Test-Time Adaptation in Point Clouds: Leveraging Sampling Variation with Weight Averaging - Supplementary Material

Ali Bahri* Moslem Yazdanpanah Mehrdad Noori Sahar Dastani
Milad Cheraghalikhani David Osowiechi Farzad Beizaee Gustavo A. Vargas Hakim
Ismail Ben Ayed Christian Desrosiers

ÉTS Montreal, Canada
International Laboratory on Learning Systems (ILLS)

1. Implementation

We used PyTorch to implement the core functionalities of our approach. The codebase is structured into two main parts: *Pretrain* and *adaptation*.

Pretrain. We begin by focusing on the initial pretraining phase of the base models (Point-MAE, PointNet, DGCNN, and CurveNet). During this phase, we pretrain the backbones in a fully supervised manner, following the standard definition of Test-Time Adaptation (TTA). The pretraining is conducted on clean datasets such as ModelNet, ShapeNet, and ScanObjectNN. This phase ensures that the models are adequately prepared for the subsequent adaptation steps.

Adaptation. After completing the pretraining phase, we transition to the adaptation stage. In this phase, we only update the Normalization Layers of the models using our method, which is built upon the TENT algorithm. By selectively adapting the normalization layers, we efficiently adjust the models to handle corrupted data without requiring full retraining. This targeted approach not only reduces computational costs but also enhances the model's ability to generalize to different data distributions. The results of this adaptation phase are directly reflected in the experimental findings presented in this paper.

To ensure complete transparency and reproducibility of our results, we have made all relevant materials publicly available. This includes:

- The full source code for both *Pretrain* and *Adaptation* phases;
- All log files containing the detailed results of our experiments;
- Pretrained the base models for all backbones.

All these resources can be accessed through our *code*. This repository includes everything needed to understand our code, covering all aspects of the implementations and the reproduction of the results. Moreover, the specific hyperparameters used for all backbones are comprehensively outlined in Table 1.

*Correspondence to ali.bahri.1@ens.etsmtl.ca

2. Resource Overhead

Time. Our method builds on the TENT algorithm but extends it by introducing multiple sampling variations $\mathcal{P}_v$ during Test-Time Adaptation (TTA). While there may be concerns about potential resource overhead, particularly regarding execution time, our method is designed to run in parallel for all $\mathcal{P}_v$. This parallelization allows the model to adapt independently for each variation, significantly reducing time costs compared to a sequential approach. The comparison between parallel and sequential adaptations is detailed in the Supplementary Material Section 3. To quantify the computational cost, we evaluated our method on the PointNet backbone, comparing it directly with TENT. Using $N^V = 6$, the average adaptation time for TENT is approximately **21 ms**, whereas our method required around **26 ms**. This marginal difference indicates that the parallelization ensures minimal resource overhead, making our approach highly efficient even with multiple sampling variations.

Table 1. Hyperparameters

Backbone	Config	Value
All	Optimizer	AdamW
All	learning rate	1e-3
All	Weight decay	0.0
All	Momentum	$\beta = 0.9$
All	Iteration	1
All	FPS	512, 1024
PointMAE-PointNet	Batch size	128
DGCNN-CurveNet	Batch size	16, 64

	Method	uni	gauss	backg	impul	upsam	rbf	rbf-inv	den-dec	dens-inc	shear	rot	cut	distort	oclsion	lidar	Mean
Point-MAE	Source-Only	66.6	59.1	7.2	31.8	74.6	67.7	69.8	59.3	75.1	74.4	38.0	53.7	70.0	38.6	23.4	53.9
	Ours (BN)	85.4	84.7	29.9	74.8	87.1	80.9	82.3	85.1	88.4	82.4	67.9	83.9	80.7	55.7	54.8	74.9
	Ours (BN & LN)	85.0	83.9	33.0	74.6	87.0	80.9	82.3	85.1	88.0	82.7	66.9	84.0	80.5	56.2	55.3	75.0

Table 2. Top-1 Classification Accuracy (%) for all distribution shifts in the ModelNet-40C dataset.

Memory. Given that our method adapts only the learnable parameters of the normalization layers, keeping the other weights frozen and shared, it involves a limited number of parameters in the adaptation process. For instance, in the PointNet backbone, there are approximately 3,500,000 parameters, and we adapt only around 12,000 parameters, which constitutes **0.3%** of all parameters. Consequently, when using $N^V = 6$, the memory resource overhead is approximately **1.8%** of the whole backbone, which is negligible.

3. Additional Experiments

Parallel vs Sequential WA. We investigated two different strategies to handle model adaptation across multiple variations:

- **Parallel Mode**: After adapting the model using each variation $\mathcal{P}_V$, the model is reset to its initial state before the next adaptation begins. The weights adapted from each variation θ_v are stored individually. The final model weights θ_{avg} are then calculated by averaging all the adapted weights across the variations. This approach enables the model to process each variation independently, offering faster adaptation.
- **Sequential Mode**: In this method, the model does not reset after each adaptation. Instead, the adapted model from one variation serves as the starting point for the next variation. This results in iterative adaptation, where the model progressively refines its parameters after each variation $\mathcal{P}_V$, creating a cumulative adaptation process. The final model weights θ_{avg} are then calculated by averaging all the adapted weights across the variations.

As shown in Figure 1, both modes offer similar performance as the number of variations N^V increases. However, since speed and efficiency are critical for TTA, we select the parallel mode, as it allows for faster processing by adapting the model simultaneously across all variations. This experiment was conducted using the Point-MAE backbone on the ModelNet-40C dataset.

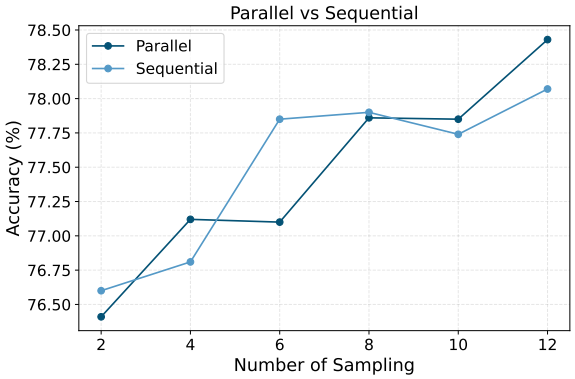

Figure 1. Impact of Parallel vs Sequential on Accuracy

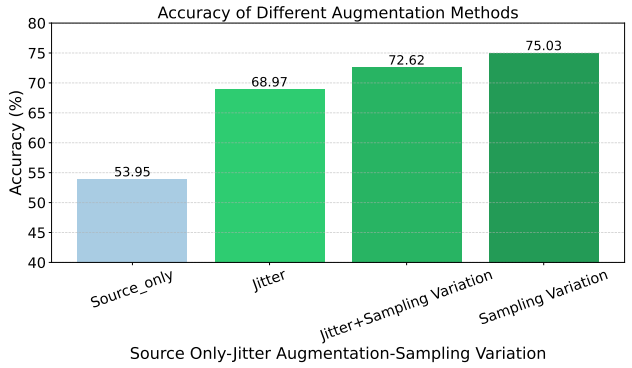

Figure 2. Comparison between Sampling Variation and Different Augmentations

Integration of Jitter and Sampling Variation. In this experiment, we investigate the effect of combining jitter augmentation with the sampling variation as a new strategy $\mathcal{P}$ to generate model diversity in our method. As seen in Figure 2, jitter is selected for this combination because it shows the best performance among other augmentation techniques (as noted in Figure 3 of the main paper). However, while combining jitter with sampling variation yields better results compared to using jitter alone, it does not surpass the performance of our method when using sampling variation

	Method	uni	gauss	backg	impul	upsam	rbf	rbf-inv	den-dec	dens-inc	shear	rot	cut	distort	oclsion	lidar	Mean
CurveNet	Source-Only	67.3	77.1	7.6	47.6	70.1	78.6	80.6	79.2	88.1	77.0	68.8	78.6	77.6	35.5	26.5	64.0
	SHOT [2]	75.5	78.3	22.4	61.1	68.7	72.9	69.1	62.3	64.7	39.2	31.0	30.6	27.1	10.7	8.0	48.1
	DUA [3]	81.5	84.3	27.5	71.1	81.3	82.6	84.5	85.5	89.0	82.1	76.9	85.2	81.7	46.6	45.8	73.7
	PL [1]	79.5	84.0	29.5	72.6	82.7	82.0	83.1	85.9	88.7	81.2	78.9	85.3	81.6	52.8	52.5	74.7
	TENT [4]	80.9	84.9	29.0	73.9	83.8	83.1	**85.5**	85.2	**89.3**	**83.0**	79.8	**85.8**	**83.6**	50.2	51.0	75.3
	Ours	**80.9**	**85.6**	**30.0**	**74.7**	**83.9**	**83.2**	84.3	**86.1**	88.7	82.8	**81.2**	85.7	82.7	**56.3**	**56.4**	**76.2**

Table 3. Top-1 Classification Accuracy (%) for all distribution shifts in the ModelNet-40C dataset.

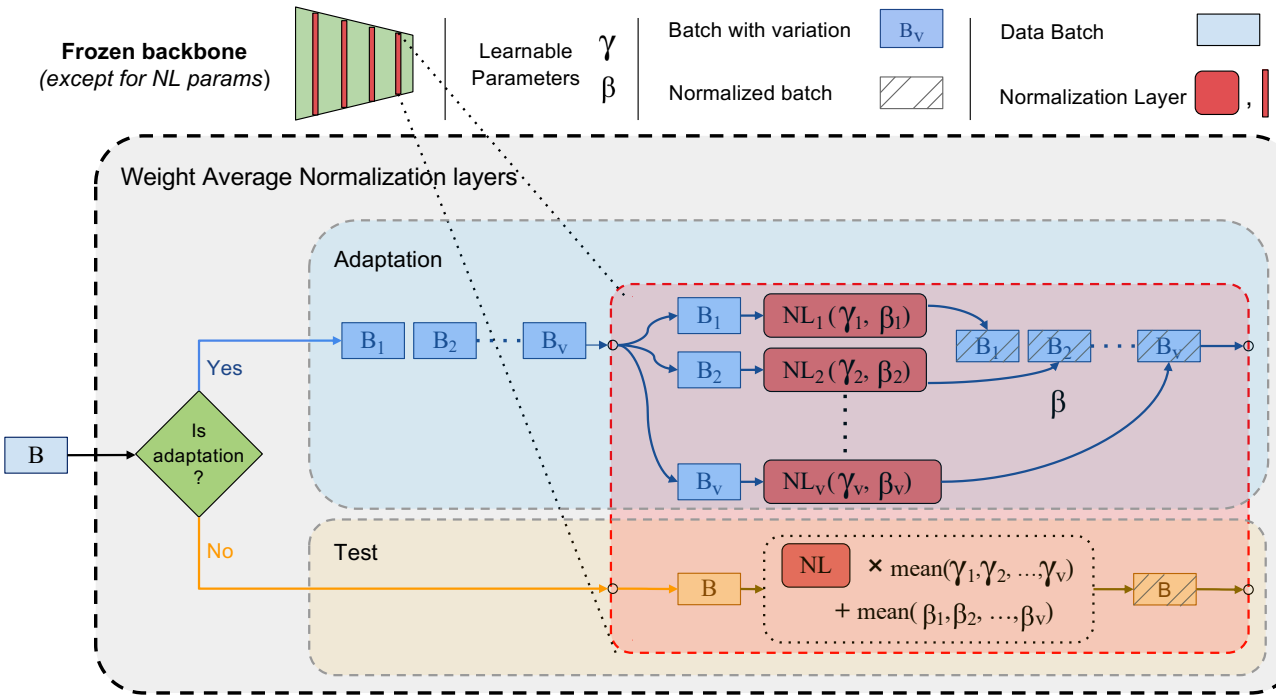

Figure 3. Detailed diagram of our method's Parallel mode.

exclusively.

Impact of Batch Normalization and Layer Normalization. In Table 2, we investigate the effect of updating Batch Normalization (BN) layers only versus updating both Batch Normalization (BN) and Layer Normalization (LN) layers during test-time adaptation. The results demonstrate that updating only BN layers significantly improves performance over the Source-Only baseline. Furthermore, updating both BN and LN layers leads to a slight but consistent improvement across most corruptions, resulting in a higher mean accuracy (75.0%) compared to updating BN layers alone (74.9%). The experiment was conducted with a batch size of 128 and 5 iterations, using PointMAE as the backbone. The dataset used was ModelNet40-C, and weight averaging was performed in parallel mode.

Evaluation on the CurveNet Backbone. In order to further assess the robustness and generalizability of our method, we conducted additional experiments using a different backbone architecture, CurveNet, on the ModelNet-40C dataset. The results are summarized in Table 3. As can be seen, our method demonstrates consistent improvements over baseline approaches, achieving a mean accuracy of **76.2%**, which is notably higher than TENT's accuracy of **75.3%**. The improvements are particularly significant in corruptions like *occlusion* (56.3%), and *lidar* (56.4%), where our method consistently outperforms the other approaches.

Efficient Parallel Implementation. Figure 3 illustrates the detailed implementation of our method in parallel mode. When adapting only the normalization layers, we handle N^V variations $\mathcal{P}_v$ in parallel. For each sampling variation $\mathcal{P}_v$, our method adapts the corresponding normalization layers independently. This means that the weights of the rest of the

network (the majority) are shared across variations, reducing the memory overhead significantly. As shown in Figure 3, we construct a "Weight Average Normalization Layer," which comprises the N^V individual normalization layers.

During adaptation, all the variations are processed through their respective normalization layers. After adaptation, the normalization layer parameters γ and β are then averaged to produce the final set of normalized parameters. With this technique, we avoid saving or reloading the backbone weights for each variation, which leads to memory efficiency. For example, in the PointNet backbone, the normalization layers constitute only 0.3% of the total network parameters. Hence, by adapting only these layers, we reduce the memory resource overhead to a mere 1.8% when using $N^V = 6$, compared to the 500% memory overhead of the naive implementation.

References

[1] Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896. Atlanta, 2013. 3

[2] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International conference on machine learning*, pages 6028–6039. PMLR, 2020. 3

[3] M Jehanzeb Mirza, Jakub Micorek, Horst Possegger, and Horst Bischof. The norm must go on: Dynamic unsupervised domain adaptation by normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14765–14775, 2022. 3

[4] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 3