DEBUG-HD: Debugging TinyML models on-device using Hyper-Dimensional computing

Nikhil P Ghanathe
The University of British Columbia
nikhilghanathe@ece.ubc.ca

Steven J E Wilton
The University of British Columbia
stevew@ece.ubc.ca

Abstract

TinyML models often operate in remote, dynamic environments without cloud connectivity, making them prone to failures. Ensuring reliability in such scenarios requires not only detecting model failures but also identifying their root causes. However, transient failures, privacy concerns, and the safety-critical nature of many applications—where systems cannot be interrupted for debugging—complicate the use of raw sensor data for offline analysis. We propose DEBUG-HD, a novel, resource-efficient on-device debugging approach optimized for KB-sized tinyML devices that utilizes hyper-dimensional computing (HDC). Our method introduces a new HDC encoding technique that leverages conventional neural networks, allowing DEBUG-HD to outperform prior binary HDC methods by 27% on average in detecting input corruptions across various image and audio datasets.

1 Introduction

Recent advances in machine learning (ML) and embedded systems have enabled ML to run on KB-sized, milliwatt-powered devices known as *TinyML*. These *always-on* devices perform all computations locally without any cloud connectivity [60]. TinyML has increasingly been used in mission-critical scenarios such as autonomous navigation [57] and healthcare diagnostics [49]. However, when deployed in uncertain environments, where inputs can become unpredictably *corrupted* (e.g., sensor failure or weather changes) [35], maintaining model *reliability* becomes crucial, especially when device access is limited. A model is considered *reliable* if its failures can be detected, and the root cause of those failures can be identified and addressed effectively. While recent works [15] have proposed resource-efficient monitoring mechanisms for detecting failures in tinyML systems, ensuring reliability often requires identifying these root causes, sometimes *in the field*. However, this is challenging on KB-sized tinyML devices, often running bare-metal applications, as transmitting raw sensor data for offline diagnosis can raise privacy and regulatory concerns [7], especially without secure communication hardware. Moreover, many tinyML applications are safety-critical (e.g., remote patient monitoring [3], autonomous navigation [44]), and thus cannot be interrupted for transmitting data [13]. In such scenarios, on-device debugging becomes crucial to ensure reliability, offering advantages such as 1) triggering *corrective actions* by identifying the root cause of failures [36], and 2) enhancing *active learning* by uncovering *hard-to-obtain* yet highly informative samples [53, 11].

Unlike most prior works on on-device debugging [22, 50, 16], we focus on *What input changes caused model failure?* instead of *Why a model fails for given inputs?* Thus, we focus on *diagnosing failures* in field-deployed models in this work, the first crucial step in debugging. To combat resource scarcity, we investigate hyper-dimensional computing (HDC) [34] as a resource-efficient paradigm for on-device debugging in TinyML, leveraging HDC's lightweight operations and memory-optimized representations. To explore this idea, we focus on the task of identifying the source of *corruptions* in input images (e.g., noise, weather, blur etc.) that cause model accuracy to drop. We propose DEBUG-HD, a novel binary HDC classifier that can effectively classify these corruptions (data shifts)

Machine Learning for Systems Workshop at (NeurIPS 2024).

in real-time. In addition, we address the limitations of HDC in the tinyML context by formulating an efficient encoding method with the assistance of a conventional neural network (NN), thereby allowing DEBUG-HD to outperform all prior binary HDC works even in low hyper-dimensions (hyper-d). DEBUG-HD achieves 12% higher accuracy on average than the best-performing prior work on various datasets. To the best of our knowledge, we are among the first to explore on-device debugging in remote, field-deployed tinyML devices. We believe that the proposed work offers developers a practical solution for diagnosing and addressing model failures, enhancing autonomy, reliability, and real-time diagnostic capabilities in tinyML.

2 Background and Preliminaries

Traditional ML debug TensorFlow Debugger (*tfdbg*) [8] provides deep insights into TensorFlow graphs for software-like debugging, while TensorBoard [61] and TensorWatch [54] allow visual monitoring of model statistics. Typically, lossy compression (e.g., statistical summaries) is sufficient for machine learning debugging. Tools like ML-Exray [50] catch deployment issues (e.g., quantization/preprocessing bugs), while Nazar [16] performs root-cause analysis. U-TOE [25] and RIOT [26] add over-the-air (OTA) debugging capabilities. Other debugging techniques [35, 10] focus on model assertions and characterizing model instability. The closest works on on-device ML debug [22, 47, 21] track statistics like sparsity and magnitude distributions. Another line of research focuses on *interpretability/explainability* methods [4, 2, 37, 1, 56, 5, 63, 38, 55, 31], which help explain the inner workings of a ML model in a human-understandable way. However, almost all prior works assume easy hardware access, which is often impractical for field-deployed tinyML systems.

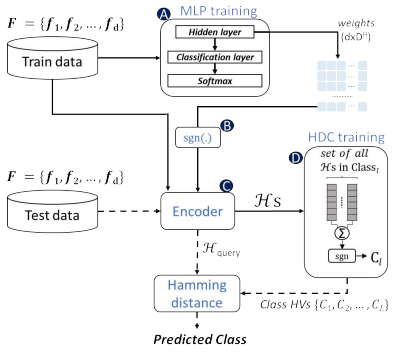


Figure 1: DEBUG-HD overview with training (solid lines) and inference (dotted lines) flows

2.1 HDC preliminaries

HDC *encodes/projects* inputs into hyper-dimensions to exploit the higher *discriminative* properties of hyper-dimensional representations, which in turn enables simpler learning processes that are both compute and memory friendly. There are three key stages in HDC model development (see Figure 1).

1) *Encoding*: All feature vectors in train set with dimension d are encoded/projected into hyper-dimension D^H using a projection matrix (usually randomly-generated [9]) of size $d \times D^H$ such that $D^H >> d$. This is a simple matrix-vector multiplication. The encoded vector is called a *hypervector*.

2) *Training*: The class hypervectors (HV) are learned by summing all encoded HVs associated with the class. For example, consider a balanced dataset $\mathcal{D}$ with size $|\mathcal{D}|$ and L classes. Let $\{f_{l_1}, f_{l_2}, ...f_{l_N}\}$ be the N samples from $\mathcal{D}$ belonging to class l such that $|\mathcal{D}| = N \times L$. The encoded versions are given by $\{\mathcal{H}_{l_1}, \mathcal{H}_{l_2}, ...\mathcal{H}_{l_N}\}$. Then, $\mathcal{C}_l = \sum_{i=1}^{N} \mathcal{H}_{l_i}$. $\{\mathcal{C}_1, \mathcal{C}_2, .., \mathcal{C}_L\}$ is the set of all class HVs.

3) *Inference*: The unseen test input is encoded using the same projection matrix as before to create the query HV ($\mathcal{H}_{query}$). Next, the similarity score of $\mathcal{H}_{query}$ is calculated with respect to each class HV and the class with the highest score is predicted. $l_{predicted} = \underset{l \in \{1,2,..,L\}}{argmax} \ (\delta(\mathcal{H}_{query}, \mathcal{C}_l))$

where $\delta(.)$ denotes similarity function, and for real-valued HVs it is usually cosine similarity. For bipolar HVs, this reduces to Hamming distance, where class with lowest distance is predicted. Traditional HDC classifiers project input data into a hyper-d $> 10,000$ [34], causing overhead that is impractical for extreme edge devices . Prior works address this using sparsity [30], compression [46], and quantization [28, 14], while others convert pretrained neural networks to HDC models [45], which we find suboptimal. Using smaller hyper-dimensions (e.g., < 1000) reduces memory footprint, but also reduces accuracy sharply, as shown in Figure 2. Figure 2 plots the top-1 accuracy of a Vanilla HDC classifier on the task of identifying 19 different types of corruptions in CIFAR10 images for hyper-d values from 200 to 10K. We find

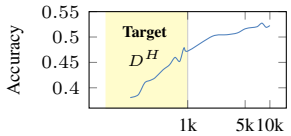


Figure 2: Top-1 accuracy of Vanilla HDC for hyper-d from 200 to 10k (x-axis logscale)

that Vanilla HDC, like most prior works, uses a randomly-generated projection matrix since random HVs are nearly orthogonal in large hyper-dimensions [33]. However, this property weakens for hyper-d< 1000, causing a significant drop in accuracy. Some prior works improve representation quality by using randomly-generated real-valued projection matrices instead of binary ones [58, 20, 65]. While this improves accuracy, it greatly increases memory.

3 DEBUG-HD

We aim to develop a diagnostic instrument to identify the source/type of input corruption, which is highly challenging [24]. Our preliminary evaluations show that a simple multi-layer perceptron (MLP) can effectively identify various corruptions but incurs significant overhead; for instance, an 8-bit 2-layer MLP for detecting 19 types of CIFAR10 corruptions uses nearly twice the resources of the ResNet-8 [17] base network. Thus, we explore a HDC classifier for this task due to its lower memory/compute footprint, enabled by its bipolar nature and simple parallel operations. Despite the limitations of hyper-d < 1000, we aim to enhance performance through our proposed approach.

3.1 MLP-assisted Encoding

To address the limitations of HDC in hyper-d < 1000, we design a novel initialization scheme for the *HDC encoder* where we learn the encoding/projection matrix from a simple 2-layer MLP. We observe that a 2-layer MLP (hidden+classification layer) with no bias vector closely resembles the HDC classifier. In particular, the hidden layer *encodes/projects* the inputs into a higher dimension using a matrix-vector multiplication operation, which is similar to the encoding method of a HDC classifier. Figure 1 illustrates our proposed approach.

Training and Inference: As shown, we first train a 2-layer MLP model with the input dataset. The hidden layer size is set to the hyper-dimension value of the HDC classifier to match HDC encoder dimensions (Ⓐ). The weights learned by the hidden layer of the MLP with dimensions $d \times D^H$ is fed to the *encoder* module after passing through the $sgn(.)$ function (Ⓑ). Next, the encoder module uses the *learnt projection matrix* to convert all feature vectors in the train set to hypervectors ($\mathcal{H}s$) with hyper-dimension D^H (Ⓒ). Finally, the training module computes all the class HVs (Ⓓ), and performs inference as described in Section 2.1. The MLP's projection matrix allows HDC's encoder to better separate data in the hyperspace, leading to a significant improvement in its performance.

4 Results and Discussion

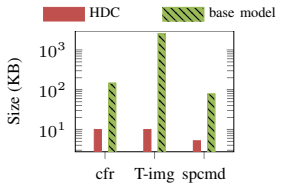
We evaluate DEBUG-HD on one audio and two image classification tasks. We use the in-distribution (ID) and corrupted-in-distribution (CID) versions of 1) SpeechCmd [59] on a 4-layer depthwise-separable model (DSCNN), 2) TinyImagnet [40] on a MobilenetV2 [23] and 3) CIFAR10 [39] on Resnet-8 from MLPerf Tiny benchmarck suite [6]. For CID datasets, we use CIFAR10-C (19 corruptions) and TinyImagenet-C (15 corruptions) from [19]. For SpeechCmd, we use the audiomentations library [32] to obtain SpeechCmd-C with 11 corruptions. To avoid the extra costs of on-device preprocessing and feature engineering, we directly utilize the output of an intermediate layer of the base network as input to HDC. However, this output may not be the most informative, which limits overall performance of the HDC classifier. Appendix A.1 has more details.

Evaluation and metrics We evaluate the accuracy of all HDC baselines on the task of detecting corruptions. In our evaluations with prior HDC works, we differentiate them into binary [14, 29, 46, 28, 30] and non-binary [58, 65, 20] HDC. We use the torch-hd library [18] to implement prior works. We envision a scenario where a finite set of *commonly-occurring* corruptions [19] affecting inputs are used to train the HDC before deployment. Post-deployment, the HDC classifier is invoked only when the model monitoring mechanism detects an accuracy drop in the base network to identify corruption causing the drop. To balance resource usage and accuracy, we use hyper-d values of 300, 400 and 300 for CIFAR10, T-imgnet and SpCmd respectively. See Appendix A.4 for more details.

HDC Results: Table 3a reports the top-1 accuracy of all baselines. DEBUG-HD outperforms all binary HDC methods across evaluated datasets, achieving an average accuracy **improvement of 12%** over the best-performing binary HDC work, highlighting the effectiveness of the MLP-assisted encoding scheme. Surprisingly, Vanilla often surpasses many state-of-the-art binary HDC works. Further analyses reveal that unlike Vanilla and DEBUG-HD's single-pass training, binary baselines

	Baseline	CIFAR10	T-imgnet	SpCmd
BINARY	Vanilla	0.42	0.71	0.35
	LeHDC	0.29	0.64	0.22
	AdaptHD	0.34	0.75	0.32
	CompHD	0.45	0.69	0.32
	QuantHD	0.38	0.65	0.23
	SparseHD	0.34	0.65	0.25
	DEBUG-HD	**0.68**	**0.78**	**0.45**
REAL	DistHD	**0.78**	0.84	0.43
	NeuralHD	0.74	0.88	**0.51**
	OnlineHD	0.75	**0.89**	0.50
	MLP	0.85	0.90	0.54

(a) Top-1 Accuracy



(b) Base model and DEBUG-HD size comparison. (y-axis logscale)

Dataset	Top-2	Top-3
CIFAR10	0.86	0.95
T-imgnet	0.92	0.95
SpCmd	0.58	0.67

(c) Top-2 and Top-3 accuracy of DEBUG-HD

Figure 3: Experimental results

employ a retraining step over multiple epochs, with some methods [14, 28] regenerating the least contributing hyper-dimensions by changing the encoding. However, we find that with lower hyper-d values (e.g., 300, 400), changes to encoder after a train-pass leads to an immediate drop in accuracy, with the next epoch barely recovering the lost performance, resulting in suboptimal outcomes. In contrast, higher hyper-d values allow for less accuracy drop between epochs during regeneration. For instance, accuracies of both Vanilla and AdaptHD on SpCmd for hyper-d=1000 is 0.39. Additionally, most binary HDC methods maintain two versions of class HV: non-binary for training updates and binary for inference/validation, typically obtained by applying the $sgn(.)$ function. This approach also causes sharp accuracy drops at lower hyper-d values. We also report the accuracy of a conventional MLP as a reference. While non-binary HDC approaches are close to MLP in performance due to superior data representation, binary HDC methods lag behind. However, DEBUG-HD still achieves strong performance relative to the MLP.

Debugging With multiple *similar* corruptions (see Table 3 and Table 1) and inherent model limitations, HDC's top-1 accuracy is constrained. For SpCmd, we observe that *highly similar* corruptions (Table 3) and the quality of the intermediate layer data limits DEBUG-HD's performance, impacting even the MLP. This reflects an intrinsic limitation of the data, which could potentially be addressed through improved data or feature engineering. Table 3c presents the top-2 and top-3 accuracy of DEBUG-HD, demonstrating that on-device diagnosis can effectively narrow down corruptions to the top candidates with high precision. This advancement is vital as it improves the efficiency of diagnosing model failures by reducing the number of potential error sources for developers to investigate, particularly when faced with multiple corruption types. Figure 3b illustrates the overhead of DEBUG-HD relative to the base network. We observe that the trade-off between accuracy and overhead improves with larger model sizes and higher-resolution images, both of which are realistic scenarios in practical applications. While our work focuses on identifying image corruptions—a specific subset of potential model failures—there are numerous other diagnostics that can be applied on-device across a wide range of applications [48, 52, 42]. This work serves as a strong foundation to facilitate the exploration.

5 Conclusion

We present DEBUG-HD, a resource-efficient approach for on-device debugging in TinyML devices using HDC. DEBUG-HD leverages an improved HDC encoding technique, learned through neural networks, to outperform previous binary HDC methods in identifying corruptions in inputs across various datasets. With HDC's inherent fault tolerance [43] alongside the proposed enhanced encoding, DEBUG-HD strengthens TinyML reliability. This work underscores HDC's potential as a robust foundation for integrating diagnostics in TinyML, facilitating advancements in model reliability and performance through innovative, real-time, on-device debugging solutions.

Limitations: HDC methods are generally outperformed by conventional neural networks (NNs) and often require specialized parallel hardware, such as FPGAs, to fully leverage their computational efficiency [51]. Their performance is also influenced by the number of corruptions to identify; however, it can improve if developers narrow the focus to specific corruptions (see Appendix A.5). Additionally, the accuracy gain from MLP-assisted encoding is notable only in smaller hyperspaces. For instance, the accuracies of AdaptHD and DEBUG-HD are 0.83 and 0.84, respectively, for hyper-d=2000 on TinyImageNet. Finally, we observe that incorporating DEBUG-HD into extremely small models on toy datasets (e.g., Ward [62], USPS [27], MNIST [41]) incurs significant overhead relative to the base model size. Nevertheless, DEBUG-HD operates efficiently with more realistic models.

References

[1] Abubakar Abid, Mert Yuksekgonul, and James Zou. 2022. Meaningfully debugging model mistakes using conceptual counterfactual explanations. In *International Conference on Machine Learning*. PMLR, 66–88.

[2] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access* 6 (2018), 52138–52160.

[3] Khaled Ahmed and Mohamed Hassan. 2022. tinycare: A tinyml-based low-cost continuous blood pressure estimation on the extreme edge. In *2022 IEEE 10th International Conference on Healthcare Informatics (ICHI)*. IEEE, 264–275.

[4] Sajid Ali, Tamer Abuhmed, Shaker El-Sappagh, Khan Muhammad, Jose M Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz-Rodríguez, and Francisco Herrera. 2023. Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence. *Information Fusion* (2023), 101805.

[5] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. 2010. How to explain individual classification decisions. *The Journal of Machine Learning Research* 11 (2010), 1803–1831.

[6] Colby R. Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Király, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Pete Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier M. Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Wenxu Niu, and Xuesong Xu. 2021. MLPerf Tiny Benchmark. *CoRR* abs/2106.07597 (2021). arXiv:2106.07597 `https://arxiv.org/abs/2106.07597`

[7] Mamta Bhamare, Pradnya V. Kulkarni, Rashmi Rane, Sarika Bobde, and Ruhi Patankar. 2024. Chapter 14 - TinyML applications and use cases for healthcare. In *TinyML for Edge Intelligence in IoT and LPWAN Networks*, Bharat S. Chaudhari, Sheetal N. Ghorpade, Marco Zennaro, and Rytis Paškauskas (Eds.). Academic Press, 331–353. `https://doi.org/10.1016/B978-0-44-322202-3.00019-1`

[8] Shanqing Cai, Eric Breck, Eric Nielsen, Michael Salib, and D Sculley. 2016. Tensorflow debugger: Debugging dataflow graphs for machine learning. (2016).

[9] Cheng-Yang Chang, Yu-Chuan Chuang, Chi-Tse Huang, and An-Yeu Wu. 2023. Recent Progress and Development of Hyperdimensional Computing (HDC) for Edge Intelligence. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 13, 1 (2023), 119–136. `https://doi.org/10.1109/JETCAS.2023.3242767`

[10] Eyal Cidon, Evgenya Pergament, Zain Asgar, Asaf Cidon, and Sachin Katti. 2021. Characterizing and taming model instability across edge devices. *Proceedings of Machine Learning and Systems* 3 (2021), 624–636.

[11] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2020. Selection via Proxy: Efficient Data Selection for Deep Learning. arXiv:1906.11829 [cs.LG] `https://arxiv.org/abs/1906.11829`

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[13] Duarte Dias and João Paulo Silva Cunha. 2018. Wearable health devices—vital sign monitoring, systems and technologies. *Sensors* 18, 8 (2018), 2414.

[14] Shijin Duan, Yejia Liu, Shaolei Ren, and Xiaolin Xu. 2022. Lehdc: Learning-based hyperdimensional computing classifier. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 1111–1116.

[15] Nikhil P Ghanathe and Steve Wilton. 2024. QUTE: Quantifying Uncertainty in TinyML models with Early-exit-assisted ensembles. arXiv:2404.12599 [cs.LG] `https://arxiv.org/abs/2404.12599`

[16] Wei Hao, Zixi Wang, Lauren Hong, Lingxiao Li, Nader Karayanni, Chengzhi Mao, Junfeng Yang, and Asaf Cidon. 2023. Monitoring and Adapting ML Models on Mobile Devices. *arXiv preprint arXiv:2305.07772* (2023).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. `https://doi.org/10.1109/CVPR.2016.90`

[18] Mike Heddes, Igor Nunes, Pere Vergés, Denis Kleyko, Danny Abraham, Tony Givargis, Alexandru Nicolau, and Alexander Veidenbaum. 2023. Torchhd: An open source python library to support research on hyperdimensional computing and vector symbolic architectures. *Journal of Machine Learning Research* 24, 255 (2023), 1–10.

[19] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261* (2019).

[20] Alejandro Hernández-Cano, Namiko Matsumoto, Eric Ping, and Mohsen Imani. 2021. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 56–61.

[21] Daniel Holanda Noronha, Ruizhe Zhao, Jeff Goeders, Wayne Luk, and Steven JE Wilton. 2019. On-chip FPGA debug instrumentation for machine learning applications. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 110–115.

[22] Daniel Holanda Noronha, Ruizhe Zhao, Zhiqiang Que, Jeffrey Goeders, Wayne Luk, and Steve Wilton. 2019. An Overlay for Rapid FPGA Debug of Machine Learning Applications. In *2019 International Conference on Field-Programmable Technology (ICFPT)*. 135–143. `https://doi.org/10.1109/ICFPT47387.2019.00024`

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[24] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. 2020. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10951–10960.

[25] Zhaolan Huang, Koen Zandberg, Kaspar Schleiser, and Emmanuel Baccelli. 2023. U-TOE: Universal TinyML On-Board Evaluation Toolkit for Low-Power IoT. In *2023 12th IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. IEEE, 1–6.

[26] Zhaolan Huang, Koen Zandberg, Kaspar Schleiser, and Emmanuel Baccelli. 2024. RIOT-ML: toolkit for over-the-air secure updates and performance evaluation of TinyML models. *Annals of Telecommunications* (2024), 1–15.

[27] Jonathan J. Hull. 1994. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence* 16, 5 (1994), 550–554.

[28] Mohsen Imani, Samuel Bosch, Sohum Datta, Sharadhi Ramakrishna, Sahand Salamat, Jan M Rabaey, and Tajana Rosing. 2019. Quanthd: A quantization framework for hyperdimensional computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2268–2278.

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

[30] Mohsen Imani, Sahand Salamat, Behnam Khaleghi, Mohammad Samragh, Farinaz Koushanfar, and Tajana Rosing. 2019. SparseHD: Algorithm-Hardware Co-optimization for Efficient High-Dimensional Computing. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 190–198. https://doi.org/10.1109/FCCM.2019.00034

[31] Md Shafayat Jamil, Sirdarta Prashad Banik, GM Atiqur Rahaman, and Sajib Saha. 2023. Advanced GradCAM++: Improved Visual Explanations of CNN Decisions in Diabetic Retinopathy. In *Computer Vision and Image Analysis for Industry 4.0*. CRC Press 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, 64–75.

[32] Iver Jordal. 2024. Audiomentations. https://github.com/iver56/audiomentations Version 0.32.0.

[33] Pentti Kanerva. 1998. Encoding Structure in Boolean Space. In *ICANN 98*, Lars Niklasson, Mikael Bodén, and Tom Ziemke (Eds.). Springer London, London, 387–392.

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

[35] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2018. Model assertions for debugging machine learning. In *NeurIPS MLSys Workshop*, Vol. 3. 10.

[36] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model assertions for monitoring and improving ml models. *Proceedings of Machine Learning and Systems* 2 (2020), 481–496.

[37] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*. PMLR, 2668–2677.

[38] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. 2017. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598* (2017).

[39] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[40] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[42] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256kb memory. *arXiv preprint arXiv:2206.15472* (2022).

[43] Jialong Liu, Mingyuan Ma, Zhenhua Zhu, Yu Wang, and Huazhong Yang. 2019. Hdc-im: Hyperdimensional computing in-memory architecture based on rram. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 450–453.

[44] Antonio Loquercio, Ana I. Maqueda, Carlos R. del Blanco, and Davide Scaramuzza. 2018. DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters* 3, 2 (2018), 1088–1095. https://doi.org/10.1109/LRA.2018.2795643

[45] Dongning Ma, Cong Hao, and Xun Jiao. 2024. Hyperdimensional computing vs. neural networks: Comparing architecture and learning process. In *2024 25th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 1–5.

[46] Justin Morris, Mohsen Imani, Samuel Bosch, Anthony Thomas, Helen Shu, and Tajana Rosing. 2019. CompHD: Efficient hyperdimensional computing using model compression. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[47] Daniel Holanda Noronha, Zhiqiang Que, Wayne Luk, and Steven J. E. Wilton. 2021. Flexible Instrumentation for Live On-Chip Debug of Machine Learning Training on FPGAs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 20–28. https://doi.org/10.1109/FCCM51124.2021.00011

[48] Keiichi Ochiai, Kohei Senkawa, Naoki Yamamoto, Yuya Tanaka, and Yusuke Fukazawa. 2019. Real-time On-Device Troubleshooting Recommendation for Smartphones. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 2783–2791. https://doi.org/10.1145/3292500.3330669

[49] Shishir G. Patil, Don Kurian Dennis, Chirag Pabbaraju, Nadeem Shaheer, Harsha Vardhan Simhadri, Vivek Seshadri, Manik Varma, and Prateek Jain. 2019. GesturePod: Enabling On-Device Gesture-Based Interaction for White Cane Users. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 403–415. https://doi.org/10.1145/3332165.3347881

[50] Hang Qiu, Ioanna Vavelidou, Jian Li, Evgenya Pergament, Pete Warden, Sandeep Chinchali, Zain Asgar, and Sachin Katti. 2022. ML-EXray: Visibility into ML deployment on the edge. *Proceedings of Machine Learning and Systems* 4 (2022), 337–351.

[51] Alexander Redding, Xiaofan Yu, Shengfan Hu, Pat Pannuto, and Tajana Rosing. 2023. EmbHD: A Library for Hyperdimensional Computing Research on MCU-Class Devices. In *Proceedings of the 2nd Workshop on Networked Sensing Systems for a Sustainable Society*. 187–192.

[52] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[53] Burr Settles. 2009. Active learning literature survey. (2009).

[54] Shital Shah, Roland Fernandez, and Steven Drucker. 2019. A system for real-time interactive analysis of deep learning training. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 1–6.

[55] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[56] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. PMLR, 3319–3328.

[57] Jorge Vargas, Suleiman Alsweiss, Onur Toker, Rahul Razdan, and Joshua Santos. 2021. An overview of autonomous vehicles sensors and their vulnerability to weather conditions. *Sensors* 21, 16 (2021), 5397.

[58] Junyao Wang, Sitao Huang, and Mohsen Imani. 2023. Disthd: A learner-aware dynamic encoding method for hyperdimensional classification. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[59] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

[60] Pete Warden and Daniel Situnayake. 2019. *TinyML*. O'Reilly Media, Incorporated.

[61] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2017. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 1–12.

[62] Jingjing Yang, Yuanning Li, Yonghong Tian, Lingyu Duan, and Wen Gao. 2009. Group-sensitive multiple kernel learning for object categorization. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 436–443.

[63] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 818–833.

[64] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. https://doi.org/10.48550/ARXIV.1711.07128

[65] Zhuowen Zou, Yeseong Kim, Farhad Imani, Haleh Alimohamadi, Rosario Cammarota, and Mohsen Imani. 2021. Scalable edge-based hyperdimensional learning system with brain-like neural adaptation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

A Appendix

A.1 Dataset details

In this section, we describe the details of the datasets and models evaluated.

A.1.1 Datasets

In our evaluations, we use three in-distribution datasets for training all baselines methods we evaluate: 1) SpeechCmd [59], 2) CIFAR10 [39] and 3) TinyImagenet [40].

SpeechCommands Speech-Cmd is a collection of short audio clips, each spanning 1 second. The dataset consists of utterances for 35 words and is commonly used for benchmarking keyword spotting systems. We train our model (DSCNN) to recognize ten words out of 35: *Down, Go, Left, No, Off, On, Right, Stop, Up, Yes*. Thus, the number of classes is 10. The audio files in WAV format are preprocessed to compute Mel-frequency cepstral coefficients (mel-spectograms). The mel-spectograms are of size 49×10 with a single channel.

CIFAR10 CIFAR10 dataset consists of 60,000 32×32 rgb images out of which 10,000 images are in the test set. It contains 10 classes and thus 6000 images per class.

TinyImagenet TinyImagenet is a smaller version of the Imagenet [12] dataset containing 200 classes instead of 1000 classes of the original Imagenet. Each class in TinyImagenet has 500 images in the train set and the validation set contains 50 images per class. The size of the images are resized and fixed at $64\times64\times3$.

A.1.2 Corrupted-in-distribution (CID) datasets

We use the following corrupted versions of ID: 1) CIFAR10-C and 2) TinyImagenet-C from [19]. For both, we use the corruptions with severity level=5 in our evaluations. All corruptions are drawn from 4 major sources: noise, blur, weather and digital. We create SpeechCmd-C from the audiomentations libary [32].

CIFAR10-C CIFAR10-C includes 19 different types of corruptions with 5 severity levels. The list of corruptions are: *gaussian_noise, brightness, contrast, defocus_blur, elastic, fog, frost, frosted_glass_blur, gaussian_blur, impulse_noise, jpeg_compression, motion_blur, pixelate, saturate, shot_noise, snow, spatter, speckle_noise and zoom_blur*.

TinyImagenet-C TinyImagenet-C includes 15 different types of corruptions with 5 severity levels. The list of corruptions are: *gaussian_noise, brightness, contrast, defocus_blur, elastic_transform, fog, frost, glass_blur, impulse_noise, jpeg_compression, motion_blur, pixelate, shot_noise, snow and zoom_blur*.

SpeechCmd-C For keyword spotting (KWS) [64], an audio classification task on the Speech Commands dataset, we introduce noise and other corruptions to the audio using the audiomentations library [32]. We apply the following 11 corruptions: *gaussian noise, air absorption, band pass filter, band stop filter, high pass filter, high shelf filter, low pass filter, low shelf filter, peaking filter, tanh distortion, time mask and time stretch*.

A.2 Experimental Setup

We evaluate three models in our experiments:1) 4-layer DSCNN on SpeechCmd [64], 2) Resnet-8 with 3 residual stacks from the MLPerf Tiny benchmark suite [6] on CIFAR10 and 3) MobilenetV2 [23] on TinyImagenet.

The 4-layer DSCNN is trained for 10 epochs with a batch size of 100, the Resnet-8 model is trained for 200 epochs with batch size of 32 and the MobilenetV2 model is trained for 200 epochs with a batch size of 128. All models are trained with Adam optimizer with momentum of 0.9 and an initial learning rate of 0.001, expect DSCNN on SpeechCmd which uses an initial learning rate of 0.0005. The learning rate is decayed by a factor of 0.99 every epoch for image classification datasets, and we follow a step function for SpeechCmd that reduces learning rate by half every 2 epochs.

We train the MLP used to learn the encoding matrix for 20 epochs with a batch size of 256.

A.3 Dataset creation for HDC classifier

Since the HDC classifier aims to distinguish between different types of corruptions in the input images, the natural input to the classifier would be the images themselves. However, the high dimensionality of the input images (e.g., 12K for TinyImageNet) significantly increases the resource usage of the HDC classifier. One potential solution is to reduce the dimensionality of the input images, but this introduces additional processing costs that may impact the execution of the base network.

To avoid such preprocessing or feature engineering overhead, we propose tapping into an appropriate intermediate layer of the base model directly. This approach offers two key advantages: 1) as the input image passes through the base network, its dimensionality is reduced, which in turn minimizes the size of the HDC classifier, and 2) the execution of the base network remains uninterrupted—an important consideration for safety-critical applications.

The dataset for the HDC classifier is created as follows.

1) First, we iterate through all the corrupted-ID images for each dataset by passing each corrupted image through the pretrained base network. The base network has been trained only on ID.

2) Second, we select an appropriate intermediate layer of the base network, and record the output of this intermediate layer as we iterate through the corrupted images. We use this to create our train and test datasets for the HDC classifier denoted by $\mathcal{D}_{HDC-train}$ and $\mathcal{D}_{HDC-test}$ respectively.

3) Next, we use $\mathcal{D}_{HDC-train}$ to train the HDC classifier as described in Section 3, and use $\mathcal{D}_{HDC-test}$ for validation/testing.

A.3.1 Selecting an appropriate intermediate layer

Since tinyML models often have only a few layers, the number of intermediate layers available for tapping is limited. Therefore, we perform an exhaustive search. We construct $\mathcal{D}_{HDC-train}$ and $\mathcal{D}_{HDC-test}$ as described above by taping into each intermediate layer's output. Next, we train and test a Vanilla HDC using each version of the dataset. We select the intermediate layer whose dataset yields the highest validation accuracy. For SpCmd using a 4-layer DSCNN, the tapping location is after the first depthwise layer. For CIFAR10 using a 3-residual stack Resnet, the best location is after the third residual stack. In the case of TinyImageNet, which employs a larger MobileNetV2 model with 17 inverted residual blocks, we refine our search using a binary search approach. This systematic method helps us narrow down the most informative layer by evaluating performance at progressively finer intervals. For MobileNetV2, we find that the output of the 5^{th} inverted residual block is the most informative. The number of features in the HDC datasets thus obtained are 256, 128 and 192 for CIFAR10, SpCmd and T-imgnet respectively.

A.4 Evaluation details

We distinguish the HDC baselines we compare against into two types: 1) Binary HDC methods: LeHDC [14], AdaptHD [29], CompHD [46], QuantHD [28] and SparseHD [30], and 2) Non-binary HDC methods: DistHD [58], NeuralHD [65] and OnlineHD [20].

Simulating model accuracy drop In our evaluations, we trigger the proposed debug tool only when the model monitoring mechanism detects an accuracy drop. Ghanathe and Wilton [15] proposes a resource-efficient accuracy monitoring mechanism for tinyML models with minimal resource overhead. We follow the procedure outlined in [15] to simulate the accuracy drop detection. First, we use the base network to iterate over all in-distribution (ID) samples using a sliding window of size 100, calculating accuracy over the past 100 input samples. The resulting accuracy distribution is denoted as $\mathcal{A}_{ID}$, with mean μ_{ID} and standard deviation σ_{ID}. Next, we create ID+CID datasets by appending ID with all corrupted-ID versions. We then iterate over the ID+CID datasets using the base network and the same sliding window approach. The HDC classifier is only invoked when $\mathcal{A}_{SW} < \mu_{ID} - 3 \cdot \sigma_{ID}$, where $\mathcal{A}_{SW}$ is accuracy of sliding window. Once invoked, the HDC classifier reads the output of the predetermined intermediate layer and identifies the type of corruption that caused the model accuracy to drop.

A.4.1 MLP-assisted encoding

A HDC classifier for debugging in tinyML devices necessitates maintaining the bipolar nature of the network while using hyper-dimensional values of less than 500 to ensure that the associated overhead remains practical. However, we observe that in a lower hyper-dimensional space (few hundreds of dimensions), the random projection matrix is inadequate in separating the data in the hyperspace, which in turn significantly limits the performance of the HDC model. This phenomenon is illustrated in Figure 2. In our initial experiments, we observe two things: 1) a simple 2-layer MLP has a close resemblance to a HDC model and 2) the MLP is able to achieve significantly higher accuracy compared to its HDC counterpart. This demonstrates that the single hidden layer of the MLP is able to project data into the hyperspace much more efficiently than the random-projection matrix. Since the hidden layer functions similar to the HDC encoder, we import the weight matrix into the encoder module of the HDC classifier, which proves to be crucial in extracting maximum performance from the HDC classifier, especially when working with low hyper-d values (< 1000).

We made several modifications to the MLP architecture used for learning the encoding matrix. In one iteration, we included a bias vector followed by an activation function. However, we found that the encoding matrix was negatively impacted by the additional assistance from the bias vector and activation function. As a result, we adopted a configuration where the hidden layer output has neither a bias vector nor an activation function.

SSIM	ID	GN	BR	CT	DB	EL	FG	FR	FGB	GB	IN	JC	MB	PX	ST	SN	SW	SP	SpN	ZB
ID	1.00	0.46	0.81	0.34	0.76	0.64	0.56	0.68	0.46	0.75	0.65	0.80	0.64	0.78	0.70	0.50	0.71	0.83	0.53	0.61
GN	0.46	1.00	0.35	0.10	0.34	0.25	0.25	0.34	0.19	0.29	0.32	0.40	0.25	0.35	0.35	0.32	0.37	0.41	0.34	0.26
BR	0.83	0.37	1.00	0.34	0.61	0.50	0.50	0.64	0.40	0.61	0.52	0.62	0.54	0.64	0.54	0.38	0.70	0.65	0.39	0.51
CT	0.45	0.15	0.45	1.00	0.56	0.45	0.57	0.38	0.37	0.60	0.30	0.41	0.50	0.46	0.38	0.17	0.36	0.38	0.19	0.49
DB	0.75	0.32	0.60	0.37	1.00	0.72	0.53	0.55	0.54	0.98	0.50	0.75	0.71	0.78	0.55	0.36	0.52	0.65	0.38	0.73
EL	0.63	0.29	0.52	0.26	0.72	1.00	0.44	0.44	0.46	0.72	0.41	0.59	0.57	0.62	0.47	0.32	0.43	0.53	0.34	0.57
FG	0.57	0.27	0.51	0.38	0.56	0.46	1.00	0.46	0.33	0.56	0.39	0.50	0.48	0.50	0.44	0.29	0.43	0.49	0.30	0.47
FR	0.69	0.35	0.68	0.26	0.56	0.44	0.50	1.00	0.31	0.55	0.45	0.59	0.47	0.53	0.47	0.35	0.57	0.57	0.35	0.44
FGB	0.45	0.20	0.37	0.20	0.52	0.45	0.30	0.34	1.00	0.53	0.32	0.43	0.39	0.48	0.38	0.23	0.32	0.41	0.21	0.53
GB	0.77	0.31	0.62	0.41	0.98	0.72	0.57	0.54	0.54	1.00	0.50	0.71	0.74	0.76	0.55	0.36	0.50	0.64	0.38	0.72
IN	0.64	0.34	0.51	0.19	0.48	0.40	0.34	0.46	0.31	0.48	1.00	0.53	0.40	0.51	0.48	0.38	0.45	0.55	0.35	0.38
JC	0.79	0.41	0.60	0.25	0.71	0.58	0.47	0.57	0.44	0.71	0.54	1.00	0.60	0.70	0.60	0.41	0.59	0.69	0.46	0.53
MB	0.65	0.30	0.56	0.36	0.69	0.59	0.48	0.48	0.40	0.73	0.42	0.60	1.00	0.63	0.49	0.29	0.44	0.54	0.33	0.57
PX	0.76	0.35	0.63	0.29	0.77	0.63	0.46	0.54	0.46	0.76	0.52	0.73	0.60	1.00	0.60	0.41	0.54	0.68	0.41	0.58
ST	0.68	0.35	0.56	0.24	0.55	0.44	0.40	0.46	0.33	0.54	0.49	0.56	0.48	0.58	1.00	0.37	0.46	0.60	0.38	0.45
SN	0.48	0.34	0.37	0.10	0.32	0.29	0.26	0.35	0.22	0.35	0.36	0.45	0.29	0.38	0.38	1.00	0.37	0.42	0.36	0.26
SW	0.72	0.40	0.72	0.24	0.54	0.43	0.42	0.58	0.30	0.53	0.45	0.60	0.46	0.53	0.45	0.40	1.00	0.58	0.42	0.40
SP	0.83	0.40	0.67	0.26	0.62	0.52	0.47	0.58	0.40	0.64	0.56	0.69	0.54	0.68	0.57	0.45	0.57	1.00	0.46	0.50
SpN	0.50	0.36	0.40	0.12	0.34	0.29	0.25	0.36	0.24	0.38	0.37	0.45	0.30	0.41	0.38	0.40	0.38	0.47	1.00	0.29
ZB	0.61	0.25	0.52	0.33	0.71	0.59	0.45	0.44	0.53	0.71	0.41	0.57	0.55	0.61	0.46	0.29	0.42	0.53	0.31	1.00

Table 1: Structural Similarity Index Matrix for CIFAR10 corruptions. Legend in Table 2. Yellow for $0.5 < \text{SSIM} \leq 0.75$ and Green for $\text{SSIM} > 0.75$

Full Name	Abbreviation
In-distribution	ID
Gaussian Noise	GN
Brightness	BR
Contrast	CT
Defocus Blur	DB
Elastic	EL
Fog	FG
Frost	FR
Frosted Glass Blur	FGB
Gaussian Blur	GB
Impulse Noise	IN
JPEG Compression	JC
Motion Blur	MB
Pixelate	PX
Saturate	ST
Shot Noise	SN
Snow	SW
Spatter	SP
Speckle Noise	SpN
Zoom Blur	ZB

Table 2: Legend for Abbreviations Used in CIFAR10 corruptions in Table 1

A.5 Additional insights into debugging

The HDC classifiers are generally outperformed by conventional NNs due to their inherent architecture limitations and binarized implementations. Another important factor impacting HDC performance is the presence of *similar corruptions*. Table 3 and Table 1 present the structural similarity index measure (SSIM) matrices for Speech Commands (SpCmd) and CIFAR10 corruptions, respectively. These tables indicate the degree of similarity between each pair of corruptions. To compute the SSIM between two corruption types, we randomly sample 50 images from each type and calculate the mean SSIM between them. Each sample pair consists of identical images, each corrupted by a different type of corruption. A SSIM value of 0 indicates no similarity between images, while a value of 1 signifies complete identity. The cells in Tables 3 and 1 are color coded for identification of similar corruptions. In these tables, a yellow cell indicates SSIM between 0.5 and 0.75 (moderate similarity), and a green cell indicates SSIM>0.75 (high similarity). As seen in the SSIM table for SpCmd (Table 3), there are many corruptions with high degree of similarity, which limits the performance of even a conventional MLP classifier.

The SSIM tables can help developers narrow down the number of corruptions to detect on-device, which in turn leads to a massive accuracy improvement. For example, when we reduce the number of CIFAR10 corruptions from 19 to 12, we immediately improve the top-1 accuracy from 0.68 to 0.79. The SSIM tables help the developers eliminate similar corruption types from the debugging task. For example, in the SSIM table for CIFAR10 (Table 1), we observe that defocus blur (DB) and gaussian blur (GB) are highly similar. Therefore, the developer can choose to remove either of them from the classification task to obtain a higher precision in debugging.

SSIM	ID	GN	AA	BPF	BSF	HPF	HSF	LPF	LSF	PF	TD	TM	TS
ID	1.00	0.31	0.80	0.45	0.40	0.44	0.93	0.82	0.93	0.89	0.76	0.95	0.77
GN	0.45	1.00	0.41	0.31	0.21	0.32	0.47	0.43	0.46	0.43	0.44	0.69	0.59
AA	0.77	0.24	1.00	0.48	0.38	0.43	0.80	0.84	0.78	0.72	0.60	0.85	0.67
BPF	0.45	0.12	0.46	1.00	0.27	0.55	0.46	0.46	0.42	0.41	0.33	0.61	0.41
BSF	0.40	0.09	0.35	0.25	1.00	0.24	0.34	0.33	0.35	0.38	0.29	0.59	0.39
HPF	0.40	0.15	0.44	0.55	0.27	1.00	0.46	0.45	0.47	0.49	0.29	0.65	0.47
HSF	0.94	0.31	0.77	0.48	0.39	0.45	1.00	0.82	0.90	0.86	0.71	0.93	0.79
LPF	0.85	0.26	0.82	0.44	0.36	0.38	0.83	1.00	0.81	0.73	0.61	0.86	0.65
LSF	0.93	0.26	0.77	0.43	0.35	0.47	0.89	0.84	1.00	0.83	0.69	0.93	0.74
PF	0.87	0.25	0.78	0.46	0.37	0.46	0.85	0.73	0.81	1.00	0.69	0.91	0.72
TD	0.80	0.31	0.64	0.38	0.31	0.38	0.78	0.71	0.73	0.71	1.00	0.85	0.66
TM	0.90	0.29	0.73	0.44	0.37	0.40	0.85	0.76	0.84	0.80	0.72	1.00	0.74
TS	0.69	0.24	0.56	0.37	0.32	0.32	0.69	0.60	0.64	0.67	0.56	0.85	1.00

Table 3: Structural Similarity Index Matrix for SpeechCmd corruptions. Legend in Table 4. Yellow for $0.5 < \text{SSIM} \leq 0.75$ and Green for $\text{SSIM} > 0.75$

Abbreviation	Corruption type
ID	In-distribution
GN	Gaussian Noise
AA	Air Absorption
BPF	Band Pass Filter
BSF	Band Stop Filter
HPF	High Pass Filter
HSF	High Shelf Filter
LPF	Low Pass Filter
LSF	Low Shelf Filter
PF	Peaking Filter
TD	Tanh Distortion
TM	Time Mask
TS	Time Stretch

Table 4: Legend for Abbreviations Used in SpeechCmd corruptions in Table 3

A.5.1 Effect of removing corruptions from debugging

Demonstrating the usability of a debugging tool is a complex task, as each developer may use it differently. We attempt to explore a plausible debugging scenario, where corruptions are removed from the HDC classification task to improve accuracy.

While removing corruptions from the classification may potentially reduce the coverage of the debugging task, we find that this might not always be the case. For example, consider a scenario where Gaussian blur (GB) is removed from the classification task, but defocus blur (DB) remains. Post-deployment, if the input is corrupted by GB, the HDC classifier might still identify it, though misclassifying it as DB. In response, the developer may choose to retrain the model to handle DB corruption. Interestingly, this retraining phase may indirectly improve the model's ability to handle GB corruption as well, due to the similarity between DB and GB. We empirically verify this claim by showing that after retraining on DB, the model exhibits improved performance on GB-corrupted inputs, even though GB was not explicitly included during retraining. This is also supported by the findings of Kindermans et al. [38], where they developed an *explanation* for neural networks. According to Kindermans et al. [38], input data consists of two components: the *signal*, which contains task-relevant information, and the *distractor*, which obfuscates the signal. During training, neural networks learn to filter out the distractor to recover the signal. Applied to our scenario, if the network learns to remove DB to recover the signal, it should similarly be able to remove GB. Thus, developers can safely exclude similar corruptions from the classification task, improving HDC accuracy with minimal impact on the debugging process.