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 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-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. We believe this work present a deeper understanding to the community of the impact of tooling noise, as well as raise the attention to the trade-off between determinism and training overhead. We release the source code used in experiments[1].

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 [52, 46, 77, 74, 69, 45, 22]. Less well explored or understood is how our choice of tooling

[1]https://github.com/usyd-fsalab/NeuralNetworkRandomness

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 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 and autonomous driving. 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 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 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 \[31\] 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% on a spectrum of widely used GPU accelerator architectures, relative to non-deterministic training (Section 4).

2 **Methodology**

We consider a supervised learning setting,

\[
D\{ (x_1, y_1), \ldots, (x_N, y_N) \} \subset \mathcal{X} \times \mathcal{Y}
\]

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** \((ALGO)\) - noise that ultimately comes from pseudo-random number generator. This includes model design choices which are stochastic by design. For example, random initialization \([17, 21]\), data augmentation \([38, 23]\), data shuffling ordering \([73]\), and stochastic layers \([76, 24, 81, 53, 49]\). In 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 that ultimately comes from floating-point number accumulation ordering error.\(^2\) This includes noise introduced by software choices (e.g. Tensorflow \([2]\), PyTorch \([60]\), cuDNN \([9]\) as well as hardware accelerators’ architectures (e.g., modern GPU hardware designs \([56, 57, 58]\)). 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 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 \([10]\).

- **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.

\(^2\)Non-associativity of floating-point arithmetic, e.g., \((a+b)+c \neq a+(b+c)\).
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 a consistent visualization scale across a variety of experiments.

**Churn** (\(\text{churn}\)) - 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 [12]. Thus, understanding the factors that amplify churn is of considerable research interest with several different proposed definitions of predictive churn [8, 68, 74]. We define churn between two models under the same experimental framework and tooling, the variation of the training outcome for as done by [51] as the fraction of test examples where the predictions of two models disagree.:

\[
C(f_1, f_2) = E_x [\mathbb{I}_{\{\hat{y}_{x,f_1} \neq \hat{y}_{x,f_2}\}}]
\]

(2)

where \(\mathbb{I}\) is an indicator function for whether the predictions by each model match.

**L2 norm** (\(\|w_1 - w_2\|\)) - L2 norm of the trained weights 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** (\(\text{stddev}\)) - 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 all measures over 10 independent runs unless indicated otherwise.
Figure 3: 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.

2.2 Experimental Setup

We conduct extensive experiments across large-scale datasets (CIFAR-10 and CIFAR-100 [37], ImageNet [64] and CelebA [44]) and widely-used networks including ResNet-18 and ResNet-50 [21], DenseNet-121 and DenseNet-201 [30], Inception-v3 [78], MobileNet [65], EfficientNet [79], three-layer small CNN and six-layer medium CNN (Appendix F). 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 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 C. We preserve the same hyperparameter choices across hardware types and use Tensorflow [2] 2.4.1, CUDA 11, and cuDNN [9] for all the experiments.

**GPU** - we evaluate NVIDIA P100 with an older Pascal architecture [56] and later generations V100 [57], RTX5000 and T4 [58] 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 [34] is a custom ASIC leverage systolic arrays [39] 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 F elaborates technical details to achieve this.

**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.
Figure 4: 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 2: Data points distribution in CelebA dataset

<table>
<thead>
<tr>
<th></th>
<th>Male</th>
<th>Female</th>
<th>Young</th>
<th>Old</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive Data Points</td>
<td>1387 (0.8%)</td>
<td>22880 (14.1%)</td>
<td>20230 (12.4%)</td>
<td>4037 (2.5%)</td>
</tr>
<tr>
<td>Negative Data Points</td>
<td>66874 (41.1%)</td>
<td>71629 (44.0%)</td>
<td>106558 (65.5%)</td>
<td>31945 (19.6%)</td>
</tr>
</tbody>
</table>

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 std dev 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 std dev, churn and l2 for all sources of noise. The small CNN is the only architecture we benchmark without batch normalization (BN) [31], a standard technique for stabilizing training [1]. 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 the std dev 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 [37] and CelebFaces Attributes (CelebA) dataset [44]. 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 compared to standard deviation of top-1 accuracy. Interestingly, even for uniformly distributed dataset, per-class accuracy variance still have a large range of divergence.

CelebA [44] 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 suffering 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 3 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 chern 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** We find that IMPL impact on chern 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 3 for ALGO+IMPL TPUs incurs a lower level of chern 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 obverse that while TPU lower chern 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 chern and l2, but does not have an observable relationship with stdev 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 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, 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.

### 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 [55]. We select networks that are widely used such as MobileNet [29], EfficientNets [79], DenseNet-121/201 [50], VGG-16/19 [72] and ResNet-50/152 [21]. We profile all models on ImageNet dataset with input shape 224*224 and batch size of 64 on 100 training steps. Note that the ratio of the overhead remain unchanged no matter how many epochs the model trained for.
How does model architecture impact overhead? Figure 8 (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% × 211% and 101% ∼ 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 D). 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% ∼ 746% on P100, 129% ∼ 241% on V100, and 117% ∼ 196% on T4 respectively. For all kernel size we evaluate on each GPU, larger kernel size is always comes with larger overhead.

How does hardware impact overhead? GPU architecture deterministic overhead varies considerably. 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 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 7, 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.

5 Related Work

Impact of Algorithmic Factors Several works have evaluated the impact on randomness introduced by algorithm design choices. 52 evaluate the role of random initialization in reinforcement learning. 46 measure how random seeds impact explanations for NLP tasks provided by interpretability methods. 77 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 241, 117, hyper-parameter choices 45, 22, 35, the use of data parallelism 67 and test set construction 75, 41, 48.

Impact of Software Dependencies 25 evaluate the role of different compilers for the specialized task of weather simulation. Recent work by 61 and 3 in the machine learning domain evaluates the impact of randomness introduced by popular deep neural network libraries (Pytorch, CNTK, Theano and Tensorflow). 3 evaluates a segmentation task for mouse neo-cortex data and MNIST on LeNet 42. 61 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.

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 70, 40, partition over optimizer state 63, and asynchronous gradients update 43. An important area of future work involves understanding how distributed training impacts model stability.
References


[55] NVIDIA. Profiler user’s guide.

[56] NVIDIA. Nvidia tesla p100, 2016.

[57] NVIDIA. Nvidia tesla v100 gpu architecture, 2017.


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments (e.g., for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]