A Supplementary Material

A.1 Elaborations on Existing Methods

In this section, we will elaborate on the three baselines considered in this paper, namely, the *conventional method* [34], *StolenEnoder* [26], and *Cont-Steal* [34].

• **Conventional method:** Given a surrogate dataset $D_S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, the conventional method first queries the target encoder E_T once with each sample $\boldsymbol{x}_i \in D_S$. The output embedding $E_T(\boldsymbol{x}_i), i = 1, 2, \ldots, N$, will be stored in a memory bank to serve as a "ground truth" for the sample across the whole training. Next, to train the surrogate encoder E_S, it feeds each sample $\boldsymbol{x}_i \in D_S$ to E_S and pulls the output embedding $E_S(\boldsymbol{x}_i)$ close to its corresponding optimization objective, i.e., $E_T(\boldsymbol{x}_i)$.

• **StolenEncoder:** Similar to the conventional method, StolenEncoder first queries the target encoder E_T once with each sample $\boldsymbol{x}_i \in D_S$ and stores the output embedding $E_T(\boldsymbol{x}_i), i = 1, 2, \ldots, N$, in a memory bank. Next, it not only feeds each sample $\boldsymbol{x}_i \in D_S$ but also an augmentation $\boldsymbol{x}'_i$ of it to the surrogate encoder E_S. The output embeddings $E_S(\boldsymbol{x}_i)$ and $E_S(\boldsymbol{x}'_i)$ will be pulled close to $E_T(\boldsymbol{x}_i)$ simultaneously for optimizing the surrogate encoder E_S.

• **Cont-Steal:** In particular, Cont-Steal adopts an end-to-end training strategy to train the surrogate encoder E_S. In each epoch, Cont-Steal first augments each image $\boldsymbol{x}_i \in D_S$ into two perspectives, denoted as $\boldsymbol{x}'_{i,t}$ and $\boldsymbol{x}'_{i,s}$, respectively. Next, $\boldsymbol{x}'_{i,t}$ is used to query the target encoder E_T, while $\boldsymbol{x}'_{i,s}$ is fed to the surrogate encoder E_S. The surrogate encoder then is optimized via contrastive learning that pulls $E_T(\boldsymbol{x}'_{i,t})$ and $E_S(\boldsymbol{x}'_{i,s})$ close while pushing $E_S(\boldsymbol{x}'_{k,s}), k \neq i$, and $E_T(\boldsymbol{x}'_{i,t})$ apart. Moreover, to enhance contrastive learning, Cont-Steal also includes $E_S(\boldsymbol{x}'_{i,s})$ and $E_S(\boldsymbol{x}'_{k,s}), i \neq k$, as negative pairs to train E_S.

Table 7: Query cost analysis of three baseline methods.

Method	Surrogate Dataset Size	Training Epoch	Query Cost
Conventioanl [34]	N	L	N
StolenEncoder [26]	N	L	N
Cont-Steal [34]	N	L	$N \times L$

Query Cost Analysis. Assuming the attacker has a surrogate dataset $D_S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ consisting of N images. To train a surrogate encoder, the attacker will use D_S to query the target encoder and optimize the surrogate encoder to mimic its outputs. Regarding the conventional method and StolenEncoder, each image $\boldsymbol{x}_i \in D_s$ will be used to query only once. Therefore, the query cost of them is N, and nothing about the number of training epochs. As for Cont-Steal, each image in D_S will be augmented into two perspectives and one is used to query the target encoder in each epoch. Therefore, assuming the training will last for L epochs, the query cost of Cont-Steal is $N \times L$. Although Cont-Steal achieves superior results, its query cost is formidable since the training typically

Algorithm 1: Detailed steps of RDA

input : Surrogate dataset D_S, target encoder E_T, surrogate encoder E_S, patch number n for generating prototypes and m for training.
output: A high-performance surrogate encoder E_S

Load E_T and initialize E_S.
Prototype generation:
for *each image* $\boldsymbol{x}_i \in D_S$ **do**
  augment $\boldsymbol{x}_i$ into multiple patches $\{\boldsymbol{x}'_{i,t,c}\}_{c=1}^{n} = \{\boldsymbol{x}'_{i,t,1}, \ldots, \boldsymbol{x}'_{i,t,n}\}$;
  query E_T with $\{\boldsymbol{x}'_{i,t,c}\}_{c=1}^{n}$ and obtain $\{E_T(\boldsymbol{x}'_{i,t,c})\}_{c=1}^{n} = \{E_T(\boldsymbol{x}'_{i,t,1}), \ldots, E_T(\boldsymbol{x}'_{i,t,n})\}$;
  calculate the sample-wise prototype for $\boldsymbol{x}_i$ as follows: $p_{\boldsymbol{x}_i} = \frac{1}{n}\sum_{c=1}^{n} E_T(\boldsymbol{x}'_{i,t,c})$;
Training:
for *each epoch* **do**
  for *each image* $\boldsymbol{x}_i \in D_S$ **do**
    augment $\boldsymbol{x}_i$ into multiple patches $\{\boldsymbol{x}'_{i,s,q}\}_{q=1}^{m} = \{\boldsymbol{x}'_{i,s,1}, \ldots, \boldsymbol{x}'_{i,s,m}\}$;
    feed $\{\boldsymbol{x}'_{i,s,q}\}$ into E_S and obtain $\{E_S(\boldsymbol{x}'_{i,s,q})\}_{q=1}^{m} = \{E_S(\boldsymbol{x}'_{i,s,1}), \ldots, E_S(\boldsymbol{x}'_{i,s,m})\}$;
    for *each* $E_S(\boldsymbol{x}'_{i,s,1}) \in \{E_S(\boldsymbol{x}'_{i,s,q})\}_{q=1}^{m}$ **do**
      optimize E_S with $\mathcal{L} = \lambda_1 \cdot \mathcal{L}_D + \lambda_2 \cdot \mathcal{L}_A$ based on Eq. 2-10;
return: the trained surrogate encoder E_S

Table 8: Results of different loss designs. The results are SAs on different downstream datasets, with highlighting the optimal.

Loss Type	CIFAR100	F-MNIST	GTSRB	SVHN
$\mathcal{L}$ (current used)	44.27	89.32	62.75	73.74
$\mathcal{L}_{v2}$	41.99	85.45	61.92	72.88
$\mathcal{L}_{v3}$	42.23	88.63	62.46	71.86

requires over 100 epochs to converge. We have the query cost of each method summarized in Table 7.

A.2 Details about Our Method

Practical Formulation of the Memory Bank. In practice, we assign a unique label to each image in the attacker's surrogate dataset as its key in the memory bank. Specifically, we transform $D_s = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ into a set of image-key pairs, i.e., $D_s = \{(\boldsymbol{x}_1, 1), (\boldsymbol{x}_2, 2), \ldots, (\boldsymbol{x}_n, n)\}$. Then we generate a prototype for each image and get a set of key-prototype pairs that can be expressed as $P = \{1 : p_{\boldsymbol{x}_1}, 2 : p_{\boldsymbol{x}_2}, \ldots, n : p_{\boldsymbol{x}_n}\}$. The prototype set P is stored in a memory bank and we do not need to query the target model during training anymore.

Detailed Steps. Detailed steps of RDA are summarized in Algorithm 1.

Table 9: Ablation studies on the weights of $\mathcal{L}_{amp}$ and $\mathcal{L}_{ang}$. The results are SAs on different downstream datasets, with highlighting the optimal.

$\mathcal{L}_{amp} : \mathcal{L}_{ang}$	CIFAR100	F-MNIST	GTSRB	SVHN
5 : 1	42.90	86.52	61.32	70.57
2 : 1	43.61	86.80	62.55	72.90
1 : 1	44.27	89.32	62.75	73.74
1 : 2	42.70	88.64	60.59	72.05
1 : 5	43.24	89.16	62.04	73.11

A.3 Loss Designs

Alternative Loss Designs. In this section, we consider two alternative loss designs about $\mathcal{L}_A$. For simplicity, we denote them as $\mathcal{L}_{A,v2}$ and $\mathcal{L}_{A,v3}$, respectively.

• $\mathcal{L}_{A,v2}$: To investigate the benefits of penalizing different deviations with different levels, we define $\mathcal{L}_{A,v2}$ as a naive combination of $\mathcal{L}'_{amp}$ and $\mathcal{L}'_{ang}$ as follows:

$$\mathcal{L}_{A,v2} = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{L}'_{amp}(\boldsymbol{x}_i) - \mathcal{L}'_{ang}(\boldsymbol{x}_i)). \tag{11}$$

• $\mathcal{L}_{A,v3}$: To investigate the benefits of our current penalizing regime employed by $\mathcal{L}_A$, we reverse it here by an "exp", i.e., penalizing the MSE increase from 0.8 to 0.9 more harshly than that from 0.3 to 0.4, with the same rule employed on the reciprocal of the cosine similarity. Therefore, we formulate the $\mathcal{L}_{A,v3}$ as follows:

$$\mathcal{L}_{A,v3} = \frac{1}{N}\sum_{i=1}^{N}(\exp\left(\mathcal{L}'_{amp}(\boldsymbol{x}_i)\right) + \exp\left(1/\mathcal{L}'_{ang}(\boldsymbol{x}_i)\right)). \tag{12}$$

Finally, we define:

$$\mathcal{L}_{v2} = \mathcal{L}_D + \mathcal{L}_{A,v2}, \tag{13}$$

$$\mathcal{L}_{v3} = \mathcal{L}_D + \mathcal{L}_{A,v3}. \tag{14}$$

We use the encoder pre-trained on CIFAR10 as the target encoder and evaluate the trained surrogate encoders on four different downstream datasets, i.e., CIFAR100, F-MNIST, GTSRB, and SVHN. Each experiment is run three times and we report the mean value of the achieved SAs by each loss design in Table 8.

As the results show, both $\mathcal{L}_{v2}$ and $\mathcal{L}_{v3}$ underperform $\mathcal{L}$. On the one hand, $\mathcal{L}$ outperforms $\mathcal{L}_{v2}$ indicating that incorporating our current penalizing regime is beneficial. On the other hand, $\mathcal{L}$ outperforms $\mathcal{L}_{v3}$ indicating that penalizing a deviation from a more favorable value to a worse one more harshly is more favorable upon the opposite penalizing regime.

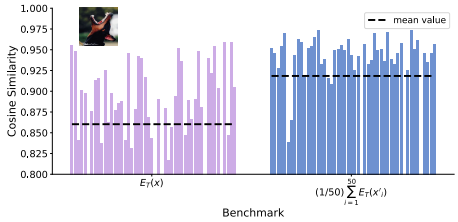

Fig. 10: We augment an image $\boldsymbol{x}$ into 50 patches and feed them into an encoder E_T that pre-trained on CIFAR10 (KNN Accuracy = 88.31%, tested on CIFAR10). This figure depicts the cosine similarity between each augmentation patch's embedding and two different benchmarks, i.e., (1) the embedding of the original image and (2) the prototype of the image, with the mean value over the 50 similarities marked in the black dashed line. We can see that the prototype is significantly more similar to each patch's embedding, showcasing it is less biased.

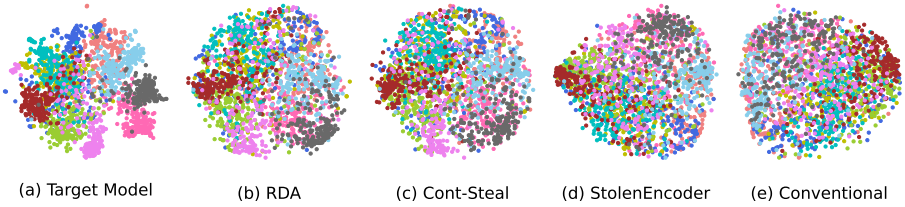

Fig. 11: t-SNE of embeddings of 2,000 images randomly sampled from CIFAR10 generated by the target encoder and surrogate encoders trained with different methods.

The Weights of Amplitude and Angle Deviations. Recall our default setting that we assume the amplitude and angle deviations are of equal importance and thus formulate $\mathcal{L}_A(\boldsymbol{x}_i) = \mathcal{L}_{amp}(\boldsymbol{x}_i) + \mathcal{L}_{ang}(\boldsymbol{x}_i)$. Here we ablate on five different weight pairs as shown in Table 9 to investigate the importance level of each deviation type. From the results, we can see that setting the weights of $\mathcal{L}_{amp}$ and $\mathcal{L}_{ang}$ to $1:1$ generally achieves the best result.

A.4 Supplementary Experiments

Benifit of Using Prototypes. We augment an image $\boldsymbol{x}$ (stamped at the upper left corner of Figure 10) into 50 patches and denote each of them as $\boldsymbol{x}'_i, i = 1, \ldots, 50$. Next, we input each patch as well as the image's original version into E_T, an encoder pre-trained on CIFAR10. To show the image's prototype (i.e., $\frac{1}{50}\sum_{i=1}^{50} E_T(\boldsymbol{x}'_i)$) is a less biased optimization objective compared to the embedding of its original version (i.e., $E_T(\boldsymbol{x})$), we depict the cosine similarity between each patch's embedding and the two different benchmarks in Figure 10. We can see that the prototype has significantly higher similarity with each patch's embedding, showcasing it is less biased.

Table 10: The time cost of each method over the 100 training epochs. Each result consists of two parts, i.e., the time for training the surrogate encoder and the time for testing after each epoch.

Method	Time (min)
Conventional	3.87 + 13.18
StolenEncoder	5.53 + 13.18
Cont-Steal	4.08 + 13.18
RDA	10.65 + 13.18

Visualization of the Embedding Space. To visualize the embedding space of the surrogate encoder trained by each method, we randomly sample 200 images from each class of CIFAR10 (i.e., 2,000 images in total) and feed them to the surrogate encoder. From Figure 11, we can observe that embeddings of different classes from surrogate encoders trained by StolenEncoder and the conventional method overlap with each other and exhibit less structural clarity. In contrast, RDA and Cont-Steal train surrogate encoders that generate more distinguishable embeddings for different classes. This indicates that encoder stealing benefits from contrastive learning.

Time Cost. In particular, the total training time comprises two parts, i.e., the time for training the surrogate encoder and the time for testing the trained surrogate encoder after each training epoch to choose the best one. We present the total time cost of each method in Table 10. We can see that the testing time is 13.18 minutes and is identical across all methods over 100 epochs, while the training time of RDA is the longest. This is because RDA augments each image into 5 patches by default to train the surrogate encoder, which means 5 times forward encoding for each image. For a fair comparison, we prolong the training for another 200 epochs for each baseline method and show that our RDA still surpasses them by a large margin, as demonstrated by Figure 6.

Effectiveness of RDA on Different Encoder Architectures. To demonstrate the effectiveness of our RDA against pre-trained encoders of various architectures, we further evaluate it on ResNet34, VGG19_bn [37], DenseNet121 [15], and MobileNetV2 [14]. Table 11 shows that RDA can achieve comparable performances with the target encoders of various architectures and even outperforms them on multiple datasets.

Ablation Studies on the Weight of Each Part in the Loss Function. To investigate the impact of each part in our loss function on the stealing performance, we fix λ_1 in Eq. 10 to 1 and vary the value of λ_2 (i.e., the weight of the aligning loss $\mathcal{L}_A$) from 0 to 10. We use the encoder pre-trained on CIFAR10 as the target encoder and evaluate the trained surrogate encoder on three different downstream datasets, i.e., CIFAR10, GTSRB, and SVHN. As we can observe in

Table 11: Results of RDA against target encoders of different architectures. The pre-training dataset is CIFAR10 and the architecture of the surrogate encoder is ResNet18.

Target Encoder Architecture	Downstream Dataset	TA	SA	$\frac{\text{SA}}{\text{TA}} \times 100\%$
ResNet34	MNIST	96.42	96.19	99.76
	F-MNIST	89.51	87.03	97.23
	GTSRB	62.79	54.57	86.91
	SVHN	61.68	70.12	113.68
VGG19_bn	MNIST	90.76	91.57	100.89
	F-MNIST	68.54	74.28	108.37
	GTSRB	13.27	11.25	84.78
	SVHN	26.32	41.30	156.91
DenseNet121	MNIST	95.81	96.15	100.35
	F-MNIST	86.75	88.79	102.35
	GTSRB	54.69	52.05	95.17
	SVHN	49.52	67.19	135.68
MobileNetV2	MNIST	90.56	94.53	104.38
	F-MNIST	77.69	83.38	107.32
	GTSRB	33.88	24.00	70.84
	SVHN	27.54	58.54	212.56

Fig. 12: Ablation studies on the weight of each part in the loss, where we fix λ_1 to 1 and vary the value of λ_2 from 0 to 10.

Figure 12, the surrogate encoders' performances on CIFAR10 and SVHN positively correlate to the weight of $\mathcal{L}_A$, i.e., λ_2. This is because CIFAR10 is the pre-training dataset of the target encoder. As the weight of the aligning loss $\mathcal{L}_A$ increases, the outputs of the surrogate and the target encoder become more aligned, and thus the trained surrogate encoder will perform better on CIFAR10. On the other hand, a larger weight for $\mathcal{L}_A$ will make the surrogate encoder fit the surrogate dataset more. In this sense, since SVHN is more similar to the surrogate dataset (i.e., Tiny ImageNet), the surrogate encoder naturally performs better on it as the λ_2 increases. On the contrary, as the weight of $\mathcal{L}_A$ increases, the surrogate encoder's performance on GTSRB which shares little similarity with both the pre-training and surrogate datasets becomes worse. Similar phenomenons have been discussed in Section 4.4, where $\mathcal{L}_D$ trains the surrogate encoder that achieves the optimal performance on GTSRB. A larger weight for

Table 12: Ablations on each part of $\mathcal{L}_A$.

Loss Function	CIAFR10	GTSRB	SVHN	CIFAR100	AVG
$\mathcal{L}_D + \mathcal{L}_A$	79.39	62.75	73.74	44.27	65.03
$\mathcal{L}_D + \mathcal{L}_{amp}$	76.17	56.21	74.40	40.49	61.81
$\mathcal{L}_D + \mathcal{L}_{ang}$	77.70	55.85	75.81	42.78	63.03

Table 13: Ablations on the surrogate data.

Surrogate Data	CIAFR10	GTSRB	SVHN
CIFAR10	81.33	59.34	75.21
GTSRB	68.54	54.84	73.70
SVHN	63.63	42.00	68.78

$\mathcal{L}_A$ will make the surrogate encoder fit the surrogate dataset more, and thus perform worse on GTSRB.

Ablation Studies on the Effect of Each Part in $\mathcal{L}_A$. To investigate whether our $\mathcal{L}_A$ can acquire better performance compared to each part (i.e., $\mathcal{L}_{amp}$ and $\mathcal{L}_{ang}$) of it, we do ablation experiments as presented in Table 12. All settings follow the default configurations we use except the loss function. From the results, we can observe that $\mathcal{L}_D + \mathcal{L}_A$ achieves the best performance in three downstream datasets among four. Averagely, $\mathcal{L}_D + \mathcal{L}_A$ still largely surpasses the other two loss functions. This suggests the necessity of both $\mathcal{L}_{amp}$ and $\mathcal{L}_{ang}$ in our loss design of $\mathcal{L}_A$.

Ablation Studies on the Choice of The Surrogate Data. To investigate the impact of the choice of the surrogate data used by attackers, we randomly sample 2,500 images from three different datasets, namely CIFAR10, GTSRB, and SVHN respectively to build three different surrogate datasets. Other settings follow the default setting. As Table13 shows, using images sampled from CIFAR10 as the surrogate data offers the most effective performance. However, comparing Table 6 and Table 13, we can find that sampling data from CIFAR10 as the surrogate data underperformance sampling from Tiny-ImageNet on GTSRB. We suspect the reason is that GTSRB differs much from the pertaining data and thus using CIFAR10 as the surrogate data offers minimal benefits. Additionally, the diversity of Tiny-ImageNet enables it to extract broader potential representations of the target encoder.

More Complex Downstream Datasets. We have also conducted small-scale experiments on three more complex downstream datasets: Tiny-ImageNet, CUB-200-2011, and Food 101. For these datasets, we center-crop and resize each image to 224×224, followed by training the downstream classifier for 300 epochs to

Table 14: Additional experiments on more difficult datasets. The reported result is $\frac{\mathbf{SA}}{\mathbf{TA}} \times 100\%$, which shows the attack efficacy.

Dataset	Description	Conventional	StolenEncoder	Cont-Steal	RDA
Tiny-ImageNet	200 classes	34.29	79.81	92.14	94.76
CUB-200-2011	200 classes	31.35	44.58	54.88	57.37
Food101	101 classes	28.77	40.04	79.29	79.77

Table 15: Attack efficacy ($\frac{\mathbf{SA}}{\mathbf{TA}} \times 100\%$) against third-party models. **These target models can be downloaded via our code link.**

Provider / Platform	URL	CIFAR10	CIFAR100
OpenMMLab	https://github.com/open-mmlab	84.72	107.59
Hugging Face	https://huggingface.co	99.53	91.41

attain relatively decent performances. The results presented in Table 14 demonstrate that our RDA consistently delivers superior performances.

Stealing Third-Party Models. To mitigate the potential risks of violating relevant laws, we refrain from unauthorized usage of commercial APIs for stealing attacks. Consequently, to further validate the applicability of our method, we provide additional results using open-sourced third-party models in accordance with our default settings for a simulation. After deploying the target model, we assume knowledge only of the dimension of the output embeddings. Results are presented in Table 15, illustrating the applicability of RDA on these third-party models.

A.5 Possible Defenses against RDA

Since our RDA augments each image into multiple semantically similar patches, this may result in frequent similar queries at the embedding level. Therefore, a potential defense against RDA could involve rejecting such frequent and similar queries. For instance, if a user's recent queries exhibit high similarity, the defender might reject subsequent queries. However, this approach has several challenges. Determining the appropriate threshold for rejecting similar queries is complex and requires careful consideration. Moreover, the defender might need to dynamically adjust the threshold and rejection criteria. Additionally, if the user distributes the queries across multiple accounts, the defense becomes even more challenging. Effective defense strategies still require significant development.