Appendices

A. Few-shot learning

A.1. Datasets

We experiment with four datasets for few-shot learning: Omniglot [25], MiniImageNet [51], TieredImageNet [38], and CIFAR-FS [6]. The Omniglot dataset consists of handwritten characters from 50 different alphabets and 1623 characters. There are 20 handwritten examples of each character. MiniImageNet contains 100 classes form ImageNet [13], which are split to 64, 16, and 20 classes for meta-training, meta-validation, and meta-test, respectively. TieredImageNet has 608 classes from ImageNet, which are grouped into 34 higher-level categories following the ImageNet taxonomy. They are split into 20 meta-training categories, 6 meta-validation categories, and 8 meta-test categories. Due to this partition scheme, the meta-test classes are less similar to the meta-training classes in TieredImageNet than in other datasets. CIFAR-FS re-purposes CIFAR-100 [24], splitting its 100 classes into 64, 16, and 20 classes for meta-training, meta-validation, and meta-test, respectively.

A.2. Experiment protocols and hyper-parameters

Our experiment protocols and implementation details largely follow MAML [15] and Reptile [33]. In particular, we use a convolutional neural network that comprises four modules in all the experiments. Each module has 3x3 convolutions, a batch-normalization layer, 2x2 max-pooling, and the ReLU activation, and every convolutional layer contains 64 filters for the experiments on Omniglot and 32 filters for other datasets. For fair comparison, we also re-implement some of the existing methods using this network architecture. We report more details for the "Lazy" Reptile in Table 6. For "Lazy" MAML, we set k = 10 for 1-shot and 5-shot tasks, respectively, on both MiniImageNet and TieredImageNet.

A.3. Many-Way Classification

We have presented many-way results on TieredImageNet for "Lazy" MAML in section 5.1.1. Here, we describe the implementation details for these experiments. We set the inner learning rate (η) to 0.005 and the outer learning rate (β) to 0.001 for all the settings. We let the student run k = 15, 15 and 18 steps for 20-way, 30-way and 50-way, respectively during meta-training.

A.4. Computational Analysis

In our evaluation, we also want to answer the following question empirically. How does the memory requirements of "Lazy" MAML compare with MAML?. Figure 4 shows the memory trade-off for "Lazy" MAML and MAML on 5-shot 20-way MiniImageNet. "Lazy" MAML decouples the dependency of inner and outer loop by teacher-student scheme which allows it to define a very lightweight computation graph. The figure shows that the memory of the "Lazy" MAML doesn't exceed beyond 5 GB for many inner gradient steps while on the other hand, MAML reaches the capacity of 12 GB after 5 inner steps. The right panel shows the computation time per iteration with respect to multiple gradient inner steps. The time taken by "Lazy" MAML doesn't increase exponentially as compared to MAML which takes more compute time and reaches the maximum capacity of memory after 5 inner steps.

A.5. Additional results on many-way learning

The left panel of Figure 5 compares the results of Reptile and "Lazy" Reptile for N-way-five-shot learning on TieredImageNet where N varies in $\{5, 20, 30\}$. Our approach outperforms Reptile. We emphasize that not all meta-learning algorithms can be approximated by a first-order version; for example, it is not immediately clear how to do it for Algorithm 2, the two-component weighting method for long-tailed classification, in the main text. The right panel of Figure 5 shows some 20-way-5-shot results on MiniImageNet. We can see that our lazy strategy boosts both MAML and Reptile by a significant margin, which is similar to what we see in Figure 2 of the main paper. It again indicates that more training data needs more steps of exploration for a task-specific model and hence magnifies the benefit of our teacher-student scheme introduced to both MAML and Reptile.

B. Meta-Attack

Here, we formally present the algorithm of our lazy meta-learning approach to training the meta-attacker in Algorithm 3. As described in the main paper that the original meta-attack [14] uses Reptile to train the attacker, so it is straightforward to improve the approach by using a "lazy" teacher. The inputs to the meta-attacker are images, and the desired outputs are their gradients — during meta-training, the gradients are generated from different classification models. Instead of the cross-entropy loss, meta-attack adopts a mean-squared error (MSE) loss in the inner loop, i.e.,

$$\mathcal{L}^{\mathcal{T}_i}_{\mathcal{D}_{tr}} = ||\phi(\mathcal{X}_{ij}) - \mathcal{G}_{ij}||^2_2$$

where the task $\mathcal{T}_i$ is to find adversarial examples for the inputs to the i-th pre-trained classification network, $\mathcal{X}_{ij}$ is an image sampled for the task, $\mathcal{G}_{ij}$ are the gradients of the classification network with respect to (w.r.t.) the image, and $\phi(\cdot)$ is a task-specific model whose output is to approximate the gradients $\mathcal{G}_{ij}$. This model is useful because, given a blackbox classification network, we can use the task-specific model to predict the gradients of this network w.r.t. an image, followed by gradient ascent towards an adversarial example (cf. Algorithm 4).

Table 6. Hyper-parameter details for few-shot learning in ours (Reptile). The "Eval inner batch" row shows the numbers for both 1-shot and 5-shot settings.

Hyper-parameter	Omniglot	CIFAR-FS	Mini-ImageNet	TieredImageNet
Inner learning rate (η)	0.001	0.001	0.001	0.001
Inner iterations (k)	5	8	8	8
Inner batch size	10	10	10	10
Training shots	10	15	15	15
Outer step-size (β)	1.0	1.0	1.0	1.0
Total outer-iterations	100k	120k	120k	130k
Meta batch size	20	20	20	20
Eval. inner iterations	50	50	50	50
Eval. inner batch	5/15	5/15	5/15	5/15

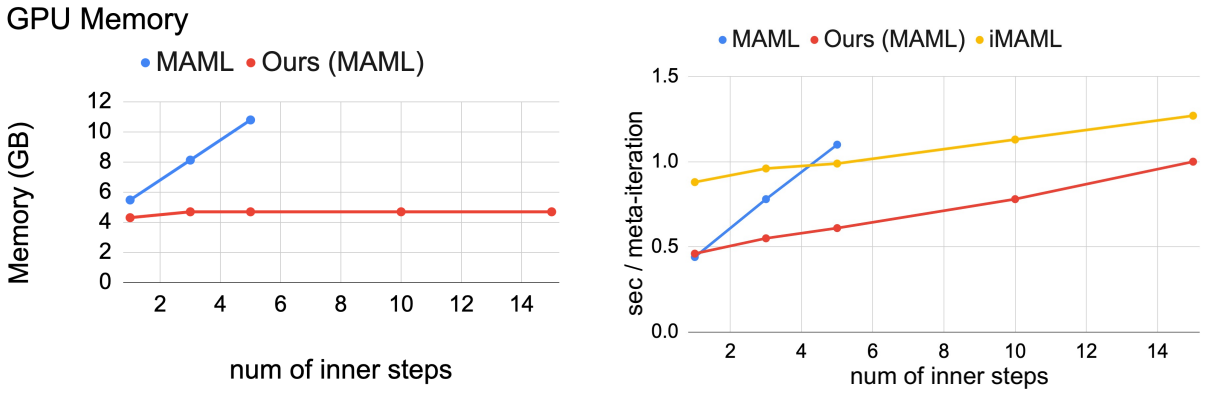

Figure 4. Left: Memory trade-offs with 4 layer CNN on 20-way-5-shot MiniImageNet task. b). Computation time (sec per meta-iteration) w.r.t the number of inner gradient steps on 20-way-5-shot MiniImageNet task with batch size 2.

Algorithm 3 Training algorithm of meta-attack using "lazy" Reptile

Require: A distribution over tasks $P_{\mathcal{T}}$
Require: Input Images $\mathcal{X}$, gradients $\mathcal{G}_i$ generated from a classification network serving task $\mathcal{T}_i$
Require: Learning rates α, β
Ensure: The meta attacker θ

Randomly initialize the meta-attacker θ
while not done **do**
  Sample a batch of tasks $\{\mathcal{T}_i \sim P_{\mathcal{T}}\}$
  for all $\{\mathcal{T}_i\}$ **do**
    Sample data $\mathcal{D}_{tr}$ and $\mathcal{D}_{val}$ for $\mathcal{T}_i$ // in the form of $\{\mathcal{X}_{ij}, \mathcal{G}_{ij}\}$
    $\phi_{i,0} \leftarrow \theta$
    for $j = 1, 2, \cdots, k$ **do**
      $\phi_{i,j} \leftarrow \phi_{i,j-1} - \alpha \nabla_{\phi} \mathcal{L}^{\mathcal{T}_i}_{\mathcal{D}_{tr}}(\phi_{i,j-1})$
    end for
    $\gamma_i \leftarrow \arg\min_{\gamma} \mathcal{L}^{\mathcal{T}_i}_{\mathcal{D}_{val}}(\gamma\theta + (1-\gamma)\phi_{i,k})$
    $\phi_i(\theta) \leftarrow \gamma_i\theta + (1-\gamma_i)\phi_{i,k}$
  end for
  $\theta \leftarrow \theta - \beta \sum_i (\theta - \phi_i(\theta))$
end while

Algorithm 4 Adversarial Meta-Attack

Require: Test image x_o with label t, meta-attacker f_θ, target model $\mathcal{M}_{tar}$, iteration interval j, selected top-q coordinates

for $t = 0, 1, 2, \cdots$ **do**
  if $(t+1) \mod j = 0$ **then**
    Perform zeroth-order gradient estimation on top-q coordinates, denoted as I_t and
    obtain g_t.
    Fine-tune meta-attacker f_θ with (x_t,g_t) on I_t by $\mathcal{L} = |[f_\theta(x_t]_{I_t} - [g_t]_{I_t}|^2_2$.
  else
    Generate the gradient map g_t directly from meta-attacker f_θ with x_t,
    select coordinates I_t.
  end if
  Update $[x']_{I_t} = [x_t]_{I_t} + \lambda[g_t]_{I_t}$.
  if $\mathcal{M}_{tar}(x') \neq t$ **then**
    $x_{adv} = x'$
    break
  else
    $x_{t+1} = x'$
  end if
end for

Ensure: adversarial example x_{adv}.

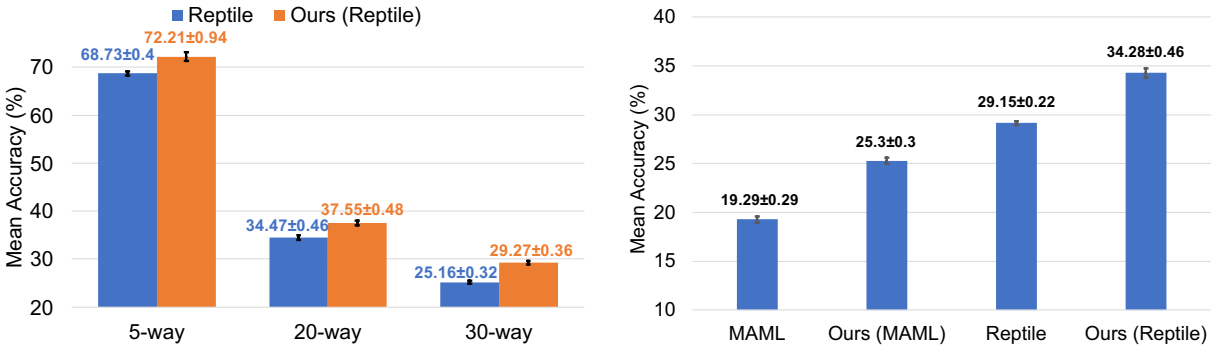

Figure 5. Left: Mean Accuracy (%) for N-way-five-shot classification on TieredImageNet. b). Mean Accuracy (%) for 20-way-5-shot classification on MiniImageNet.

Table 7. Comparison of several methods under targeted attack on MNIST and CIFAR-10. Similar to the untargeted attack, we reduce the number of queries for meta attack.

Dataset / Target model	Method	Success Rate	Avg. L_2	Avg. Queries
MNIST / Net4	Zoo [8]	1.00	2.63	23,552
	Decision Boundary [1]	0.64	2.71	19,951
	AutoZoom [47]	0.95	2.52	6,174
	Opt-attack [9]	1.00	2.33	99,661
	Meta attack [14]	1.00	2.66	1,299
	Lazy meta-attack (ours)	1.00	2.63	**1,108**
CIFAR10 / Resnet18	Zoo [8]	1.00	0.55	66,400
	Decision Boundary [1]	0.58	0.53	16,250
	AutoZoom [47]	1.00	0.51	9,082
	Opt-attack [9]	1.00	0.50	121,810
	FW-black [7]	0.90	0.73	6,987
	Meta attack [14]	0.93	0.77	3,667
	Lazy meta-attack (ours)	0.92	0.69	**3,092**

Algorithm 3 presents how to train this meta-attacker by applying our "lazy" teacher to Reptile, and we then follow Algorithm 4 for attacking blackbox networks [14].

B.1. Results under targeted attack

In Table 5 of the main paper, we report the results under untargeted attack. Here, we are presenting the results under targeted attack for both MNIST and CIFAR-10 in Table 7. Similar to untargeted attack, we achieve comparable results on success rate and average ℓ_2 distortion using a smaller number of queries.