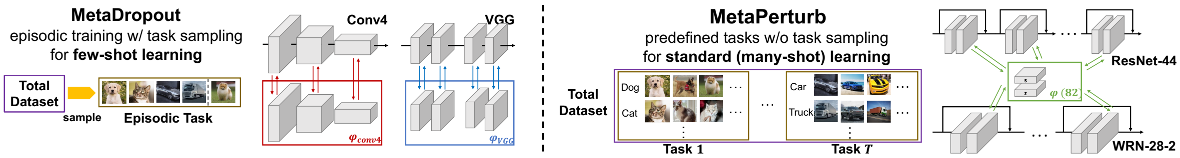

We thank all the reviewers for constructive comments. Reviewers appreciate that our paper is well-written, clear, and is tackling the important problem of scaling meta-learning, by proposing a novel distributed framework.

[Common Comments] Comparison with MetaDropout [R2, R4] Our MetaPerturb is **not** incremental over MetaDropout [18]. **1) MetaDropout cannot generalize across heterogeneous neural architectures**, since it learns an individual noise generator for each layer (Figure 2 of [18]). Thus it is tied to the specific base network architecture (Top Figure), while MetaPerturb can generalize across architectures since it is a size- and order-invariant set function shared across all layers (L74-75). **2) MetaDropout does not scale to large networks** since the noise generator should be the same size as the main network. MetaPerturb, on the other hand, requires marginal memory overhead (82 parameters) even for deep CNNs (e.g. ResNet-44, L190-192) since it shares the same lightweight noise generator across all layers and channels. MetaDropout also becomes almost infeasible to train with large networks due to the needs of computing the second-order derivatives. **3) MetaDropout cannot scale to standard learning** (Top Figure), since it uses episodic training and MAML for meta-learning. For standard learning with a large number of instances, taking a few gradient steps with few sampled instances is highly insufficient for minimizing loss on all instances, and taking large number of gradient steps over large number of episodes is infeasible. (L115-116) We overcome such a challenge by proposing a **scalable meta-learning** framework which splits the given dataset into multiple subsets (tasks) without task sampling, and jointly training the shared set-function across all tasks (L76-77) without lookahead gradient steps.

Improvement on fine-grained datasets [R1, R2] As mentioned in L252-254, we attribute the improvements to **z** and **s**, which help focus on the more relevant part of each input, that is crucial for discrimination between two very similar classes. **Missing references** [R3, R4] We will cite them and include the following discussions: FiLM uses instance-wise modulation whereas our s network is a batch-wise set function. MetaMixup meta-learns the hyperparameter of Mixup and MetaReg proposes to meta-learn the regularization parameter (ℓ_1 for domain generalization), but they consider generalization within a single task or across similar domains, while ours target heterogeneous domains.

[R4] Contributions seems limited. Please see the comparison against MetaDropout in the general comments. Also, each component is largely different from the models mentioned: **1) vs. BN:** While BN learns the scaling terms as free variables, s network outputs the scaling factor for each channel as a function of the batch. **2) vs. Deep Sets.** The DeepSets paper does not deal with channel-wise permutation equivariance for Conv layers, which we newly developed. **Analysis on approximation error.** We meta-trained MetaPerturb with Ren et al. [30] with a single lookahead step and meta-test on STL10 for empirical analysis. The Table on the right shows that Ren et al. [30] increases the training time by 6× with marginal increase in accuracy. **Why not consider other techniques?** Although there exist diverse approaches to improve generalization, we compared against the most relevant works (stochastic perturbation) since all other techniques are orthogonal to ours and thus can be used together.

Method	Train time	Accuracy
MetaPerturb	$\sim$ 1 hr	69.79$\pm$0.60
MetaPerturb w/ Ren et al	$\sim$ 6 hrs	69.88$\pm$0.50

Jutification of the parameter usage control for each dataset. Figure 6 shows that the distribution of s is different across the datasets, and the ablation study (Table 3) shows the necessity of the s network. **What if it is not CNNs?** For MLP, perturbation function can be implemented by replacing convolution with linear operations. For RNNs and Transformers, we leave it as future work. **Missing configurations of hyperparameters.** Please see Section C.4. of the supplementary file. **Definition of the optimal amount of perturbation.** We will tone down *optimal* to *proper*.

[R2] TinyImageNet may contain image classes for fine-grained datasets (e.g. aircraft). TIN contains low-resolution (32×32) images with general classes (e.g. airplane, bird), while Aircraft and CUB datasets contain high-resolution images (84×84) and contain fine-grained classes. Thus, we believe that the two datasets are sufficiently different. **Performance of finetuning.** In Table 1, finetuning significantly outperforms learning from scratch in all cases. Yet, for experiments with SVHN which contains digits and which is largely different from classes in TIN (Table 2), the performance gain become smaller. MetaPerturb obtains large performance gains on both cases, which shows that the knowledge of perturbing a sample is more generic and thus is applicable to diverse domains.

[R1] Perturb function at the top and bottom layers. We performed the suggested experiments, and it performs better than perturbing only the top or the bottom layer, but is worse than the full model. **Split of B^{tr} and B^{te}?** They both come only from the training split of the original dataset (no fairness issue). **Heterogeneous tasks for meta-test?** At meta-test time, we fix the transferred perturbation parameters and only train the main model parameters with a single target task.

Location of perturb	s-CIFAR100	Aircraft	CUB
Top layers	32.54$\pm$0.19	53.42$\pm$0.79	27.70$\pm$0.68
Bottom layers	31.75$\pm$0.97	61.93$\pm$0.86	31.40$\pm$0.24
Top&Bottom layers	33.63$\pm$0.48	61.65$\pm$1.65	32.57$\pm$0.30
MetaPerturb	**34.47**$\pm$**0.45**	**66.12**$\pm$**0.70**	**39.94**$\pm$**1.30**

[R3] Weight visualization of s network. We also visualize the weights for the 3x3 Conv filters and FC layer weight on the right. It shows that the s network outputs larger scales for feature maps with more channels and larger spatial size. **is the gradient of ϕ shared?** Yes, and ϕ is updated *synchronously* at every iteration thanks to its small dimensionality ($d = 82$). **Controversial flatter loss surface.** We agree and will tone down the claims.