RANDOMNESS IN NEURAL NETWORK TRAINING: CHARACTERIZING THE IMPACT OF TOOLING

Donglin Zhuang [1] **Xingyao Zhang** [2] **Shuaiwen Leon Song** [1] **Sara Hooker** [3]

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-the-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 suggest that the impact of non-determinism is 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%* on a spectrum of widely used GPU accelerator architectures, relative to non-deterministic training.

1 INTRODUCTION

In the pursuit of scientific progress, a key desideratum 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.

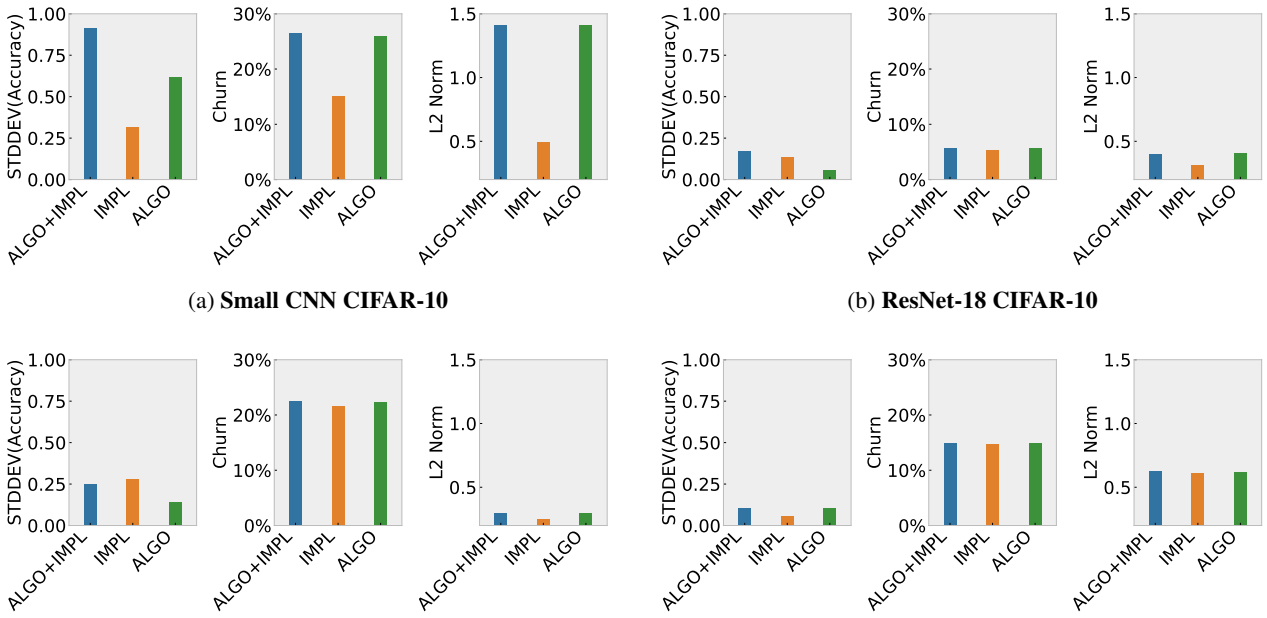
Recent work has disproportionately focused on the impact of algorithm design choices on model replicability (Nagarajan et al., 2018; Madhyastha & Jain, 2019; Summers & Dinneen, 2021; Snapp & Shamir, 2021; Shamir et al., 2020; Lucic et al., 2018; Henderson et al., 2017). Less well explored or understood 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 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 propse a rigorous benchmark to measure 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 noise introduced by tooling, consist of the combination of hardware and software used to train the model (e.g., non-deterministic GPU computation, non-deterministic operators in framework). 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 suggest that a more nuanced understanding of noise can also inform our understanding of how our tooling

[1]School of Computer Science, The University of Sydney [2]Department of Computer Science, University of Washington [3]Google Brain. Correspondence to: Shuaiwen Leon Song <shuaiwen.song@sydney.edu.au>, Sara Hooker <shooker@google.com>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2022.



(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.

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 work is the first to our knowledge to propose an ambitious benchmark for characterizing sources of noise in a machine learning system. 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 sensitivity 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 benchmark for evaluating the impact of tooling on different measures of model stability. We consolidate metrics of interest for the purpose of evaluating noise in a system, and establish results on the impact of widely used tooling across an extensive experimental set-up. We conduct large-scale experiments across different hardware, accelerators, widely used training architectures and datasets (Section 3.1). The results of our large scale benchmarking of deep neural network training provide valuable insights in the nature of stochasticity in ML systems and the cost of controlling this noise.
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 partial sources 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 & 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 distri-

Table 1. 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	$76.58\% \pm 0.10$	$76.61\% \pm 0.10$	$76.60\% \pm 0.05$

bution 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%* on a spectrum of widely used GPU accelerator architectures, relative to non-deterministic training (Section 4).

2 METHODOLOGY

We consider a supervised learning setting,

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

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

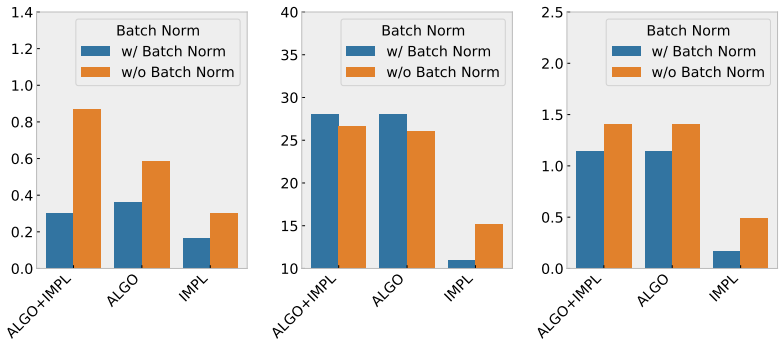
A neural network is a function $f_w : \mathcal{X} \mapsto \mathcal{Y}$ with trainable weights $w \in W$. Given training data, our model learns a set of weights w^* that minimize a loss function L. Stochastic factors that impact the distribution of the learned weights w^* at the end of training include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

Algorithmic Factors (ALGO) 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 & 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 & 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 & Hinton, 2010) or variable length backpropagration through time (Merity et al., 2017).

Implementation Factors (IMPL) - noise that ultimately comes from floating-point number accumulation ordering error[1]. This includes 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 de-

[1]Non-associativity of floating-point arithmetic, e.g., (a+b)+c != a+(b+c).



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

Figure 2. Comparison of standard deviation of accuracy, prediction churn and l2 norm of 3-layer small CNN both **with** and **without** batchnorm on CIFAR-10 dataset.

scribes two common scenarios which are known to cause 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 massive parallelism typically aims to maximize resource utilization for execution speed and throughput *rather* than output accuracy/precision. Thus, GPUs introduce stochasticity due to random floating-point accumulation ordering from parallel threads, 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 is the source 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 focus 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 & Coviello, 2020; Snapp & 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 where the predictions of two models disagree.:

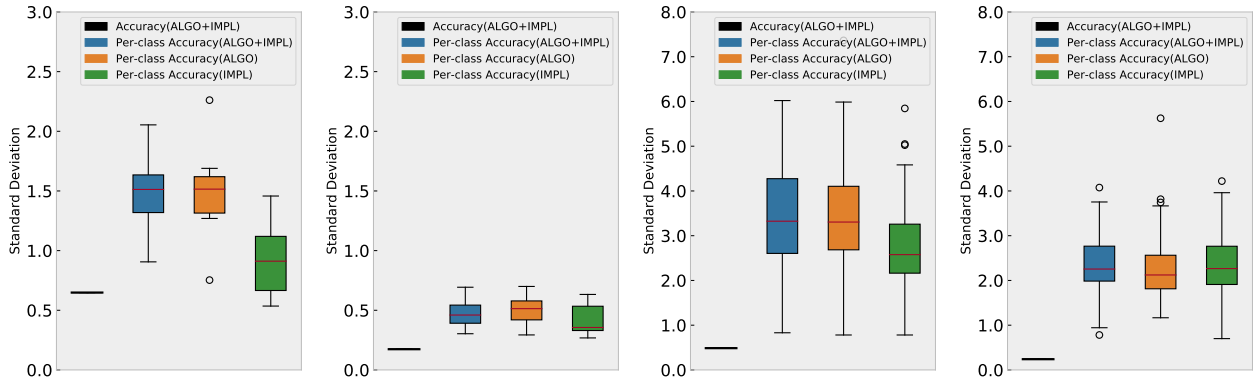
$$C(f_1, f_2) = E_{\mathcal{X}}\left[\mathbb{1}_{\{\hat{y}_{x;f_1} \neq \hat{y}_{x;f_2}\}}\right] \tag{2}$$

where $\mathbb{1}$ is an indicator function for whether the predictions by each model match. Given the sampling size in this paper is always larger than two, we report churn as the average churn of each pairwise combination of models for a given architecture and dataset.

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.

We do note that an exceptionally rare possibility is that two initializations lead to two permuted but otherwise identical final models (hence high L2 distance but otherwise identical behavior). In such a case then L2 would not be a sensitive enough measure to be informative. However, in practice we do believe this would be a extremely unlikely occurrence, as indicated by the acceptance of L2 in the machine learning community as a metric to understand divergence in trained functions (Zhang et al., 2019; Neyshabur et al., 2021).

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



(a) SmallCNN CIFAR-10 (b) ResNet18 CIFAR-10 (c) SmallCNN CIFAR-100 (d) ResNet18 CIFAR-100

Figure 3. Per-class accuracy standard deviation vs. overall accuracy standard deviation of SmallCNN and ResNet18 trained on V100 under different factors of noise. Per-class accuracy standard deviation is significant higher than standard deviation of accuracy across datasets and nerual networks.

(FNR). We compute all measures 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 & Le, 2020), three-layer small CNN and six-layer medium CNN (Appendix B). For all the experiment variants with the exception of ImageNet, we report the average performance metric over 10 models independently trained from scratupch. For ImageNet, given the higher training cost, we report average performance across 5 independent trains. Table 1 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 A. We preserve the same hyperparameter choices across hardware types and use Tensorflow (Abadi et al., 2016) 2.4.1, CUDA 11, and cuDNN 8 (Chetlur et al., 2014) for all the experiments. For CUDA cores experiments we consistently use the same precision for all accumulators. We release the source code used in experiments [2].

GPU - we evaluate NVIDIA P100 with an older Pascal architecture (NVIDIA, 2016) and later generations V100 (NVIDIA, 2017), 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 accelerate matrix multiplication.

TPU - A TPU (Jouppi et al., 2017) is a custom ASIC 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. Appendix C elaborates technical details to achieve this. 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.

[2] https://url-will-be-revealed-upon-acceptance

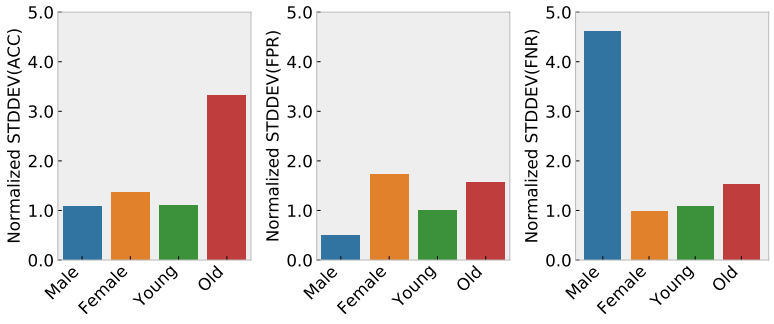


Figure 4. STDDEV(Accuracy) of each sub-group of ResNet18 trained on CelebA dataset using V100. Vertical axes is normalized against corresponding metric of overall dataset. Noise is disproportionately impacting *Old* and *Male* sub-group as these sub-groups have fewer data points for the positive class.

Table 2. 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%)

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

Our protocol is able to rigorously separate each high level grouping of noise (implementation and algorithmic). We hold all experimental details constant, and vary one aspect at a time. This allows us to rigorously say something about how all algorithmic stochastic processes compare to tooling stochastic processes.

3 RESULTS: CHARACTERIZING THE IMPACT OF RANDOMNESS

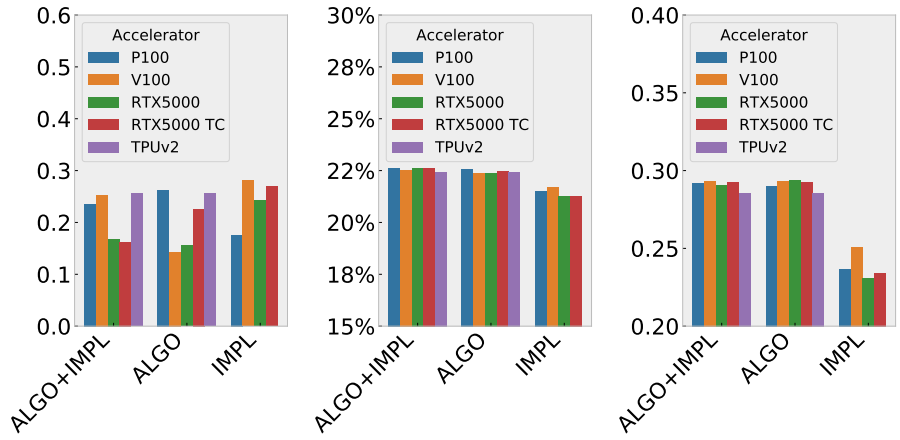
In this section we address the following questions: **1)** How do implementation and algorithmic noise contribute to system level noise? **2)** How do both impact model stability? **3)** How does varying choices of hardware, low-level vendor libraries and architecture impact the level of noise in the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

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 1, 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.

Model Stability Metrics A closer inspect of `l2`, `churn` and `stdev` measures in Figure 1 shows 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 that will keep perturb the 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. Furthermore, 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 & Szegedy, 2015), a standard technique for stabilizing training (Tessera et al., 2021). To understand the role of model design choices at curbing or amplifying noise in the system, 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



(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. Note standard deviation of accuracy is zero under IMPL noise on TPUv2.

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%).

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 3, 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 on ResNet18 compared to standard deviation of top-1 accuracy, we also observe similar effect for small CNN model, with 3X and 23X larger per-class standard deviation of accuracy on CIFAR-10/100 dataset respectively. Interestingly, even for uniformly distributed dataset, per-class accuracy variance still have a large range of divergence.

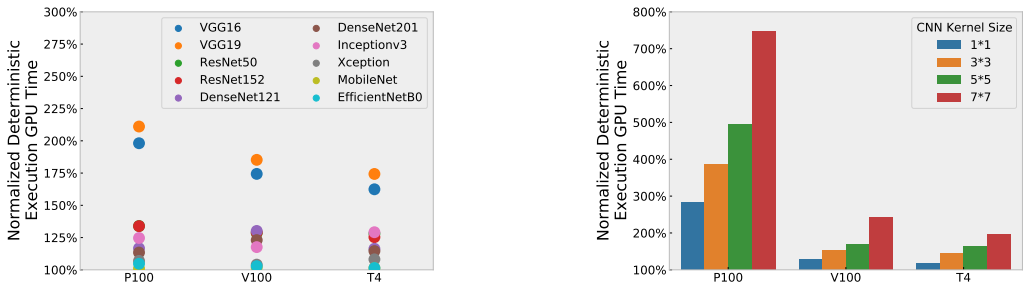
CelebA (Liu et al., 2015) is a dataset of celebrity images where each image is associated with 40 binary labels identifying attributes such as hair color, gender, and age. To understand the implications of noise on model bias and fairness considerations. Thus, we focus attention on two protected unitary attributes `Male`, `Female` and `Young` and `Old`. In Figure 4, 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 certain parts of the data distribution more sensitive to noise? We observe a correlation between underrepresented sub-groups sufferring the most pronounced impact in variance. In Figure 4, 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 2). This suggests stochasticity disproportionately impact features in the long-tail of the dataset.

3.3 How does noise level vary across hardware types?

Number of CUDA Cores In Figure 5, we compare all hardware types we evaluate on CIFAR-100. In the appendix D, 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`. One possible reason for this difference between hardware performance are the relatively larger number of CUDA cores in V100 GPUs than either P100 and RTX5000. This may suggest increased parallelism is a key driver of implementation noise.

Accelerator comparison We find that `IMPL` impact on



(a) Across Networks

(b) Across Kernels

Figure 6. 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 B) plugged with different size convolution kernels.

`churn` and `l2` is still high for RTX5000 Tensor Cores which employ systolic arrays similar to TPUs to accelerate computation. The high `IMPL` noise 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 is introducing a similar level of noise compared to CUDA Cores.

In Figure 5, 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 implementation noise since intra-batch shuffling will introduce differences in gradient accumulation order, even for deterministic accelerator like TPUs. In Figure 7, 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, but, 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.

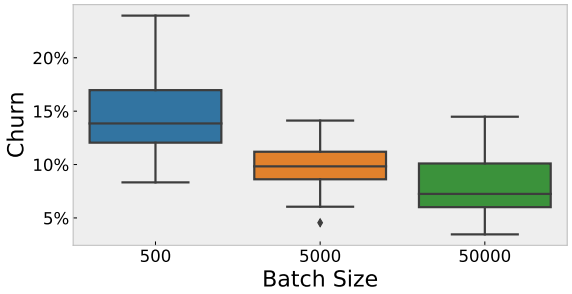


Figure 7. Data input order introduces additional non-determinism on TPU.

4 Results: The Cost of Ensuring Determinism

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). We select networks that are widely used such as MobileNet (Howard et al., 2017), EfficientNets (Tan & Le, 2020) , DenseNet-121/201 (Huang et al., 2017), VGG-16/19 (Simonyan & 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 on 100 training steps unless specified elsewhere.

How does model architecture impact overhead? Figure 6 (a) shows the relative deterministic overhead of a variety of 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 model hyperparameter To understand the rela-

Table 3. Model hyperparameter impact deterministic tooling overhead

VARY CHANNEL NUMBER			
	1X	2X	4X
RELATIVE OVERHEAD	132%	139%	154%
VARY BATCH SIZE			
	1X	2X	4X
RELATIVE OVERHEAD	132%	132%	134%
VARY INPUT IMAGE SIZE			
	0.5X	1X	2X
RELATIVE OVERHEAD	127%	132%	134%
VARY LAYER NUMBER			
	0.5X	1X	1.5X
RELATIVE OVERHEAD	127%	130%	130%

tive overhead of variation in size of model hyperparameters. we first evaluate across different kernel sizes using a six layer medium CNN (Appendix B). Assembled with convolution kernel size ranging from $1 * 1$ to $7 * 7$. As show in Figure 6 (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.

Furthermore, we report the relative overhead compared to corresponding non-deterministic settings on V100 when varying channel number, batch size, input image size, and layer number in medium CNN. When alternate one hyperparameter, the others remain unchanged [3]. We report detailed result in Table 3, similar to filter size, channel number can improve deterministic tooling overhead drastically when increased. Increase overhead from 132% to 154% on V100 when enlarge channel number in each layer four times. While other hyperparameter such as batch size, input image size, and layer number to not present obvious impact in deterministic tooling overhead. We believe tooling overhead is valuable for algorithm designers who are designing deterministic-tooling friendly neural net as well as hardware/framework designers who are working on minimizing deterministic tooling overhead.

How does hardware impact overhead? GPU architecture deterministic overhead varies considerably. In Figure 6 (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 we benchmark, GPUs with older *Pascal* architecture (P100) evidence higher overhead than GPUs with later *Volta* and *Turing* architecture. This 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 8, we plot the time spent on Top-20 kernels used 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.

[3]When vary layer number, we keep channel number constantly equal to 32 in each CNN layer to prevent channel number change with layer number.

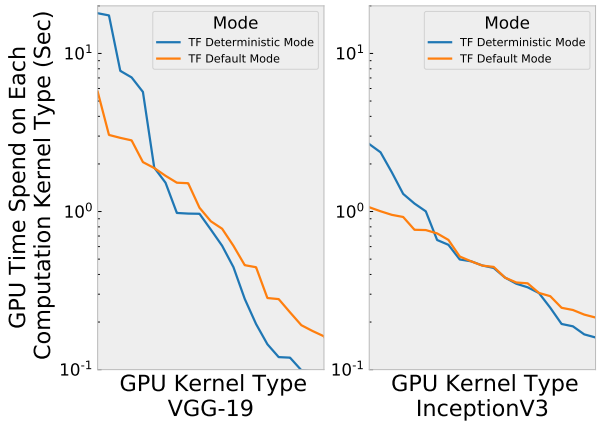


Figure 8. Top-20 GPU kernels runtime comparison (Top-1 is on the left side). X-axis indicates the different type of kernels scheduled on GPU. Y-axis indicates the time spend on each type of GPU kernel.

5 RELATED WORK

Reproducibility in machine learning As numerous works have pointed out (Goodman et al., 2016; Gundersen & 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 & Missier, 2016), to successfully executing code (Collberg & 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 & 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 & Jain, 2019) measuring how random seeds impact explanations for NLP tasks provided by interpretability methods. (Summers & 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 & Shamir, 2021; Shamir et al., 2020), hyper-parameter choices (Lucic et al., 2018; Henderson et al., 2017; Kadlec et al., 2017; Bouthillier et al., 2021), 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.

We also note that our evaluation protocols for understanding the impact of tooling on stochasticity is complementary to benchmarking efforts that evaluate the impact of hardware on other measures of performance such as energy,latency (Banbury et al., 2021; Mattson et al., 2020).

6 Conclusion

In this work, we 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.

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. In this work, we did some preliminary exploration on the role of dataset and architecture choice such as training stabilization techniques like batch normalization can impact on variation. However, we prudently regard a more thorough investigation as valuable but beyond the scope of this paper. We believe this an important future area of scholarship, as designing architectures to limit stochasticity is likely far more efficient than imposing end-to-end determinism.

REFERENCES

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

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

Badgeley, M., Zech, J., Oakden-Rayner, L., Glicksberg, B., Liu, M., Gale, W., McConnell, M., Percha, B., and Snyder, T. 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.

Banbury, C., Reddi, V. J., Torelli, P., Jeffries, N., Kiraly, C., Holleman, J., Montino, P., Kanter, D., Warden, P., Pau, D., Thakker, U., antonio torrini, jay cordaro, Guglielmo, G. D., Duarte, J., Tran, H., Tran, N., niu wenxu, and xu xuesong. MLPerf tiny benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=8RxxwAut1BI.

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

Bouthillier, X., Laurent, C., and Vincent, P. Unreproducible research is reproducible. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 725–734. PMLR, 09–15 Jun 2019.

Bouthillier, X., Delaunay, P., Bronzi, M., Trofimov, A., Nichyporuk, B., Szeto, J., Mohammadi Sepahvand, N., Raff, E., Madan, K., Voleti, V., Ebrahimi Kahou, S., Michalski, V., Arbel, T., Pal, C., Varoquaux, G., and Vincent, P. Accounting for variance in machine learning benchmarks. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems*, volume 3, pp. 747–769, 2021. URL https://proceedings.mlsys.org/paper/2021/file/cfecdb276f634854f3ef915e2e980c31-Paper.pdf.

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

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

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

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

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

Council, N. R. *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.

Cummings, R., Gupta, V., Kimpara, D., and Morgenstern, J. On the compatibility of privacy and fairness. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization, UMAP 2019, Larnaca, Cyprus, June 09-12, 2019*, pp. 309–315. ACM, 2019. doi: 10.1145/3314183.3323847.

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

Drummond, C. 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.

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

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and*

Statistics, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

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

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

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

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

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

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

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

Hong, S.-Y., Koo, M.-S., Jang, J., Kim, J.-E. E., Park, H., Joh, M.-S., Kang, J.-H., and Oh, T.-J. 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.

Hooker, S. 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.

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

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

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

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

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

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

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

Jouppi, N. P., Young, C., Patil, N., Patterson, D. A., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA*, 2017.

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

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and development team, J. Jupyter notebooks ? a publishing format for reproducible computational workflows. In Loizides, F. and Scmidt, B. (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90. IOS Press, 2016.

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

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

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

Langer, M., He, Z., Rahayu, W., and Xue, Y. 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.

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

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

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B. Scaling distributed machine learning with the parameter server. In Flinn, J. and Levy, H. (eds.), *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pp. 583–598. USENIX Association, 2014.

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

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

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

Mattson, P., Cheng, C., Diamos, G., Coleman, C., Micikevicius, P., Patterson, D., Tang, H., Wei, G.-Y., Bailis, P., Bittorf, V., Brooks, D., Chen, D., Dutta, D., Gupta, U., Hazelwood, K., Hock, A., Huang, X., Kang, D., Kanter, D., Kumar, N., Liao, J., Narayanan, D., Oguntebi, T., Pekhimenko, G., Pentecost, L., Janapa Reddi, V., Robie, T., St John, T., Wu, C.-J., Xu, L., Young, C., and Zaharia, M. Mlperf training benchmark. In Dhillon, I., Papailiopoulos, D., and Sze, V. (eds.), *Proceedings of Machine Learning and Systems*, volume 2, pp. 336–349, 2020. URL https://proceedings.mlsys.org/paper/2020/file/02522a2b2726fb0a03bb19f2d8d9524d-Paper.pdf.

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

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

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

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

Milani Fard, M., Cormier, Q., Canini, K., and Gupta, M. Launch and iterate: Reducing prediction churn. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Nagarajan, P., Warnell, G., and Stone, P. 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.

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

Neyshabur, B., Sedghi, H., and Zhang, C. What is being transferred in transfer learning?, 2021.

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

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.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. 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.

Pham, H. V., Qian, S., Wang, J., Lutellier, T., Rosenthal, J., Tan, L., Yu, Y., and Nagappan, N. 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, pp. 771–783, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367684. doi: 10.1145/3324884.3416545.

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

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

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

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

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

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

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

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

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., Sepassi, R., and Hechtman, B. A. Mesh-tensorflow: Deep learning for supercomputers. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10435–10444, 2018.

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

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

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

Snapp, R. R. and Shamir, G. I. Synthesizing irreproducibility in deep networks. *CoRR*, abs/2102.10696, 2021.

Søgaard, A., Ebert, S., Bastings, J., and Filippova, K. 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*, pp. 1823–1832, Online, April 2021. Association for Computational Linguistics.

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

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

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

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

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

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

Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1058–1066. JMLR.org, 2013.

Xie, H., Yang, D., Sun, N., Chen, Z., and Zhang, Y. 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.

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

Zhang, C., Bengio, S., and Singer, Y. Are all layers created equal?, 2019.

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

7 APPENDIX

A 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 ∼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 [4].

B 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 medium CNN, kernel size X can be 1, 3, 5, and 7.

C Tensorflow Deterministic Mode

As described in Section 2.2, when models are trained only algorithmic factors noise (ALGO), implementation noise introduced by perturbation of floating-point accumulation ordering should be eliminated wherever possible. Tensorflow (Abadi et al., 2016) can achieve this (with compromised speed) by setting TF_DETERMINISTIC_OPS and TF_CUDNN_DETERMINISTIC environment variables to true. In this way, non deterministic operations will be disabled. Furthermore, cuDNN benchmarking will be disabled as well where will try to benchmark available algorithms and find the fastest one. Pytorch (Paszke et al., 2019) also have equivalent functionalities to control implementation factors of noise by employ deterministic computation operators and disable cuDNN benchmarking. A detailed discussion of deep leaning framework support on controlling implementation factors of noise is beyond the research scope of this paper. We are here to point this out for the ease of reproduce of this paper.

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

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

D Comparison of Impact of Different Source of Noise

Table 4. 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.

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**)

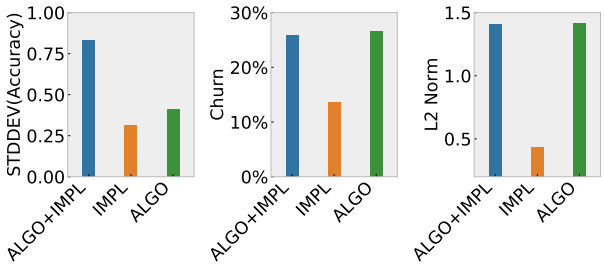
Dataset	**Training/Test Split**	**Number Classes**
Cifar-10	50000/10000	10
Cifar-100	50000/10000	100
ImageNet	1281167/50000	1000
CelebA	162770/19962	40 (Multi-label)

Table 5. Overview of each dataset benchmarked.

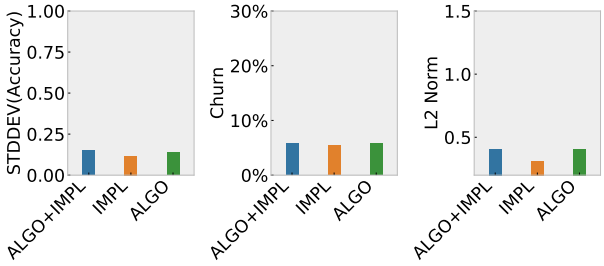
Algorithmic Sources of Randomness

Source	**Method**
Random Initialization	weights initialized by sampling random distribution (Glorot & Bengio, 2010; He et al., 2016)
Data augmentation	stochastic transformations to the input data (Kukačka et al., 2017; Hernández-García & 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 & Hinton, 2010)

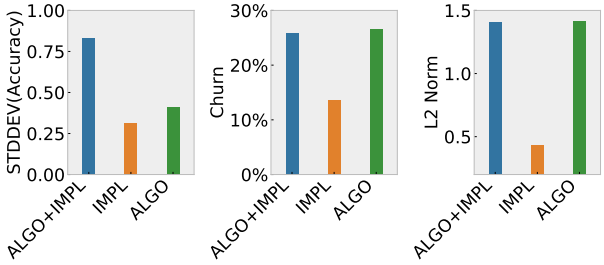
Table 6. Overview of different sources of algorithm (ALGO) noise.



(a) SmallCNN CIFAR-10

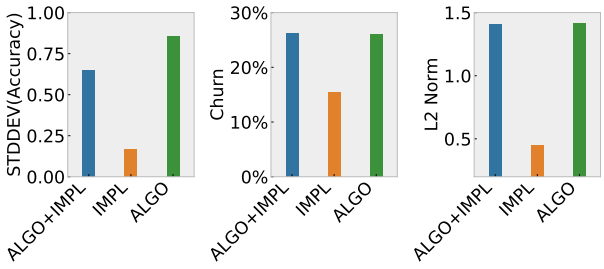


(b) ResNet18 CIFAR-10

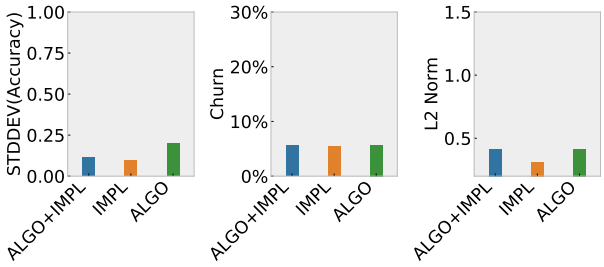


(c) ResNet18 CIFAR-100

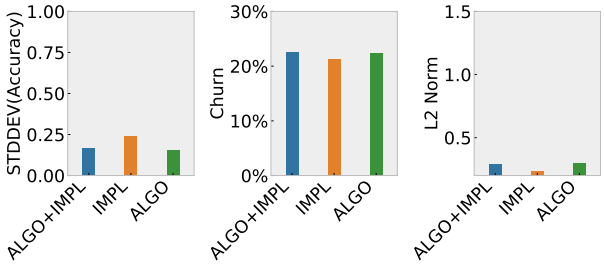
Figure 9. Comparison of impact of different source of noise across on four tasks trained on P100



(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

8 ARTIFACT APPENDIX

A Abstract

The artifact appendix helps readers to reproduce key experiments in this paper. Specifically, this appendix including all instructions needed to be reproduce Figure1, Figure2, Figure5, Figure6. For other experiments, fully automated scripts are not available yet but one can reproduce them manually using training script released on GitHub: $https://github.com/usyd-fsalab/NeuralNetworkRandomness$.

B Artifact check-list (meta-information)

- **Model:** ResNet18, ResNet50, SmallCNN, MediumCNN. All models are included in the artifact source code.
- **Data set:** CIFAR-10, CIFAR-100, CelebA, and ImageNet. All dataset takes approximately 350GB storage.
- **Run-time environment:** Ubuntu 20.04.3 LTS
- **Hardware:** Google Cloud VM with 2*NVIDIA Tesla V100 (16GB), 16 core CPU. This artifact appendix focus on validate artifact *functionality* instead of *correctness* due to computation resource constrain. All models are trained on a small sample of data. For full experiments, the recommended computation resource are listed in Section G. And remove *–fast* option in the script for full experiment (Section E).
- **Metrics:** Standard deviation on test accuracy, prediction churn, l2 distance between trained parameters, standard deviation of false positive rate, standard deviation of false negative rate, and execution time.
- **Output:** File
- **Experiments:** See below sections.
- **How much disk space required (approximately)?:** 500GB for all datasets, source code, and intermediate data during the experiments.
- **How much time is needed to prepare workflow (approximately)?:** 2 hours
- **How much time is needed to complete experiments (approximately)?:** 2 hours for functionality test. For full experiments, one week is needed with at least recommended computation resource (Section G)
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT License
- **Workflow framework used?:** Tensorflow 2.4.1, tensorflow-datasets, tensorflow-addons
- **Archived (provide DOI)?: https://zenodo.org/record/6078161**

C Description

C.1 How delivered

The artifact can be accessed via either Github:
$https://github.com/usyd-fsalab/NeuralNetworkRandomness$
or Zenodo:
$https://zenodo.org/record/6078161$.

C.2 Hardware dependencies

This experiment depends on two NVIDIA V100 GPUs, 500GB free disk space for functionality evaluation in artifact evaluation process. For full experiments, the recomended computation resource are listed in Section G.

C.3 Software dependencies

Tensorflow 2.4.1 and tensorflow_datasets. All software dependencies can be installed with single line of command:

```
pip install -r requirements.txt
```

C.4 Data sets

CIFAR-10, CIFAR-100, CelebA, and ImageNet datasets are used in experiments.

C.5 Prepare CIFAR-10 dataset

Let tensorflow datasets manage data cache. Execute below command to download and preprocess CIFAR-10 dataset

```
python -c "import tensorflow_datasets as tfds;
tfds.load('cifar10')"
```

C.6 Prepare CIFAR-100 dataset

```
python -c "import tensorflow_datasets as tfds;
tfds.load('cifar100')"
```

C.7 Prepare CelebA dataset

```
python -c "import tensorflow_datasets as tfds;
tfds.load('celeb_a')"
```

C.8 Prepare ImageNet dataset

Prepared ImageNet data following instructions in tensorflow datasets offcial website [5].

D Installation

Download artifact from either GitHub or Zenodo. When cloning from github with git, remember use $--recursive$ option:

```
git clone --recursive
https://github.com/usyd-fsalab/NeuralNetworkRandomness.git
```

Then, install package dependencies with:

```
cd NeuralNetworkRandomness &&
pip install -r requirements.txt
```

E Experiment workflow

All experimental can be launched in a single python file execution, suppose two gpu (0, 1) are used in this expermeits and imagenet data located in /mnt/data/:

```
python run_ae.py --gpus 0 1
--experiments Figure1 Figure2 Figure5 Figure6
--imagenet_data_dir /mnt/data/ --fast
```

The $--fast$ option here is indicating run experiments on small sample of data to expedite experiment process. To run experiment on full dataset, remove $fast$ option and consider use computation resource not less than specified in Section G, to get experiments results with in a reasonable time (approximate one week).

[5] https://www.tensorflow.org/datasets/catalog/imagenet2012

F Evaluation and expected result

The result can be found in ./results folder. Whereas file name of each csv file indicate which figure it corresponding to.

G Computation Resource for Full Experiments

At least below computation resources are recommended if you need to reproduce *both* functionality and correctness of experiments in this paper.

- NVIDIA Tesla P100 * 4
- NVIDIA RTX5000 * 4
- NVIDIA Tesla V100 * 20
- NVIDIA Tesla T4 * 1
- Google TPUv2 * 4