A Experimental Details

A.1 Training Details

Model Training: Following TENT [Wang et al., 2020a], we use ResNet26 models [He et al., 2016a] for all experiments besides the ImageNet experiments. For ImageNet, we use ResNet50v2 [He et al., 2016b]. All methods utilizing ensembles use 10 different models in the ensemble.

For all ResNet26 experiments, we train all models and obtain posteriors using SWAG-D [Maddox et al., 2019] and use the same hyperparameters as provided by the authors for training a WideResNet28x10. For methods that do not include the posterior density term, we simply use the mean of the learned SWAG-D posterior, which simply corresponds to the solution obtained by Stochastic Weight Averaging (SWA) [Izmailov et al., 2018]. We train for 300 epochs on each training dataset, starting to collect iterates for SWA/SWAG starting from epoch 160. We use SGD with momentum for all experiments, with a learning rate schedule given by 0.1 for the first 150 epochs, decaying linearly down to 0.01 until epoch 270, then remaining at 0.01 for the remaining 30 epochs.

For ResNet50v2 experiments on ImageNet, we first train for 90 epochs using the example training script at `https://github.com/deepmind/dm-haiku/tree/master/examples/imagenet`. We then train for an additional 10 epochs with a constant learning rate of 0.0001 to collect iterates for SWAG, collecting 4 iterates per epoch.

Adaptation Details: For test time adaptation on the small scale experiments with ResNet26, we use the same hyperparameters described in TENT [Wang et al., 2020a]. using learning rate 0.001 and batch size 128, using SGD with momentum. For the ImageNet experiments with ResNet50v2, we tune the learning rate for accuracy using the validation corruptions in ImageNet-C, and keep the batch size fixed at 64. For TENT, we found a learning rate 0.001 to perform the best, while for our method BACS, we found a smaller learning rate of 0.0001 to perform the best.

During adaptation with BACS, according to equation 8, we would optimize each model with the objective

$$\max_{\theta} \log q_i(\theta) - \frac{\tilde{\alpha}}{m} \sum_{j=1}^{m} -H(Y|\tilde{\mathbf{x}}_j, \theta). \tag{9}$$

where $q_i(\theta)$ is a diagonal Gaussian posterior density obtained by SWAG-D, and $\tilde{\alpha}$ is a hyperparameter controlling how much to weight the entropy minimization objective against the posterior term. We initialize from the mean of the SWAG posterior before adaptation. In practice, we also rewrite the objective as

$$\max_{\theta} \beta \log q_i(\theta) - \frac{1}{m} \sum_{j=1}^{m} -H(Y|\tilde{\mathbf{x}}_j, \theta), \tag{10}$$

where $\beta = \frac{1}{\tilde{\alpha}}$, and use minibatches of test inputs for the entropy minimization term. For all small-experiments with ResNet26, we use $\beta = 0.0001$. For the ImageNet experiments, we jointly tuned β with the learning rate using the extra validation corruptions and selected $\beta = 0.0003$.

Compute Resources: The initial training for each ResNet50 model on ImageNet used a Google Cloud TPU v3 instance with 8 cores, each taking approximately 20 hours to train. In total, as we needed to train 10 models for the ensemble, this utilized 200 hours of compute with TPU v3 instances with 8 cores each.

All other model training and all the adaptation were run using local GPU servers with Nvidia Titan RTX GPUs. We estimate model training time to be around 200 hours in total for all ResNet26 models (10 models for each of CIFAR10, CIFAR100, and SVHN and an estimated 6 hour training time with on one GPU). For adaptation and evaluation, we estimate the total time needed for the ImageNet-Corrupted experiments, as these would take up the bulk of the time spent on adaptation. For each of the 15 corruption types and 5 corruption levels, we estimate the total time it takes to adapt each model using entropy minimization for one epoch, evaluate the model without any adaptation, evaluate the model with adapted batchnorm statistics, and evaluate each model after entropy minimization to be 10 minutes. With 10 models for each corruption, this totals 125 hours of GPU time to compile the ImageNet Corrupted results, which take up the bulk of the time needed for adaptation and evaluation in our experiments.

	CIFAR10-C				CIFAR100-C			
Method	Acc	NLL	Brier	ECE	Acc	NLL	Brier	ECE
TENT	84.52	0.7248	0.2289	0.09914	64.37	3.265	0.6111	0.2759
BACS (ours)	87.43	0.4242	0.1845	0.02882	67.28	1.323	0.4435	0.0500

Table 4: **CIFAR-10/100 Corrupted Online** results at the highest level of corruption, averaged over all corruption types at severity level 5 (the most severe level). While online adaptation performs slightly worse than offline adaptation, our method BACS still provides substantial improvements over TENT, ensembles with BN adaptation and other baselines.

A.2 Metrics

We compute Brier score for a probability vector $p(y|x_n)$ and true label y_n as

$$\sum_{y \in \mathcal{Y}} (p(y|x_n) - \delta(y - y_n))^2. \tag{11}$$

To compute expected calibration error (ECE), we use 20 bins. We order all predictions by confidence and aggregate them into 20 bins, each with the same number of data points, instead of using fixed windows for the buckets. Given the bins B_i, we then compute ECE as

$$\text{ECE} = \frac{1}{20} \sum_{i=1}^{20} |\text{acc}(B_i) - \text{conf}(B_i)|, \tag{12}$$

where acc and conf compute the average accuracies and confidences within the predictions in each bucket.

A.3 Dataset details

Models were trained on CIFAR10/100 [Krizhevsky, 2012], ImageNet [Deng et al., 2009], CIFAR10/100-Corrupted and Imagenet-Corrupted [Hendrycks and Dietterich, 2019], STL10 [Coates et al., 2011], SVHN [Netzer et al., 2011], MNIST [LeCun et al., 2010].

For the corrupted datasets, we report results averaged over all 15 standard corruption types in [Hendrycks and Dietterich, 2019], using the 4 extra corruption types for validation.

CIFAR10-Corrupted, STL10, SVHN, and MNIST were downloaded using Tensorflow Datasets. The Imagenet training data was directly downloaded from `www.image-net.org`), while CIFAR-100-Corrupted and Imagenet-Corrupted were downloaded from the author-released images at `zenodo.org/record/3555552` and `https://zenodo.org/record/2235448` respectively.

For Imagenet-C, we initially ran experiments and performed hyperparameter tuning using the datasets generated through Tensorflow Datasets, which recomputed the the corrupted images instead of simply downloading the images released by the original authors. However, performance on the TFDS version of ImageNet-C (see appendix B.3) is not directly comparable to that using the official released dataset, with overall results being substantially stronger with the TFDS version. This discrepancy is also noted in Ford et al. [2019], who postulated the performance differences are due to the extra JPEG compression used for the officially released dataset making the tasks more difficult.

We were unable to find licensing information for CIFAR10, CIFAR100 or STL10. Imagenet is released under a custom license stipulating non-commercial research and educational use only (see `www.image-net.org/download`). SVHN is released with a note stating it should be used for non-commercial uses only. MNIST is released under the CC BY-SA 3.0 license. CIFAR10/100-Corrupted and Imagenet-Corrupted are released under the CC BY 4.0 License.

B Additional Experimental Results

B.1 Online Evaluation

For the results in the main paper, all methods that adapted to the test distribution were evaluated in an offline setting, where each method could access the full test dataset before making any predictions.

	CIFAR10				CIFAR100			
Method	Acc	NLL	Brier	ECE	Acc	NLL	Brier	ECE
Vanilla	95.50	0.1715	0.07252	0.02549	77.88	1.023	0.3389	0.1198
BN Adapt	95.49	0.1767	0.07303	0.02588	77.88	1.071	0.3437	0.1266
TENT	95.48	0.1788	0.07662	0.03180	77.86	1.083	0.3458	0.1322
BACS (MAP)	95.40	0.1827	0.07734	0.02884	77.81	1.1388	0.3494	0.1412
Vanilla ensemble	96.07	**0.122**	**0.05835**	0.009644	80.34	**0.7182**	**0.2711**	**0.04254**
Ensemble BN Adapt	**96.09**	0.1244	0.05834	**0.009433**	80.46	0.7312	0.2720	0.04438
TENT Ensemble	96.08	0.1240	0.05869	0.01065	**80.68**	0.7341	0.2719	0.05352
BACS (ours)	95.98	0.1340	0.06154	0.01091	80.32	0.7428	0.2754	0.05051

Table 6: **CIFAR-10/100 In-distribution results.** We evaluate all methods on the standard (uncorrupted) test sets.

For BACS, this involves first making one pass through the test data to update the network for one epoch of optimization (or multiple epochs when specified), then making one more pass through the dataset to make predictions with the fully adapted network.

As there might be scenarios where we do not have access to all the test data from a particular distribution at once, we also include experiments on the corrupted datasets evaluating methods in an online setting, where we adapt the network and make predictions in a single pass through the dataset. The online procedure is more computationally efficient (requiring one fewer forward pass per batch) and does not require the model to wait for all the data to arrive before making predictions. Note that all adaptive methods we evaluate still require access to batches of test data to make updates, and we utilize the same adaptation hyperparameters as with the offline experiments. For online experiments, we fix the ordering on the test dataset for all methods.

We show results in the online setting for TENT and BACS in Tables 4 and 5. Compared to the offline results in Tables 1 and 2, online BACS generally performs worse than offline BACS, but still outperforms all baselines in accuracy, NLL, and Brier score.

Method	Acc	NLL	Brier	ECE
TENT	54.12	2.330	0.6005	0.09544
BACS (MAP)	56.13	2.171	0.5771	0.07055
TENT Ensemble	60.22	2.093	0.5702	0.1828
BACS (ours)	61.81	1.911	0.5369	0.1503

Table 5: **ImageNet-C Online** results averaged over all corruption types and levels. Again, online adaptation performs worse compared to offline adaptation, but online BACS still outperforms one TENT (as well as other baseliens) in accuracy, NLL, and Brier score.

B.2 In Distribution Results

At test time, it is also possible that the distribution we encounter is actually the same as training, though we would not necessarily know a priori. We thus also evaluate the performance of different adaptive methods when there is no distribution shift at test time in Table 6. We see that BACS does slightly underperform relative to ensembles without adaptation and ensembles with batch-norm adaptation.

B.3 Expanded ImageNet-C and CIFAR-C Results

In addition to measuring the average accuracy across the ImageNet-C corruptions, we also report results using the *mean corruption error* (mCE) [Hendrycks and Dietterich, 2019], which normalizes the per-corruption error rates using the performance of an AlexNet model as a baseline before averaging across corruption types. We include these results in Table 7.

We include expanded experimental results for ImageNet-Corrupted in Figure 7. For each metric, we now use box plots to show the variability of results over different corruption types, separating out results at each level of corruption.

Method	mCE
Vanilla	73.42
BN Adapt	61.24
TENT	54.37
BACS (MAP)	53.13
Vanilla Ensemble	67.98
Ensemble BN Adapt	52.70
TENT Ensemble	47.50
BACS (ours)	**46.78**

Table 7: **ImageNet-C mCE** results. BACS (ours) outperforms all other methods, while our ablation without ensembles BACS (MAP) outperforms all non-ensembled methods.

Across all corruption levels, BACS consistently performs the best in accuracy, NLL and Brier score (with the exception being the mean accuracy at the highest level of corruption, where a single corruption where BACS has very low accuracy drags down the mean, though median performance is still higher than all baselines).

finally, we include expanded experimental results for CIFAR10 and CIFAR100 Corrupted in Figures 5 and 6.

We also include experimental results using the TFDS version of ImageNet-C in Table 8, which we note has substantially better results overall than the officially released dataset. We also included boxplots for the Tensorflow Datasets version of ImageNet-Corrupted in Figure 8, where we see BACS performs the best in accuracy, NLL, and Brier scores at each level of corruption.

Method	Acc	NLL	Brier	ECE
Vanilla	45.64	2.867	0.6750	**0.0614**
BN Adapt	55.97	2.363	0.6050	0.1606
TENT	60.82	1.803	0.522	0.0313
BACS (MAP)	61.96	1.712	0.5022	0.0294
Vanilla Ensemble	50.48	2.519	0.6211	0.07878
Ensemble BN Adapt	62.18	2.137	0.5802	0.2438
TENT Ensemble	65.83	1.586	0.4726	0.0956
BACS (ours)	**66.64**	**1.492**	**0.4548**	0.0735

Table 8: **ImageNet-C (TFDS)** results averaged over all corruption types and levels. BACS again substantially outperforms all baselines in accuracy, NLL, and Brier score.

B.4 Domain Adaptation Results

We further evaluate BACS in additional small-scale experiments commonly evaluated for domain adaptation. We evaluate transferring from CIFAR10 to STL10 (using only the 9 overlapping classes), as well as a commonly used digit recognition task transferring from SVHN to MNIST.

CIFAR10 to STL10. We further evaluate all methods on a more natural distribution shift by considering evaluating a model trained CIFAR-10 and evaluated on STL-10 [Coates et al., 2011]. We only use the 9 overlapping classes in each dataset. In this setting, BACS and BN ensembles outperform the other methods.

Method	Acc	NLL	Brier	ECE
Vanilla	82.38	0.7825	0.2886	0.1185
BN adapt	83.72	0.7926	0.2709	0.1147
TENT	84.05	0.8831	0.2753	0.1216
Vanilla Ensemble	84.03	0.5360	0.2333	0.06561
Ensemble BN Adapt	85.40	0.5301	0.2195	**0.05949**
TENT Ensemble	85.10	0.5337	0.2270	0.06844
BACS (ours)	**85.47**	**0.5284**	**0.2184**	0.06114

Table 9: **CIFAR10 to STL10**: Source model trained on CIFAR10 and evaluated on STL10 test set the (with nonoverlapping classes removed during both training and test). Here, the ensembled approaches perform the best overall and perform similarly to one another in uncertainty estimation.

While TENT is able to improve accuracy slightly over the non-ensembled baselines, we again see that uncertainty estimates degrade as entropy is minimized (as seen by the increases in NLL, Brier, and ECE scores). In contrast, BACS still remains well-calibrated, again emphasizing the importance of Bayesian marginalization for reliable uncertainty estimation when adapting models via entropy minimization.

SVHN to MNIST transfer. We also evaluate our method on a digit classification task, transferring a model trained on SVHN [Netzer et al., 2011] to MNIST [LeCun et al., 2010], which is commonly studied as a the domain adaptation setting [Ganin et al., 2015, Liang et al., 2020]. We find that BACS is able to significantly outperform the models with no adaptation as well as ensembles with batch norm adaptation in accuracy, NLL, and Brier score.

We find that this severe distribution shift requires larger changes in parameters to effectively adapt, as seen by the substantial improvements between optimizing for one epoch and optimizing for 10. We also find that the approximate posterior term used in our method overly constrains the network during adaptation, as removing the regularizer from the method (denoted BACS-posterior in the table) results in the best performance in all metrics. We note that all ensemble methods perform much better than non-ensembled methods in terms of calibration, emphasizing the benefits of marginalizing over different models for uncertainty estimation when adapting via entropy minimization.

We note that our method and the compared baselines are focused on improving results in robustness settings, and do not necessarily obtain state-of-the-art performance in common domain adaptation

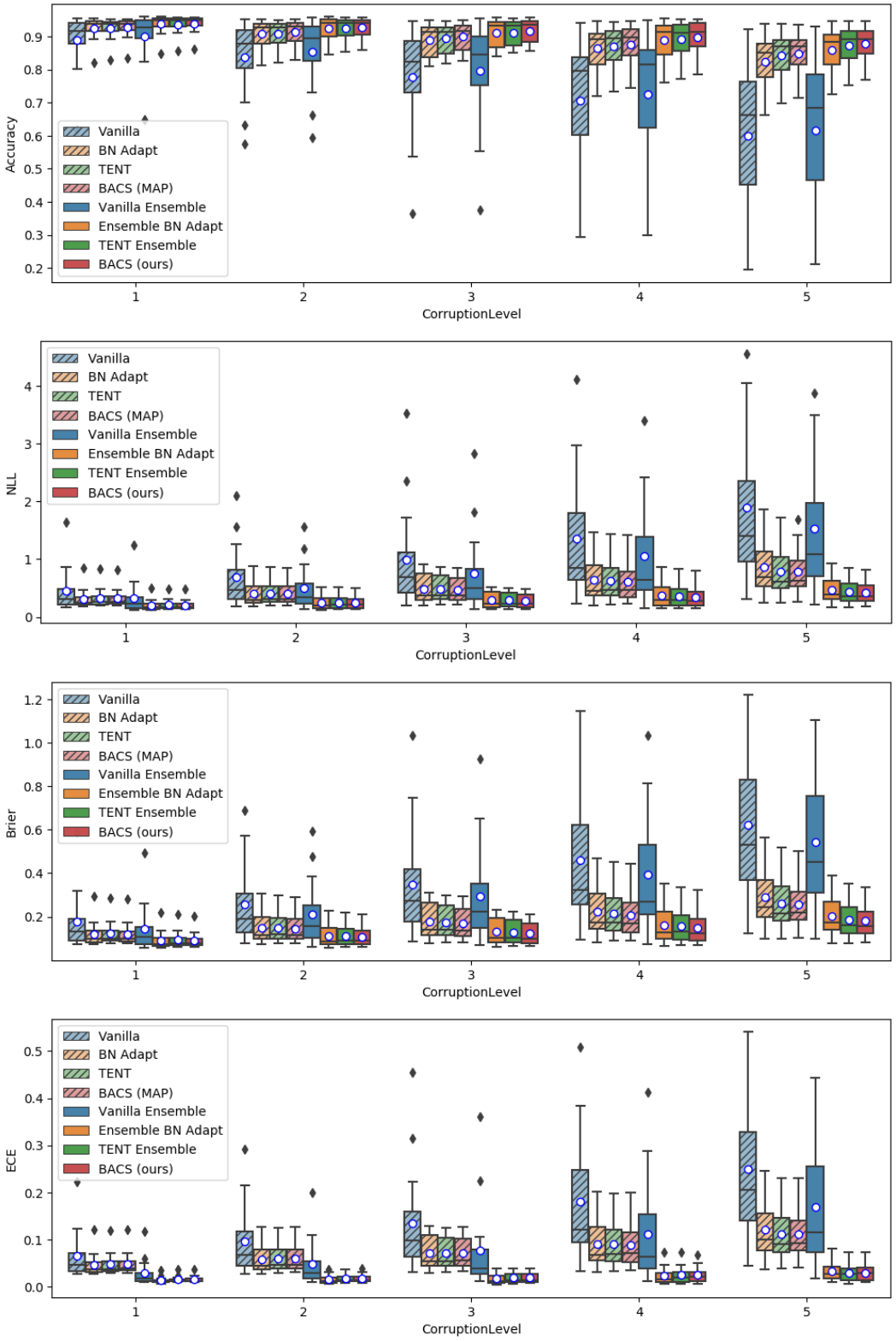

Figure 5: **CIFAR10 Corrupted Results**. For each corruption level, we use boxplots the spread of results over the different individual corruption types. Boxes are drawn at the 25th and 75th percentiles, with the median being drawn as a line in the middle of the box and the mean being shown with white dots. The ends of the whiskers show the min and max across corruption types at the level (with black diamonds for outliers).

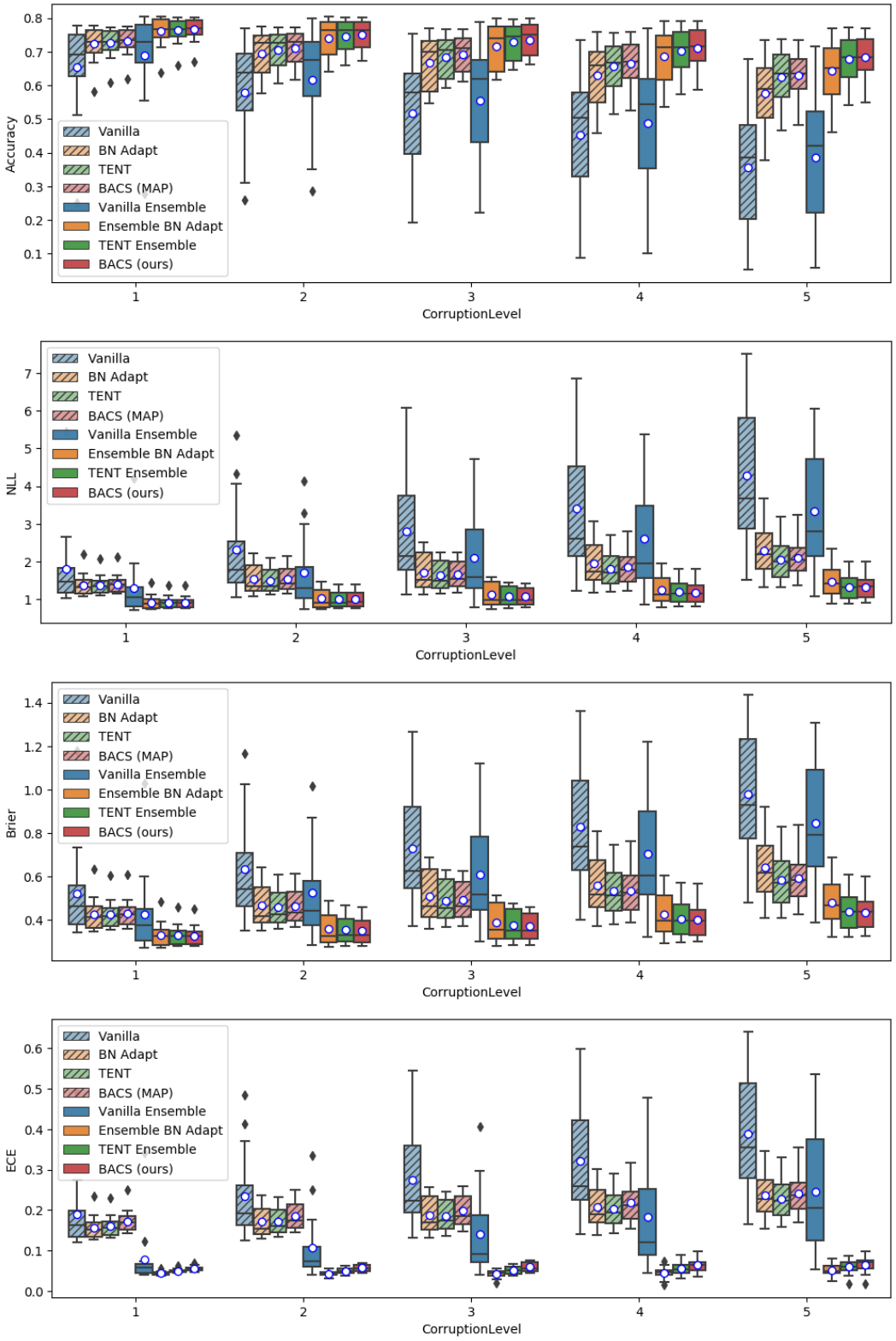

Figure 6: **CIFAR100 Corrupted Results**. For each corruption level, we use boxplots the spread of results over the different individual corruption types. Boxes are drawn at the 25th and 75th percentiles, with the median being drawn as a line in the middle of the box and the mean being shown with white dots. The ends of the whiskers show the min and max across corruption types at the level (with black diamonds for outliers).

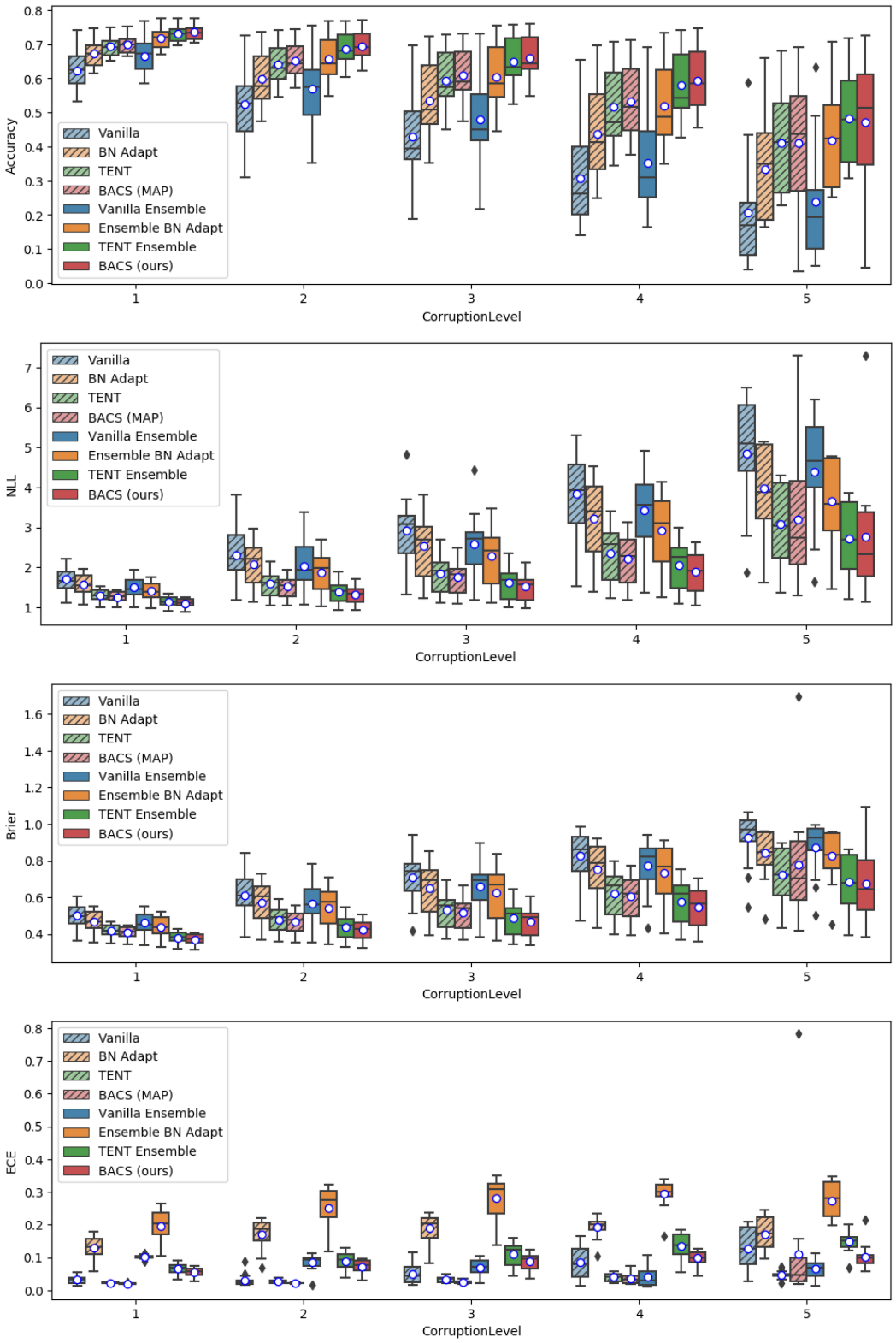

Figure 7: **ImageNet Corrupted Results**. For each corruption level, we use boxplots the spread of results over the different individual corruption types. Boxes are drawn at the 25th and 75th percentiles, with the median being drawn as a line in the middle of the box and the mean being shown with white dots. The ends of the whiskers show the min and max across corruption types at the level (with black diamonds for outliers).

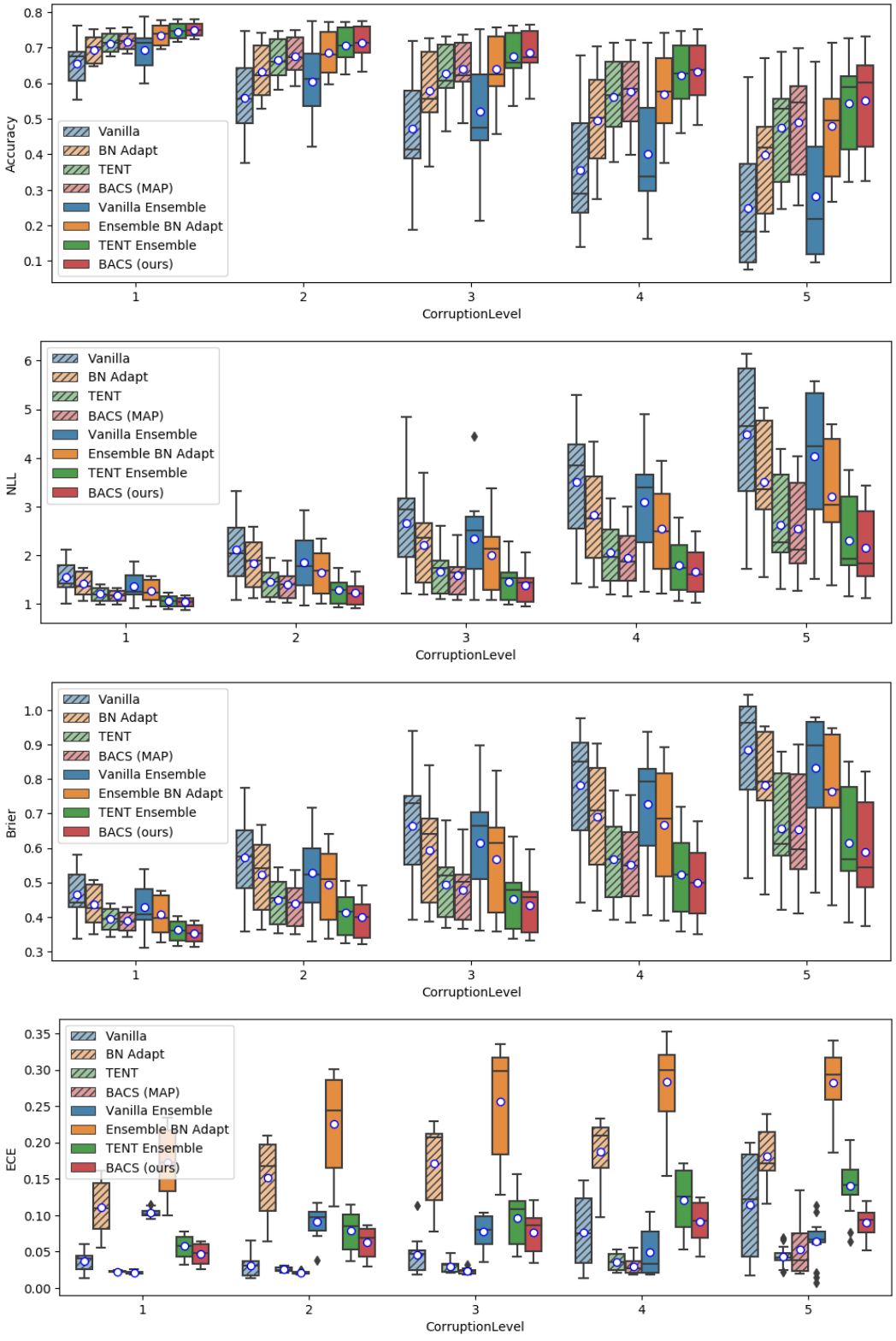

Figure 8: **ImageNet Corrupted (TFDS) Results**. For each corruption level, we use boxplots the spread of results over the different individual corruption types. Boxes are drawn at the 25th and 75th percentiles, with the median being drawn as a line in the middle of the box and the mean being shown with white dots. The ends of the whiskers show the min and max across corruption types at the level (with black diamonds for outliers). At each corruption level, BACS (ours) outperforms all baselines in accuracy, NLL, and Brier score.

Method	Acc	NLL	Brier	ECE
Vanilla	76.99	0.9967	0.3645	0.1290
Vanilla ensemble	79.52	0.7368	0.3024	0.03856
BN Adapt	73.43	1.4502	0.4453	0.1878
Ensemble BN Adapt	76.84	0.9444	0.3442	0.07855
TENT (1 epoch)	77.24	1.321	0.3896	0.1605
TENT (10 epochs)	85.53	0.9554	0.2035	0.1166
TENT Ensemble (1 epoch)	79.48	0.8869	0.3149	0.09254
TENT Ensemble (10 epochs)	86.89	0.5784	0.2035	0.06384
BACS (ours) (1 epoch)	84.14	0.6024	0.2285	0.05595
BACS (ours) (10 epochs)	86.32	0.5553	0.2094	0.05133
BACS - posterior (1 epoch)	87.28	0.4214	0.1650	0.02603
BACS - posterior (10 epochs)	**93.03**	**0.2371**	**0.0965**	**0.02404**

Table 10: **SVHN to MNIST**: In this domain adaptation setting, we find adapting batch norm statistics alone hurts performance compared to the unadapated models, but methods utilizing entropy minimization are able to improve substantially in accuracy.

settings when compared to algorithms specifically designed for these problems. For example, Liang et al. [2020] introduce a source-free domain adaptation algorithm (that also does not require access to the training data during adaptation) and report 99% accuracy transferring from SVHN to MNIST, though results are not directly comparable due to architecture and training differences. In contrast to typical source-free domain adaptation algorithms, our algorithm also focuses on improving uncertainty estimation, does not require multiple epochs of optimization during adaptation, and is amenable to online evaluations where predictions need to be made before seeing the entirety of the test data.