SHIELD: Secure Hypernetworks for Incremental Expansion Learning Defense

Supplementary Material

A. Extended related works

This work introduces a novel synthesis of two previously disjoint research areas in deep learning: continual learning and adversarial robustness. Since existing literature rarely addresses their intersection, we organize the related work section around both foundations to highlight the unique contributions of our method.

Adversarial Attack. Adversarial attacks on deep neural networks are typically categorized as white-box or black-box, depending on the attacker's access to the target model. In white-box settings, the adversary fully knows the model, including its architecture, parameters, and gradients. Prominent white-box attacks include gradient-based methods such as FGSM [12], BIM [19], MIM [9], PGD [30], and AutoAttack [8], as well as optimization-based approaches like the OnePixel [43].

In black-box settings, attackers have limited access to the model and must rely on indirect signals. Some approaches use confidence scores to estimate gradients, such as zoo [6], while others operate with only the predicted label, as seen in decision-based methods like the Boundary Attack [4]. Another strategy involves crafting adversarial examples on a surrogate model and transferring them to the target model, exploiting the cross-model transferability of adversarial perturbations [9, 10].

Simultaneously with adversarial attack techniques, there are many defence algorithms. Early adversarial defenses relied on heuristics such as input transformations [48], model ensembles [3], and denoisers [21], but many were later found to rely on obfuscated gradients [2]. More robust approaches like adversarial training (AT) [12, 20] and defensive distillation [11, 54] have become dominant. Recent work has enhanced AT with learnable strategies [20], feature perturbation [36], and optimization techniques [38]. However, the above models are typically static and struggle against evolving attack sequences. Test-Time Adaptation Defense [42, 49] addresses adaptation to new attacks but neglects previous ones.

Most existing methods aim to improve adversarial robustness, but they do so without providing formal guarantees. Interval Bound Propagation (IBP) [13, 34] offers a principled alternative by using interval arithmetic to train models that are provably robust against adversarial attacks. These methods enforce that the model classifies all points within a specified perturbation region (typically a ball in ℓ_∞) correctly, by optimizing a worst-case cross-entropy loss.

Continual Learning. Continual learning methods are often grouped into architectural, rehearsal, and regularization-based strategies [18]. Architectural approaches like Progressive Neural Networks [40] and CopyWeights with Reinit [28] prevent forgetting by expanding or modifying network structures. Further developments reuse a fixed architecture with iterative pruning, such as PackNet [31] or Supermasks [47]. Rehearsal-based methods, such as the buffer strategy in [15], retain selected past examples, while pseudo-rehearsal approaches train generative models to replay prior data [32]. Regularization-based techniques like EWC [23], SI [50], LwF [26], and MAS [1] constrain parameter updates based on past task importance, often using the Fisher Information Matrix or similar metrics.

Hypernetworks, introduced in [14], are defined as neural models that generate weights for a separate target network that solves a specific task. The authors aim to reduce the number of trainable parameters by designing a hypernetwork that often has fewer parameters than the target network. In continual learning, hypernetworks can directly generate individual weights for subsequent continual learning tasks, like in HNET [45]. Then, HNET was extended with a probabilistic approach called posterior meta-replay [17]. In HyperMask [25], the authors used the lottery ticket hypothesis to produce task-specific masks for the trainable or fixed target model. In [24], the authors employ interval arithmetic on weights to regularize the target model. These methods are positioned between those based on architecture and those focused on regularization.

The above concepts have many different modifications and upgrades [46]. But the adversarial robustness of continual learning problems is unexplored in the literature. AIR [53] introduced the first framework for continual adversarial defense, addressing robustness against evolving attack sequences. The method leverages unsupervised data augmentation to enhance general robustness in a task-agnostic manner. While innovative, AIR is limited in scalability, having been evaluated on only two to three defense tasks, raising concerns about its effectiveness in longer, continual settings.

Alternatively, the study on Double Gradient Projection (DGP) [39] highlights that this technique is notably more resilient to adversarial attacks compared to traditional continual learning approaches such as GEM [29] and GPM [41]. DGP enhances adversarial robustness by projecting gradients orthogonally concerning a critical subspace before updating weights, thus maintaining the prior gradient smoothness from earlier samples. Furthermore, [22] examines the resistance of current models against adversarial assaults.

These methodologies enhance the robustness of continual learning, yet they have several significant limitations. Firstly, such approaches are restricted to several subsequent tasks and do not give strict guarantees.

B. MixUp

Definition B.1 (MixUp)**.** Given two training examples (x_i, y_i) and (x_j, y_j) drawn independently from the data distribution, MixUp constructs a new virtual training example $(\tilde{x}, \tilde{y})$ as:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \tag{20}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j, \tag{21}$$

where $\lambda \in [0, 1]$ is sampled from the Beta distribution, $\lambda \sim \text{Beta}(\alpha, \alpha)$, with a fixed hyperparameter $\alpha > 0$.

Here, $y_i, y_j \in \mathbb{R}^M$ are typically one-hot encoded class labels, so the mixed label $\tilde{y}$ becomes a soft target, a convex combination of label vectors. To train a model with such soft labels, the loss function must support probabilistic targets. A common choice is the soft-label cross-entropy loss:

$$\mathcal{L}(\tilde{y}, \hat{y}) = -\sum_{k=1}^{M} \tilde{y}_k \log \hat{y}_k, \tag{22}$$

where $\hat{y}$ denotes the model's predicted probability distribution over classes. This formulation encourages linear behavior between training examples, often improving generalization and calibration.

C. Interval Neural Network Layers

The implementation of linear and convolutional layers in the interval version is straightforward and follows the approach described in the main part. However, implementing other interval layers requires more care and is not as trivial.

Interval Batch Normalization. Batch normalization in the interval version is more involved. Instead of simply normalizing the lower and upper bounds separately, we first concatenate the lower and upper bounds into a single tensor. We then compute the batch statistics, mean, and variance, over this combined tensor, capturing the distribution of the entire interval. Using these statistics, we apply the standard batch normalization transformation.

More concretely, given pre-activation interval bounds $[\underline{x}, \overline{x}]$, we form the concatenated tensor:

$$X_{\text{concat}} = [\underline{x}, \overline{x}], \tag{23}$$

and compute the expected mean μ and variance σ^2 over X_{concat}. We then normalize and scale the interval bounds as follows:

$$\left[\frac{\underline{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \frac{\overline{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \right], \tag{24}$$

followed by the affine transformation with learned parameters γ (scale) and β (shift):

$$\gamma \cdot \left[\frac{\underline{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \frac{\overline{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \right] + \beta. \tag{25}$$

We add a small positive constant ϵ to the denominator to ensure numerical stability. Special attention is required to handle the sign of γ correctly to maintain valid lower and upper bounds. If the sign of γ is negative, the lower and upper bounds must be swapped after scaling to preserve correct interval ordering.

Interval Pooling Layers. Pooling operations can be extended to the interval version by applying them separately to the lower and upper bounds of each input element.

For max pooling, given a set of input intervals $\{[\underline{x}_i, \overline{x}_i]\}_{i=1}^n$, the output interval is computed as:

$$\text{MaxPool}_{\text{inf}} = \max_{1 \le i \le n} \underline{x}_i, \tag{26}$$

$$\text{MaxPool}_{\text{sup}} = \max_{1 \le i \le n} \overline{x}_i, \tag{27}$$

and the resulting output interval is $[\text{MaxPool}_{\text{inf}}, \text{MaxPool}_{\text{sup}}]$.

For average pooling, the interval is computed by averaging the bounds independently:

$$\text{AvgPool}_{\text{inf}} = \frac{1}{n} \sum_{i=1}^{n} \underline{x}_i, \tag{28}$$

$$\text{AvgPool}_{\text{sup}} = \frac{1}{n} \sum_{i=1}^{n} \overline{x}_i, \tag{29}$$

so that the final interval is $[\text{AvgPool}_{\text{inf}}, \text{AvgPool}_{\text{sup}}]$.

Average pooling often produces tighter and more stable bounds, and is therefore preferred in interval-based analysis such as IBP.

D. Benchmark Datasets

Permuted MNIST introduces a different fixed pixel permutation for each task, applied to the original MNIST images. Rotated MNIST, on the other hand, creates tasks by rotating the digits by varying angles. These two datasets are widely used in the continual learning literature as synthetic benchmarks derived from MNIST [12, 27]. Split CIFAR-100 [50] is constructed by randomly partitioning the 100 classes of the CIFAR-100 dataset into 10 mutually exclusive groups, with each group forming a separate classification task of 10 classes. Similarly, Split miniImageNet is generated by selecting a subset of the original ImageNet dataset [5] and dividing it into 20 distinct groups, each containing 5 unique classes. For both Split CIFAR-100 and Split miniImageNet, each class appears in only one group, ensuring

Table 2. Comparison of learnable parameters (in millions) between our method and the baseline from [39].

Method	Permuted MNIST	Rotated MNIST	Split CIFAR-100
Baseline	2M	2M	5.5M
SHIELD	2.8M	2.8M	5.6M

no class overlap between tasks. This design allows each subset to be treated as an independent classification task, resulting in a sequence of tasks for continual learning. To match the setup of Split miniImageNet presented in [39], we split Split miniImageNet as described; however, we use only the first ten tasks. Therefore, we have 10 tasks, each with 5 classes.

E. Discussion on Learnable Number of Parameters Used in SHIELD

Since we employ hypernetworks to generate the weights of the target network, the total number of learnable parameters is typically higher than that of baseline architectures. However, we demonstrate that it is possible to significantly reduce this parameter count, bringing it in line with standard models, while still achieving state-of-the-art performance. Importantly, the number of parameters in the target network remains comparable to, or even smaller than, those of the baseline models; the additional overhead stems solely from the hypernetwork. The Interval MixUp technique is also not used in this experiment.

For the Permuted MNIST dataset, we reduce the number of learnable parameters to match the budget used in [39] by employing a hypernetwork composed of a two-layer MLP with hidden dimensions 100 and 50, and an embedding size of 96. The target network is a two-layer MLP with 50 neurons per layer. For the Rotated MNIST dataset, we adopt the same architecture but reduce the embedding size to 24 to further minimize the parameter count. We use $\varepsilon = \frac{25}{255}$ for both Permuted MNIST and Rotated MNIST during training.

For the Split CIFAR-100 dataset, again, we align with the parameter budget from [39] by using an embedding size of 512 and a hypernetwork consisting of a two-layer MLP with 200 and 50 hidden units, respectively. Additionally, we simplify the target AlexNet architecture by removing two fully connected layers, applying a 4×4 average pooling operation after the final convolutional layer in place of the max pooling, and reducing the hidden dimension of the last linear layer to 100. For Split CIFAR-100, we use $\varepsilon = \frac{2}{255}$ during training.

The total number of learnable parameters used in each configuration is reported in Tab. 2. Furthermore, we adopt the same FGSM, PGD, and AutoAttack settings as described in the main paper.

Tables 6, 4, and 5 present the AA of SHIELD and several baselines after learning all tasks sequentially. The results include performance under three adversarial attacks (AutoAttack, PGD, FGSM) and on original clean data.

On the Permuted MNIST benchmark (Tab. 6), SHIELD delivers the strongest adversarial robustness among all compared methods. It surpasses prior approaches under all attack settings, including AutoAttack, PGD, and FGSM, establishing itself as the most resilient model to adversarial perturbations in this setting. While a few baselines, particularly DGP and GPM, achieve higher clean accuracy, their robustness drops notably under stronger attacks. In contrast, SHIELD maintains both high robustness and solid performance on clean samples, without relying on overly large architectures or task-specific replay. This highlights the method's ability to generalize under distribution shifts and adversarial pressure, a balance that remains a major challenge in continual learning.

On Rotated MNIST (Tab. 4), SHIELD sets a new state-of-the-art across all evaluated attacks. It achieves the highest Average Accuracy (AA) under AutoAttack (90.36%), PGD (92.76%), and FGSM (82.21%), outperforming strong baselines like DGP and GPM by a significant margin. Moreover, SHIELD not only maintains robust performance but also retains high AA (93.12%) on original samples, closely matching the top-performing baselines in that category.

On the more complex Split CIFAR-100 benchmark (Tab. 5), SHIELD once again delivers the strongest performance under adversarial attacks. It outperforms all baselines by a wide margin in both AutoAttack and PGD evaluations, achieving 54.55% and 54.53% AA, respectively, substantially ahead of the next best method, DGP. While some baselines achieve slightly better AA on original samples, they fall short under adversarial threat models. In contrast, SHIELD consistently balances robustness and plasticity, confirming its capacity to learn durable representations without catastrophic forgetting, even in challenging continual learning settings.

While SHIELD does not achieve the highest AA on original samples in this setup, it remains competitive with top methods like DGP. We emphasize that Interval MixUp was not used; our goal is to show that SHIELD performs well even without it and with a smaller target network architecture. The slight drop in clean accuracy stems from two deliberate choices: (1) SHIELD is trained using a relatively small target network architecture, and (2) training with stronger adversarial perturbations, which often lowers clean accuracy but improves robustness. This trade-off allows SHIELD to strike a strong balance between robustness and performance, making it a reliable option for adversarially robust continual learning.

Table 3. Comparison of AA after completing all tasks on the Permuted MNIST dataset. AA results for SHIELD are averaged over 2 seeds for AutoAttack and 5 seeds for all other evaluations.

Method	Permuted MNIST			
	AutoAttack	PGD	FGSM	Original samples
	AA(%)	AA(%)	AA(%)	AA(%)
SGD	14.1	15.4	21.8	36.8
SI	14.3	16.5	22.3	36.9
A-GEM	14.1	19.7	22.9	48.4
EWC	39.4	43.1	50.0	84.9
GEM	12.1	75.5	72.8	96.4
OGD	19.7	24.1	26.0	46.8
GPM	70.4	72.9	65.7	97.2
DGP	81.6	81.2	75.8	**97.6**
SHIELD	**88.43 ± 0.35**	**90.47 ± 1.36**	**79.6 ± 1.34**	90.71 ± 1.8

Table 4. Comparison of AA after completing all tasks on the Rotated MNIST dataset. AA results for SHIELD are averaged over 2 seeds for AutoAttack and 5 seeds for all other evaluations.

Method	Rotated MNIST			
	AutoAttack	PGD	FGSM	Original samples
	AA(%)	AA(%)	AA(%)	AA(%)
SGD	14.1	9.9	20.4	32.3
SI	13.9	15.3	20.1	33.0
A-GEM	14.1	21.6	24.8	45.4
EWC	45.1	49.5	46.5	80.7
GEM	11.9	76.5	74.4	96.7
OGD	19.7	23.8	23.8	48.0
GPM	68.8	71.5	65.9	97.1
DGP	81.6	82.6	78.6	**98.1**
SHIELD	**90.36 ± 0.1**	**92.76 ± 0.22**	**82.21 ± 0.43**	93.12 ± 0.22

F. Selected Hyperparameters

For the Split CIFAR-100 experiments, we used an AlexNet-based target network along with a hypernetwork comprising two hidden layers of 100 and 50 neurons, respectively. The embedding size was set to 512, with a batch size of 32 and a learning rate of 0.001. The hypernetwork used a regularization coefficient $\beta = 0.01$, and training was performed with an ℓ_∞ perturbation of $\varepsilon = 0.005$. The optimizer was Adam, and no data augmentation was used. Training was conducted for 200 epochs with ReLU activation in both the target and hypernetwork. We applied a learning rate scheduler: ReduceLROnPlateau with monitoring set to the maximum validation accuracy, reduction factor $\sqrt{0.1}$, patience of 5 epochs, minimum learning rate of 5×10^{-7}, and no cooldown. Batch normalization was enabled throughout. When Interval MixUp is applied, we sample the mixing coefficient from a Beta distribution with parameter $\alpha = 0.3$.

For the Permuted MNIST task, we employed a 2-layer MLP with 256 neurons per layer in the target network and 100 neurons per layer in the hypernetwork. The embedding size was 24, the batch size was 128, and the learning rate was 0.001. We used $\beta = 0.001$ and performed training with $\varepsilon = 0.01$. For the Rotated MNIST experiments, the setup mirrored that of Permuted MNIST but used an embedding size of 96. Training on Permuted MNIST and Rotated MNIST was conducted for 5000 iterations. When using Interval MixUp, we set the Beta distribution parameter to $\alpha = 0.1$.

In the Split miniImageNet experiments, we used a ResNet-18 target network and a hypernetwork with two hidden layers of 100 and 50 neurons, respectively. The embedding dimension was 128, with a batch size of 16 and a learning rate of 0.001. We trained using the Adam optimizer for 50 epochs, applying ReLU activations and enabling batch normalization throughout. The hypernetwork was regularized with $\beta = 0.01$, and training included an ℓ_∞ perturbation of $\varepsilon = \frac{2}{255}$. No data augmentation was used. The learning rate scheduler matched that of the Split CIFAR-100 setup. When Interval MixUp was used, mix-

Table 5. Comparison of AA after completing all tasks on the Split CIFAR-100 dataset. AA results for SHIELD are averaged over 2 seeds for AutoAttack and 5 seeds for all other evaluations.

Method	Split CIFAR-100			
	AutoAttack	PGD	FGSM	Original samples
	AA(%)	AA(%)	AA(%)	AA(%)
SGD	10.3	12.8	19.4	46.5
SI	13.0	15.2	19.8	45.4
A-GEM	12.6	12.9	20.7	40.6
EWC	12.6	23.2	30.5	56.8
GEM	21.2	19.4	47.7	60.6
OGD	11.8	14.1	18.9	44.2
GPM	34.4	36.6	**53.7**	58.2
DGP	36.6	39.2	48.0	**67.2**
SHIELD	**54.55 ± 0.42**	**54.53 ± 0.48**	47.71 ± 0.86	59.36 ± 1.11

ing coefficients were drawn from a Beta distribution with $\alpha = 0.2$.

Model selection was based on the best validation loss. The optimizer across all experiments was Adam, and ReLU was used as the activation function throughout. In all cases, the same adversarial attack settings were used as in [39], except that the number of attack iterations for PGD was not specified in that work; we defaulted to 100 iterations for consistency across evaluations. For AutoAttack, we used the standard version with default parameters as suggested by the authors in [7].

Experiments were conducted on NVIDIA RTX 4090 and A100 GPUs. All software and package versions are listed in the README file in our code repository.

G. Training Details

To identify the most effective configuration for each dataset, we conducted an extensive grid search over key hyperparameters, evaluating combinations to maximize validation performance. Below we summarize the search spaces used for each dataset.

Permuted MNIST. For the Permuted MNIST dataset, the grid search was performed using a 2-layer MLP with 256 neurons per layer in the target network. As a hypernetwork, we used an MLP, with architecture defined by the hypernetwork hidden layers parameter. The following hyperparameters were explored:

- Embedding sizes: 24, 48, 96
- Learning rate: 0.001
- Batch sizes: 64, 128
- Regularization coefficients β: 0.0005, 0.001, 0.005, 0.01, 0.05
- Perturbation sizes ε: $\frac{2}{255}$, $\frac{20}{255}$, $\frac{25}{255}$, 0.01
- Hypernetwork hidden layers (MLP): [100, 50], [200, 50], [100, 100]
- Number of training iterations: 5000
- Number of validation samples per class in each task: 500
- Beta distribution hyperparameters (α): 0.01, 0.1, 0.2, 0.3, 0.4, 0.5

Rotated MNIST. We use the same grid search as for the Permuted MNIST dataset.

Split CIFAR-100. For the Split CIFAR-100 dataset, the grid search was conducted using AlexNet as the target network and an MLP-based hypernetwork. Batch normalization was enabled, and data augmentation was disabled. The full hyperparameter search space included:

- Embedding sizes: 128, 256, 512
- Learning rate: 0.001
- Batch sizes: 32, 64
- Regularization coefficients β: 0.01, 0.05, 0.1
- Perturbation sizes ε: $\frac{2}{255}$, $\frac{4}{255}$, 0.005, 0.01
- Hypernetwork hidden layers (MLP): [200, 50], [100, 50], [100, 100], [100], [200]
- Number of training epochs: 200
- Number of validation samples per class in each task: 500
- Beta distribution hyperparameters (α): α: 0.01, 0.1, 0.2, 0.3, 0.4, 0.5

Split miniImageNet. For the Split miniImageNet dataset, the grid search was conducted using ResNet-18 as the target network and an MLP-based hypernetwork. Batch normalization was enabled, and data augmentation was disabled. The full hyperparameter search space included:

- Embedding sizes: 96, 128, 256
- Learning rate: 0.001
- Batch size: 16

- Regularization coefficients β: 0.01, 0.05
- Perturbation sizes ε: $\frac{1}{255}$, $\frac{2}{255}$, $\frac{4}{255}$
- Hypernetwork hidden layers (MLP): [200, 50], [100, 50], [100]
- Number of training epochs: 50
- Number of validation samples per class in each task: 250
- Beta distribution hyperparameters (α): α: 0.01, 0.1, 0.2, 0.3, 0.4, 0.5

The Adam optimizer was used across all tasks. These grid searches enabled a systematic selection of optimal configurations for each dataset.

Interval Bound Propagation Training Details. Training with overly wide intervals can destabilize the learning process, especially in the early stages. To mitigate this, we begin training with $\kappa = 1$, placing full emphasis on the standard classification loss. As training progresses, the weight on the interval bound loss increases gradually, reaching $\kappa = 0.5$ by the midpoint of training. Simultaneously, we initialize the perturbation radius at $\varepsilon = 0$ and linearly increase it during the first half of training, so that it reaches the target value at the midpoint. Both κ and ε remain fixed for the remainder of training.

Formally, let E be the total number of training iterations per task, and denote by κ_i and ε_i the values of κ and ε at iteration $i \in \{1, \ldots, E\}$. We define their schedules as:

$$\kappa_i = \max\{\frac{1}{2}, 1 - \frac{i}{2E}\} \tag{30}$$

$$\varepsilon_i = \begin{cases} \frac{2i \cdot \varepsilon}{E} & \text{if } i \leq \lfloor \frac{E}{2} \rfloor, \\ \varepsilon & \text{otherwise.} \end{cases} \tag{31}$$

Architecture Details. In the main part of the paper, we use the same base architecture across Permuted MNIST, Rotated MNIST, and Split-miniImageNet. For the Split CIFAR-100, we adopt a streamlined variant of AlexNet to meet the capacity and efficiency requirements of hypernetwork-based training. Specifically, we remove dropout, include biases, and replace standard batch normalization layers with interval-aware variants. The classifier comprises two fully connected layers with 100 units each. To improve numerical stability during IBP, we replace max pooling with average pooling. Additionally, a 4×4 average pooling layer is applied after the final convolutional layer to reduce dimensionality.

This lightweight adaptation of AlexNet ensures that the target network remains compact, which is critical for training efficiency when its parameters are generated dynamically by a hypernetwork. Importantly, unlike [39], we do not allocate separate convolutional layers for each task; our model maintains a single shared architecture across tasks. Further architectural details and hyperparameter choices are discussed in Appendix Section G. A shared MLP hypernetwork is used across all datasets.

H. Ablation Study and Additional Experiments

H.1. Results on Permuted MNIST

On Permuted MNIST (Tab. 6), SHIELD achieves competitive AutoAttack performance, closely matching DGP, while outperforming all baselines under PGD, FGSM, and on original samples. Notably, it is the only method with positive BWT, indicating both effective knowledge retention and improvement on previous tasks. SHIELD$_{\text{IM}}$ further improves PGD and original accuracy, with a minor trade-off in BWT.

Table 6. AA performance after all tasks on the Permuted MNIST. Results for SHIELD and SHIELD with Interval MixUp (SHIELD$_{\text{IM}}$) are averaged over 2 seeds for AutoAttack and 5 seeds for the other evaluations. Baseline results are from [39].

Method	AutoAttack	PGD	FGSM	Original samples	
Metric	AA(%)	AA(%)	AA(%)	AA(%)	BWT
Permuted MNIST					
SGD	14.1	15.4	21.8	36.8	-0.66
SI	14.3	16.5	22.3	36.9	-0.67
A-GEM	14.1	19.7	22.9	48.4	-0.54
EWC	39.4	43.1	50.0	84.9	-0.12
GEM	12.1	75.5	72.8	96.4	-0.01
OGD	19.7	24.1	26.0	46.8	-0.57
GPM	70.4	72.9	65.7	97.2	-0.01
DGP	**81.6**	81.2	75.8	97.6	-0.01
SHIELD	80.91	90.11	78.87	93.58	**0.02**
SHIELD$_{\text{IM}}$	80.08	**97.44**	**79.09**	**97.96**	-0.05

H.2. Alternative Epsilon Decay Rate Functions in Interval MixUp

In this subsection, we investigate how different epsilon decay rates affect the training of SHIELD. In the paper, we employ a linear decay schedule. However, it is important to examine whether alternative decay strategies lead to different training behaviors. To this end, we additionally consider quadratic, logarithmic, and cosine decay functions. The corresponding plots of these decay schedules are shown in Figure 4.

Fig. 5 presents the AA over time for various decay rate strategies across three continual learning benchmarks: Permuted MNIST, Split CIFAR-100, and Split miniImageNet. The top row shows AA, computed as the mean accuracy over all tasks seen so far, immediately after learning each task. The bottom row shows final AA, calculated as the average accuracy on all tasks after the final task is learned.

Across all datasets, linear and logarithmic decay rate schedules demonstrate stable and competitive performance

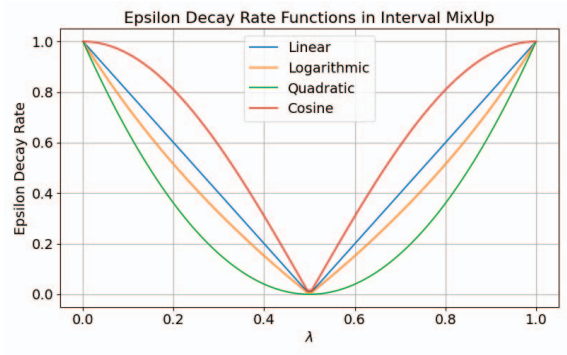


Figure 4. Comparison of different epsilon decay rates used in Interval MixUp.

throughout training. In contrast, cosine and quadratic functions result in noticeably unstable and degraded accuracy, particularly evident in later tasks (such as the 6th task of Split CIFAR-100 and Split miniImageNet). This instability may be attributed to the more abrupt fluctuations of cosine and quadratic schedules (Figure 4). These findings suggest that decay strategies, such as linear or logarithmic, better support robust learning in adversarial continual learning scenarios.

Table 7. Comparison of AA after completing all tasks on the Permuted MNIST dataset. AA results for different scheduling strategies (linear, quadratic, log, and cos) were calculated for a single seed.

Schedule	Permuted MNIST				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
Linear	80.53	97.48	79.4	98.01	**0.002**
Quadratic	80.45	**97.64**	79.51	**98.16**	0.001
Log	80.38	97.44	79.67	97.98	-0.03
Cos	**81.44**	97.4	**80.1**	97.92	-0.05

Table 8. Comparison of AA after completing all tasks on the Split CIFAR-100 dataset. AA results for different scheduling strategies (linear, quadratic, log, and cos) were calculated for a single seed.

Schedule	Split CIFAR-100				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
Linear	**63.89**	**62.9**	**46.99**	66.97	**-0.22**
Quadratic	63.14	62.58	46.72	**67.6**	-0.27
Log	62.91	62.1	46.75	67.07	-0.68
Cos	58.87	57.72	42.91	62.44	-5.29

We evaluate the performance of SHIELD trained using linear, quadratic, logarithmic, and cosine epsilon decay rate schedules in the context of adversarial attacks. The results

Table 9. Comparison of AA after completing all tasks on the Split miniImageNet dataset. AA results for different scheduling strategies (linear, quadratic, log, and cos) were calculated for a single seed.

Schedule	Split miniImageNet				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
Linear	**60.08**	**60.32**	**56.16**	**64.72**	**-0.13**
Quadratic	59.2	51.36	46.92	55.2	-7.33
Log	55.72	55.64	51.64	59.84	-2.09
Cos	55.52	55.84	51.88	60.2	-3.51

are presented in Figures 7, 8, and 9. The attacks used are the same as those described in the main paper. On Permuted MNIST, the results are nearly identical across all schedules. However, for Split CIFAR-100 and Split miniImageNet, the linear decay strategy performs the best. In this experiment, we used the best hyperparameters identified for the linear strategy in the main paper, so it is possible that further grid search for the other schedules could yield improved results.

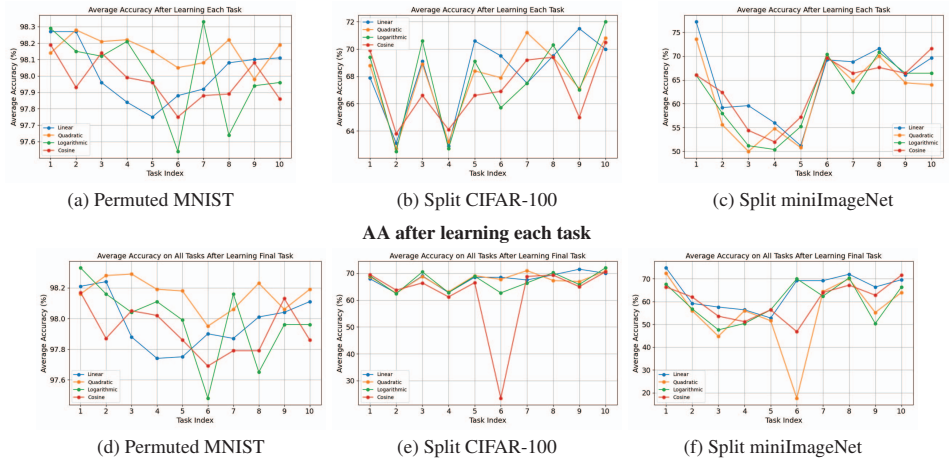
H.3. Impact of FGSM Perturbation Size on Average Accuracy

In Fig. 6, we show the effect of FGSM attacks with varying $\varepsilon_{\text{attack}}$ values on the AA metric for our method and for HNET. The results indicate that HNET is substantially more vulnerable to adversarial perturbations, with a steep drop in AA on Split CIFAR-100 and Split miniImageNet, even under small $\varepsilon_{\text{attack}}$ values. For each subfigure in Fig. 6, we evaluate our models using 10 FGSM attack strengths, with $\varepsilon_{\text{attack}}$ values evenly spaced from 0 up to twice the ε (perturbation value used in the training process) used during training. This setup allows us to assess how well the models generalize robustness beyond the training-time perturbation level. The optimal ε values used are those identified in Section F.

H.4. Impact of PGD Iterations on Average Accuracy

In Fig. 7, we present the effect of PGD attacks with an increasing number of steps on the AA metric for SHIELD and HNET. Across all evaluated datasets, SHIELD consistently achieves higher initial accuracy when subjected to no attack or only a few PGD steps. Both SHIELD and HNET exhibit a noticeable drop in AA within the first 10 to 25 steps; however, their performance then stabilizes, demonstrating resilience to stronger iterative attacks.

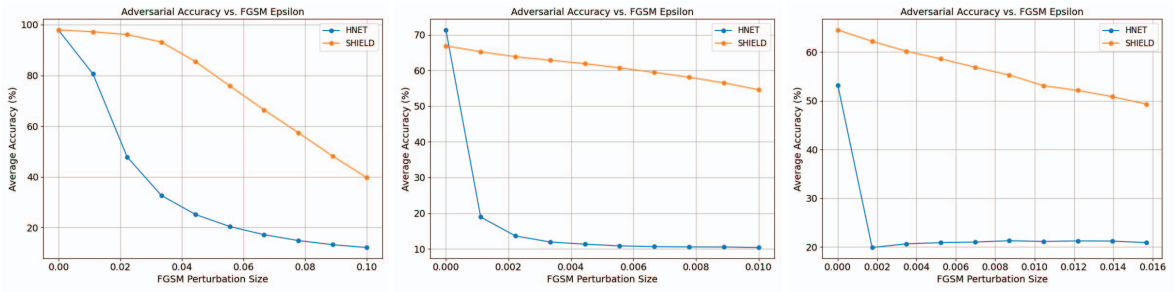
For each subfigure in Fig. 7, we evaluate both models across 10 PGD configurations, with the number of attack steps evenly spaced from 0 to 200. The remaining PGD hyperparameters (e.g., step size, $\varepsilon_{\text{attack}}$) are kept identical to those used in the main experiments. This extended evaluation allows us to assess model robustness across a broader



(a) Permuted MNIST (b) Split CIFAR-100 (c) Split miniImageNet

(d) Permuted MNIST (e) Split CIFAR-100 (f) Split miniImageNet

Figure 5. Comparison of results for different epsilon decay rates across (a) Permuted MNIST, (b) Split CIFAR-100, and (c) Split miniImageNet. Top row: AA measured immediately after learning each task. Bottom row: AA evaluated on all tasks after completing the final task.



(a) Permuted MNIST (b) Split CIFAR-100 (c) Split miniImageNet

Figure 6. AA of SHIELD and HNET under FGSM attacks with increasing perturbation sizes, evaluated on models trained on Permuted MNIST (Fig. 6a), Split CIFAR-100 (Fig. 6b), and Split miniImageNet (Fig. 6c).

attack range than previously reported. As before, we use the best-performing hyperparameters identified in Section F.

These results further confirm that SHIELD provides stronger and more stable adversarial robustness, particularly under longer and more aggressive PGD attacks.

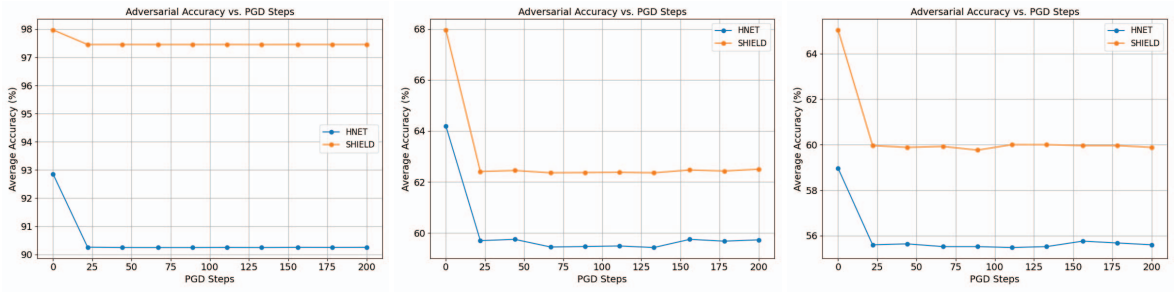
H.5. Impact of β Hyperparameter Selection on Training Process of SHIELD

We evaluate the ability of SHIELD to retain knowledge across tasks, which is controlled by the hyperparameter β. In Fig. 8, we present AA curves obtained by training the model with optimal hyperparameters while varying β. The results show that β values of 0.01 and 1.0 yield the best performance, ensuring stable predictions and minimal forgetting. In contrast, a small value of $\beta = 0.001$ leads to significant forgetting, particularly on the 5th and 6th tasks, after learning the final task.

H.6. Performance of SHIELD on TinyImageNet dataset

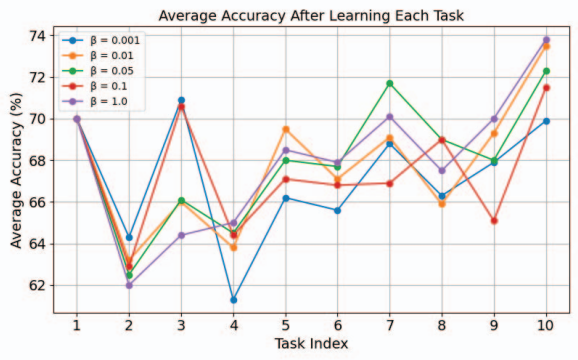
We assess the scalability of SHIELD on a more challenging benchmark involving a larger number of tasks. Specifically, we use TinyImageNet, a subset of ImageNet, consisting of 200 classes, each with 500 training images and 50 validation images at a resolution of 64×64 pixels. To construct the benchmark, we define 40 tasks, each composed of 5 randomly selected, non-overlapping classes. This setup allows us to evaluate SHIELD's performance under increased task complexity and class diversity.

We compare the robustness of SHIELD with the traditional HNET approach [45] under three adversarial attacks: FGSM, PGD, and AutoAttack. Both models are trained

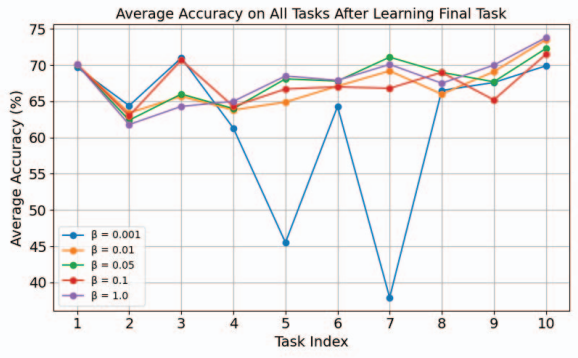


(a) Permuted MNIST (b) Split CIFAR-100 (c) Split miniImageNet

Figure 7. AA of SHIELD and HNET under PGD attacks with increasing number of gradient steps, evaluated on models trained on Permuted MNIST (Fig. 7a), Split CIFAR-100 (Fig. 7b), and Split miniImageNet (Fig. 7c).



(a) AA calculated directly after learning each task



(b) AA calculated after the final task

Figure 8. Comparison of AA curves for varying β values, which control hypernetwork regularization strength in SHIELD

specifically on TinyImageNet. For FGSM, we set the attack strength to $\varepsilon_{\text{attack}} = \frac{4}{255}$. For PGD and AutoAttack, we use $\varepsilon_{\text{attack}} = \frac{2}{255}$, with a PGD step size of $\delta = \frac{4}{255}$ and 100 iterations. The AutoAttack configuration is the same as used in the main experiments of the paper.

We use ResNet-18 as the target network. The hypernetwork is implemented as a multilayer perceptron (MLP) with a single hidden layer of 100 neurons and an embedding size of 48. Results are reported using a fixed random seed set to 1, shared across both models. We train both architectures for 50 epochs using the Adam optimizer, a learning rate of 0.001, and a batch size of 128. The same learning rate scheduler as in the Split CIFAR-100 experiments is applied. We set $\beta = 0.05$. For SHIELD, we use $\varepsilon = 0.005$ during training and apply Interval MixUp.

Table 10. Comparison of AA after completing all tasks on the TinyImageNet dataset.

Method	AutoAttack	PGD	FGSM	Original samples
HNET	46.37	45.3	49.17	63.83
SHIELD	**51.78**	**52.83**	**52.36**	**64.33**

From Tab. 10, we observe that even on a challenging dataset like TinyImageNet, our model achieves high AA on original (non-adversarial) test samples while also maintaining robustness against adversarial attacks. While HNET achieves competitive results, our method performs slightly better, especially under adversarial conditions.

H.7. Impact of κ Hyperparameter Selection on Average Accuracy

To evaluate the sensitivity of our method to the hyperparameter κ, we conducted experiments on the Split CIFAR-100 dataset, measuring AA under various adversarial attacks (AutoAttack, PGD, FGSM) as well as on original samples, along with BWT. The results, averaged over 2 seeds, are presented in Tab. 11. The table compares performance across κ values of 0.0, 0.5, and 1.0, with the best results per metric highlighted in bold. From the results, $\kappa = 0.5$ consistently achieves the highest average accuracy across all evaluated attacks and on original samples, demonstrating superior robustness and generalization. While BWT is slightly negative for $\kappa = 0.5$, indicating minimal forgetting,

Table 11. Impact of the hyperparameter κ on AA under various attacks and backward transfer on Split CIFAR-100. Results are averaged over 2 seeds. Bold values indicate the best performance per metric.

κ	Split CIFAR-100				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
0.0	61.24	60.1	45.01	64.93	**0.06**
0.5	**63.08**	**62.39**	**46.48**	**67.45**	-0.41
1.0	56.58	58.94	45.54	64.55	- 8.39

it outperforms the other values, particularly $\kappa = 1.0$, which exhibits significant degradation in both accuracy and BWT. In contrast, $\kappa = 0.0$ shows moderate performance but falls short of the balanced improvements offered by $\kappa = 0.5$. Based on these findings, we select $\kappa = 0.5$ for all experiments in the main paper, as it provides the optimal trade-off between adversarial robustness, clean accuracy, and knowledge retention in continual learning settings.

H.8. Hypernetwork Ablation

To assess the importance of the hypernetwork component in SHIELD, we compare the hypernetwork with IBP training against a variant where the hypernetwork is replaced with EWC while keeping IBP training. In both settings, we additionally employ Interval MixUp to ensure a fair comparison. The backbone architectures (AlexNet/MLP, depending on the dataset) remain unchanged. Results are reported in Table 12.

Table 12. Ablation study comparing SHIELD and EWC + IBP (both with Interval MixUp). Results report AA on original samples and are averaged over 5 seeds.

Method	Permuted MNIST	Split CIFAR-100
EWC + IBP	58.77 ± 0.21	15.97 ± 0.04
SHIELD	**97.96 ± 0.04**	**67.45 ± 0.28**

Replacing the hypernetwork with EWC leads to a noticeable drop in performance, particularly on Split CIFAR-100. On Permuted MNIST, EWC with IBP achieves $58.77 \pm 0.21\%$ AA, while performance on Split CIFAR-100 decreases to $15.97 \pm 0.04\%$. Notably, AA on original samples in the EWC-based variant is already substantially lower, which further limits adversarial robustness under strong attacks. This suggests that the hypernetwork plays a crucial role in maintaining stable representations across tasks, which in turn supports stronger adversarial robustness.

Overall, the results indicate that the hypernetwork is an essential component of SHIELD, contributing significantly to both continual knowledge retention and robustness under adversarial evaluation.

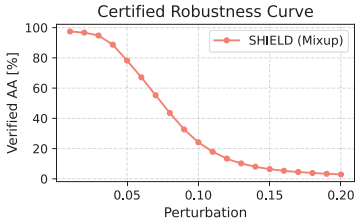


Figure 9. IBP-verified AA as a function of the perturbation radius ε for SHIELD on Permuted MNIST, evaluated after the final task.

H.9. IBP-Verified Robustness Curve on Permuted MNIST

In Fig. 9, we report the IBP-verified AA as a function of the perturbation radius ε. The curve is obtained using the model that achieved the best performance in the main paper and is evaluated after training on the final task. For each value of ε, a prediction is counted as correct only if the IBP bounds certify that the predicted class remains unchanged for all inputs within the corresponding ℓ_∞ neighborhood.

I. Extended Results from Main Paper

In this section, we extend results from Table 1 to attach standard deviations. We present extended results in Table 13, 14, 15, and 16.

J. Class-Incremental Learning Results

Tab. 17 presents a comparison between SHIELD and the HNET baseline in terms of adversarial robustness across four datasets: Permuted MNIST, Rotated MNIST, Split CIFAR-100, and Split miniImageNet, within the Class-Incremental Learning (CIL) setting. The results include AA under three adversarial attacks (AutoAttack, PGD, FGSM) as well as accuracy on original test samples, all reported after learning the full task sequence.

The results indicate that SHIELD consistently achieves higher adversarial robustness than HNET under AutoAttack across all datasets. On Permuted MNIST and Rotated MNIST, SHIELD shows large margins of improvement across all attack types, with particularly high AA under PGD on Permuted MNIST (97.21%) and AutoAttack (36.9%) on Rotated MNIST. For Split CIFAR-100, both methods perform similarly. On the more challenging Split miniImageNet benchmark, SHIELD again significantly outperforms HNET in all scenarios, achieving over 12% of AA under AutoAttack compared to just 2% for the baseline.

These findings confirm that SHIELD offers a substantial improvement in adversarial robustness in continual learning, even under challenging conditions. Notably, this is the first method, to our knowledge, that demonstrates strong ro-

Table 13. Comparisons of AA and BWT after learning all the tasks on Permuted MNIST dataset. The results for baselines were extracted from Table 1 in [39].

Method	Permuted MNIST				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
SGD	14.1	15.4	21.8	36.8	-0.66
SI	14.3	16.5	22.3	36.9	-0.67
A-GEM	14.1	19.7	22.9	48.4	-0.54
EWC	39.4	43.1	50.0	84.9	-0.12
GEM	12.1	75.5	72.8	96.4	-0.01
OGD	19.7	24.1	26.0	46.8	-0.57
GPM	70.4	72.9	65.7	97.2	-0.01
DGP	**81.6**	81.2	75.8	97.6	-0.01
SHIELD	80.91 ± 1.18	90.11 ± 0.8	78.87 ± 2.06	93.58 ± 0.52	**0.02 ± 0.01**
$\text{SHIELD}_{\text{IM}}$	80.08±0.01	**97.44 ± 0.05**	**79.09 ± 0.87**	**97.96 ± 0.04**	-0.05 ± 0.03

Table 14. Comparisons of AA and BWT after learning all the tasks on Rotated MNIST dataset. The results for baselines were extracted from Table 2 in [39].

Method	Rotated MNIST				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
SGD	14.1	9.9	20.4	32.3	-0.71
SI	13.9	15.3	20.1	33.0	-0.72
A-GEM	14.1	21.6	24.8	45.4	-0.57
EWC	45.1	49.5	46.5	80.7	-0.18
GEM	11.9	76.5	74.4	96.7	-0.01
OGD	19.7	23.8	23.8	48.0	-0.55
GPM	68.8	71.5	65.9	97.1	-0.01
DGP	81.6	82.6	78.6	98.1	**-0.00**
SHIELD	**85.64 ± 0.03**	92.94 ± 0.16	**83.82 ± 0.2**	95.62 ± 0.06	-0.03 ± 0.05
$\text{SHIELD}_{\text{IM}}$	82.91 ± 0.37	**97.88 ± 0.08**	83.03 ± 0.45	**98.32 ± 0.06**	-0.08 ± 0.04

bustness against adversarial attacks in the CIL setting, highlighting the practical effectiveness and novelty of the proposed framework.

The results reported in Tab. 17 were averaged over multiple random seeds to ensure statistical reliability. Specifically, results for Permuted MNIST, Rotated MNIST, and Split CIFAR-100 were averaged over 5 seeds, while Split miniImageNet results were averaged over 3 seeds. Due to the computational cost of AutoAttack, its results were averaged over 2 seeds for all datasets.

We present the pseudocode for SHIELD in the CIL setting in Algorithm 1. During inference, since task identity is unknown, the hypernetwork generates task-specific models for all tasks, and each is evaluated on the given input. The task is inferred by selecting the model with the lowest predictive entropy, assuming that the correct task model yields the most confident prediction. When adversarial attacks are considered, task inference is always performed on perturbed inputs to simulate realistic evaluation conditions. A prediction is counted as correct only if both the task is correctly inferred and the predicted class matches the true label; otherwise, it is treated as misclassified.

K. Comparison with AIR Method

Tab. 18 reports the performance of SHIELD under the AIR evaluation protocol (None → FGSM → PGD) on CIFAR-100. SHIELD achieves the highest accuracy on Task 2 and Task 3, outperforming all baselines under both FGSM and PGD attacks in the later tasks.

Importantly, SHIELD is attack-agnostic - it does not rely on generating adversarial examples during training, in contrast to AIR. Instead, robustness is encouraged through interval-based training. For a fair comparison, we adopt the Wide ResNet architecture used in AIR, but employ a 10-layer variant with a widening factor of 10 to reduce the number of learnable parameters and ensure feasibility on our available hardware. Apart from this architectural adjust-

Table 15. Comparisons of AA and BWT after learning all the tasks on Split CIFAR-100 dataset. The results for baselines were extracted from Table 3 in [39].

Method	Split CIFAR-100				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
SGD	10.3	12.8	19.4	46.5	-0.49
SI	13.0	15.2	19.8	45.4	-0.48
A-GEM	12.6	12.9	20.7	40.6	-0.48
EWC	12.6	23.2	30.5	56.8	-0.35
GEM	21.2	19.4	47.7	60.6	-0.13
OGD	11.8	14.1	18.9	44.2	-0.50
GPM	34.4	36.6	**53.7**	58.2	**-0.10**
DGP	36.6	39.2	48.0	67.2	-0.13
SHIELD	60.91	59.77	45.37	64.24	-0.34
$\text{SHIELD}_{\text{IM}}$	**63.08** $\pm$ **0.64**	**62.39** $\pm$ **0.38**	46.48 $\pm$ 0.33	**67.45** $\pm$ **0.28**	-0.41 $\pm$ 0.19

Note: In the original Table 3 from [39], the reported accuracy for FGSM was higher than for original samples, which is inconsistent with expected adversarial behavior. We assume this was a mistake and have swapped the columns accordingly for a fair comparison.

Table 16. Comparisons of AA and BWT after learning all the tasks on Split miniImageNet dataset. The results for baselines were extracted from Figure 3 in [39].

Method	Split miniImageNet				
	AutoAttack	PGD	FGSM	Original samples	
	AA(%)	AA(%)	AA(%)	AA(%)	BWT
SGD	20.5	22.0	23.5	30.8	-0.24
A-GEM	19.0	19.8	21.2	29.2	-0.28
EWC	21.3	22.7	24.3	29.9	-0.25
SI	20.4	21.3	22.7	28.1	-0.27
GEM	22.3	23.8	25.4	31.8	-0.20
OGD	17.9	18.8	20.7	29.6	-0.29
GPM	26.3	27.1	28.8	36.8	-0.12
DGP	32.1	33.8	35.5	44.8	**-0.05**
SHIELD	56.22 $\pm$ 0.37	56.8 $\pm$ 0.95	53.08 $\pm$ 0.63	59.52 $\pm$ 0.67	-0.16 $\pm$ 0.3
$\text{SHIELD}_{\text{IM}}$	**57.9** $\pm$ **3.08**	**58.47** $\pm$ **1.9**	**54.1** $\pm$ **2.42**	**62.67** $\pm$ **1.9**	-0.18 $\pm$ 0.43

ment, all experimental settings follow [53]. Our hypernetwork consists of a two-layer MLP with 100 and 50 hidden units, respectively, and an embedding size of 512.

L. Training Algorithm

The training procedure of SHIELD is detailed in Algorithm 2. Note that subtracting a scalar from a tensor denotes a broadcasted operation, where the scalar is automatically expanded to match the shape of the tensor. This broadcasting allows element-wise arithmetic without the need for explicit reshaping. In particular, such operations occur during the construction of perturbed inputs (e.g., interval bounds), enabling efficient implementation.

M. Training Time of SHIELD

Tab. 19 reports the training time of SHIELD model across different datasets. The durations are shown in the HH:MM:SS format and include standard deviations where available. For Permuted MNIST, Rotated MNIST, Split CIFAR-100, and Split miniImageNet, results are averaged over 5 training runs using the best-performing hyperparameters. For TinyImageNet, the time is reported from a single run with no standard deviation.

N. Interval MixUp Samples

Fig. 10 presents representative examples of samples generated using Interval MixUp. Each row corresponds to a single mixed input (the center of a hypercube), and each column shows perturbations with increasing noise magnitude.

Table 17. Adversarial robustness of SHIELD across multiple datasets in the CIL scenario. We report AA under AutoAttack, PGD, and FGSM attacks, as well as on original samples after learning all tasks.

Dataset	Method	AutoAttack	PGD	FGSM	Original samples
Permuted MNIST	SHIELD	**79.76 ± 0.18**	**97.21 ± 0.32**	**77.76 ± 1.5**	**97.46 ± 0.27**
	HNET	3.96 ± 2.45	89.16 ± 0.92	12.02 ± 0.26	93.97 ± 0.49
Rotated MNIST	SHIELD	**36.9 ± 0.21**	**39.12 ± 0.47**	**12.01 ± 0.28**	**42.32 ± 0.56**
	HNET	7.77 ± 0.07	31.86 ± 0.75	3 ± 0.36	38.64 ± 0.8
Split CIFAR-100	SHIELD	**15.17 ± 0.23**	13.08 ± 0.43	9.8 ± 0.47	**23.11 ± 0.44**
	HNET	10.57 ± 0.21	**13.47 ± 0.5**	**11.91 ± 0.45**	21.15 ± 0.69
Split miniImageNet	SHIELD	**12.66 ± 0.88**	**9.65 ± 1.79**	**9.53 ± 1.76**	**19.16 ± 1.25**
	HNET	2.06 ± 0.42	4.33 ± 0.76	6.19 ± 0.92	12.84 ± 2.53

Algorithm 1 The pseudocode of SHIELD in CIL setting

Require: Trained task embeddings $\{\boldsymbol{e}_t\}_{t=1}^{T}$, number of tasks T, trained hypernetwork $\mathcal{H}(\cdot;\boldsymbol{\Phi})$ with weights $\boldsymbol{\Phi}$, test sample x, number of classes C_t per task
Ensure: Predicted label $\hat{y}$ and inferred task ID $\hat{t}$
1: **for** $t = 1$ to T **do**
2: $\quad\boldsymbol{\theta}_t \leftarrow \mathcal{H}(\boldsymbol{e}_t;\boldsymbol{\Phi})$
3: $\quad\hat{y}^{(t)} \leftarrow \text{Softmax}(f(x;\boldsymbol{\theta}_t))$
4: $\quad$Compute entropy: $\psi_t \leftarrow -\sum_{j=1}^{C_t} \hat{y}_j^{(t)} \cdot \log(\hat{y}_j^{(t)})$
5: **end for**
6: $\hat{t} \leftarrow \arg\min_t \psi_t$
7: $\hat{y} \leftarrow \arg\max_j \hat{y}_j^{(\hat{t})}$

Table 18. Accuracy results on CIFAR-100 for the None → FGSM → PGD training sequence. Results for SHIELD are averaged over 3 random seeds. Baseline results are extracted from Tab. 4 in [53].

Method	Task 1	Task 2	Task 3
Vanilla	42.01	22.30	17.80
EWC	**48.35**	22.54	16.59
AIR	47.08	27.34	23.04
SHIELD	46.25 ± 0.29	**28.36 ± 0.01**	**27.82 ± 0.08**
Upper bound	45.33	30.23	21.25

Table 19. Comparison of training time of the SHIELD model on considered datasets.

Dataset	SHIELD
Permuted MNIST	00 : 37 : 59 ± 00 : 00 : 02
Rotated MNIST	00 : 36 : 18 ± 00 : 00 : 19
Split CIFAR-100	02 : 10 : 23 ± 00 : 02 : 00
Split miniImageNet	09 : 21 : 05 ± 00 : 12 : 11
TinyImageNet	05 : 16 : 04

The samples were randomly selected from the Split miniImageNet dataset. This visualization highlights how the neighborhood around each mixed point is explored, showing that small perturbations preserve the semantic content of the image, while larger ones introduce gradual variations.

O. Theoretical Analysis of SHIELD

We now present the theoretical analysis of SHIELD together with the proof of the main theorem.

O.1. Robustness Preservation Condition Theorem

Proof. Using the previously introduced notation, and following Definition 3.1, our goal is to show that for any task s and any $(x, y_{\text{true}}) \in \mathcal{D}_s$, the following condition holds:

$$\underline{z}_{y_{\text{true}}}^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) > \max_{j\neq y_{\text{true}}} \overline{z}_j^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}), \tag{32}$$

where $\boldsymbol{h}$ denotes the hypernetwork output change induced by the gradient update of the hypernetwork weights while learning the $(t+1)$-th task. This parameter change affects the classifier outputs. For each classifier logit l_x, we can write:

$$f_{l_x}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) \in \big[\underline{z}_{l_x}^{(K)}(x;\boldsymbol{\theta}_{s,t}) - \Delta_{\max}^{(t)}(\boldsymbol{h}), \overline{z}_{l_x}^{(K)}(x;\boldsymbol{\theta}_{s,t}) + \Delta_{\max}^{(t)}(\boldsymbol{h})\big]. \tag{33}$$

Since each logit changes by at most $\Delta_{\max}^{(t)}(\boldsymbol{h})$, it follows that

$$\underline{z}_{y_{\text{true}}}^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) - \max_{j\neq y_{\text{true}}} \overline{z}_j^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) \tag{34}$$

$$\geq \underline{z}_{y_{\text{true}}}^{(K)}(x;\boldsymbol{\theta}_{s,t}) - \max_{j\neq y_{\text{true}}} \overline{z}_j^{(K)}(x;\boldsymbol{\theta}_{s,t}) - 2\,\Delta_{\max}^{(s,t)}(\boldsymbol{h}). \tag{35}$$

By the assumption stated in Eq. (14), the right-hand side of the inequality above is strictly positive. Therefore, we obtain:

$$\underline{z}_{y_{\text{true}}}^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) > \max_{j\neq y_{\text{true}}} \overline{z}_j^{(K)}(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}), \tag{36}$$

which concludes the proof. □

Algorithm 2 The pseudocode for SHIELD training

Require: Task embeddings $\{\boldsymbol{e}_t\}_{t=1}^{T}$, hypernetwork $\mathcal{H}(\boldsymbol{e}_t;\boldsymbol{\Phi})$, target network $f(\cdot;\mathcal{H}(\boldsymbol{e}_t;\boldsymbol{\Phi}))$, number of tasks T, training data $D_t = \{(x_i, y_i)\}_{i=1}^{N_t}$, regularization weight $\beta > 0$, perturbation value ε, cross-entropy weight $\kappa \in (0, 1)$, number of training steps n, Beta distribution parameter $\alpha \geq 0$

Ensure: Trained parameters $\boldsymbol{\Phi}$ and $\{\boldsymbol{e}_t\}_{t=1}^{T}$

Randomly initialize $\boldsymbol{\Phi}$ and $\{\boldsymbol{e}_t\}_{t=1}^{T}$
for $t = 1$ to T **do**
  if $t > 1$ **then**
    Freeze $\boldsymbol{e}_{t-1}$
    for $j = 1$ to $t-1$ **do**
      $\boldsymbol{\theta}_j \leftarrow \mathcal{H}(\boldsymbol{e}_j;\boldsymbol{\Phi})$
    end for
  end if
  $\varepsilon' \leftarrow 0$
  $\kappa' \leftarrow 1$
  for step $= 1$ to n **do**
    $\boldsymbol{\theta}_t \leftarrow \mathcal{H}(\boldsymbol{e}_t;\boldsymbol{\Phi})$
    Sample minibatch $\{(x_i, y_i)\}_{i=1}^{B}$ from D_t
    Sample $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$
    $\varepsilon_{\text{IM}} \leftarrow |2\lambda - 1| \cdot \varepsilon'$
    Sample $x_i \neq x_j$ from minibatch
    $\tilde{x} \leftarrow \lambda \cdot x_i + (1-\lambda) \cdot x_j$
    $[\hat{y}, \bar{y}] \leftarrow f([\tilde{x} - \varepsilon_{\text{IM}}, \tilde{x} + \varepsilon_{\text{IM}}]; \boldsymbol{\theta}_t)$
    $\tilde{y} \leftarrow f(\tilde{x}; \boldsymbol{\theta}_t)$
    if $t = 1$ **then**
      $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{IMixUp}}$ {From Eq. (17), using ε' and κ'}
    else
      $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{IMixUp}} + \beta \cdot \mathcal{L}_{\text{out}}$ {From Eq. (11), using ε' and κ'}
    end if
    Update $\boldsymbol{\Phi}$ and $\boldsymbol{e}_t$ using gradient descent
    if step $\leq \lfloor \frac{n}{2} \rfloor$ **then**
      $\varepsilon' \leftarrow \varepsilon \cdot \frac{2 \cdot \text{step}}{n}$
      $\kappa' \leftarrow \max\{\frac{1}{2}, 1 - \frac{\text{step}}{2n}\}$
    end if
  end for
  Store $\boldsymbol{e}_t$
end for

O.2. Justification via Empirical Validation of the Theoretical Condition

We now provide an empirical analysis to justify the practical efficacy of our method with respect to the theoretical conditions established in Theorem 3.1.

The theorem defines a strict, per-sample sufficient condition for preserving certified robustness: a sample (x, y_{true}) that is certifiably robust at time t (i.e., $M(x;\boldsymbol{\theta}_{s,t}) > 0$) will remain so at time $t+1$ if its margin is large enough to absorb the logit change caused by the parameter update $\boldsymbol{h}$:

$$M(x;\boldsymbol{\theta}_{s,t}) > 2\,\Delta^{(s,t)}(x,\boldsymbol{h}), \tag{37}$$

where $\Delta^{(s,t)}(x,\boldsymbol{h}) = \|f(x;\boldsymbol{\theta}_{s,t}+\boldsymbol{h}) - f(x;\boldsymbol{\theta}_{s,t})\|_\infty$.

The Practical Proxy Argument

Explicitly enforcing this theoretical constraint for all samples at every step is computationally prohibitive in a data-free continual learning setting. We argue that the intrinsic regularization properties of our hypernetwork architecture serve as an effective, automatic, and computationally efficient proxy for this hard constraint.

Our hypernetwork regularization (see Eq. (10)) restricts changes in the hypernetwork weights ($\boldsymbol{\Phi}$). Since the output classifier weights ($\boldsymbol{\theta}_{s,t}$) are a smooth and continuous function of $\boldsymbol{\Phi}$, this regularization implicitly limits the magnitude of the resulting parameter update $\boldsymbol{h}$. Consequently, it constrains the logit change $\Delta^{(s,t)}(x,\boldsymbol{h})$.

Empirical Validation

To validate this claim, Tab. 20 reports the percentage of samples that satisfy the theoretical guarantee; i.e., samples that were initially robust (at time t) and for which the inequality $M(x;\boldsymbol{\theta}_{s,t}) - 2\,\Delta^{(s,t)}(x,\boldsymbol{h}) > 0$ holds after all tasks are completed. This directly measures how well our implicit regularization fulfills the conditions of the theorem.

The results are strong and consistent across all benchmarks. On Permuted MNIST, 97.70% of samples satisfy the theoretical guarantee; on Rotated MNIST, 95.63%; and on Split CIFAR-100, 99.00%. These high rates indicate that our implicit regularization effectively limits the magnitude of the logit change $\Delta^{(s,t)}(x,\boldsymbol{h})$, such that, for the vast majority of inputs, this change is absorbed by the margin $M(x;\boldsymbol{\theta}_{s,t})$, thereby preserving the theoretical guarantee in practice.

In addition, Fig. 11 provides a more detailed empirical verification of the underlying assumption. For each previous task s, we compute $\Delta_{\text{max}}^{(s,T)}(x;\boldsymbol{h})$, where T denotes the final task and $\boldsymbol{h} = \boldsymbol{\theta}_{s,T} - \boldsymbol{\theta}_{s,s}$. Figure 11 shows histograms (log-scale on the x-axis) of the ratio $\Delta_{\text{max}}^{(s,T)}(x;\boldsymbol{h}) / \left(\frac{1}{2} M(x, y_{\text{true}};\boldsymbol{\theta}_{s,T})\right)$ for Permuted MNIST and Split CIFAR-100. The dashed red line indicates the critical threshold at 1: values exceeding this threshold violate Eq. (14), meaning that the parameter drift exceeds half of the certified margin and robustness is no longer guaranteed. Across both benchmarks, the mass of the distributions lies well below this threshold, indicating that the sufficient condition holds for nearly all IBP-certified samples, with a substantial margin. Therefore, robustness is preserved in practice.

Figure 10. Each row corresponds to a single Interval MixUp-generated sample (hypercube center), and each column shows perturbations with increasing noise magnitude. This visualization demonstrates how added noise affects interpolated inputs, preserving semantic structure under small perturbations and gradually introducing variability.

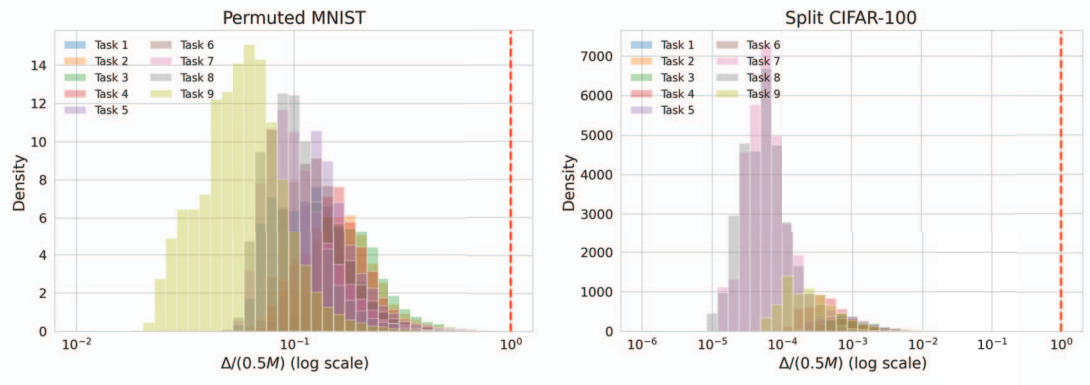


Figure 11. Empirical verification of Eq. (14) from Theorem 3.1. The dashed red line marks the threshold 1. Left: Permuted MNIST. Right: Split CIFAR-100.

In conclusion, our method maintains high certified robustness across tasks. The empirical analysis shows that knowledge of certified robustness is effectively transferred between tasks via the hypernetwork's regularization mechanism. This provides a practical solution that satisfies the necessary condition for robustness preservation without incurring the computational cost of explicitly enforcing the hard theoretical constraint derived from Theorem 3.1.

Table 20. Percentage of samples that remain robustly classified after learning the final task (Theorem 3.1). Higher values indicate better preservation of robustness. Results are averaged over 5 seeds.

Task ID	Permuted MNIST	Rotated MNIST	Split CIFAR-100
1	97.67 ± 0.42	94.49 ± 0.53	100.00 ± 0.00
2	97.09 ± 0.47	95.26 ± 0.87	95.21 ± 4.10
3	96.83 ± 0.50	94.77 ± 1.07	100.00 ± 0.00
4	97.38 ± 0.35	94.70 ± 1.53	99.04 ± 1.92
5	97.23 ± 0.47	97.07 ± 0.78	99.00 ± 1.39
6	98.02 ± 0.45	91.95 ± 4.03	98.95 ± 2.10
7	97.84 ± 0.34	95.76 ± 0.63	99.00 ± 1.17
8	98.55 ± 0.23	98.34 ± 0.14	99.76 ± 0.48
9	98.73 ± 0.59	98.33 ± 0.22	100.00 ± 0.00
Average	97.70 ± 0.11	95.63 ± 0.49	99.00 ± 0.59