

## Appendix Organization

Section A describe the dataset classes and split details. Section B presents the implementation details. Section C presents the additional results details, including comparisons to continual meta learning, 1-shot learning results, effect of domain ordering, effect of hyperparameters, etc. Section D presents additional discussion about the connection with the interpretation of proposed method. Section E presents the theorem proof for the theorem in the main text. Section F presents the additional related work comparisons between incremental few-shot learning (IFSL) and SDML. Also, we present meta testing algorithms.

## A. Dataset Details

**Miniimagenet** [70] A dataset selected from ImageNet with 100 different classes, each with 600 images. All images are the same size of  $84 \times 84$ . We adopt the same splits of [53] with meta train/validation/test splits being 64/16/20 classes.

**CIFARFS** [10] A dataset randomly sampled from CIFAR-100. The meta train/validation/test splits are of 64/16/20 classes respectively. We follow the split of [10].

**Omniglot** [35] An image dataset of 1623 handwritten characters from 50 different alphabets, with 20 examples per class. We follow the same setup and data split used in [70].

**CUB** [77] A dataset for fine-grained classification on 200 different bird species. The meta train/validation/test splits are of 100/50/50 classes respectively. We follow the same split of [16].

**AIRCRAFT** [45] A dataset for aircraft image classification consisting of 102 categories, with 100 images per class. The dataset is split into 70/15/15 classes for meta- training/validation/test. We follow the split in [72].

**Fungi** [62] A large dataset for Fungi classification. It consists of approximately 100K images with nearly 1,500 wild mushrooms species. We only consider the classes containing more than 15 images to form few shot tasks. The dataset is randomly split into 817/175/175 classes for meta- training/validation/test.

**Quickdraw** [31] A large dataset consisting of 50 million black-and-white drawings with 345 categories. This dataset is provided by the players of the game Quick, Draw! We follow the setup of [37] and split the dataset into 241/52/52 classes for meta- training/validation/test.

**Plantae** [28] A large dataset consisting of approximately 100K images with 200 randomly selected plant species. We follow the split of [66] and split the dataset into 100/50/50 classes for meta- training/validation/test.

**Necessities** *Necessities* Logo images from the large-scale publicly available dataset Logo-2K+ [73]. The dataset is randomly split into 100/41/41 classes for meta- training/validation/test.

**Electronic** *Electronic* Logo images from the large-scale publicly available dataset Logo-2K+ [73]. The

dataset is randomly split into 111/40/40 classes for meta-training/validation/test.

## B. Implementation details

### B.1. Memory buffer update

Reservoir sampling (RS) for SDML is operated as the following descriptions. Initially, first  $k$  tasks in the task stream will be stored in memory as long as the memory is not full. When the memory buffer is full, and the task  $i$  arrives, RS then generates a random number  $j$  between 1 and  $i$ . If  $j$  is at most  $k$ , then the current task  $i$  is selected to be stored in memory and replaces the task currently in the  $j^{th}$  position in the reservoir. Otherwise, the task  $i$  is discarded. A total of  $k = 20$  tasks from previous domains are maintained in memory.

### B.2. Detailed Implementation

**Data Preprocessing** . All images are resized into  $84 \times 84$ . for one channel image, we repeat the channel to be 3 so that all images have 3 channels.

For PNet-based [65] baselines, we use a four-layer CNN with 64 filters of kernel size being 3 as a domain-shared feature extractor for all the domains. One layer of CNN with 64 filters is appended to the last layer for domain-specific learning when a new domain arrives for the domain-aware case or when a new domain is detected for the domain-agnostic case. For ANIL-based [51] baselines, following [7], we use a three-layer CNN with 48 filters as a domain-shared feature extractor, and one convolutional layer with 48 filters and one fully-connected layer for domain-specific learning for both domain-aware and domain-agnostic cases. Similar architecture is commonly used in existing meta-learning literature. The memory buffer maintains 20 tasks by reservoir sampling. We sample 10K tasks from each dataset and thus 100K tasks in total. We perform experiments on different domain orderings, with the default ordering being Quickdraw, Aircraft, CUB, MiniImagenet, Omniglot, Plantae, Electronic, CIFARFS, Fungi, and Necessities. For the domain-aware setting, we only adopt learning rate adaptation and online adaptive freeze. For the domain-agnostic setting, a domain shift detection component is additionally adopted. We do not use any pre-trained feature extractor here since the agent in SDML is learning on the fly (tasks sequentially arrive). Thus, not all training data are available beforehand and pre-trained feature extractor is not suitable here, similar to the recent works of [1, 22]. Each experiment is conducted over five runs.

The number of tasks sampled from each dataset ideally should be between 6K and 12K to work well in practice. The hyper learning rate  $\eta$  is set to  $1e-4$  for PNet and  $5e-5$  for ANIL. The gradients on all the learnable learning rates are clipped to  $[-10, 10]$  for both PNet and ANIL. This gradient

clipping is used for stabilizing the training processing and avoid learning rate change too much. Each CNN layer is evenly divided into 4 learning blocks (each block consists of equal number of filters and each block is associated with one learnable learning rate). A total of 20 tasks from previous domains are maintained in memory by reservoir sampling.

**BOCPD implementation** . We use public implementation of multivariate BOCPD [36] adapted to our problem.

### B.2.1 domain-aware setting details

The learning rates adaptation mechanism is applicable at different levels of resolution. In this work, the mechanism functions on the block of filters, with each block is associated with one learnable learning rate. It is straightforward to extend it at layer level or for every single filter. We randomly sample 750 evaluation tasks from each domain for meta testing.

For online adaptive freeze mechanism, the online average ELBO (proposed in Algorithm 3) is evaluated every 30 iterations. The network is temporally freezed if it does not improve for 150 iterations (5 time interval).

For ANIL and Prototypical Network, we adopt the following hyperparameter settings.

- **ANIL** [51] The meta batch size (number of training tasks at each iteration) for each domain is 2. Meta training is performed for a total of 50000 iterations, with 5000 iterations for each domain. The number of training data for each class are set to 1 and 5 respectively corresponding to the 1-shot and 5-shot setting, and the number of query (testing) data is 10 for both cases. During meta training, the inner loop is updated for 5 steps. The inner loop learning rate is set to 1e-2 and the outer loop learning rate is 1e-3. The meta training loss is optimized by Adam [32].
- **Prototypical Network** [65] We maintained a meta batch size of 2 for each domain at each iteration, with a total number of 50000 meta training iterations. The number of meta training iterations for each domain is 5000. The number of query (testing) data is 10 for both 1-shot and 5-shot learning settings. The meta training loss is optimized by Adam [32], with learning rate set to 1e-3.

### B.2.2 domain-agnostic setting details

Here the training setting is mostly the same as the domain aware training details introduced above for both ANIL-based and PNet-based approaches, other than domain identity is unavailable during training and testing. When generating the latent space, the best performance is achieved with context size  $m$  (for constructing the distance metric vector  $\mathbf{d}_t$  in Section 4.3) set to 7. The smoothing factor  $\alpha$  is set to be 0.5 for

weighing the relative importance of current task embedding and past moving average.

### B.3. CL baselines details

For comparison baselines, we adopt the following hyperparameter settings

- **HAT settings** [63] This baseline is adapted from the authors' implementation<sup>1</sup>. The output  $\mathbf{h}_l$  of layer  $l$  performed an element-wise multiplication:  $\mathbf{h}'_l = \mathbf{a}_l^t \odot \mathbf{h}_l$ , where  $\mathbf{a}_l^t$  is the annealed version of the single layer gated domain embedding  $\mathbf{e}_l^t$ , defined as

$$\mathbf{a}_l^t = \sigma(\mathbf{se}_l^t) \quad (5)$$

The stability parameter or scaling parameter  $s$  is annealed according to [63]:

$$s = \frac{1}{s_{max}} + (s_{max} - \frac{1}{s_{max}}) \frac{b-1}{B-1}, \quad (6)$$

where  $s_{max} \in \{100, 200, 400\}$ .  $b \in \{1, \dots, B\}$  is batch index and  $B$  is the total number of batches. The best performing results are selected as baseline for comparison.

- **EWC settings** [33] The weight penalty  $\lambda$  for controlling magnitude of change in parameters ranges in  $\{5, 1 \times 10^1, 5 \times 10^1, 1 \times 10^2, 5 \times 10^2, 1 \times 10^3, 5 \times 10^3\}$  and we select the best model as our comparison baseline.
- **A-GEM settings** [14] We adopt the same settings as the publicly available repository provided by the author<sup>2</sup>.
- **MER settings** [58] The implementation is based on the publicly available repository<sup>3</sup>. The within batch meta-learning rate  $\beta \in \{0.01, 0.03, 0.05\}$ , across batch meta-learning rate  $\gamma \in \{0.3, 0.5, 1.0\}$ . The best performing model is chosen as comparison baseline.
- **UCB settings** [18] Following [18], we scale the learning rate  $\mu$  and  $\rho$  for each parameter proportional to its importance  $\Omega$ , which is measured by its variance, to reduce changes in important parameters. The implementation is adapted from the public repository provided by the authors<sup>4</sup>. We used from 5 to 10 Monte Carlo samples respectively at each iteration. We select the best performing model as our baseline.
- **ER settings** [15] We use reservoir sampling to maintain a fixed memory buffer to jointly train with current domain training tasks.

Our code is provided in the supplementary material for reproduction purposes. It is developed on top of the meta learning framework introduced in Torchmeta [17].

<sup>1</sup><https://github.com/joansj/hat>

<sup>2</sup><https://github.com/GMvandeVen/continual-learning>

<sup>3</sup><https://github.com/matttriemer/MER>

<sup>4</sup><https://github.com/SaynaEbrahimi/UCB>

Table 6. Domain-aware SDML results with domain order 1

Algorithm	ACC	5-Way 1-Shot
		BWT
PNet-Sequential	$32.34 \pm 0.46$	$-23.54 \pm 0.43$
PNet-EWC	$35.92 \pm 0.27$	$-14.12 \pm 0.29$
PNet-HAT	$36.52 \pm 0.16$	$-14.70 \pm 0.15$
PNet-UCB	$35.58 \pm 0.18$	$-14.28 \pm 0.23$
PNet-A-GEM	$34.86 \pm 0.25$	$-20.85 \pm 0.21$
PNet-RS	$34.50 \pm 0.33$	$-21.85 \pm 0.12$
PNet-MER	$33.18 \pm 0.15$	$-17.54 \pm 0.14$
PNet-DEGCL	$35.15 \pm 0.37$	$-13.77 \pm 0.32$
PNet-GPM	$34.98 \pm 0.32$	$-14.65 \pm 0.38$
Ours	<b><math>40.23 \pm 0.32</math></b>	<b><math>-10.72 \pm 0.19</math></b>
Joint-training	$50.23 \pm 0.21$	N/A

Table 7. Domain-aware SDML results with domain order 1

Algorithm	ACC	5-Way 1-Shot
		BWT
ANIL-Sequential	$35.71 \pm 0.36$	$-16.91 \pm 0.45$
ANIL-EWC	$34.90 \pm 0.47$	$-18.44 \pm 0.52$
ANIL-HAT	$34.86 \pm 0.17$	$-19.14 \pm 0.15$
ANIL-UCB	$38.29 \pm 0.28$	$-18.37 \pm 0.31$
ANIL-A-GEM	$35.97 \pm 0.22$	$-17.57 \pm 0.27$
ANIL-RS	$36.28 \pm 0.15$	$-16.82 \pm 0.15$
ANIL-MER	$38.61 \pm 0.23$	$-13.89 \pm 0.12$
ANIL-DEGCL	$37.58 \pm 0.35$	$-15.26 \pm 0.29$
ANIL-GPM	$37.19 \pm 0.31$	$-15.73 \pm 0.35$
Ours	<b><math>40.82 \pm 0.29</math></b>	<b><math>-10.37 \pm 0.14</math></b>
Joint-training	$52.93 \pm 0.12$	N/A

## C. Supplementary results

We considered three different representative scenarios of domain ordering in our experiment:

**Order 1** Quickdraw, Aircraft, CUB, MiniImagenet, Omniglot, Plantae, Electronic, CIFARFS, Fungi and Necessities (default ordering in main text).

**Order 2** Necessities, Fungi, Omniglot, Plantae, Aircraft, MiniImagenet, CIFARFS, CUB, Quickdraw, Electronic.

**Order 3** Omniglot, Aircraft, Necessities, Fungi, Plantae, CUB, Quickdraw, MiniImagenet, CIFARFS, Electronic.

### C.1. 5-way 1-shot results with domain order 1

Table 6 and 7 show results on 5-way 1-shot learning with domain order 1 (default order in main text).

### C.2. Accuracy of Domain shift detection

We perform experiments with 20 runs for domain shift detection accuracy evaluation, each consists of 9 changepoints since each run consists of 10 domains and there are 9 domain shift happens along the domain sequence. The detection accuracy is 175/180, only miss the domain shift changepoint for 5 times. The performance meets our expectation.

Table 8. Independence test for the context before and after the domain shift for first training on Quickdraw and then Aircraft.

Detection space	5-Way 5-Shot	
	MIC	TIC
$e_t$	0.2546	0.1435
$d_t$	<b>0.1633</b>	<b>0.0723</b>

Table 9. Independence test for the context before and after the domain shift for first training on Aircraft and then CUB.

Detection space	5-Way 5-Shot	
	MIC	TIC
$e_t$	0.2438	0.1366
$d_t$	<b>0.1687</b>	<b>0.0718</b>

## C.3. Independence of Latent Space

In section 4.3, we construct a special latent space for domain shift detection. Since BOCPD assumes the contexts before and after the changepoints are independent, we analyze the correlation before and after the domain shift with maximal information coefficient (MIC) [56] and total information coefficient (TIC) [57]. These two metrics can be used for testing nonlinear dependency between two random variables and can be viewed as smoothing of mutual information. Thus, both metrics are non-negative. Since mutual information between two variables are zero when the two random variables are independent. Thus, if the two metrics are more closer to zero, they are more independent. This is to further verify the contexts before and after the domain shift are more independent for our proposed latent space. More precisely, we verify whether latent space  $d_t$  are more independent than  $e_t$ . We use context size  $m = 2$  for our method. We use 500 (latent) task embeddings before the domain shift and another 500 (latent) task embeddings after the domain shift for evaluating the independence. Table 8, 9 and 10 shows the evaluation results, indicating that the constructed latent space  $d_t$  are more independent than  $e_t$ . This can be explained from an information-theoretic perspective, using  $d_t$  compresses the task information in a low-dimensional space by reducing the image semantic information in raw task embeddings  $e_t$  in high dimensional space and only retain the information useful for domain shift detection. This can reduce the overlapping mutual information before and after the domain shift.

### C.4. Effect of domain ordering

We perform additional experiments on the 2nd and 3rd domain ordering listed above, with results specified below.

**Domain order 2 (domain aware)** Table 11, 12, 13, 14 show the results of 5-way 5-shot learning and 5-way 1-shot

Table 10. Independence test for the context before and after the domain shift for first training on CIFARFS and then Fungi.

Detection space	5-Way 5-Shot	
	MIC	TIC
$e_t$	0.2649	0.1504
$d_t$	<b>0.1614</b>	<b>0.0710</b>

Table 11. PNet-based methods 5-way 5-shot learning for domain-aware SDML results with domain order 2

Algorithm	ACC	5-Way 5-Shot
		BWT
PNet-Sequential	$48.15 \pm 0.25$	$-20.72 \pm 0.32$
PNet-EWC	$50.29 \pm 0.28$	$-9.30 \pm 0.24$
PNet-HAT	$48.60 \pm 0.18$	$-18.32 \pm 0.12$
PNet-UCB	$49.51 \pm 0.20$	$-17.53 \pm 0.19$
PNet-A-GEM	$46.32 \pm 0.27$	$-20.81 \pm 0.21$
PNet-RS	$52.09 \pm 0.21$	$-17.00 \pm 0.24$
PNet-MER	$50.10 \pm 0.15$	$-14.81 \pm 0.16$
PNet-DEGCL	$53.27 \pm 0.34$	$-13.25 \pm 0.41$
PNet-PGM	$51.16 \pm 0.38$	$-15.32 \pm 0.49$
Ours	<b><math>57.21 \pm 0.34</math></b>	<b><math>-7.62 \pm 0.25</math></b>
Joint-training	$66.32 \pm 0.18$	N/A

Table 12. PNet-based methods 5-way 1-shot learning for Domain-aware SDML results with domain order 2

Algorithm	ACC	5-Way 1-Shot
		BWT
PNet-Sequential	$36.26 \pm 0.39$	$-19.95 \pm 0.50$
PNet-EWC	$34.52 \pm 0.18$	$-10.18 \pm 0.16$
PNet-HAT	$37.09 \pm 0.16$	$-16.07 \pm 0.11$
PNet-UCB	$37.95 \pm 0.22$	$-15.68 \pm 0.20$
PNet-A-GEM	$33.96 \pm 0.25$	$-20.19 \pm 0.21$
PNet-RS	$38.40 \pm 0.51$	$-17.61 \pm 0.50$
PNet-MER	$37.36 \pm 0.21$	$-16.43 \pm 0.19$
PNet-DEGCL	$38.91 \pm 0.36$	$-15.49 \pm 0.56$
PNet-GPM	$37.72 \pm 0.43$	$-16.87 \pm 0.40$
Ours	<b><math>41.76 \pm 0.36</math></b>	<b><math>-7.23 \pm 0.12</math></b>
Joint-training	$50.23 \pm 0.21$	N/A

learning results for both PNet-based and ANIL-based models.

**Domain order 2 (domain agnostic)** Table 15 and 16 show the results of 5-way 5-shot learning and 5-way 1-shot learning results for both PNet-based and ANIL-based models under domain-agnostic setting.

**Domain order 3** Table 17 shows 5-way 5-shot learning results for domain order 3.

Table 13. ANIL-based methods 5-way 5-shot learning for domain-aware SDML results with domain order 2

Algorithm	ACC	5-Way 5-Shot
		BWT
ANIL-Sequential	$47.24 \pm 0.30$	$-20.54 \pm 0.26$
ANIL-EWC	$46.41 \pm 0.31$	$-21.51 \pm 0.31$
ANIL-HAT	$48.79 \pm 0.18$	$-21.24 \pm 0.25$
ANIL-UCB	$49.29 \pm 0.23$	$-20.17 \pm 0.20$
ANIL-A-GEM	$45.92 \pm 0.21$	$-22.82 \pm 0.23$
ANIL-RS	$48.69 \pm 0.12$	$-21.03 \pm 0.19$
ANIL-MER	$49.72 \pm 0.17$	$-19.48 \pm 0.19$
Ours	<b><math>52.95 \pm 0.29</math></b>	<b><math>-13.40 \pm 0.42</math></b>
Joint-training	$68.16 \pm 0.11$	N/A

Table 14. ANIL-based methods 5-way 1-shot learning for domain-aware SDML results with domain order 2

Algorithm	ACC	5-Way 1-Shot
		BWT
ANIL-Sequential	$36.69 \pm 0.15$	$-16.20 \pm 0.21$
ANIL-EWC	$36.77 \pm 0.22$	$-17.06 \pm 0.27$
ANIL-HAT	$37.96 \pm 0.18$	$-15.42 \pm 0.21$
ANIL-UCB	$37.28 \pm 0.34$	$-15.03 \pm 0.25$
ANIL-A-GEM	$37.75 \pm 0.17$	$-15.22 \pm 0.16$
ANIL-RS	$37.91 \pm 0.29$	$-15.68 \pm 0.23$
ANIL-MER	$38.81 \pm 0.23$	$-13.62 \pm 0.25$
Ours	<b><math>40.97 \pm 0.35</math></b>	<b><math>-10.23 \pm 0.26</math></b>
Joint-training	$52.93 \pm 0.12$	N/A

Table 15. ANIL-based methods for domain-agnostic SDML results with domain order 2

Algorithm	5-Way 1-Shot	5-Way 5-Shot
	ACC	ACC
ANIL-Sequential	$36.69 \pm 0.15$	$47.24 \pm 0.30$
ANIL-RS	$37.91 \pm 0.29$	$48.69 \pm 0.12$
ANIL-AGEM	$37.75 \pm 0.17$	$45.92 \pm 0.21$
ANIL-GSS	$38.06 \pm 0.23$	$48.37 \pm 0.28$
Ours	<b><math>40.53 \pm 0.32</math></b>	<b><math>52.18 \pm 0.25</math></b>
Joint-training	$52.93 \pm 0.12$	$68.16 \pm 0.11$

Table 16. PNet-based methods for domain-agnostic SDML results with domain order 2

Algorithm	5-Way 1-Shot	5-Way 5-Shot
	ACC	ACC
PNet-Sequential	$36.26 \pm 0.39$	$48.15 \pm 0.25$
PNet-RS	$38.40 \pm 0.51$	$52.09 \pm 0.21$
PNet-AGEM	$33.96 \pm 0.25$	$46.32 \pm 0.27$
PNet-GSS	$37.91 \pm 0.32$	$50.98 \pm 0.34$
Ours	<b><math>41.18 \pm 0.23</math></b>	<b><math>56.73 \pm 0.30</math></b>
Joint-training	$50.23 \pm 0.21$	$66.32 \pm 0.18$

Table 17. PNet-based methods 5-way 5-shot learning for domain-aware SDML results with domain order 3

Algorithm	ACC	5-Way 5-Shot BWT
PNet-Sequential	49.96 $\pm$ 0.22	-19.15 $\pm$ 0.29
PNet-EWC	46.57 $\pm$ 0.24	-13.83 $\pm$ 0.22
PNet-HAT	51.72 $\pm$ 0.27	-16.97 $\pm$ 0.35
PNet-UCB	51.58 $\pm$ 0.36	-15.76 $\pm$ 0.23
PNet-A-GEM	50.79 $\pm$ 0.31	-17.37 $\pm$ 0.25
PNet-RS	52.18 $\pm$ 0.23	-16.35 $\pm$ 0.31
PNet-MER	51.30 $\pm$ 0.28	-15.16 $\pm$ 0.20
Ours	<b>56.52 <math>\pm</math> 0.25</b>	<b>-8.69 <math>\pm</math> 0.29</b>
Joint-training	66.32 $\pm$ 0.18	N/A

Table 18. Effect of number of memory tasks. Top, memory with 10 tasks; Bottom, memory with 20 tasks

Algorithm	ACC	5-Way 5-Shot BWT
PNet-A-GEM	47.17 $\pm$ 0.37	-20.81 $\pm$ 0.22
PNet-RS	48.22 $\pm$ 0.27	-19.89 $\pm$ 0.24
PNet-MER	48.37 $\pm$ 0.22	-17.05 $\pm$ 0.19
Ours	54.65 $\pm$ 0.17	-12.85 $\pm$ 0.30
PNet-A-GEM	49.21 $\pm$ 0.31	-20.01 $\pm$ 0.39
PNet-RS	49.56 $\pm$ 0.18	-18.87 $\pm$ 0.19
PNet-MER	50.38 $\pm$ 0.24	-15.10 $\pm$ 0.24
Ours	<b>55.28 <math>\pm</math> 0.19</b>	<b>-11.15 <math>\pm</math> 0.27</b>
Joint-training	66.32 $\pm$ 0.18	N/A

**Effect of memory size** we explore the change on model performance when the number of tasks that memory stores differs. Table 18 show results with memory storing 10 and 20 tasks respectively. It matches with our expectation that with smaller number of tasks from previous domains in memory, performance of all memory-based baselines drop significantly, while our method maintains relatively stable performance. This might be due to (1) with a smaller memory size, all memory-based baselines either overfit on the small number of memory tasks; (2) or are impacted by unstable gradient updates.

**Running time.** We further report the running time of baselines and our method with 200 iterations in Table 19 to demonstrate the running efficiency of different methods. Our method is slightly slower than PNet-AGEM, but faster than the other two baselines.

Table 19. Running time of different methods.

Algorithm	PNet-AGEM	PNet-UCB	PNet-MER	Ours
Time (s)	<b>58</b>	357	405	73

Table 20. effect of context size

Algorithm	5-Way 1-Shot ACC	5-Way 5-Shot ACC
PNet-Ours ( $m=10$ )	38.29 $\pm$ 0.17	52.45 $\pm$ 0.12
PNet-Ours ( $m=7$ )	38.17 $\pm$ 0.21	54.67 $\pm$ 0.20
PNet-Ours ( $m=5$ )	38.36 $\pm$ 0.19	54.26 $\pm$ 0.29
PNet-Ours ( $m=1$ )	36.82 $\pm$ 0.16	53.41 $\pm$ 0.25

Table 21. effect of moving average smoothing factor  $\alpha$

Algorithm	5-Way 1-Shot ACC	5-Way 5-Shot ACC
PNet-Ours ( $\alpha=0.1$ )	38.09 $\pm$ 0.37	54.02 $\pm$ 0.29
PNet-Ours ( $\alpha=0.3$ )	38.68 $\pm$ 0.32	54.29 $\pm$ 0.26
PNet-Ours ( $\alpha=0.5$ )	38.17 $\pm$ 0.21	54.67 $\pm$ 0.20
PNet-Ours ( $\alpha=0.7$ )	38.03 $\pm$ 0.23	54.49 $\pm$ 0.25

Table 22. effect of hyper learning rate  $\eta$

Algorithm	ACC	5-Way 5-Shot BWT
PNet-Ours ( $\eta=1e-4$ )	55.28 $\pm$ 0.19	-11.15 $\pm$ 0.27
PNet-Ours ( $\eta=1e-5$ )	54.97 $\pm$ 0.28	-11.86 $\pm$ 0.35
PNet-Ours ( $\eta=1e-6$ )	55.65 $\pm$ 0.23	-10.73 $\pm$ 0.32

## C.5. Effect of hyperparameters

**context size** For domain-agnostic case, the proposed method calculate the distance of current task embedding to the moving average of task embeddings from previous time steps. Involving more steps from past means more context, but at the same time the distribution dimension will also be higher. Table 20 shows the comparison among different context size (value of  $m$  when calculating the distance metric vector  $d_t$  to construct the latent space in Section 4.3). We compare model performance with different choice  $m \in [1, 5, 7, 10]$  respectively. Interestingly, the performance drops with context size larger than 7. This shows when distribution dimension is higher, it is relatively inaccurate to estimate the changepoint compared to lower dimensional case.

**moving average smoothing factor** Table 21 shows the results on the effect of moving average smoothing factor  $\alpha$ , indicating the performance of our method is not sensitive to this parameter.

**hyper learning rate** Table 22 shows the results on the effect of hyper learning rate  $\eta$  to model performance. It provides the range of  $\eta$  that is effective for forgetting mitigation.



Table 23. effect of number of blocks of filters per layer

Algorithm	ACC	5-Way 5-Shot BWT
PNet-Ours (number blocks = 8)	55.79 ± 0.28	−10.82 ± 0.36
PNet-Ours (number blocks = 4)	55.28 ± 0.19	−11.15 ± 0.27
PNet-Ours (number blocks = 2)	54.95 ± 0.22	−11.68 ± 0.31

**number of blocks per layer** We compare different number of blocks, i.e., 2, 4 and 8 blocks per layer to investigate on its effect, where each block consists of equal number of CNN filters and each block is associated with one learnable learning rate. Table 23 shows the results. A reasonable choice on the number of blocks per layer ranges from 2 to 8.

Table 24. Effectiveness of online adaptive freeze

Algorithm	ACC	5-way 5-shot BWT
PNet-Ours (w/o OAF)	52.92 ± 0.20	−13.27 ± 0.25
PNet-Ours (w/o meta optimizer)	49.36 ± 0.29	−17.86 ± 0.36
PNet-Ours (w/ OAF)	<b>55.28 ± 0.19</b>	<b>−11.15 ± 0.27</b>

**Ablation Study.** To verify the effectiveness of each individual component of our method, we benchmark our approach with and without online adaptive freeze (OAF) and meta optimizer. The results are shown in Table 24. We find that the *meta optimizer alone* outperforms best baselines by 2.1%; the OAF mechanism provides additional 2.3% improvement, which demonstrates its significant effectiveness.

### C.6. Experiments on real-world dataset

To further show the effectiveness of the proposed method on a real-world dataset, we use LSTM with hidden layer size of 128 as the base natural language generation model on the dialogue dataset, MultiWoZ-2.0 dataset [11], which contains six domains (attraction, hotel, restaurant, booking, taxi, and train). We use the meta-learning strategy from [50] to sequentially learn on each domain, similar to [48]. The results are measured by BLEU-4 and slot error rate (SER) [48], which is the ratio of the number of missing and redundant slots in a generated utterance to the total number of ground truth slots in the dialog act. We compare to Sequential, RS [15], AGEM [14], DEGCL [12] and GPM [61] baselines. The domain order is booking, taxi, restaurant, train, attraction, and hotel. We got the following results:

The results are evaluated on SER (the lower the better) and BLEU (the higher the better). ALL-SER and ALL-BLEU metrics evaluate the overall performance. First-SER and First-BLEU metrics evaluate the ability of mitigating forgetting [48]. We can observe that our method outperforms baselines in terms of SER and BLEU metrics. We believe that the performance differences are due to the heterogeneous

Table 25. real world dialogue dataset results (↓ indicates lower is better; ↑ indicates higher is better)

Algorithm	ALL SER ↓	ALL BLEU ↑	First SER ↓	First BLEU ↑
MAML-Sequential	68.594	0.417	149.937	0.287
MAML-RS	9.576	0.634	5.769	0.651
MAML-A-GEM	9.833	0.626	6.021	0.649
MAML-DEGCL	9.936	0.612	6.187	0.658
MAML-GPM	10.781	0.593	6.385	0.619
Ours	<b>8.156</b>	<b>0.662</b>	<b>3.671</b>	<b>0.685</b>

nature of different domains in the dialogue system.

### D. Additional Discussion

Interpretation of the learning rate adaptation,

$$\min_{\theta} \left[ \mathcal{I}_{\theta} = \mathcal{L}_{\theta}(\mathcal{T}_t) + \mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} (\mathcal{L}_{\theta}(\mathcal{T}_j) - \rho \nabla_{\theta}^t \cdot \nabla_{\theta}^j) \right]. \quad (7)$$

We first derive the gradient of  $\mathcal{I}_{\theta}$  is

$$\frac{\partial \mathcal{I}_{\theta}}{\partial \theta} = \nabla_{\theta}^t + \mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} \nabla_{\theta}^j - \rho \frac{\partial (\mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} \nabla_{\theta}^t \cdot \nabla_{\theta}^j)}{\partial \theta} \quad (8)$$

$$\frac{\partial (\mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} \nabla_{\theta}^t \cdot \nabla_{\theta}^j)}{\partial \theta} = \mathcal{H}_{\theta}^t \cdot (\mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} \nabla_{\theta}^j) + (\mathbb{E}_{\mathcal{T}_j \sim \mathcal{M}} \mathcal{H}_{\theta}^j) \cdot \nabla_{\theta}^t \quad (9)$$

The first term is to minimize on the current task, and the second term corresponds to the gradient on memory tasks  $\mathcal{M}$ . where  $\mathcal{H}_{\theta}^j$  is the Hessian matrix

$$\min_{\theta} [\mathcal{F}(\theta) = \mathbb{E}_{\mathcal{T} \in \mathcal{M}} \mathcal{L}_{\theta'}(\mathcal{T})], \text{ where, } \theta' = \theta - \lambda \frac{\partial \mathcal{L}_{\theta}(\mathcal{T}_t)}{\partial \theta}. \quad (10)$$

$$\frac{\partial \mathcal{F}(\theta)}{\partial \lambda} = - \frac{\partial \mathcal{F}(\theta)}{\partial \theta'} \cdot \frac{\partial \mathcal{L}_{\theta}(\mathcal{T}_t)}{\partial \theta}. \quad (11)$$

By Taylor expansion at  $\theta$

$$\frac{\partial \mathcal{F}(\theta)}{\partial \theta'} = \frac{\partial \mathcal{F}(\theta)}{\partial \theta} + \mathcal{H}(\mathcal{F}(\theta))(\nabla) \quad (12)$$

Plug the Equation 12 into the Equation 11, we find that the second order terms coincides with the second term of Equation 9. Thus, the proposed meta optimizer approximately solves the problem 7 by approximating the gradient in Equation 8.

### E. Theorem proof

We provide a theoretical analysis of the domain shift detection algorithm, with a lower bound on the expected

domain shift detection delay shown in Theorem 1. This theorem shows that the detection delay is inverse proportional to domain similarity, i.e., the detection delay is shorter if they are more dissimilar. We put the detailed proof in the Appendix E.

**Theorem 1.** Assume the actual domain change time is  $\omega$ , the detected domain shift time is  $\Omega$ . Assume the false detection rate of domain shift is upper bounded by  $\xi$ . Further, suppose the exponential family distribution for fitting  $\mathbf{d}_i$  before and after  $\omega$  are  $P_{\eta_0}$  and  $P_{\eta_1}$  respectively. For some  $z$  (the number of steps from starting time to  $\omega$ ),  $\limsup_{\xi \rightarrow 0} \frac{z}{|\log \xi|} > \frac{1}{\mathbb{KL}(P_{\eta_0}, P_{\eta_1})}$ . If  $\lim_{\xi \rightarrow \infty} \omega / |\log \xi| = \infty$ , then the expected domain shift detection delay is lower bounded by

$$\mathbb{E}_{\eta_0, \eta_1}(\Omega - \omega)^+ \geq \left\{ \frac{P_{\eta_0}(\Omega \geq \omega)}{\mathbb{KL}(P_{\eta_0}, P_{\eta_1})} + o(1) \right\} |\log \xi|. \quad (13)$$

**Proof of Theorem 1** The theorem follows from [34]. Before proving the above theorem, we first define related notations.

Let  $\mathbf{d}_1, \dots, \mathbf{d}_{\omega-1}$  be independent random variables with a common density function  $P_{\eta_0}(\mathbf{d}_i | \mathbf{d}_{1:i-1})$  and let  $\mathbf{d}_\omega, \mathbf{d}_{\omega+1}, \dots$  be independent with a common density function  $P_{\eta_1}$ . We shall use  $P^{(\omega)}$  to denote such probability measure (with change time  $\omega$ ) and use  $P_0$  to denote the case  $\omega = \infty$  (no change point). We define  $I = \mathbb{KL}(P_{\eta_0}, P_{\eta_1})$  as the KL-divergence between  $P_{\eta_0}$  and  $P_{\eta_1}$ .

define the random variable  $Z_i = \log \frac{P_{\eta_0}(\mathbf{d}_i | \mathbf{d}_{1:i-1})}{P_{\eta_1}(\mathbf{d}_i | \mathbf{d}_{1:i-1})}$

$$C_\delta = \{0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi, \sum_{i=\omega}^{\Omega} Z_i < (\delta^2 - 1) \log \xi\} \quad (14)$$

**Lemma 2.**  $P^{(\omega)}(C_\delta) \leq \exp\{(\delta^2 - 1) \log \xi\} \cdot P_0(0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi) \leq \xi^{\delta^2} (\log \xi)^2$

**proof**

For the chosen  $\omega$  and every  $0 < \delta < 1$ . Let  $\mathcal{F}_n$  be the  $\sigma$ -field generated by  $\mathbf{d}_1, \dots, \mathbf{d}_n$  and let  $P_n^{(\omega)}$  be the restriction of  $P^{(\omega)}$  to  $\mathcal{F}_n$ . Then, [59, 67, 68]

$$\frac{dP_n^{(\omega)}}{dP_{0,n}} = \exp \left( \sum_{i=\omega}^n Z_i \right), \quad \text{for } n \geq \omega$$

and, therefore,

$$\begin{aligned} P^{(\omega)}(C_\delta) &= \int_{C_\delta} \exp \left( \sum_{i=\omega}^{\Omega} Z_i \right) dP_0 \\ &\leq \exp \{ (\delta^2 - 1) \log \xi \} P_0(C_\delta) \end{aligned}$$

noting that  $\sum_{i=\omega}^{\Omega} Z_i < (1 - \delta^2) \log \xi$  on  $C_\delta$ . Because

$$\{\Omega \geq \omega\} \in \mathcal{F}_{\omega-1} \Rightarrow P^{(\omega)}\{\Omega \geq \omega\} = P_0\{\Omega \geq \omega\}$$

it then follows that for all large  $\xi$

$$\begin{aligned} P^{(\omega)}\{C_\delta\} &\leq \exp \{ (\delta^2 - 1) \log \xi \} \\ &\quad \cdot P_0\{0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi\} \\ &\leq \xi^{1-\delta^2} P_0(\Omega < \omega + m \mid \Omega \geq \omega) \\ &\leq \xi^{\delta^2} (\log \xi)^2 \rightarrow 0 \text{ as } \xi \rightarrow 0 \end{aligned}$$

**proof**

$$\text{Suppose } \sup_{w \geq 1} P^{(\omega)} \left\{ \max_{t \leq n} \sum_{i=\omega}^{\omega+t} Z_i \geq I(1 + \delta)n \right\} \rightarrow 0$$

**Lemma 3.**  $\sup_{w \geq 1} P^{(\omega)}\{0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi, \sum_{i=\omega}^{\Omega} Z_i \geq (\delta^2 - 1) \log \xi\} \rightarrow 0$

$$\sup_{w \geq 1} P^{(\omega)} \left\{ \max_{t \leq n} \sum_{i=\omega}^{\Omega} Z_i \geq (1 + \delta)n \right\} \rightarrow 0 \quad (15)$$

**proof**  $\sup_{w \geq 1} P^{(\omega)}\{0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi, \sum_{i=\omega}^{\Omega} Z_i \geq (\delta^2 - 1) \log \xi\} \leq$

$\sup_{w \geq 1} P^{(\omega)} \left\{ \max_{t < (\delta - 1)I^{-1} \log \xi} \sum_{i=\omega}^{\Omega} Z_i \geq (\delta + 1)(\delta - 1)I^{-1} \log \xi \right\} \rightarrow 0$ . The RHS follows from applying the assumption 15.

**Theorem 4.** Assume the actual domain change time is  $\omega$ , the detected domain shift time is  $\Omega$ . Assume the false detection rate of domain shift is upper bounded by  $\xi$ . Further, suppose the exponential family distribution for fitting  $\mathbf{d}_i$  before and after  $\omega$  are  $P_{\eta_0}$  and  $P_{\eta_1}$  respectively. For some  $z$  (the number of steps from starting time to  $\omega$ ),  $\limsup_{\xi \rightarrow 0} \frac{z}{|\log \xi|} > \frac{1}{\mathbb{KL}(P_{\eta_0}, P_{\eta_1})}$ . If  $\lim_{\xi \rightarrow \infty} \omega / |\log \xi| = \infty$ , then the expected domain shift detection delay is lower bounded by

$$\mathbb{E}_{\eta_0, \eta_1}(\Omega - \omega)^+ \geq \{P_{\eta_0}(\Omega \geq \omega) / \mathbb{KL}(P_{\eta_0}, P_{\eta_1}) + o(1)\} |\log \xi| \quad (16)$$

**proof**

By lemma 2 and lemma 3, we have

$$\sup_{w \geq 1} P^{(\omega)}\{0 \leq \Omega - \omega < (\delta - 1)I^{-1} \log \xi\} \rightarrow 0 \quad (17)$$

Which implies that

$$\sup_{w \geq 1} P^{(\omega)}\{\Omega - \omega \geq (\delta - 1)I^{-1} \log \xi\} \rightarrow 1 \quad (18)$$

Therefore,

$$\begin{aligned} \mathbb{E}^{(\omega)}(\Omega - \omega)^+ &\geq (1 - \delta)I^{-1} |\log \xi| P^{(\omega)}\{\Omega - \omega \geq (1 - \delta)I^{-1} \log \xi\} \\ &= (1 - \delta)I^{-1} |\log \xi| (P^{(\omega)}(\Omega \geq \omega) + o(1)) \\ &= (1 - \delta) |\log \xi| (P^{(\omega)}(\Omega \geq \omega) / I + o(1)) \end{aligned}$$

## F. Algorithm

We use Welford’s online algorithm [76] that is much less susceptible to precision loss for calculating the running std  $\sigma$  within a small running window. We use parameters from fixed interval (window) as particle approximation to  $q(\theta)$  for calculating the running std. This can reduce the online ELBO estimation variance.

---

### Algorithm 3 Online ELBO calculation.

---

**Require:**  $C = 0, V = 0, \log P = 0$  (ELBO),  $\bar{\theta} = 0$ , size of time interval  $m$

- 1: **while**  $\theta \neq \emptyset$  **do**
- 2:    $\Delta = \theta - \bar{\theta}$
- 3:    $\bar{\theta} = \bar{\theta} + \Delta/C$
- 4:    $\Delta' = \theta - \bar{\theta}$
- 5:    $V = V + \Delta \cdot \Delta'$
- 6:    $C = C + 1$  (count number of steps)
- 7:   compute likelihood of current tasks and memory tasks  
 $\mathcal{L}_{sum} = -\mathbb{E}_{\mathcal{T}_j \in \mathcal{M}} \mathbb{E}_{q(\theta)} \mathcal{L}_{\theta}(\mathcal{T}_j) - \mathbb{E}_{q(\theta)} \mathcal{L}_{\theta}(\mathcal{T}_t)$
- 8:   compute the running average of  $\mathcal{L}_{sum}$   
 $\log P = \log P + (\mathcal{L}_{sum} - \log P)/C$
- 9:   **if**  $C > m$  **then**
- 10:     **break**
- 11:   **end if**
- 12: **end while**
- 13: running standard deviation  $\sigma = V/C$
- 14: entropy of  $q(\theta)$  is  $H(q(\theta)) = \log(\sigma\sqrt{2\pi e})$
- 15: calculate ELBO,  $\log P = \log P + H(q(\theta))$
- 16: Return  $\log P$

---

Domain-aware and domain-agnostic meta testing algorithm is shown in Algorithm 4 and 5 respectively.

---

### Algorithm 4 Domain-aware meta testing.

---

**Require:** A sequence of testing domain data  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_N$ ; the model parameters for each domain  $\theta^1, \theta^2, \dots, \theta^N$ , where  $\theta^q = \{\theta^S, \theta_q^D\}, \forall q \in \{1, \dots, N\}$ ,  $\theta^S$  is the domain-shared parameters and  $\theta_q^D$  is the domain-specific parameters;  $N$  is the number of training domains .

- 1: **for**  $q = 1$  to  $N$  **do**
- 2:   **for**  $i = 1$  to  $M_N$  **do**
- 3:     Sample testing task  $\mathcal{T}_i = \{\mathcal{T}_i^{tr}, \mathcal{T}_i^{test}\}$  from  $P(\mathcal{D}_q)$ , the distribution over tasks in domain  $\mathcal{D}_q$
- 4:     evaluate the performance of task  $\mathcal{T}_i$  using model  $f_{\theta_q}(\mathcal{T}_i)$
- 5:   **end for**
- 6: **end for**

---



---

### Algorithm 5 Domain-agnostic meta testing.

---

**Require:** A collection of testing tasks  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_N$ ;  $\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_L$ ; the model parameters for each detected domain  $\theta^1, \theta^2, \dots, \theta^L$ , where  $\theta^q = \{\theta^S, \theta_q^D\}, \forall q \in \{1, \dots, L\}$ ,  $\theta^S$  is the domain-shared parameters and  $\theta_q^D$  is the domain-specific parameters;  $L$  is the number of detected domains.

- 1: **for**  $i = 1$  to  $N$  **do**
- 2:   For task  $\mathcal{T}_i = \{\mathcal{T}_i^{tr}, \mathcal{T}_i^{test}\}$ , either  $\mathcal{T}_i^{tr}$  or  $\mathcal{T}_i^{test}$  could be used for calculating task embedding. Assume we use  $\mathcal{T}_i^{tr}$  consisting of  $K$  data examples,  $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k=1}^K$ , they are embedded by  $\mathbf{e}_q = f_{\theta_q}(\{\mathbf{x}^k\}_{k=1}^K), q \in \{1, \dots, L\}$ .
- 3:   calculate the distance of  $d(\mathbf{e}_q, \mathcal{E}_q), \forall q \in \{1, \dots, L\}$
- 4:   infer the domain identity with  $q_o = \underset{q \in \{1, \dots, L\}}{\operatorname{argmin}} d(\mathbf{e}_q, \mathcal{E}_q)$
- 5:   evaluate the performance of task  $\mathcal{T}_i$  with  $f_{\theta_{q_o}}$
- 6: **end for**

---

## G. Related Work

### G.1. Continuous domain adaptation

Continuous domain adaptation [42] is a recent application of continual learning to domain adaptation. A labeled source domain is adapted to the target domain sequentially with the same class labels by aligning source and target domains with Maximum Mean Discrepancy. They assume that the classification task is always the same across different domains during the learning process. However, in practice, at each time  $t$ , the agent may solve a different task, and each domain may have different classes. Our more general SDML naturally considers these factors. In addition, we are not able to align different domains in SDML since previous domains are not available when meta training on the current domain, e.g., due to data privacy. Therefore, these techniques are not applicable in SDML.

### G.2. Difference between IFSL and SDML

Incremental few-shot learning [24, 55, 79] (IFSL) aims to handle new categories with limited resources while preserving knowledge on old categories. They implicitly assume unlimited access to the base categories. However, this would be impractical for many scenarios, such as the medical care system, the data is usually unavailable after use due to privacy issues or the number of tasks are too large to be stored in memory, such as in our case with 100K tasks (episodes). Therefore, this paper, by contrast, we consider the case where previous domains are not available and generalization to unseen categories in previous domains is required.

For incremental few-shot learning, the increasing new classes are assumed to be in the same domain as the base classes, which is significantly different from the changing domains with long domain sequences in SDML.

IFSL focuses on offline learning. However, in SDML, the agent is learning on-the-fly in online fashion. In addition, for SDML, most previous learned tasks are not available



Table 26. Comparison between IFSL and SDML.

Settings	Online/Offline	Multiple/Single domain	Availability of previous tasks
IFSL	Offline	Single	Available
SDML	<b>Online</b>	<b>Multiple</b>	<b>Mostly Unavailable</b>

after learning on current task (there are no base categories). Although we allocate a small memory buffer, the number of tasks in memory buffer is too small, e.g., 20 tasks, so that they are negligible compared to total number of tasks. The domains in the sequence are significantly different. In other words, we work on non-stationary task distributions with online learning fashion. Thus, not all tasks are all available before training so that we cannot pretrain the networks like IFSL. From this aspect, SDML is much more challenging and fundamentally different from IFSL, and the methods in IFSL are not applicable in SDML.

For example, for IFSL, we take the Mini-imagenet as example (the following paragraph is from Xtnet [79]), the dataset is divided into 64 base classes and 36 novel classes. The network backbone is first pre-trained on the 64 base classes by regular supervised training and are fixed afterwards. After that, the problem goes into the meta training phase. To sample a episode,  $K$  per-class data samples are randomly chosen from the set of novel categories for constructing a support set for this episode, To construct a novel query set  $Q_{novel}$ , additional samples are randomly chosen from the novel categories. For base query samples, randomly selecting the samples of base categories with the same size to construct a base query set  $Q_{base}$ . The overall query set is the union of  $Q_{base}$  and  $Q_{novel}$ . From this learning procedure, IFSL uses unlimited access to the base categories throughout the learning process and also within the same dataset. Table 26 shows the difference between SDML and IFSL.