RANDOMNESS IN NEURAL NETWORK TRAINING: CHARACTERIZING THE IMPACT OF TOOLING

Donglin Zhuang
School of Computer Science
The University of Sydney
dzhu9887@sydney.edu.au

Xingyao Zhang
Department of Computer Science
University of Washington
xingyaoz@cs.washington.edu

Shuaiwen Leon Song
School of Computer Science
The University of Sydney
shuaiwen.song@sydney.edu.au

Sara Hooker
Google Research, Brain
shooker@google.com

ABSTRACT

The quest for determinism in machine learning has disproportionately focused on characterizing the impact of noise introduced by algorithmic design choices. In this work, we address a less well understood and studied question: how does our choice of tooling introduce randomness to deep neural network training. We conduct large scale experiments across different types of hardware, accelerators, state of art networks, and open-source datasets, to characterize how tooling choices contribute to the level of non-determinism in a system, the impact of said non-determinism, and the cost of eliminating different sources of noise.

Our findings are surprising, and suggest that the impact of non-determinism in nuanced. While top-line metrics such as top-1 accuracy are not noticeably impacted, model performance on certain parts of the data distribution is far more sensitive to the introduction of randomness. Our results suggest that deterministic tooling is critical for AI safety. However, we also find that the cost of ensuring determinism varies dramatically between neural network architectures and hardware types, e.g., with overhead up to *746%, 241%, and 196%* on a spectrum of widely used GPU accelerator architectures, relative to non-deterministic training. The source code used in this paper is available at https://github.com/usyd-fsalab/NeuralNetworkRandomness.

1 Introduction

In the pursuit of scientific progress, a key desiderata is to eliminate noise from a system. As scientists, we typically regard noise as all the random variations *independent* of the signal we are trying to measure. In the field of machine learning, the urgency to remove noise from training is often motivated by 1) concerns around replicability of experiment results, 2) having full experimental control and/or 3) the need to precisely audit AI behavior in safety-critical domains where human welfare may be harmed.

Recent work has disproportionately focused on the impact of algorithm design choices on model replicability [Nagarajan et al., 2018, Madhyastha and Jain, 2019, Summers and Dinneen, 2021, Snapp and Shamir, 2021, Shamir et al., 2020, Lucic et al., 2018, Henderson et al., 2017]. Less well explored or understood is how our choice of tooling impacts the level of noise in a machine learning system. While some recent work has evaluated the role of software dependencies [Pham et al., 2020, Hong et al., 2013], this has been evaluated in the context of a single machine. In parallel, the quest for determinism has spurred the design of hardware

(a) **Small CNN CIFAR-10**

(b) **ResNet-18 CIFAR-10**

(c) **ResNet-18 CIFAR-100**
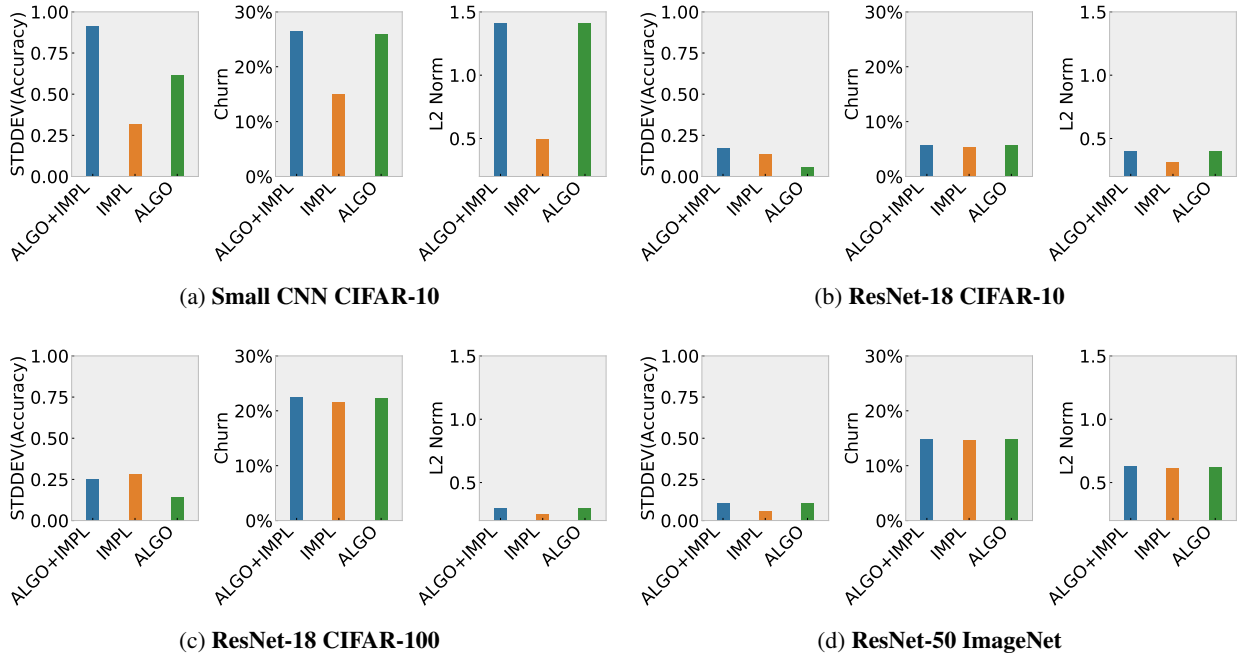
(d) **ResNet-50 ImageNet**

Figure 1: Comparison of different source of noise on standard deviation of accuracy, predictive churn and L2 distance between trained weights (on V100 GPU). *Implementation noise* (IMPL) introduces less uncertainty than *algorithmic noise* (ALGO) in terms of Churn and L2 distance, but each is a significant source of uncertainty

that is inherently deterministic Jooybar et al. [2013], Chou et al. [2020], Jouppi et al. [2017] and software patches that ensure determinism in popular deep learning libraries such as Tensorflow [Abadi et al., 2016], Jax [Bradbury et al., 2018], Pytorch [Paszke et al., 2019], and cuDNN [Chetlur et al., 2014].

In our rush to eliminate noise from ML systems, we seem to have skipped a crucial step – characterizing the origins of the problem and the cost of controlling noise in the system. Understanding the sources of noise in ML systems and the downstream impact is critical in order to weigh the benefits of controlling noise at different levels of the technology stack. How does the choice of hardware, software and algorithm individually contribute to the overall system-level noise? Here, we seek to identify individual sources of randomness at different levels of the technology stack. We separately isolate and evaluate the contribution of both *algorithmic choices* (i.e. random initialization, data shuffling, random layers and stochastic data augmentation), and *implementation choices* which is the combination of hardware and software used to train the model. Our work is the first to our knowledge to evaluate the impact of different widely used hardware types, and also quantify differences in the cost of controlling noise across hardware.

Our results are surprising, and suggest that a more nuanced understanding of noise can also inform our understanding of how our tooling impacts generalization. We find that both algorithmic and hardware factors exert minimal difference in top-line metrics. However, we observe a far more pronounced impact on the level of predictive divergence between different model runs, the standard deviation of per-class metrics and sub-group performance. Here, we find that the presence of noise can amplify uncertainty disproportionately on certain subsets of the dataset. While models maintain similar top-line metrics, randomness present during training often causes unacceptable differences in performance on subsets of the population. Notably, we find that non-determinism at all levels of the technology stack can amplify model bias by disproportionately increasing variance in performance on underrepresented sensitive sub-groups.

Our results suggest that deterministic tooling is critical for ensuring AI safety in sensitive domains such as credit scoring, health care diagnostics [Xie et al., 2019, Gruetzemacher et al., 2018, Badgeley et al., 2019, Oakden-Rayner et al., 2019] and autonomous driving [NHTSA, 2017]. However, our work also establishes that the cost of fully ensuring determinism is large and *highly* variably due to the sensitivty

Algorithmic Sources of Randomness

Source	Method
Random Initialization	weights initialized by sampling random distribution [Glorot and Bengio, 2010, He et al., 2016]
Data augmentation	stochastic transformations to the input data [Kukačka et al., 2017, Hernández-García and König, 2018, Dwibedi et al., 2017, Zhong et al., 2017]
Data shuffling	inputs shuffled randomly and batched during training [Smith et al., 2018]
Stochastic Layers	e.g. dropout [Srivastava et al., 2014, Hinton et al., 2012, Wan et al., 2013], noisy activations [Nair and Hinton, 2010]

Table 1: Overview of different sources of algorithm (`ALGO`) noise.

to model design and underlying hardware. Controlling implementation noise comes with non-negligible training speed overhead for which researchers should weigh the price and benefit based on their tolerance of uncertainty and the sensitivity of the task.

Our core contributions can be enumerated as follows:

1. We establish a rigorous framework for evaluating the impact of tooling on different measures of model stability. We establish consistent results across an extensive experimental set-up, conducting large-scale experiments across different hardware, accelerators, widely used training architectures and datasets (Section 3.1).

2. *Non-determinism must be controlled at all levels of the technical stack or is not worth controlling at all.* Even if algorithmic factors are controlled, the noise from tooling alone is substantial. This suggests that removing a single source of noise cannot effectively reduce the level of uncertainty of trained models (Section 3.2). The overall level of system noise is highly dependent on model design, with choices such as the presence of batch-normalization [Ioffe and Szegedy, 2015] driving differences in model stability.

3. *Non-determinism has a pronounced impact on sub-aggregate measures of model stability.* While we observe minimal impact on top-line metrics, we find that model performance on certain sub-sets of the distribution is far more sensitive, with underrepresented attributes disproportionately impacted by the introduction of stochasticity (Section 3.2).

4. *Large variance in overhead introduced by deterministic training.* Controlling for implementation noise poses significant overhead to model training procedures – with overhead up to *746%, 241%, and 196%* on a spectrum of widely used GPU accelerator architectures, relative to non-deterministic training (Section 4).

2 Methodology

We consider a supervised learning setting,

$$\mathcal{D}\big\{(x_1, y_1), \ldots, (x_N, y_N)\big\} \subset \mathcal{X} \times \mathcal{Y} \tag{1}$$

where $\mathcal{X}$ is the data space and $\mathcal{Y}$ is the set of outcomes that can be associated with an instance.

A neural network is a function $f_w : \mathcal{X} \mapsto \mathcal{Y}$ with trainable weights $w \in W$. Given training data, our model learns a set of weights w^* that minimize a loss function L. Stochastic factors that impact the distribution of the learned weights w^* at the end of training include both algorithm design choices (`ALGO`) that introduce noise to the training process and implementation choices (`IMPL`).

Algorithmic Factors (`ALGO`) - model design choices which are stochastic by design. For example, random initialization [Glorot and Bengio, 2010, He et al., 2016], data augmentation [Kukačka et al., 2017, Hernández-García and König, 2018], data shuffling ordering [Smith et al., 2018], and stochastic layers [Srivastava et al., 2014, Hinton et al., 2012, Wan et al., 2013, Nair and Hinton, 2010, Merity et al., 2017]. In

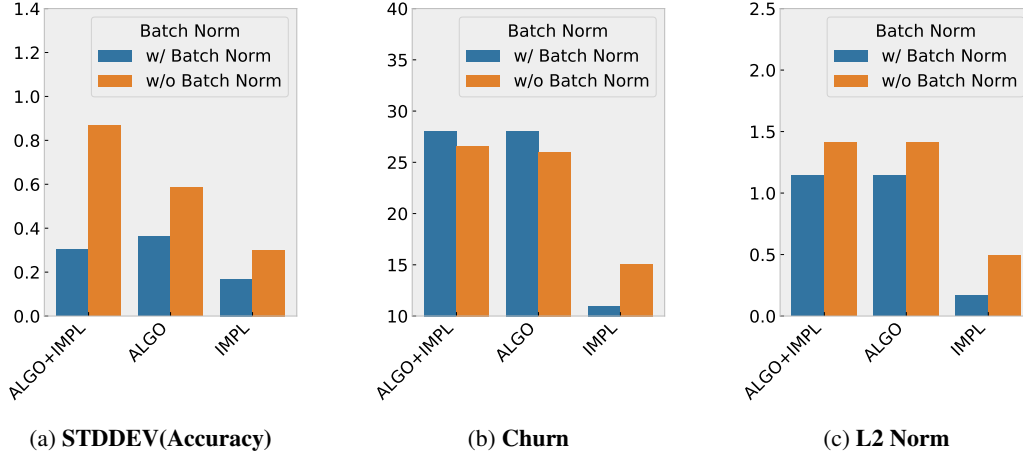(a) **STDDEV(Accuracy)** (b) **Churn** (c) **L2 Norm**

Figure 2: Model design choices can amplify or curb impact of noise. Comparison of standard deviation of accuracy, prediction churn and l2 norm of 3-layer small CNN on CIFAR-10 dataset when trained with and without batch normalization in each layer.

Appendix A, we include a more detailed treatment of the widely-used model design choices that introduce stochasticity in DNN training.

Implementation Factors (`IMPL`) - noise introduced by software choices (e.g. Tensorflow [Abadi et al., 2016], PyTorch [Paszke et al., 2019], cuDNN [Chetlur et al., 2014]) as well as hardware accelerators' architectures (e.g., modern GPU hardware designs [NVIDIA, 2016, 2017, 2018]). The following describes two typical scenarios causing implementation noises.

- **Parallel Execution** - Popular general-purpose DNN accelerators (e.g., GPUs) leverage highly parallel execution for speed-up in execution. However, these sophisticated software-hardware designs for fine-grained massive parallelism typically aims to maximize resource utilization for execution speed and throughput *rather* than output accuracy/precision. GPUs introduce stochasticity due to random floating-point accumulation ordering, which often cause inconsistent outputs between multiple runs due to the truncation of fraction part in floating point number in the accumulation procedure [Chou et al., 2020].

- **Input Data Shuffling and Ordering** - While input data shuffling induces algorithmic noise, it also induces implementation noise due to the different input ordering. Differences in input data ordering can result in different floating point accumulation orders for the reduction operations across data points which are often a overlooked source of implementation noise.

2.1 Measures of Model Stability

In this work, we are focused on measuring the impact of randomness on model stability, defined as *ensuring that given the same experimental framework and tooling, the variation of the training outcome for given input dataset*. To this end, we evaluate the impact on both top-line metrics, but also more granular measures of model stability such as predictive churn, l2 norm and sub-group performance, as different measures of model stability. We briefly introduce each below.

Churn (`churn`)- Predictive churn is a measure of predictive divergence between two models. In sensitive domains such as medicine, consistent individualized predictions are of paramount importance, as there can be severe costs for inconsistent model behavior with a risk to human life [Council, 2011]. Thus, understanding the factors that amplify churn is of considerable research interest with several different proposed definitions of predictive churn [Chen et al., 2020, Shamir and Coviello, 2020, Snapp and Shamir, 2021]. We define churn between two models f_1 and f_2 as done by [Milani Fard et al., 2016] as the fraction of test examples

Table 2: Test-set accuracy with standard deviation under each type of noise. We report the average of 10 models trained independently from scratch.

Hardware	Task	Test Accuracy		
		ALGO+IMPL	ALGO	IMPL
P100	SmallCNN CIFAR-10	$62.28\% \pm 0.83$	$61.44\% \pm 0.41$	$61.61\% \pm 0.31$
	ResNet18 CIFAR-10	$93.33\% \pm 0.14$	$93.32\% \pm 0.13$	$93.12\% \pm 0.11$
	ResNet18 CIFAR-100	$73.37\% \pm 0.23$	$73.42\% \pm 0.26$	$73.36\% \pm 0.17$
RTX5000	SmallCNN CIFAR-10	$62.24\% \pm 0.64$	$62.13\% \pm 0.85$	$62.36\% \pm 0.16$
	ResNet18 CIFAR-10	$93.34\% \pm 0.11$	$93.44\% \pm 0.19$	$93.13\% \pm 0.09$
	ResNet18 CIFAR-100	$73.30\% \pm 0.16$	$73.52\% \pm 0.15$	$73.34\% \pm 0.24$
V100	SmallCNN CIFAR-10	$62.03\% \pm 0.91$	$62.35\% \pm 0.61$	$61.69\% \pm 0.31$
	ResNet18 CIFAR-10	$93.32\% \pm 0.17$	$93.44\% \pm 0.05$	$93.41\% \pm 0.13$
	ResNet18 CIFAR-100	$73.42\% \pm 0.25$	$73.35\% \pm 0.14$	$73.41\% \pm 0.28$
	ResNet50 ImageNet	$76.58\% \pm 0.10$	$76.61\% \pm 0.10$	$76.60\% \pm 0.05$

where the predictions of two models disagree.:

$$C(f_1, f_2) = \mathbb{E}_{\mathcal{X}} \left[\mathbb{1}_{\{\hat{y}_{x;f_1} \neq \hat{y}_{x;f_2}\}} \right] \qquad (2)$$

where $\mathbb{1}$ is an indicator function for whether the predictions by each model match.

L2 norm (`l2`) - L2 norm of the trained weights $\|w_1^* - w_2^*\|$ between f_1 and f_2 at the end of training indicates the divergence of each run in function space. We normalize the weight vector to a unit vector before computing l2 norm, for a consistent visualization scale across a variety of experiments.

Standard Deviation of top-line and sub-group metrics (`stdev`) - In addition to the standard deviation of top-1 test-set accuracy over independent runs, we measure deviation in sub-group performance as measured by sub-group error rate, false positive rate (FPR) and false negative rate (FNR). We compute standard deviation over 10 independent runs unless indicated otherwise.

2.2 Experimental Setup

We conduct extensive experiments across large-scale datasets (CIFAR-10 and CIFAR-100 [Krizhevsky, 2012], ImageNet [Russakovsky et al., 2015] and CelebA [Liu et al., 2015]) and widely-used networks including ResNet-18 and ResNet-50 [He et al., 2016], DenseNet-121 and DenseNet-201 [Huang et al., 2017], Inception-v3 [Szegedy et al., 2015], MobileNet [Sandler et al., 2018], EfficientNet [Tan and Le, 2020], three-layer small CNN and six-layer medium CNN (Appendix C). For all the experiment variants with the exception of ImageNet, we report the average performance metric over 10 models independently trained from scratch. For ImageNet, given the higher training cost, we report average performance across 5 independent trains. Table 2 includes the baseline accuracy given each dataset/model combination we train. A detailed description of training methodology for each dataset and model architecture combination is included in Appendix B. We preserve the same hyperparameter choices across hardware types and use Tensorflow [Abadi et al., 2016] 2.4.1, CUDA 11, and cuDNN [Chetlur et al., 2014] 8 for all the experiments.

GPU - we evaluate NVIDIA P100 with an older Pascal architecture [NVIDIA, 2016] and later generations NVIDIA V100 [NVIDIA, 2017], Nvidia RTX5000 and T4 [NVIDIA, 2018] with Volta and Turing architecture respectively. Our choice of GPUs allows us to evaluate the impact of different levels of parallelism, as P100, V100, RTX5000, and T4 GPU are each equipped with 3584, 5120, 3072, and 2560 CUDA Cores for floating point computation, respectively. In addition, we compare GPUs with and without Tensor Cores accelerators by evaluating both Pascal and Turing architectures. GPU generations with Turing architectures have multiple dedicated matrix multiplication units called Tensor Cores to provide massive computation throughput.
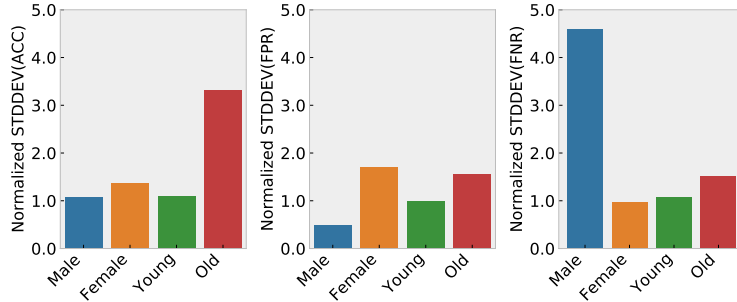
Figure 3: STDDEV(Accuracy) of each sub-group of ResNet18 trained on CelebA dataset using V100. Y axes is normalized against corresponding metric of overall dataset. Noise is disproportionately impacting *Old* and *Male* sub-group as these sub-groups have fewer data points for the positive class.

Table 3: Data points distribution in CelebA dataset

	Male	Female	Young	Old
Positive Data Points	**1387 (0.8%)**	22880 (14.1%)	20230 (12.4%)	**4037 (2.5%)**
Negative Data Points	66874 (41.1%)	71629 (44.0%)	106558 (65.5%)	31945 (19.6%)

TPU - A TPU [Jouppi et al., 2017] is a custom ASIC which differs from GPUs that employ arithmetic logic unit (ALU) as the basic building blocks to offer massive parallel computation. In contrast, TPUs leverage systolic arrays [Kung, 1982] in matrix unit (MXU) to provide massive computation throughput with a single-threaded, deterministic computation model. Thus, TPUs are designed to be deterministic, which differs from the time-varying optimizations of CPUs and GPUs such as caches, out-of-order-execution, multithreading, MIMD/SIMD and prefetching, etc.

We benchmark four key experimental variants which allows us to independently measure the impact of both algorithm (`ALGO`) and implementation (`IMPL`) factors on downstream model performance:

Both Algorithm + Implementation noise - (`ALGO + IMPL`). Here, we do not control for either algorithmic or implementation factors that introduce randomness. This is the default setting of the model training procedure.

Only Algorithm noise - (`ALGO`). We measure the impact of stochastic algorithmic factors by fully controlling all noise introduced by tooling. The implementation noise controlling feature is supported by many prevalent deep learning frameworks such as Tensorflow [Abadi et al., 2016] and Pytorch [Paszke et al., 2019]. Note that controlling implementation noise is far from free (Section 4).

Only Implementation noise - (`IMPL`). We measure the impact of implementation noise by using a fixed random seed for all stochastic algorithm factors. This results in deterministic weights initialization, data augmentation and batch shuffling.

Control - This `Control` variant both sets a fixed random seed to control algorithmic noise and uses software patches to eliminate implementation noise.

3 Results: Characterizing the Impact of Randomness

In this section we address the following questions: **1)** How do implementation and algorithmic noise contribute to system level noise? **2)** How do both impact model stability? **3)** How does varying choices of hardware, low-level vendor libraries and architecture impact the level of noise in the system, and **(4)** Why are certain model design choices far more sensitive to noise?
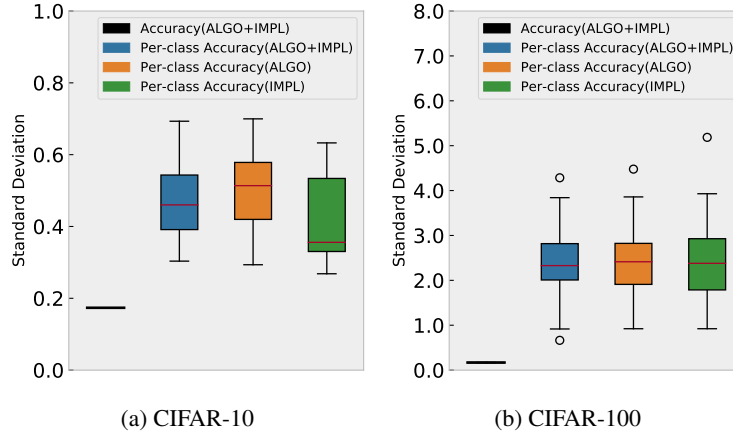
(a) CIFAR-10   (b) CIFAR-100

Figure 4: Per-class accuracy variance vs. overall accuracy variance of ResNet18 trained on V100 under different factors of noise. Per-class accuracy variance is up to **Left:** 4X larger for Cifar-10 and **Right:** 23X larger CIFAR-100 than overall accuracy.

3.1 Impact of Randomness on Top-Line Metrics

Top-1 Accuracy Across all experiments, we observe small variance in Top-1 accuracy. In Table 2, the maximum standard deviation in accuracy is 0.91% for the small cnn trained on CIFAR-10, and the minimum standard deviation is 0.05% for ResNet-10 trained on ImageNet. Top-line metrics do not differ substantially between algorithmic and implementation factors, we observe there is less than 1% standard deviation between `ALGO`, `IMPL` and `ALGO+IMPL` aross all datasets and networks. We observe a maximum absolute difference of 0.84% on the small cnn network trained on CIFAR-10.

Model Stability Metrics A closer inspect of `l2`, `churn` and `stdev` measures in Figure 1 show that both `ALGO` and `IMPL` factors create significant levels of model instability across each of these measures. While for most networks and measures, `ALGO` contributes higher levels of instability relative to `IMPL` factors, this is not always a pronounced gap. For example, on ResNet-50 ImageNet, the impact of predictive churn of `IMPL` factors is 14.68% versus `ALGO` factors is 14.89%. Our results show that `IMPL` can be a significant source of non-determinism accumulated along the trajectory of model training procedure. Due to the non-linearities in deep neural network training, simply removing a single source of noise cannot effectively reduce the level of uncertainty of trained models.

Combined sources of noise (`ALGO + IMPL`) are a non-additive combination of individual factors. For example, the impact of (`ALGO + IMPL`) factors on churn for ResNet-18 and ResNet-50 is on par or only slightly higher than the impact of *only* `IMPL` or `ALGO` noise. The lack of an additive relationship between different sources of noise suggests there is an upper bound in what level of overall system noise is possible.

The role of model design choices In Figure 1, we observe pronounced amplification of noise in the small CNN relative to ResNet-18 for CIFAR-10 with far higher `stdev`, `churn` and `l2` for all sources of noise. The small CNN is the only architecture we benchmark without batch normalization (BN) [Ioffe and Szegedy, 2015], a standard technique for stabilizing training [ab Tessera et al., 2021]. To understand the role of model design choices at curbing or amplifying overall noise in the system, we evaluate the impact to training with and without BN. We compare the small CNN trained without BN to the same architecture trained with BN. In Figure 2 (a), we show that BN has a pronounced impact with a decline in the `stddev` of the accuracy from 0.86% without BN to a much small 0.30% with BN.

We note that architecture appears to play a larger role than dataset in the amplification or curbing of system noise. For example, in Figure 1 the difference in standard deviation between small CNN (0.91%) and ResNet-18 (0.17%) is far larger than the difference between ResNet-18 trained on CIFAR-10 (0.17%) vs the same architecture trained on CIFAR-100 (0.25%).

7

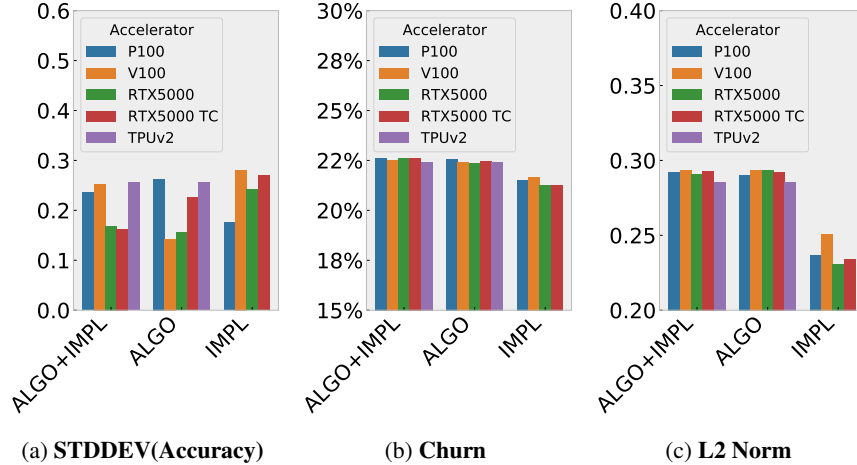(a) **STDDEV(Accuracy)**   (b) **Churn**   (c) **L2 Norm**

Figure 5: Comparison of standard deviation of accuracy, prediction churn and l2 norm of ResNet18 on CIFAR-100 dataset between different training accelerators.
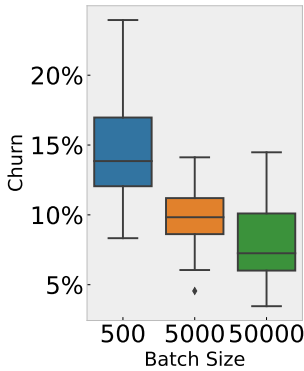

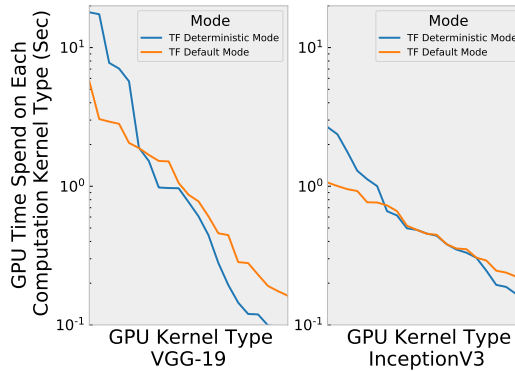Figure 6: Data input order introduces additional non-determinism on TPU.

Figure 7: Top-20 GPU kernels cumulative runtime comparison (Top-1 is on the left side). X-axis indicates the different type of kernels scheduled on GPU. Y-axis is in log scale indicates the cumulative time spend on each type of GPU kernel.

3.2 Impact of Randomness on Sub-Group Performance

How does noise impact sub-group performance? We decompose top-line metrics along class label dimension on CIFAR-10/100 dataset [Krizhevsky, 2012] and CelebFaces Attributes (CelebA) dataset Liu et al. [2015]. In Figure 4, we train models on CIFAR-10/100 under `ALGO+IMPL`, `ALGO`, and `IMPL` respectively. We observe high variance of per-class accuracy of `ALGO` and `IMPL` group similar to models trained under `ALGO+IMPL`. It is clear that removing partial source of noise does not effectively improve model stability. The maximum per-class standard deviation of accuracy is 4X and 23X on CIFAR-10 and CIFAR-100 dataset compared to standard deviation of top-1 accuracy.

CelebA is a dataset of celebrity images where each image is associated with 40 binary labels identifying attributes such as hair color, gender, and age. Our goal is to understand the implications of noise on model bias and fairness considerations. Thus, we focus attention on two protected unitary attributes `Male`, `Female` and `Young` and `Old`. In Figure 3, we can see that (`ALGO+IMPL`) noise is resulting unstable metrics on underrepresented *Male* and *Old* subgroups leading to disproportionate high-variance up to 3.3X on standard deviation on accuracy of *Old* group and 4.6X standard deviation on FNR of *Male* group. Thus, We conclude that even if the top-line metric variation is small enough, noise still imposes disproportionate high variance on dis-aggregated metrics.

Why are certain parts of the data distribution more sensitive to noise? We observe a correlation between underrepresented sub-groups and the sub-groups with the most pronounced impact in variance. In Figure 3,
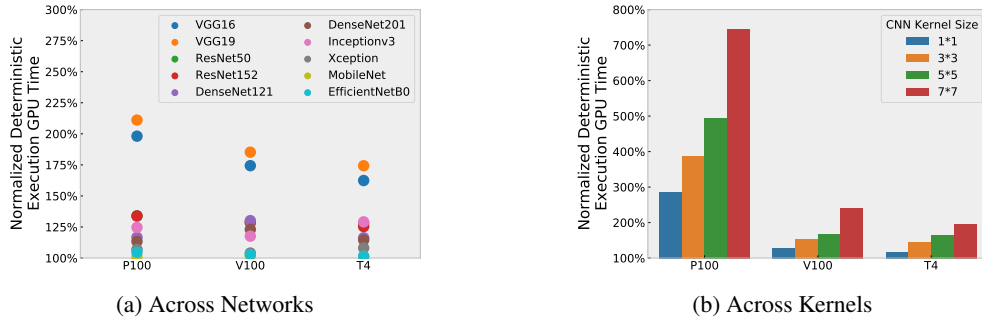
(a) Across Networks

(b) Across Kernels

Figure 8: Comparison of GPU overhead of deterministic setting relative to non-deterministic training setting on **Left:** Ten widely used neural networks, **Right:** A six-layers medium CNN (Appendix C) plugged with different size convolution kernels.

the classes disproportionately impacted *Male* and *Old* as they are heavily underrepresented in the training dataset with 0.8% and 2.5% positive labels as a fraction of the entire dataset (see Table. 3). This suggests stochasticity disproportionately impact features in the long-tail of the dataset.

3.3 How does noise level vary across hardware types?

We trained ResNet-18 on CIFAR-10 and CIFAR-100 dataset using different accelerators including GPU using CUDA Cores (P100, V100, RTX5000), GPU using Tensor Cores (RTX5000 TC), and TPUv2-8 chip [Norrie et al., 2021]. For each hardware except TPU, we measure model divergence under each variant of source of noise (i.e. `ALGO+IMPL`, `ALGO`, and `IMPL`).

Number of CUDA Cores In Figure 5, we compare all hardware types we evaluate on CIFAR-100. In the appendix, we include additional breakdowns for each dataset/model/hardware evaluated (Figure 9 and Figure 10). For all GPUs we evaluate, V100 results in larger divergence under implementation noise in terms of both `churn` and `l2`. We attribute this to the relatively larger number of CUDA cores in V100 GPUs than either P100 and RTX5000, which suggests increased parallelism is a key driver of implementation noise.

Accelerator comparison Surprisingly, we find that `IMPL` impact on `churn` and `l2` is still high for RTX5000 Tensor Cores which employ systolic arrays similar to TPUs to accelerate computation. The high level of `IMPL` despite the systolic design appears to be due to the reliance of Tensor Cores on non-deterministic CUDA cores on GPU for computations that not supported. Thus, model training leveraging Tensor Cores computation remains non-deterministic and is introducing a similar level of noise compared to CUDA Cores.

In Figure 5, it is visible that for `ALGO+IMPL` TPUs incurs a lower level of churn and l2 in weights compared to GPUs. This difference is due to the inherently deterministic design of TPUs, such that any stochasticity is only introduced algorithmic factors even under `ALGO+IMPL` setting. We oberve that while TPU lower `churn` and `l2` relative to GPUs, there is not a pronounced impact on `stdev`. This is consistently with our wider observation across experiments, we note that removing individual sources of noise tends to slightly reduce `churn` and `l2`, but does not have an observable relationship with `stddev` which appears far more sensitive to the presence of *any* source of noise.

Non-determinism based upon differences in ordering Both GPUs and TPUs can introduce hardware noise because of it. In Figure 6, we train ten small CNNs on CIFAR-10 dataset for each batch size, with all source of noise fixed except data shuffling order. When the batch is 50000, the full dataset is packed into a signal training batch, mathematically in this case all models should produce identical result. Interestingly, we still observe divergence of predictions between end runs for all batch size we evaluate. TPUs are designed for single-threaded, deterministic execution mode but are not ensured to be deterministic to ordering in data. This is because the difference in input data order will result in different float-point accumulation order in gradients accumulation stage thus introducing latent implementation noise.

9

4   Results: The Cost of Ensuring Determinism

Algorithmic noise can be controlled by simply setting a fixed random seed. In contrast, controlling implementation noise comes with non-negligible overhead. Most popular deep learning frameworks leverage cuDNN for high-performance computation on GPU. Some convolution algorithm implementation in cuDNN are designed to trade determinism for execution speed. Thus, the cost of controlling implementation noise should be thoroughly analyzed.

Profiling Experiments We profile the overhead of deterministic settings relative to normal training (`ALGO` + `IMPL`) by measuring GPU time spend on CUDA kernel computation using nvprof profiler [NVIDIA]. This metric is well suited for our experiments since it is focusing how much time GPU spends on computation and excludes time intervals we do not care such as latency on data input pipeline. We select networks that are widely used such as MobileNet [Howard et al., 2017], EfficientNets [Tan and Le, 2020] , DenseNet-121/201 [Huang et al., 2017], VGG-16/19 [Simonyan and Zisserman, 2015] and ResNet-50/152 [He et al., 2016]. We profile all models on ImageNet dataset with input shape 224*224 and batch size of 64.

How does the choice of model architecture impact overhead? Figure 8 (a) shows the relative deterministic overhead of a variety of popular CNN models. VGG-19 has the most significant overhead among the models we profiled on all GPUs, with a 185% relative GPU time compared to non-deterministic counterpart on V100 whereas MobileNet has only 101% relative GPU time compared to to non-deterministic counterpart. P100 and T4 also present a large variation of deterministic overhead associate with different model architectures with range $101\% \sim 211\%$ and $101\% \sim 196\%$ respectively.

The role of filter size To further understand the relative overhead of variation in size of convolutional filters, we evaluate across different kernel sizes using a six layer medium CNN (Appendix C). Assembled with convolution kernel size ranging from $1 * 1$ to $7 * 7$. As show in Figure 8 (b), the GPU overhead time is remarkably sensitive to the size of kernel, with $284\% \sim 746\%$ on P100, $129\% \sim 241\%$ on V100, and $117\% \sim 196\%$ on T4 respectively. For all kernel size we evaluate on each GPU, larger kernel size is always comes with larger overhead.

How does the choice of hardware impact overhead? We observe that GPU architecture overhead varies considerably and is highly dependent on model design choices. For example, as shown in Figure 8 (b), we observe overhead for a 7*7 kernel relative to default mode is up to 746%, 241%, and 196% on *P100*, *V100*, and *T4* respectively. Consistently, across all models ranging from six-layer medium CNN to ResNet50, GPUs with older *Pascal* architecture (P100) evidence higher overhead than GPUs with later *Volta* (V100) and *Turing* (T4) architecture.

The large and highly variable overhead suggests deterministic training comes with non-negligible overhead on which researchers should weigh the price and benefit based on their tolerance to uncertainty. However, even the minimum observed overhead poses significant hurdles to efficient training. In Figure 7, we plot the time spent using the Top-20 kernels selected across 100 steps of training. The more skewed time allocation of deterministic mode shows the heavy dependency on a narrower set of kernels instead tuning the best one heuristically. This cost can be attributed to the narrow range of kernels the compiler is forced to use when deterministic training is selected.

5   Related Work

Reproducibility in machine learning As numerous works have pointed out [Goodman et al., 2016, Gundersen and Kjensmo, 2018, Barba, 2018, Drummond, 2009], the word reproducibility can correspond to very different standards, ranging from the ability to reproduce statistically similar values [Raff, 2019, McDermott et al., 2019, Thavasimani and Missier, 2016], to successfully executing code [Collberg and Proebsting, 2016], to the ability to reproduce a relative relationship (a model remains state of art even when the experimental set-up is changed) [Bouthillier et al., 2019]. In this work, we are concerned with replicability, a subset of reproducibility where the standard is reproducing the exact results given the same experimental framework. Advances in tooling have aimed to simplify replication, ranging from shareable notebooks [Kluyver et al., 2016], dockerization [Merkel, 2014], machine learning platforms where code and data is

easily shareable [Isdahl and Gundersen, 2019] and software patches to ensure determinism for a subset of operations. Less mature ideas include research around automatic generation of code from research papers [Sethi et al., 2017]. In the computer architecture research community, researchers have proposed several deterministic GPU architectures [Jooybar et al., 2013, Chou et al., 2020] to boost the reproducibility and debuggability of GPU workloads.

Impact of Algorithmic Factors A substantial amount of work has considered the impact of different sources of randomness introduced by algorithm design choices. Several works have evaluated the impact of a random seed, with [Nagarajan et al., 2018] evaluating the role of random initialization in reinforcement learning, and [Madhyastha and Jain, 2019] measuring how random seeds impact explanations for NLP tasks provided by interpretability methods. [Summers and Dinneen, 2021] benchmark the separate impact of choices of initialization, data shuffling and augmentation. Work mentioned thus far is focused on how design choices that introduce randomness impact training. However, there is a wider body of scholarship that has focused on sensitivity to non-stochastic factors including choice of activation function and depth of model [Snapp and Shamir, 2021, Shamir et al., 2020], hyper-parameter choices [Lucic et al., 2018, Henderson et al., 2017, Kadlec et al., 2017], the use of data parallelism [Shallue et al., 2019] and test set construction [Søgaard et al., 2021, Lazaridou et al., 2021, Melis et al., 2018].

Impact of Software Dependencies [Hong et al., 2013] evaluate the role of different compilers for the specialized task of weather simulation. Recent work by [Pham et al., 2020] and [Alahmari et al., 2020] in the machine learning domain evaluates the impact of randomness introduced by popular deep neural network libraries (Pytorch, CNTK, Theano and Tensorflow). [Alahmari et al., 2020] evaluates a segmentation task for mouse neo-cortex data and MNIST on LeNet [Lecun et al., 1998]. [Pham et al., 2020] finds the biggest variance across all deep learning libraries on LeNet5. These works and others only evaluate the role of software dependencies on a single type of hardware. Our contribution is the first to our knowledge to vary the hardware, and measure the cost of ensuring determinism across different types of hardware.

Trade-off with fairness objectives Recent work [Hooker, 2021, Yona et al., 2021, D'Amour et al., 2020, Hooker et al., 2019] has identified that models with similar top-line metrics can evidence unacceptable performance on subsets of the distribution. Design choices such as compression [Hooker et al., 2020] and privacy [Cummings et al., 2019] can impact disparate impact on sensitive attributes. However, ours is the first to our knowledge that evaluates the impact of tooling and sources of randomness on disparate harm.

6   Discussion and Future Work

There has been increasing urgency to ensure non-determinism in deep neural network training. However, in the rush to mitigating or eliminating noise in deep learning system, a natural question that have not been discussed thoroughly would be: *What is the impact of noise? Does the impact merit the cost of controlling it?*.

Limitations In this work, our focus is evaluating the impact of tooling in a non-distributed setting. However, increasingly training deep neural networks involves data and model parallelism [Shazeer et al., 2018, Langer et al., 2020], partition over optimizer state [Rajbhandari et al., 2020], and asynchronous gradients update [Li et al., 2014]. An important area of future work involves understanding how distributed training impacts model stability.

7   Conclusion

In this work, we seek to characterize the impact and cost of controlling noise at *all* levels of the technical stack. We empirically demonstrate that both algorithmic and implementation noise are significant sources of noise. Thus, simply removing noise from one part of the technical stack is not a robust way to improve training stability. Secondly, we show that even with minimal changes to top-line metrics, there is a disproportionately impact on sub-group performance which can incur fairness trade-offs when protected attributes are underrepresented. Finally, we evaluate the cost of ensuring determinism and find it is highly variable and dependent on hardware type and model design choices.

References

K. ab Tessera, S. Hooker, and B. Rosman. Keep the gradients flowing: Using gradient flow to study sparse network optimization, 2021.

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI*, 2016.

S. S. Alahmari, D. B. Goldgof, P. R. Mouton, and L. O. Hall. Challenges for the repeatability of deep learning models. *IEEE Access*, 8:211860–211868, 2020. doi: 10.1109/ACCESS.2020.3039833.

M. Badgeley, J. Zech, L. Oakden-Rayner, B. Glicksberg, M. Liu, W. Gale, M. McConnell, B. Percha, and T. Snyder. Deep learning predicts hip fracture using confounding patient and healthcare variables. *npj Digital Medicine*, 2:31, 04 2019. doi: 10.1038/s41746-019-0105-1.

L. A. Barba. Terminologies for reproducible research, 2018.

X. Bouthillier, C. Laurent, and P. Vincent. Unreproducible research is reproducible. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 725–734. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/bouthillier19a.html.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Z. Chen, Y. Wang, D. Lin, D. Z. Cheng, L. Hong, E. H. Chi, and C. Cui. Beyond point estimate: Inferring ensemble prediction variation from neuron activation strength in recommender systems, 2020.

S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

Y. Chou, C. Ng, S. Cattell, J. Intan, M. D. Sinclair, J. Devietti, T. G. Rogers, and T. M. Aamodt. Deterministic atomic buffering. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, 2020.

C. Collberg and T. A. Proebsting. Repeatability in computer systems research. *Commun. ACM*, 59(3):62–69, Feb. 2016. ISSN 0001-0782. doi: 10.1145/2812803. URL https://doi.org/10.1145/2812803.

N. R. Council. *Toward Precision Medicine: Building a Knowledge Network for Biomedical Research and a New Taxonomy of Disease*. The National Academies Press, Washington, DC, 2011. ISBN 978-0-309-22222-8. doi: 10.17226/13284. URL https://www.nap.edu/catalog/13284/toward-precision-medicine-building-a-knowledge-network-for-biomedical-research.

R. Cummings, V. Gupta, D. Kimpara, and J. Morgenstern. On the compatibility of privacy and fairness. UMAP'19 Adjunct, page 309–315, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367110. doi: 10.1145/3314183.3323847. URL https://doi.org/10.1145/3314183.3323847.

A. D'Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, F. Hormozdiari, N. Houlsby, S. Hou, G. Jerfel, A. Karthikesalingam, M. Lucic, Y. Ma, C. McLean, D. Mincu, A. Mitani, A. Montanari, Z. Nado, V. Natarajan, C. Nielson, T. F. Osborne, R. Raman, K. Ramasamy, R. Sayres, J. Schrouff, M. Seneviratne, S. Sequeira, H. Suresh, V. Veitch, M. Vladymyrov, X. Wang, K. Webster, S. Yadlowsky, T. Yun, X. Zhai, and D. Sculley. Underspecification presents challenges for credibility in modern machine learning, 2020.

C. Drummond. Replicability is not reproducibility: Nor is it good science. In *Evaluation Methods for Machine Learning Workshop at the 26th International Conference on Machine Learning, ICML*, 2009.

D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection, 2017.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

S. N. Goodman, D. Fanelli, and J. P. A. Ioannidis. What does research reproducibility mean? *Science Translational Medicine*, 8(341):341ps12–341ps12, 2016. ISSN 1946-6234. doi: 10.1126/scitranslmed. aaf5027. URL `https://stm.sciencemag.org/content/8/341/341ps12`.

R. Gruetzemacher, A. Gupta, and D. B. Paradice. 3d deep learning for detecting pulmonary nodules in ct scans. *Journal of the American Medical Informatics Association : JAMIA*, 25 10:1301–1310, 2018.

O. E. Gundersen and S. Kjensmo. State of the art: Reproducibility in artificial intelligence. In *AAAI*, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL `http://arxiv.org/abs/1709.06560`.

A. Hernández-García and P. König. Further advantages of data augmentation on convolutional neural networks. *Lecture Notes in Computer Science*, page 95–103, 2018. ISSN 1611-3349. doi: 10.1007/978-3-030-01418-6_10. URL `http://dx.doi.org/10.1007/978-3-030-01418-6_10`.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

S.-Y. Hong, M.-S. Koo, J. Jang, J.-E. E. Kim, H. Park, M.-S. Joh, J.-H. Kang, and T.-J. Oh. An evaluation of the software system dependency of a global atmospheric model. *Monthly Weather Review*, 141(11): 4165 – 4172, 2013. doi: 10.1175/MWR-D-12-00352.1. URL `https://journals.ametsoc.org/view/journals/mwre/141/11/mwr-d-12-00352.1.xml`.

S. Hooker. Moving beyond "algorithmic bias is a data problem". *Patterns*, 2(4):100241, 2021. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2021.100241. URL `https://www.sciencedirect.com/science/article/pii/S2666389921000611`.

S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome. What Do Compressed Deep Neural Networks Forget? *arXiv e-prints*, art. arXiv:1911.05248, Nov. 2019.

S. Hooker, N. Moorosi, G. Clark, S. Bengio, and E. Denton. Characterising bias in compressed models, 2020.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. doi: 10.1109/CVPR.2017.243.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

R. Isdahl and O. E. Gundersen. Out-of-the-box reproducibility: A survey of machine learning platforms. In *2019 15th International Conference on eScience (eScience)*, pages 86–95, 2019. doi: 10.1109/eScience. 2019.00017.

H. Jooybar, W. W. L. Fung, M. O'Connor, J. Devietti, and T. M. Aamodt. GPUDet: a deterministic GPU architecture. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS '13*, 2013.

N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony,

K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA*, 2017.

R. Kadlec, O. Bajgar, and J. Kleindienst. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 69–74, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2609. URL `https://www.aclweb. org/anthology/W17-2609`.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. development team. Jupyter notebooks ? a publishing format for reproducible computational workflows. In F. Loizides and B. Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. URL `https://eprints.soton.ac.uk/403913/`.

A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

J. Kukačka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy, 2017.

H. T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982. doi: 10.1109/MC.1982.1653825. URL `https://doi.org/10.1109/MC.1982.1653825`.

M. Langer, Z. He, W. Rahayu, and Y. Xue. Distributed training of deep learning models: A taxonomic perspective. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2802–2818, Dec 2020. ISSN 2161-9883. doi: 10.1109/tpds.2020.3003307. URL `http://dx.doi.org/10.1109/TPDS.2020. 3003307`.

A. Lazaridou, A. Kuncoro, E. Gribovskaya, D. Agrawal, A. Liska, T. Terzi, M. Gimenez, C. de Masson d'Autume, S. Ruder, D. Yogatama, K. Cao, T. Kocisky, S. Young, and P. Blunsom. Pitfalls of static language modelling, 2021.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su. Scaling distributed machine learning with the parameter server. In J. Flinn and H. Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pages 583–598. USENIX Association, 2014. URL `https://www.usenix.org/ conference/osdi14/technical-sessions/presentation/li_mu`.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *ICCV*, 2015.

M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? a large-scale study, 2018.

P. Madhyastha and R. Jain. On model stability as a function of random seed. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 929–939, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1087. URL `https://www. aclweb.org/anthology/K19-1087`.

M. B. A. McDermott, S. Wang, N. Marinsek, R. Ranganath, M. Ghassemi, and L. Foschini. Reproducibility in machine learning for health. *ArXiv*, abs/1907.01463, 2019.

G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. *ArXiv*, abs/1707.05589, 2018.

S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models, 2017.

D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014. ISSN 1075-3583.

M. Milani Fard, Q. Cormier, K. Canini, and M. Gupta. Launch and iterate: Reducing prediction churn. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information*

Processing Systems, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/dc5c768b5dc76a084531934b34601977-Paper.pdf`.

P. Nagarajan, G. Warnell, and P. Stone. The impact of nondeterminism on reproducibility in deep reinforcement learning. In *Reproducibility in ML Workshop at the 35th International Conference on Machine Learning, ICML*, 2018.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

NHTSA. Technical report, U.S. Department of Transportation, National Highway Traffic, Tesla Crash Preliminary Evaluation Report Safety Administration. *PE 16-007*, Jan 2017.

T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. P. Jouppi, and D. A. Patterson. The design process for google's training chips: Tpuv2 and tpuv3. *IEEE Micro*, 41(2):56–63, 2021. doi: 10.1109/MM.2021.3058217. URL `https://doi.org/10.1109/MM.2021.3058217`.

NVIDIA. Profiler user's guide. URL `https://docs.nvidia.com/cuda/profiler-users-guide/index.html`.

NVIDIA. NVIDIA Tesla P100, 2016. URL `https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf`.

NVIDIA. NVIDIA TESLA V100 GPU ARCHITECTURE, 2017. URL `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

NVIDIA. NVIDIA TURING GPU ARCHITECTURE, 2018. URL `https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf`.

L. Oakden-Rayner, J. Dunnmon, G. Carneiro, and C. Ré. Hidden Stratification Causes Clinically Meaningful Failures in Machine Learning for Medical Imaging. *arXiv e-prints*, art. arXiv:1909.12475, Sep 2019.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019.

H. V. Pham, S. Qian, J. Wang, T. Lutellier, J. Rosenthal, L. Tan, Y. Yu, and N. Nagappan. Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, page 771–783, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367684. doi: 10.1145/3324884.3416545. URL `https://doi.org/10.1145/3324884.3416545`.

E. Raff. A step toward quantifying independently reproducible machine learning research, 2019.

S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: memory optimizations toward training trillion parameter models. In C. Cuicchi, I. Qualters, and W. T. Kramer, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020. doi: 10.1109/SC41405.2020.00024. URL `https://doi.org/10.1109/SC41405.2020.00024`.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 2015.

M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

A. Sethi, A. Sankaran, N. Panwar, S. Khare, and S. Mani. Dlpaper2code: Auto-generation of code from deep learning research papers, 2017.

C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training, 2019.

G. I. Shamir and L. Coviello. Anti-distillation: Improving reproducibility of deep networks. *CoRR*, abs/2010.09923, 2020. URL `https://arxiv.org/abs/2010.09923`.

G. I. Shamir, D. Lin, and L. Coviello. Smooth activations and reproducibility in deep networks, 2020.

N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman. Mesh-tensorflow: Deep learning for supercomputers. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10435–10444, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/3a37abdeefe1dab1b30f7c5c7e581b93-Abstract.html`.

I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. Anderson. Manipulating sgd with data ordering attacks, 2021.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le. Don't decay the learning rate, increase the batch size, 2018.

R. R. Snapp and G. I. Shamir. Synthesizing irreproducibility in deep networks. *CoRR*, abs/2102.10696, 2021. URL `https://arxiv.org/abs/2102.10696`.

A. Søgaard, S. Ebert, J. Bastings, and K. Filippova. We need to talk about random splits. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1823–1832, Online, Apr. 2021. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2021.eacl-main.156`.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435.

C. Summers and M. J. Dinneen. Nondeterminism and instability in neural network optimization, 2021.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.

M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

P. Thavasimani and P. Missier. Facilitating reproducible research by investigating computational metadata. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3045–3051, 2016. doi: 10.1109/BigData.2016.7840958.

L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/wan13.html`.

H. Xie, D. Yang, N. Sun, Z. Chen, and Y. Zhang. Automated pulmonary nodule detection in ct images using deep convolutional neural networks. *Pattern Recognition*, 85:109 – 119, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2018.07.031. URL `http://www.sciencedirect.com/science/article/pii/S0031320318302711`.

G. Yona, A. Ghorbani, and J. Zou. Who's responsible? jointly quantifying the contribution of the learning algorithm and training data, 2021.

Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation, 2017.

8 Appendix

A Sources of Randomness During Deep Neural Network Training

Algorithmic Factors (`ALGO`) includes model design choices which are stochastic by design. Often, there are widely used implementation choices as introducing stochasticity to deep neural network training has been found to improve top-line metrics:

- **Random Initialization** - the weights of a deep neural network are randomly initialized, typically with the goal is maintaining variance of activations within a narrow range at the beginning of training to avoid gradient saturation ((Glorot and Bengio, 2010, He et al., 2016)).
- **Data augmentation** - the quality of a trained model depends upon the training data. Often, when faced with limited data an effective strategy is to generate new samples by applying stochastic transformations to the input data ((Kukačka et al., 2017, Hernández-García and König, 2018)). Examples of stochastic data augmentation include random crops, noise injection, and random distortions to color channels ((Dwibedi et al., 2017, Zhong et al., 2017)).
- **Data shuffling and ordering** - for mini-batch stochastic gradient optimization, datasets are typically shuffled randomly during training and batched into a subset of observations.. Thus, each training process will observe a different ordering of inputs. Batching examples introduces noise through stochastic mini-batch gradient descent ((Smith et al., 2018)). Even when batching is not used (all data is processed in a single batch), a difference in ordering can introduce stochasticity that may introduce security vulnerabilities ((Shumailov et al., 2021)).
- **Stochastic Layers** - techniques such as dropout which entails randomly dropping a subset of weights each iteration ((Srivastava et al., 2014, Hinton et al., 2012, Wan et al., 2013)), noisy activation functions ((Nair and Hinton, 2010)) or variable length backpropagation through time ((Merity et al., 2017)).

B Training Methodology

We employ random crop and flip for data augmentation on all experiments except experiments on CelebA dataset.

CIFAR-10 and CIFAR-100 ((Krizhevsky, 2012)) We train a small CNN on CIFAR-10 which consists of three convolutional layers, followed by a dense layer and a output layer. Additionally, we evaluate both CIFAR-10 and CIFAR-100 on ResNet-18. For all networks, we train for 200 epochs with a batch size of 128 and $4e - 4$ learning rate which decays by a factor of ten every 50 epochs.

CelebA ((Liu et al., 2015)) CelebA dataset consist of $\sim$200K celebrity's facial images, each image associated with labels with forty binary attributes such as identifying hair color, gender, age. Our goal is to understand the implications of noise on model bias and fairness considerations. Thus, we focus attention on two protected unitary attributes `Male`, `Female` and `Young` and `Old`. Our goal is to understand the implications of noise on model bias and fairness considerations. we measure standard deviation of sub-group accuracy, false positive rate (FPR) and false negative rate (FNR). We train ResNet18 on CelebA dataset for 20 epochs with batch size of 128 and learning rate of $1e - 3$ decays by a factor of ten every 5 epochs.

ImageNet ((Russakovsky et al., 2015)) On ImageNet dataset, we train ResNet-50 for 90 epochs with batch size of 256 with learning rate 0.1 using SGD optimizer with momentum of 0.9, the learning rate is warming up in the first epoch and using cosine decay in the following epochs. We conduct out experiment on Imagenet dataset based on ResNet50 implementation from Tensorflow Model Garden [1]

---

[1] https://github.com/tensorflow/models

C CNN Architecture

Architecture of three-layer small CNN and six-layer medium CNN. Downsampling is performed in pooling layers, all convolutional layers are using stride=1. For six-layer small CNN, kernel size X can be 1, 3, 5, and 7.

Three-layer small CNN		Six-layer medium CNN	
Layer	Output Shape	Layer	Output Shape
Input	$32 * 32 * 3$	Input	$224 * 224 * 3$
$\begin{bmatrix} Conv\ 3*3 \\ Relu \\ MaxPool \end{bmatrix}$	$16 * 16 * 16$	$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$112 * 112 * 16$
$\begin{bmatrix} Conv\ 3*3 \\ Relu \\ MaxPool \end{bmatrix}$	$8 * 8 * 32$	$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$56 * 56 * 32$
$\begin{bmatrix} Conv\ 3*3 \\ Relu \\ MaxPool \end{bmatrix}$	$4 * 4 * 32$	$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$28 * 28 * 64$
		$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$14 * 14 * 128$
		$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$7 * 7 * 256$
		$\begin{bmatrix} Conv\ X*X \\ BN \\ Relu \\ MaxPool \end{bmatrix}$	$3 * 3 * 512$
GlobalAveragePooling			
Dense	32		
Dense	10	Dense	1000

Dataset	Training/Test Split	Number Classes
Cifar-10	50000/10000	10
Cifar-100	50000/10000	100
ImageNet	1281167/50000	1000
CelebA	162770/19962	40 (Multi-label)

Table 4: Overview of each dataset benchmarked.
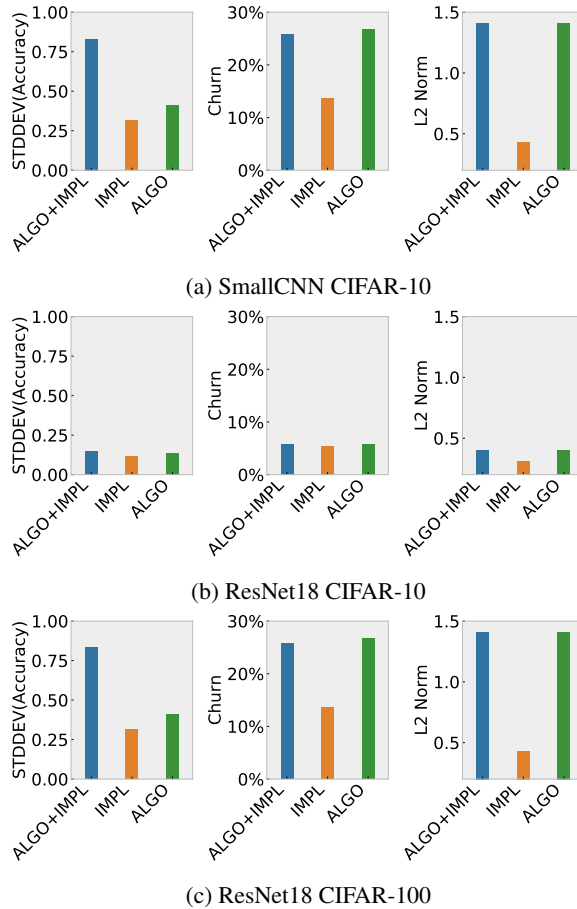
D Comparison of Impact of Different Source of Noise

(a) SmallCNN CIFAR-10

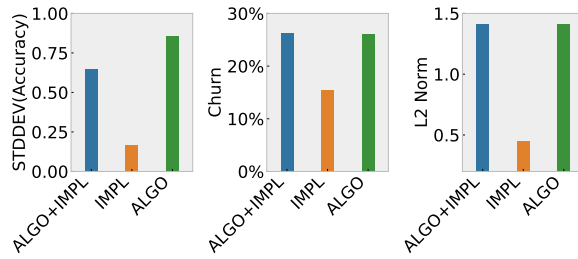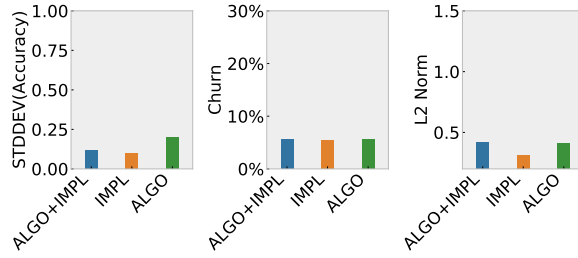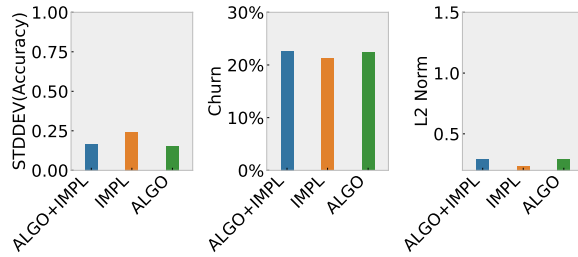(b) ResNet18 CIFAR-10

(c) ResNet18 CIFAR-100

Figure 9: Comparison of impact of different source of noise across on four tasks trained on P100

(a) SmallCNN CIFAR-10

(b) ResNet18 CIFAR-10

(c) ResNet18 CIFAR-100

Figure 10: Comparison of impact of different source of noise across on four tasks trained on RTX5000

Subgroup	ALGO+IMPL	ALGO	IMPL
		STDDEV(Accuracy)	
All	0.045 (1X)	0.051 (1X)	0.090 (1X)
MALE	0.049 (1.07X)	0.048 (0.94X)	0.058 (0.64X)
FEMALE	0.062 (1.36X)	0.083 (1.62X)	0.126 (1.39X)
YOUNG	0.050 (1.10X)	0.047 (0.93X)	0.091 (1.00X)
OLD	0.151 (**3.31X**)	0.094 (1.83X)	0.214 (**2.36X**)
		STDDEV(FPR)	
All	0.077 (1X)	0.051 (1X)	0.070 (1X)
MALE	0.039 (0.50X)	0.052 (1.01X)	0.043 (0.61X)
FEMALE	0.133 (1.71X)	0.094 (1.81X)	0.103 (1.48X)
YOUNG	0.077 (1.00X)	0.051 (0.99X)	0.065 (0.93X)
OLD	0.122 (1.57X)	0.093 (1.81X)	0.155 (**2.21X**)
		STDDEV(FNR)	
All	0.537 (1X)	0.389 (1X)	0.445 (1X)
MALE	2.475 (**4.60X**)	1.816 (**4.66X**)	1.610 (**3.61X**)
FEMALE	0.527 (0.98X)	0.349 (0.89X)	0.399 (0.89X)
YOUNG	0.585 (1.08X)	0.430 (1.10X)	0.566 (1.27X)
OLD	0.815 (1.51X)	0.335 (0.86X)	0.939 (**2.10X**)

Table 5: Standard deviation of mean accuracy, false positive rate (FPR), and false negative rate (FNR) across 10 models trained under baseline setting on the CelebA dataset (trained on V100 (using cuda cores)). Metrics are dis-aggregated across two binary dimensions Male/Female and Young/Old. In parentheses, we report relative scale of standard deviation metrics relative to overall dataset.