# LoPrune: Efficient Data Pruning for LoRA-Based Fine-Tuning of Vision Transformer

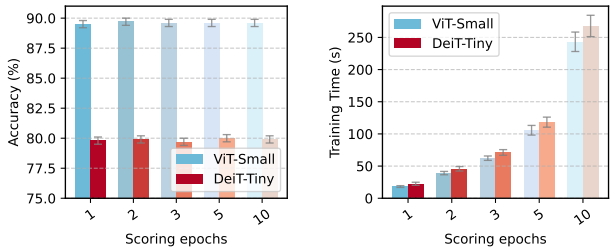
## Supplementary Material

**Algorithm 1** LoPrune: an efficient data pruning for LoRA-based fine-tuning of vision transformer

1: **Input:** Training set $\mathcal{D}$; LoRA trainable parameter vector $\phi$; pruning ratio $p$; EMA $\beta$; damping $\lambda$
2: **Output:** Retained set $\mathcal{S}$; scores $\{s(z)\}$
3: **Init**: For each block $t$, set $\Sigma_{x,t} \leftarrow I$, $\Sigma_{\delta,t} \leftarrow I$
4: **Score samples:**
5: **for each** minibatch $\mathcal{B} \subset \mathcal{D}$ **do**
6: Do one fwd/bwd to get $\{x_t, \delta_t\}$
7: Update $\Sigma_{x,t} \leftarrow (1-\beta)\Sigma_{x,t} + \beta\,\mathbb{E}[x_t x_t^\top]$,
8: $\Sigma_{\delta,t} \leftarrow (1-\beta)\Sigma_{\delta,t} + \beta\,\mathbb{E}[\delta_t \delta_t^\top]$
9: Add damping:
10: $\tilde{\Sigma}_{x,t} \leftarrow \Sigma_{x,t} + \lambda I$, $\tilde{\Sigma}_{\delta,t} \leftarrow \Sigma_{\delta,t} + \lambda I$
11: **for each** sample $z \in \mathcal{B}$ **do**
12: Calculate $S_B(z)$ and $S_A(z)$.
13: ▷ Defined in Eq. (14) and Eq. (15)
14: Update TSA scores for all samples
15: ▷ Defined in Eq. (14)
16: **end for**
17: **end for**
18: **Data pruning:** Sort $\{s(z)\}$ in descending sequence; set $\mathcal{S} \leftarrow$ top-$(1-p)N$ indices
19: **return** $\mathcal{S}$ and $\{s(z)\}$

## 7. Efficient Implementation of LoPrune

As shown in Algorithm 1, LoPrune implements sample scoring and data pruning in the LoRA-trainable subspace. For each LoRA-instrumented linear block $t = 1, \ldots, T$, LoPrune maintains two K-FAC statistics including the covariance of input activations $\Sigma_{x,t}$ and the covariance of back-propagated signals $\Sigma_{\delta,t}$. These statistics are updated online with an exponential moving average (EMA), and are then used to compute per-sample TSA scores. At the start of training, we set $\Sigma_{x,t} \leftarrow I$ and $\Sigma_{\delta,t} \leftarrow I$ for each block $t$, and initialize the score container $s(z) \leftarrow 0$ for all $z \in \mathcal{D}$. For each mini-batch $\mathcal{B} \subset \mathcal{D}$, we run one forward/backward pass to obtain cached per-sample quantities $\{x_t(z), \delta_t(z)\}$ for all $z \in \mathcal{B}$ and all LoRA blocks $t$. We then update the K-FAC statistics using EMA in Line 7 and 8. In line 9 and 10, we apply Tikhonov damping to ensure numerical stability. Reusing the cached $\{x_t(z), \delta_t(z)\}$ from the same pass, we compute for each sample $z$ the two LoRA-subspace quadratic forms $S_B(z)$ and $S_A(z)$ as defined in Eq. (14) and Eq. (15), and then aggregate them into the TSA score $s(z)$. After processing the full training set, we sort the scores $\{s(z)\}$ in descending order and retain the top-$(1-\rho)N$ samples to form the retained subset $\mathcal{S}$. The subsequent fine-tuning of the model is performed on $\mathcal{S}$. The procedure requires only a single standard forward/backward pass per mini-batch to update EMA statistics and compute scores, plus inexpensive linear solves per block. This yields approximately linear scaling in the number of samples while significantly reducing memory footprint compared to storing per-epoch or per-sample trajectories.




(a) Best Top-1 accuracy (b) Data pruning Time

Figure 6. Fine-tuning performance and data pruning time with different scoring epochs. It is clearly observed that the increase in scoring epochs has a little contribution to performance but introduces significant additional pruning overhead.

## 8. Effect of Scoring Epochs

As mentioned in the Sec. 1 and Sec. 2, fine-tuning typically converges with very few training epochs. Data pruning methods for fine-tuning must be insensitive to the number of epochs, otherwise they would struggle to perform well within such a limited number of epochs. To explore the impact of the number of epochs for computing importance scores on LoPrune, we select 1, 3, 5, and 10 epochs to collect the importance scores of the samples, using the average Mahalanobis distance across multiple epochs as the importance score. As shown in Fig. 6, with an increase in training epochs, the improvement in accuracy is not significant, with an increase of only 0.09% - 0.15%. In contrast, the training overhead escalates significantly with the increment of epochs. This suggests that, although summing over multiple epochs can slightly increase the upper bound of LoPrune, the additional computational overhead it introduces is not worthwhile given the marginal accuracy gains. Therefore, LoPrune typically performs a single pruning epoch early in the fine-tuning process.