mathcal{T}_i$. We assume all data points are drawn i.i.d. from q.

Given distributions over a set of tasks $p(\mathcal{T})$, we aim at learning a general representation function $f(\cdot;\theta)$ using a handful of data points of each class from them. The $f(\cdot;\theta)$ is essentially a representation learning network parameterized by θ, which can then fast adapt to previously unseen tasks in new devices by learning another adaption function $g(f(\theta);\phi)$ with little local data examples and (closed-form) gradient updates on ϕ with minimal computation. In a nutshell, we propose to train the LDC backbone as a reusable template to fast adapt to new tasks on tiny devices in the end.

With the goal to obtain a good initialization, we train the LDC model in a meta-learning manner. Our MetaLDC consists of two phases: meta-training and meta-testing (also referred to as fast adaption). We meta-train on m tasks $S_i \sim p(\mathcal{T})$, $i = 1, \cdots, m$ to learn the representation, and meta-test on a *different* task $T \sim p(\mathcal{T})$.

In the training process of MetaLDC, we introduce two updating loops as shown in our Algorithm 1. In a nutshell, the inner loop updates model parameters with respect to an individual task using K data points by one (or few) gradients steps, while the outer loop updates the entire model's parameters with regard to the loss after the inner loop updates.

In the fast adaption stage, we apply M-shot N-way evaluation, where N is the number of classes per task. We would use M data examples from each class of the new task to update the partial model, and then carry out the evaluation on the testing dataset from the new task.

4 THE DESIGN OF METALDC

In this section, we present our architecture, referred to as the MetaLDC. It uses a carefully-crafted interleaved training algorithm to train the LDC classifier. The primary objective is to achieve fast adaption to unseen (but related) tasks on edge tiny devices by updating partial learned model parameters with minimal computational cost. We provide an overview of the MetaLDC in Figure 1.

4.1 Meta Training

The original MAML [12] takes the *training-and-fine-tuning* pipeline, which optimizes θ and ϕ together in both meta-training and fast adaption. But, updating the non-binary weights θ can be too computationally expensive for tiny devices. Thus, in MetaLDC, we instead use the **training-and-probing** pipeline. The key idea of MetaLDC is to separate the representation function $f(\cdot;\theta)$ and the prediction function $g(\cdot;\phi)$ for meta-training and fast adaption on tiny devices. Thus, we only optimize the representation function $f(\cdot;\theta)$ in the meta-training, and optimize the prediction function $g(\cdot;\phi)$ in fast adaption.

Specifically, in the meta-training, for each update step, we first learn a randomly initialized prediction function $g(\cdot;\phi)$ to classify examples based on a given representation $f(\cdot;\theta)$. An updated parameter ϕ_i is obtained using K examples from the sampled tasks S_i through one (or more) gradient steps w.r.t. the loss on the sampled tasks. We then resample K new examples from each class in S_i and optimize the whole model w.r.t. (θ,ϕ) across those tasks from $p(\mathcal{T})$. The full algorithm is outlined in the Algorithm 1, where α, β represent tunable step size. Intuitively, the ϕ serves as the task-specific embedding, which modulates the behaviour of the model. In the outer loop updates, by considering how the errors on specific task changes with respect to the updated model parameters, we expect to obtain a model initialization such that small changes in the model parameters could lead to substantial performance improvement for any task. This improvement has been empirically attested in our evaluations of section 5.

Algorithm 1 MetaLDC— Training

Input: $\mathcal{T}$: the whole task set; t: number of outer gradient steps; m: Number of inner gradient steps (*i.e.*, number of sampled meta-training tasks); α, β: step size parameters.
Output: θ, ϕ

```
1:  Randomly initialize parameters $\theta, \phi$
2:  for $j$ in 1, 2, ..., $t$ do                                  ▷ outer loop
3:      for $i$ in 1, 2, ..., $m$ do                              ▷ inner loop
4:          Sample batches $B_i$          ▷ each batch $B_i$ contains $K$ examples for each class in $S_i \sim p(\mathcal{T})$
5:          Derive task-specific $\phi_i'$: $\phi_i' \leftarrow \phi - \alpha\nabla_\phi L(B_i;\theta,\phi)$
6:          Re-sample another batch $B_i'$ of the same batch size
7:      end for
8:      Update both parameters $\theta$, $\phi$: $(\theta,\phi) \leftarrow (\theta,\phi) - \beta\nabla_{\theta,\phi}\sum_{i=1}^{m} L(B_i';\theta,\phi_i')$
9:  end for
10: Return $\theta$, $\phi$
```

It is also worth noting that as the meta training involves potentially many tasks to learn a good initialization, the whole process of meta-training can be done on either the GPU or a powerful CPU. In contrast, adaptation is performed on the local tiny devices with minimal computation cost as we explained in the subsequent subsections.

4.2 Fast Adaption

Given the learned initialization, we can fast adapt to a new task T_i under a M-shot N-way setup. To adapt to a new task on each tiny device, we freeze the parameters of the representation network θ and only update the class layer's parameter ϕ by using only a few samples. By this means, we preserve the broad knowledge learned from various tasks in the representation network $f(\cdot;\theta)$. Moreover, updating the last layer ϕ rather than the entire model is far less costly and therefore more affordable for tiny devices which are usually with stringent resource constraints. Furthermore, we use the hinge loss instead of the commonly used cross-entropy for gradient updates, as the former requires less complicated arithmetic operations. Instead of requiring the model to compute gradients by itself, we feed the gradients of the hinge loss in a closed-form to the model directly using the Eqn. (1):

$$\nabla_{w_j} L_i = \begin{cases} -\sum_{j\neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0), & j = y_i \\ \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \cdot x_i, & j \neq y_i, \end{cases} \tag{1}$$

where Δ is the desired margin, w_{y_i} is the parameters corresponding to the correct class, whereas w_j is the rest of parameters, and x_i is the vector representation for the data example i. We outline the fast adaption process in the Algorithm 2.

Interestingly, compared to fine-tuning entire parameters of all layers in the fast adaption, our experiments results have shown the

Algorithm 2 MetaLDC - Fast adaption to a new task T

Input: θ, ϕ: learned model parameters; t: Number of gradient steps; γ: step size parameters.
Output: ϕ

```
1: Sample $M$ examples for each class of $T$
2: for $j$ in 1, 2, ..., $t$ do
3:     Update $\phi$ by: $\phi \leftarrow \phi - \gamma \nabla_{\phi} L(\cdot; \theta, \phi)$
4: end for
5: Return $\phi$
```

accuracy achieved by the MetaLDC, which only updates last layer's parameters, is not lower. This can be attributed to the alleviated meta over-fitting issue, as updating the entire model using a few data points from a task in the adaption stage can easily cause the model to overfit to this partial data distribution.

5 EVALUATION

In this section, we empirically evaluate the performance of MetaLDC compared to several baselines on both vision and non-vision datasets. We have considered various performance perspectives, including adaption accuracy on unseen tasks with a small amount of data points, robustness against hardware random bit errors, the latency and hardware energy consumption, as well as the change of model's accuracy w.r.t. the hyperparameters values selection. Additionally, we have also evaluated the efficacy of the learned representation $f(\cdot; \theta)$.

5.1 Setup

We describe the datasets and tasks, baselines, as well as implementation details here.

Datasets. We have leveraged two benchmarking datasets from both the vision and non-vision regimes. One is the Rotated MNIST, which is originated from the MNIST [7]. In Rotated MNIST, each image contains the digit rotated by a certain degree. The other is referred to as Split ISOLET, derived from the UCI ISOLET dataset [29]. The ISOLET contains 26 classes in total, and we divide them into separate tasks, where each task contains 4 unique randomly sampled classes.

Training and Evaluation Tasks. In the Rotated MNIST, the rotation degree of tasks we use to train the methods are between $[10^\circ, 20^\circ)$. The learned models are then evaluated on the testing dataset of Rotated MNIST with rotation degree from $\{0^\circ, 2^\circ, 4^\circ, 6^\circ, 8^\circ\}$. For the Split ISOLET, we use 20 classes from the ISOLET to generate the training tasks, while the remaining classes form the candidate classes pool to produce evaluation tasks.

Baselines. We compare MetaLDC with both HDC methods and other LDC-based models. In the *Pretrained LDC* method, we pretrain the LDC model with standard supervised learning using the whole training datasets from the training tasks. Then we also use the Algorithm 2 for fine-tuning. Another baseline we designed is the *MetaLDC-full*. The training algorithm of MetaLDC-full is the same as MetaLDC, as shown in the Algorithm 1. The difference is in the fine-tuning stage, where we update the parameters of the entire model, not only the last layer for the MetaLDC-full. Note that MetaLDC-full is prohibitively expensive for tiny devices, as it requires keeping the entire weights of the LDC model and full backpropagation calculations throughout the fast adaptation process. Besides the LDC variants, we also compare with a SOTA HDC method, the *HDC with retraining*. The HDC w/ retraining would give more weights to a misclassified sample in its correct class hypervector and subtracted from the wrong class hypervector in training to improve the HDC classification accuracy [17]. We set $D = 8,000$ for the HDC models in our experiments following the setup of [11]. In addition, we put the multi-layer perceptrons (MLP) here as an upper bound, although it is not feasible to be deployed on extremely resource-constrained tiny devices. In the MLP, we use Algorithm 1 for training and Algorithm 2 for fast adaption.

Implementation Details. In the meta training stage, for the Rotated MNIST, we train the model for 60 epochs with batch size of 10. In the Split ISOLET, we use 30 epochs w.r.t. its smaller dataset size. In the Rotated MNIST experiments, we used $K = \{1, 5\}$ for gradient updates in the meta learning involved methods, while setting $K = 1$ in the Split ISOLET. We train all models with the Adam optimizer [20] except the HDC classifier, which is not feasible to fit in any standard training optimizer.

In the fast adaption stage, we set $M = 10$ for the Rotated MNIST and $M = 5$ in Split ISOLET, to update ϕ. To ensure fair comparison, the same data points are used across different methods. Note that we don't fine-tune the HDC classifier as there seems no feasible way to update the learned hypervectors which are essentially composed of zeros and ones for the new incoming data examples.

5.2 Testing Accuracy

We show our evaluation results on the Rotated MNIST in Figure 2. From Figure 2, we can see that MetaLDC outperforms other baselines across all evaluation tasks. We observe that the MetaLDC has achieved higher accuracy compared to the MetaLDC-full, which updates the entire model parameters to adapt to a new task in the fine-tuning stage. We attribute this to over-fitting, as the entire model focuses on learning a very small amount of data from the task. In comparison, MetaLDC, which only updates the last layer while keeping the former layers untouched, has alleviated this over-fitting issue to certain extent. We can also observe that as the rotation degree of the evaluation data becomes larger, the testing accuracy of the MetaLDC also increases due to higher similarity between the training and the evaluation tasks.

For the Split ISOLET, the evaluation results are reported in the Figure 2. Based on the Figure 2, the accuracy achieved by MetaLDC is the highest on different tasks. The second highest is the MetaLDC-full, which is not as computationally efficient as the MetaLDC for tiny devices.

5.3 Robustness against Hardware Bit Errors

One of the most appreciated merits of the HDC-based models is the robustness against random bit errors on the hardware. The plain LDC model has been shown that it can achieve comparable robustness due to the uniform distributed information in each compact vector, although the dimensionality of the vector is largely reduced [11]. Our empirical results in Figure 3 have shown that

Methods	K-shot	Evaluation Tasks $T_1 : 0°$	$T_2 : 2°$	$T_3 : 4°$	$T_4 : 6°$	$T_5 : 8°$
HDC w/ retraining	-	$76.74_{\pm 0.74}$	$77.75_{\pm 0.65}$	$77.01_{\pm 0.57}$	$77.87_{\pm 0.44}$	$79.13_{\pm 0.53}$
Pretrained LDC	-	$78.25_{\pm 2.37}$	$78.26_{\pm 3.03}$	$78.67_{\pm 2.97}$	$79.25_{\pm 2.50}$	$80.89_{\pm 1.76}$
MetaLDC-full	$K = 1$	$80.03_{\pm 0.84}$	$81.01_{\pm 0.57}$	$82.37_{\pm 0.81}$	$84.04_{\pm 0.47}$	$85.34_{\pm 0.23}$
	$K = 5$	$80.35_{\pm 0.66}$	$81.78_{\pm 0.53}$	$83.25_{\pm 0.57}$	$84.97_{\pm 0.18}$	$86.31_{\pm 0.05}$
MetaLDC	$K = 1$	$82.78_{\pm 0.97}$	$82.99_{\pm 1.07}$	$84.84_{\pm 1.22}$	$86.33_{\pm 0.55}$	$87.97_{\pm 0.42}$
	$K = 5$	$82.83_{\pm 0.71}$	$83.11_{\pm 0.58}$	$85.74_{\pm 0.83}$	$86.54_{\pm 0.23}$	$88.01_{\pm 0.36}$
Upper Bound	$K = 1$	$87.00_{\pm 0.54}$	$87.69_{\pm 0.42}$	$88.77_{\pm 0.33}$	$90.19_{\pm 0.33}$	$92.94_{\pm 0.19}$
	$K = 5$	$87.53_{\pm 0.43}$	$87.74_{\pm 0.39}$	$89.01_{\pm 0.27}$	$90.87_{\pm 0.29}$	$93.11_{\pm 0.11}$

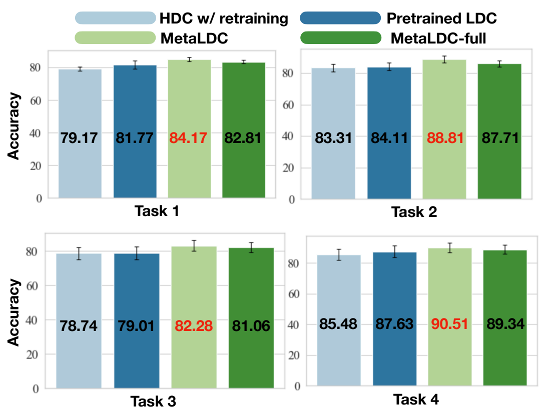

Figure 2: Left: Accuracy of MetaLDC compared to other methods on the Rotated MNIST. Right: Accuracy of MetaLDC compared to other approaches on the Split-ISOLET. Red marks the results of the highest accuracy, and blue marks the results of the second highest.

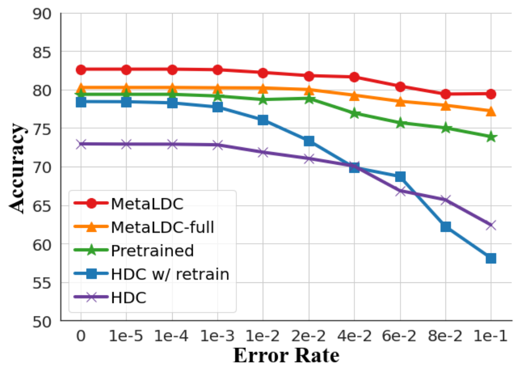

Figure 3: Bit error robustness of different models on the MNIST testing dataset.

MetaLDC exhibits even stronger robustness than other methods when the rate of bit error increases. The reason behind the improvement could be that a general representation is learned from a set of different tasks via our meta training process. The broad prior knowledge stored in the reusable template helps in adapting to tasks with perturbation, and fighting against the random bit errors. Compared to the pretrained LDC which also preserve acquired knowledge, MetaLDC has shown slightly slower decline of accuracy.

5.4 Inference Cost on Hardware

Here, we evaluate the inference efficiency of MetaLDC and HDC w/ retraining following the same hardware pipeline setup as in the [11]. The hardware platform we use is the Zynq UltraScale+, where we transform the bipolar values {1, -1} to {0, 1} in the implementation. In the experiment, we limit the resource usage (e.g., lookup table (LUT) < 10k) to approach the common practice in tiny devices.

We report the numerical results in the Table 1. From the Table 1, we can see that the LDC models are at least $100x$ faster than HDC w/ retraining classifiers. The model size based on LDC is $150x$ smaller than the HDC ones. The energy consumption of the MAML LDC (tiny) are less than $100nJ$ in the evaluation datasets, which has demonstrated great improvement on hardware acceleration compared to the HDC-based models. Note we don't measure the MLP-based model cost here, as its inference requires matrix multiplication via floating-point operators rather than simple binary arithmetic, making it too resource-intensive to run on a tiny device. On top of that, the MLP architecture cannot be trivially supported by the FPGA platform due to the floating point computation [13], and inter-platform comparison of algorithm performance is considered neither instructive or fair. Even though there is MLP with the fix-point format which could be implemented on FPGA, the required utilization of DSP and other resources are still tremendous to carry out the involved matrix multiplication, far above the resource budget of a tiny device.

Table 1: Inference cost comparison between MetaLDC and the HDC w/ retraining on the Zynq UltraScale+.

DataSet	Model	Size (KB)	Latency (us)	Energy (nJ)
R-MNIST	**MetaLDC**	**6.48**	**3.99**	**64**
	HDC	1050	499	36926
S-ISOLET	**MetaLDC**	**5.10**	**3.13**	**38**
	HDC	877	388	29488

5.5 Hyper-parameter Ablation Studies

To showcase the performance of MetaLDC with different choice of hyperparameters, we conduct the ablation study for K, the number of data examples to train the model; and M, the number of samples we used from each class of a new task to update ϕ in the fast adaption stage. Depending on different dataset size, for the Rotated MNIST, we set the $K = \{1, 5, 10, 20, 40\}$, while $K = \{1, 2, 3, 4, 5\}$ for the Split-ISOLET dataset. The M is set as $\{10, 50, 100, 150\}$ in the Rotated MNIST, and $\{1, 5, 10, 15\}$ in the Split ISOLET, respectively.

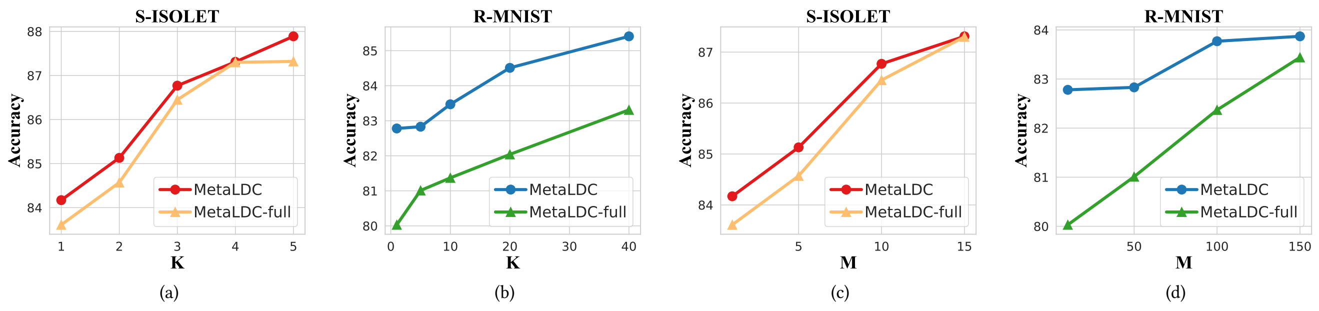

Figure 4: Hyper-paramter ablation study on different datasets w.r.t. K, the number of examples sampled from each class in the meta-training; and M, the number of examples sampled from each class of a new task in the fast adaption.

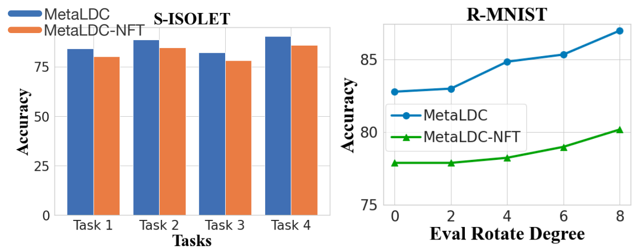

Figure 5: Accuracy comparison between MetaLDC and MetaLDC-NFT on S-ISOLET and R-MNIST with $K = 1$.

The evaluation dataset we used in the Rotation MNIST is the the original MNIST, while the images in the training data has rotation degree in $[10, 20)$. In the Split-ISOLET, we use the Task 1 as the evaluation task.

The results are provided in Figure 4. We observe that as the values of K and M increase, the accuracy of both MetaLDC and MetaLDC-full has improved. We also notice that the accuracy gap between MetaLDC-full and MetaLDC become smaller as K or M increase, especially on the Split ISOLET dataset. We attribute the performance increases of MetaLDC-full to larger percentage of data points sampled from the task to update the model, which gradually diminishes the over-fitting effect.

5.6 Efficacy of Learned Representation

To study the efficacy of learned representation by our meta-training process, we have designed another method, referred to as the MetaLDC-NonFineTuning (MetaLDC-NFT). In this method, we use the Algorithm 1 to train the LDC model. We then test its accuracy on the new task *without* any fine-tuning. As shown in the Figure 5, we can see that the accuracy gap between MetaLDC-NFT and MetaLDC is within 5% on the Split ISOLET and 10% on the Rotated MNIST, which has reflected our Algorithm 1 has produced a good initialization to some extent.

6 RELATED WORKS

By distilling the learning experience from a broad set of related tasks, MAML [12] has achieved great success in the fast adaption regime [8, 14, 30, 34, 38]. The [22, 37] have proposed distributed collaborative frameworks to leverage knowledge between egde nodes via MAML. To reduce the computational cost, the [36] has presented a divide-and-conquer approach where the linear approximation is utilized to estimate the Hessian, while [28] has discussed using MAML with synthetic gradients in a feed forward manner for deep neural networks. However, most approaches are still quite costly, not viable for tiny devices with severe resource constraints.

HDC has been known as an efficient alternative to expensive deep neural networks for tiny devices [3, 4, 9, 10, 15, 16, 27, 35]. The study [9] has proposed to use vector quantization to further reduce the model size. Besides, some works have optimized HDC's encoding and training to improve its accuracy on a single data distribution [18]. Nonetheless, the required HDC model size to obtain an acceptable accuracy is still prohibitive large for tiny devices. More recently, LDC has been studied to significantly improve the efficiency of HDC [11], where the encoded vectors are only tens, yet the accuracy is even higher. Nevertheless, fast adaption problem to unseen but related tasks has not been well addressed in either HDC or LDC.

The fast adaption issue has become even more pressing for edge tiny devices due to their low latency tolerance and limited computational power [2, 24, 26, 31, 32]. The [21] has proposed a framework based on neural architecture search to find the optimal neural architecture under resource constraints of different tiny devices, whereas the MetaLDC does not require additional search efforts but provide a reusable lightweight template for unseen tasks.

7 CONCLUSION

In this paper, we propose MetaLDC, a LDC-based approach to fast adapt to unseen tasks via interleaved meta training for resource-constrained tiny devices. In MetaLDC, the LDC architecture is first trained across a set of different tasks, where we separately train the task-specific parameters ϕ in the inner loop of the meta-training algorithm. The learned model can then fast adapt to a new task by only updating the last layer using a handful of data points, while preserving learned prior knowledge in the former layers. Our empirical results have shown that our method has achieved higher accuracy compared to other HDC methods and LDC variants.

REFERENCES

[1] Neena Aloysius and M. Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592, 2017.

[2] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, 2020.

[3] Toygun Basaklar, Yigit Tuncel, Shruti Yadav Narayana, Suat Gumussoy, and Umit Y Ogras. Hypervector design for efficient hyperdimensional computing on edge devices. *arXiv preprint arXiv:2103.06709*, 2021.

[4] Simone Benatti, Fabio Montagna, Victor Kartsch, Abbas Rahimi, Davide Rossi, and Luca Benini. Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing. *IEEE Transactions on Biomedical Circuits and Systems*, 13(3):516–528, 2019.

[5] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery.

[6] Sohum Datta, Ryan Albert Antonio, Aldrin Rolf Ison, and Jan M. Rabaey. A programmable hyper-dimensional processor architecture for human-centric iot. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9:439–452, 2019.

[7] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[8] Shumin Deng, Ningyu Zhang, Zhanlin Sun, Jiaoyan Chen, and Huajun Chen. When low resource nlp meets unsupervised language model: Meta-pretraining then meta-learning for few-shot text classification (student abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(10):13773–13774, Apr. 2020.

[9] Cameron Diao, Denis Kleyko, Jan M. Rabaey, and Bruno A. Olshausen. Generalized learning vector quantization for classification in randomized neural networks and hyperdimensional computing. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2021.

[10] Shijin Duan, Yejia Liu, Shaolei Ren, and Xiaolin Xu. Lehdc: Learning-based hyperdimensional computing classifier. *arXiv preprint arXiv:2203.09680*, 2022.

[11] Shijin Duan, Xiaolin Xu, and Shaolei Ren. A brain-inspired low-dimensional computing classifier for inference on tiny devices. In *tinyML*, 2022.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[13] Nikhil B. Gaikwad, Varun Tiwari, Avinash Keskar, and N. C. Shivaprakash. Efficient fpga implementation of multilayer perceptron for real-time human activity classification. *IEEE Access*, 7:26696–26706, 2019.

[14] Ning Gao, Hanna Ziesche, Ngo Anh Vien, Michael Volpp, and Gerhard Neumann. What matters for meta-learning vision regression tasks? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14776–14786, June 2022.

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

[16] Chi-Tse Huang, Cheng-Yang Chang, Yu-Chuan Chuang, and An-Yeu (Andy) Wu. Pq-hdc: Projection-based quantization scheme for flexible and efficient hyperdimensional computing. In Ilias Maglogiannis, John Macintyre, and Lazaros Iliadis, editors, *Artificial Intelligence Applications and Innovations*, pages 425–435, Cham, 2021. Springer International Publishing.

[17] Mohsen Imani, Samuel Bosch, Sohum Datta, Sharadhi Ramakrishna, Sahand Salamat, Jan M. Rabaey, and Tajana Simunic Rosing. Quanthd: A quantization framework for hyperdimensional computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:2268–2278, 2020.

[18] Mohsen Imani, Justin Morris, Samuel Bosch, Helen Shu, Giovanni De Micheli, and Tajana Rosing. Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2019.

[19] Mohsen Imani, Xunzhao Yin, John Messerly, Saransh Gupta, Michael Niemier, Xiaobo Sharon Hu, and Tajana Rosing. Searchd: A memory-centric hyperdimensional computing with stochastic training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2422–2433, 2020.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[21] Ji Lin, Wei-Ming Chen, Yujun Lin, john cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11711–11722. Curran Associates, Inc., 2020.

[22] Sen Lin, Guang Yang, and Junshan Zhang. A collaborative learning framework via federated meta-learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 289–299. IEEE, 2020.

[23] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[24] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[25] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021.

[26] Davide Nadalini, Manuele Rusci, Giuseppe Tagliavini, Leonardo Ravaglia, Luca Benini, and Francesco Conti. Pulp-trainlib: Enabling on-device training for risc-v multi-core mcus through performance-driven autotuning. In Alex Orailoglu, Marc Reichenbach, and Matthias Jung, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 200–216, Cham, 2022. Springer International Publishing.

[27] Fateme Rasti Najafabadi, Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. Hyperdimensional computing for text classification. 2016.

[28] Robby Neven, Marian Verhelst, Tinne Tuytelaars, and Toon Goedemé. Feed-forward on-edge fine-tuning using static synthetic gradient modules. In Adrien Bartoli and Andrea Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 131–146, Cham, 2020. Springer International Publishing.

[29] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. Uci repository of machine learning databases, 1998.

[30] Binh D. Nguyen, Thanh-Toan Do, Binh X. Nguyen, Tuong Do, Erman Tjiputra, and Quang D. Tran. Overcoming data limitation in medical visual question answering. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pages 522–530, Cham, 2019. Springer International Publishing.

[31] Haoyu Ren, Darko Anicic, and Thomas Runkler. Tinyol: Tinyml with online-learning on microcontrollers, 2021.

[32] Ramon Sanchez-Iborra and Antonio F. Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18, 2020.

[33] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[34] Ting Wang, Zongkai Wu, and Donglin Wang. Visual perception generalization for vision-and-language navigation via meta-learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–7, 2021.

[35] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher De Sa. Understanding hyperdimensional computing for parallel single-pass learning. *arXiv preprint arXiv:2202.04805*, 2022.

[36] Sheng Yue, Ju Ren, Jiang Xin, Sen Lin, and Junshan Zhang. Inexact-admm based federated meta-learning for fast and continual edge learning. In *Proceedings of the Twenty-Second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, MobiHoc '21, page 91–100, New York, NY, USA, 2021. Association for Computing Machinery.

[37] Liang Zhang, Chuanting Zhang, and Basem Shihada. Efficient wireless traffic prediction at the edge: A federated meta-learning approach. *IEEE Communications Letters*, 26(7):1573–1577, 2022.

[38] Cong Zhao, Xinyue Sun, Shusen Yang, Xuebin Ren, Peng Zhao, and Julie McCann. Exploration across small silos: Federated few-shot learning on network edge. *IEEE Network*, 36(1):159–165, 2022.