A Broader Impacts

The goal of our work is to actively defend self-supervised encoders against model stealing attacks. Since we are directly defending encoders, any negative societal impacts of our work are minimal. One potentially negative impact could be the degradation of performance for legitimate users. However, as shown in our experimental results, we are able to preserve high utility for standard users.

B Limitations

We show how our defense method is tuned for SimSiam and DINO. There are more types of SSL encoders that can be tested with our method. The B4B defense method requires tuning the parameters, such as the number of occupied buckets that is allowed without any penalty for the cost function, or the selection of the transformations. These steps are rather difficult to automate but can be replaced with more data-driven approaches. For example, instead of designing a cost function from scratch, we could create an ML model to obtain a cost for a given occupation of the representation space. We explain more details in the Appendix C.2.

C Alternative Building Blocks to Instantiate B4B

While we present a reference implementation of B4B in our work that instantiates the three building blocks with (1) Local Sensitive Hashing, (2) Utility of the Representations, and (3) a set of concrete transformations, there exists a multitude of alternatives to concretely implement our B4B framework. In the following, we present these alternatives, grouped by building block.

C.1 Alternative Estimation of the Coverage of Embedding Space

We also explore alternative methods to measure the distances between representations for queries sent to an API. One of them is to apply the cosine distance (where for two representations a and b, it is defined as: $1 - \frac{a^T b}{||a||_2 \cdot ||b||_2}$) since it can be measured between individual data points in a pair-wise fashion. If the total pair-wise cosine distance between representations for a given user is small, then the user queries presumably come from a single downstream task distribution. Otherwise, a user might be malicious and would like to cover a large part of the representation space, then the total pair-wise cosine distance for the user's representations would be high. Note that in this case, the cosine distance can be replaced with any other distance measure, such as the Minkowski distance. We opt for the LSH in our reference implementation, since it is much less expensive to compute than cosine distance. LSH requires only $2^{12} = 4096$ buckets that can be expressed as a binary table with the same number of elements, which requires in the worst case iterating over all of them to count how many are occupied. With more than 4096 queries sent by a given user, the computation on the LSH is sublinear $< O(n)$ with respect to the number of user queries. For the cosine distance approach, the required computation grows quadratically $O(n^2)$ with the number of queries.

C.2 Alternative Cost Functions

The cost functions can be designed from scratch manually or learned, for example, via an ML model, such as a neural network or SVM. In our initial version, the function was designed manually, where the underlying premise is that once a specified number of buckets is occupied, the cost should grow exponentially. Instead of defining such a function or providing the high-level parameters for functions that we contributed, one could learn an ML model that for a given number of buckets occupied, it should output an estimated cost, or even directly, the desired σ (standard deviation) of the noise added to representations. This method requires a relatively large number of data points to be provided for training the model, however, lowers the burden on a defender to either decide on the specific function or adjust its parameters. Thus, it could be more user-friendly, for example, not necessitating any mathematical background, but can be precise enough to obtain the desired behavior.

Note that instead of adding the calibrated noise (proportional to the estimated cost) to the representations, we could rather require a given user to pay a higher monetary cost for queries that cover a large fraction of the representation space, or force a user to solve a puzzle in a form of the proof-of-work [17], wait a specified amount of time via proof-of-elapsed time (PoET) [4], or prove that a specified amount of disk space was reserved [18, 19]. For example, consider the approach with PoET. A user sends queries to the API, which we cost based on their occupation of the embedding space. The user is sent a waiting time. The users' resource (e.g., a CPU) has to be occupied for this specific waiting time without performing any work. At the end of the specified amount of time, the user sends proof of the elapsed time, which can be easily verified by the server. PoET requires access to specialized hardware, for example, secure CPU instructions that are becoming widely available in consumer and enterprise processors. If a user does not own such hardware, proof of elapsed time could be produced using a service exposed by a cloud provider (e.g., Azure VMs that feature TEE 2). Note that if a server sends the time based on the calculated cost, the adversary might learn the cost function. Instead, the exact waiting time should be split in random *subwaiting* times and sent to the user one by one. Thus, a server should rather have a few rounds of exchange with the client to incur the additional cost.

C.3 Alternative to Transformations

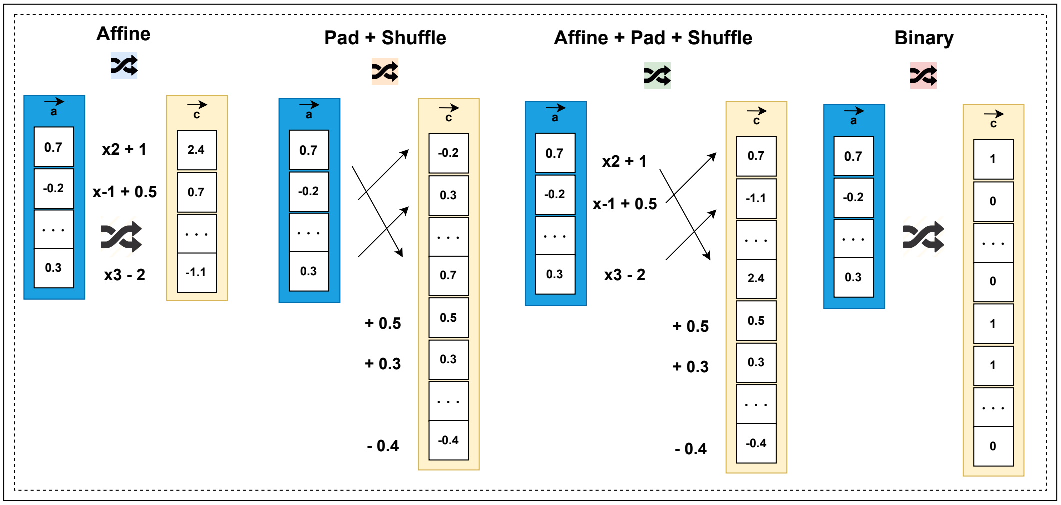


Figure 6: **Overview on Transformations.** We depict the inner-workings of the transformations considered in this work.

As an alternative to the transformations used within this work (see Figure 6), one could use a different set of transformations or combinations thereof. The padding can be done with different constant values and combined with adding constant values within the representations. The padding and adding the constant values can be followed by shuffling the elements within the representations. We can apply the affine of binary transformations on top of the padding and shuffling. Additionally, we can also use other pre-defined linear transformations like rotations or shearing.

The representations could also be compressed to smaller vectors and the compression rate would depend on the occupation of the representation space, for example, the higher the number of occupied buckets in our hash table, the more compressed the output representations could be. Such representations could be compressed via FFT, a cosine transform, or standard compression techniques such as snappy [5]. If the information from the representations should not be lost, then the lossless compression techniques can be applied, for instance, zstd [6]. The only requirement of the compression techniques is to ensure that they do not decrease the accuracy on downstream tasks for legitimate users.

Another alternative is to incorporate an additional neural network layer for transforming the returned representations. The training of this supplementary layer should primarily focus on preserving the usability of the representations for legitimate users. This approach grants the API provider with additional capabilities, as it allows for the utilization of customized training objectives. For instance, if the API provider employs LSH (Locality-Sensitive Hashing) to estimate the coverage of the representation space, they can leverage buckets and train the additional layer to maintain high-quality representations exclusively for frequently-used buckets and their surrounding areas, while not prioritizing the rest of the representation space. This approach safeguards legitimate users from any adverse effects, as their coverage of the representation space is minimal. Simultaneously, it ensures that adversaries are unable to exploit representations from the entire representation space.

D Sybil Attacks

We consider an adversary who generates n sybil accounts to steal the encoder from the API. For each of the accounts, the representations are transformed in a different way. Therefore, to replicate the victim model using all the obtained representations, the adversary has to map these representations into one single space. This can be done, for example, by training a neural network to perform the mapping.

We assume the adversary obtains $\{N_1, N_2, \ldots, N_n\}$ many representations from the victim for each of the n sybil accounts. Without loss of generality, we assume the adversary maps them back to the embedding space of the first sybil account. To learn the mapping, the adversary can apply different strategies.

D.1 Sybil Strategies

We present three potential approaches that Sybils might want to apply to circumvent our defense. Consider three users: A, B, and C, with their respective datasets D_A, D_B, and D_C, each with different distributions to maximize extraction effectiveness. First, user A is selected to unify representations from other users B and C. User A would have to query from at least two different datasets D_B and D_C, while other users would act legitimately. Sybil attackers want to deploy as many users as possible but with more fake accounts, user A incurs high coverage of the representation space, and this is prevented by our single-user defense. In all other cases neither of the sybil users can act legitimately, thus they are already affected by the single-user defense. Second, user A would query from their own dataset D_A and partially from dataset D_B. Then user B would query from their own dataset D_B and partially from dataset D_C, and so on. This method is the most inconspicuous but requires a number of remappings that scales super-exponentially with the number of fake accounts, which is impractical. Finally, each user would query from their respective dataset, for example, user B would query from dataset D_B and additionally from a remapping dataset, *e.g.,* D_A. Representations could be unified by mapping them to A's representations. The last approach as well as all other cases reduce to the minimum of remapping between representations of a pair of users. We show that our defense cuts such attempts short by ensuring that the remapping between representations is prohibitive even for a pair of users.

E Additional Related Work

One of the main workhorse techniques used in the encoders is contrastive learning, where the representations are trained so that the positive pairs (two augmented versions of the same image) have similar representations while negative pairs (augmentations of two different images) have representations which are far apart.

SimSiam utilizes Siamese networks (two encoders with shared weights) but with a simplified training process and architecture. In contrast to the previous frameworks, such as SimCLR [10], SimSiam's authors show that negative samples are unnecessary and collapsing solutions can be avoided by applying the projection head to of one of the encoders, and a stop-gradient operation to the other. SimSiam minimizes the negative cosine similarity between two randomly augmented views of the same image from the Siamese encoders, which is expressed via a symmetrized loss [21]. This creates a simple yet highly effective representation learning method.

DINO is another popular representation learning framework. While SimSiam uses CNNs, DINO employs vision transformers (ViTs). It trains a student and teacher encoder with the same architecture, updating the teacher with an (exponential moving) average of the student. Different random transformations of the same image are passed through both encoders. The student receives smaller image crops, forcing it to generate representations restoring parts of the original image. The training objective is minimizing cross-entropy loss between teacher and student representations.

F Additional Experimental Results

F.1 Details on Experimental Setup

The end-to-end experiments on stealing SimSiam and ViT DINO encoders were done using 3 A100 GPUs. Detailed experiments including mapping, transformations and the evaluation was performed using a single computer equipped with two Nvidia RTX 2080 Ti GPUs.

F.2 Datasets Used

CIFAR10 [27]: The CIFAR10 dataset consists of 32x32 colored images with 10 classes. There are 50000 training images and 10000 test images.

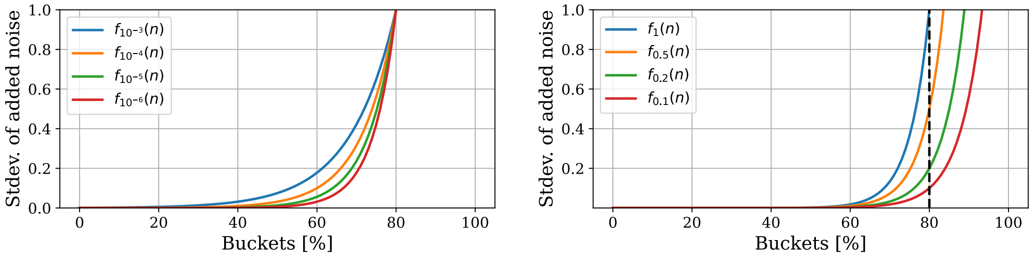
SVHN [31]: The SVHN dataset contains 32x32 coloured images with 10 classes. There are roughly 73000 training images, 26000 test images and 530000 "extra" images.

ImageNet[14]: Larger sized coloured images with 1000 classes. As is commonly done, we resize all images to be of size 224x224. There are approximately 1 million training images and 50000 test images.

STL10 [12]: The STL10 dataset contains 96x96 coloured images with 10 classes. There are 5000 training images, 8000 test images, and 100000 unlabeled images.

F.3 More Results for the End2End Empirical Evaluation

We consider fine-tuning parameters β, λ, and α for our cost function and the intuitive meaning behind these parameters. In general, our recommendation is to adjust the parameter β that specifies how many buckets are allowed to be filled by users' downstream tasks. On the other hand, when parameter λ is increased, this causes a higher amount of added noise before we reach the number of buckets specified by β, which lowers the performance of a given downstream task relatively early. For example, a higher value of λ in Figure 4, would cause an increase in the amount of added noise much earlier than for the target value of β. Finally, parameter α controls the amount of noise once the number of buckets specified by β is reached. Thus, in Figure 4, we set $\alpha = 1$ and the maximum standard deviation of the added Gaussian noise is 1.



(a) Cost Function for different λ parameter values.

(b) Cost Function for different α parameter values.

Figure 7: **Effects of λ and α parameters on the Cost Function.** We present the Cost Function for α=1, β=80 and different values of λ (*left*) and $\lambda = 10^{-6}$, β=80 and different values of α (*right*).

Table 2: **Stealing and Using Encoders With and Without our Defense**. The model used in the experiments is Simsiam, with the following parameters for the cost function $\lambda = 10^{-4}$, $\alpha = 1$, and $\beta = 80\%$, and the number of buckets equal to 2^{12}. Due to the higher value of the parameter λ, we observe lower performance on downstream tasks for the attackers since the magnitude of noise added to the representations is higher. However, for more complicated tasks than CIFAR10, this change might cause a potential drop in accuracy for the legitimate users.

User	Defense	# Queries	Dataset	Type	CIFAR10	STL10	SVHN	F-MNIST
N/A	N/A	*Victim*	*ImageNet*	*Train*	90.41±0.02	95.08±0.13	75.47±0.04	91.22±0.11
Legit	B4B	50k	CIFAR10	Query	90.02 ±0.1	N/A	N/A	N/A
Legit	B4B	5k	STL10	Query	N/A	94.88 ±0.17	N/A	N/A
Legit	B4B	73k	SVHN	Query	N/A	N/A	74.72 ±0.13	N/A
Legit	B4B	60k	F-MNIST	Query	N/A	N/A	N/A	91.76 ±0.09
Attack	None	50K	ImageNet	Steal	65.2 ±0.03	64.9 ±0.01	62.1 ±0.01	88.5 ±0.01
Attack	B4B	50k	ImageNet	Steal	28.22 ± 0.04	26.62 ± 0.02	19.62 ± 0.02	78.41 ±0.01
Attack	None	100K	ImageNet	Steal	68.1 ±0.03	63.1 ±0.01	61.5 ±0.01	89.0 ±0.07
Attack	B4B	100k	ImageNet	Steal	17.73 ± 0.18	15.59 ± 0.61	19.53± 0.01	55.11 ± 0.05
Sybil	B4B	50k+50k	ImageNet	Steal	33.43 ± 0.03	31.18 ± 0.12	22.91 ± 0.01	75.35 ± 0.05

Table 3: **Stealing and Using Encoders With and Without our Defense**. The model used in the experiments is Simsiam, with the following parameters for the cost function $\lambda = 10^{-6}$, $\alpha = 1$, and $\beta = 50\%$, and the number of buckets equal to 2^{12}. This experiment corresponds to considering 50% of buckets filled as a too-large coverage of the embedding space. This improves the defense but again might potentially harm the performance of more complicated tasks than CIFAR10 since they could occupy more buckets than 50%.

User	Defense	# Queries	Dataset	Type	CIFAR10	STL10	SVHN	F-MNIST
N/A	N/A	*Victim*	*ImageNet*	*Train*	90.41±0.02	95.08±0.13	75.47±0.04	91.22 ±0.11
Legit	B4B	50k	CIFAR10	Query	90.27 ±0.07	N/A	N/A	N/A
Legit	B4B	5k	STL10	Query	N/A	95.12 ±0.13	N/A	N/A
Legit	B4B	73k	SVHN	Query	N/A	N/A	74.94 ±0.16	N/A
Legit	B4B	60k	F-MNIST	Query	N/A	N/A	N/A	91.66 ±0.05
Attack	None	50K	ImageNet	Steal	65.2 ±0.03	64.9 ±0.01	62.1 ±0.01	88.5 ±0.01
Attack	B4B	50k	ImageNet	Steal	15.52 ± 0.37	12.57 ± 0.23	19.53± 0.01	23.17 ± 0.01
Attack	None	100K	ImageNet	Steal	68.1 ±0.03	63.1 ±0.01	61.5 ±0.01	89.0 ±0.07
Attack	B4B	100k	ImageNet	Steal	16.27 ± 0.04	13.93 ± 0.35	19.54 ± 0.02	54.69 ± 0.02
Sybil	B4B	50k+50k	ImageNet	Steal	30.14 ± 0.01	29.57 ± 0.08	19.99 ± 0.03	71.72 ± 0.01

Table 4: **Stealing and Using Encoders With and Without our Defense**. The model used in the experiments is Simsiam, with the following parameters for the cost function $\lambda = 10^{-6}$, $\alpha = 1$, and $\beta = 30\%$, and the number of buckets equal to 2^{12}. Since the value of parameter β is decreased substantially to 30%, we observe a drop in accuracy for legitimate users. For example, more than 1% for CIFAR10. In the next Table 5, we show that by also decreasing the parameter α, we can attenuate this harmful effect and retain higher accuracy for legitimate users. In case of an attack, for 100k stealing queries, we observe much lower accuracy levels than for $\beta = 50\%$ shown in Table 3.

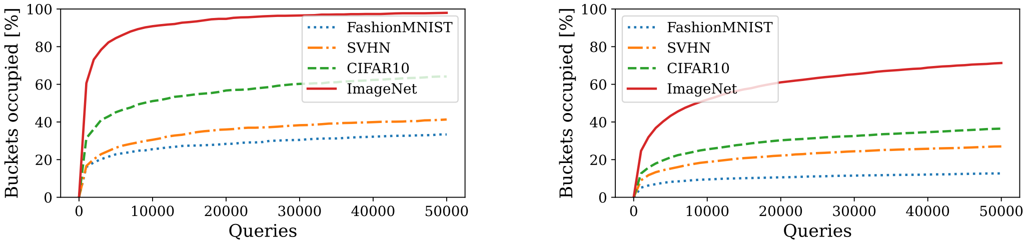
User	Defense	# Queries	Dataset	Type	CIFAR10	STL10	SVHN	F-MNIST
N/A	N/A	*Victim*	*ImageNet*	*Train*	90.41±0.02	95.08±0.13	75.47±0.04	91.22 ±0.11
Legit	B4B	50k	CIFAR10	Query	88.1 ±0.11	N/A	N/A	N/A
Legit	B4B	5k	STL10	Query	N/A	94.92 ±0.11	N/A	N/A
Legit	B4B	73k	SVHN	Query	N/A	N/A	74.37 ±0.02	N/A
Legit	B4B	60k	F-MNIST	Query	N/A	N/A	N/A	91.67 ±0.07
Attack	None	50K	ImageNet	Steal	65.2 ±0.03	64.9 ±0.01	62.1 ±0.01	88.5 ±0.01
Attack	B4B	50k	ImageNet	Steal	30.82 ± 0.09	26.37 ± 0.07	21.87 ± 0.03	66.0 ± 0.02
Attack	None	100K	ImageNet	Steal	68.1 ±0.03	63.1 ±0.01	61.5 ±0.01	89.0 ±0.07
Attack	B4B	100k	ImageNet	Steal	9.57 ± 0.17	9.83 ± 0.09	19.57 ± 0.01	27.06 ± 0.46
Sybil	B4B	50k+50k	ImageNet	Steal	29.15 ± 0.02	28.67 ± 0.06	19.98 ± 0.03	70.62 ± 0.03

Table 5: **Stealing and Using Encoders With and Without our Defense**. The model used in the experiments is Simsiam, with the following parameters for the cost function $\lambda = 10^{-6}$, $\alpha = 0.1$, and $\beta = 30\%$, and the number of buckets equal to 2^{12}. Due to the lower performance on downstream tasks observed in Table 4 while keeping the parameter β fixed to 30% and λ fixed to 10^{-6}, we decrease the value of parameter α to 0.1, which increases the performance of legitimate users on their downstream tasks. In this experiment, we also carry out a sybil attack with more accounts (4 instead of 2), but observe that this modification does not improve the performance of the attacker. With more accounts, a sybil has to sacrifice more queries for the remappings between the representations from different accounts. Additionally, note that each account introduces a different remapping error by the dint of different transformations applied to each account by B4B.

User	Defense	# Queries	Dataset	Type	CIFAR10	STL10	SVHN	F-MNIST
N/A	N/A	*Victim*	*ImageNet*	*Train*	90.41±0.02	95.08±0.13	75.47±0.04	91.22±0.11
Legit	B4B	50k	CIFAR10	Query	90.17 ±0.1	N/A	N/A	N/A
Legit	B4B	5k	STL10	Query	N/A	94.92 ±0.09	N/A	N/A
Legit	B4B	73k	SVHN	Query	N/A	N/A	74.97 ±0.13	N/A
Legit	B4B	60k	F-MNIST	Query	N/A	N/A	N/A	91.71 ±0.08
Attack	None	50K	ImageNet	Steal	65.2 ±0.03	64.9 ±0.01	62.1 ±0.01	88.5 ±0.01
Attack	B4B	50k	ImageNet	Steal	19.95 ±0.19	15.54 ± 0.34	19.57 ± 0.01	23.50 ± 0.19
Attack	None	100K	ImageNet	Steal	68.1 ±0.03	63.1 ±0.01	61.5 ±0.01	89.0 ±0.07
Attack	B4B	100k	ImageNet	Steal	10.35 ± 0.19	12.37 ± 0.69	19.34 ± 0.01	68.93 ± 0.17
Sybil	B4B	4×25k	ImageNet	Steal	33.15 ± 0.04	30.23 ± 0.07	20.87 ± 0.01	72.19 ± 0.02

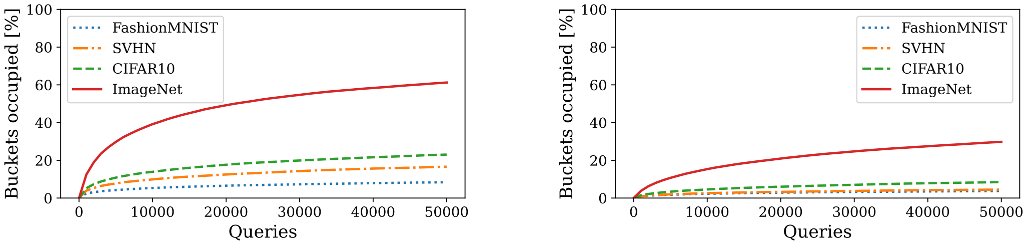
F.4 Setting the number of buckets

We present our procedure to find an optimal number of buckets in Figure 8.



(a) **Number of buckets =** 2^8

(b) **Number of buckets =** 2^{10}



(c) **Number of buckets =** 2^{12}

(d) **Number of buckets =** 2^{14}

Figure 8: **Estimating Embedding Space Coverage through LSH on SimSiam Encoder.** We extend the results from Figure 7a(a) and present the fraction of buckets occupied by representations of different datasets as a function of the number of queries posed to the encoder. We consider different number of buckets in the LSH table. We observe that 2^8 buckets is to small since queries from the ImageNet dataset saturate all the buckets after around 50k queries, while the number 2^{14} of buckets is too large since it is never occupied more than 40%. Thus, the number 2^{12} buckets is a good middle ground. Subfigure (c) corresponds to Figure 7a from the main paper. We also use the same notation and carry out our experiments in the same way as in Figure 7a.

F.5 Results DINO

We show that our defense is also applicable to the DINO encoder. The occupation of the representations space is presented visually in Figure 9. We also show that the number of buckets 2^{12} is optimal for DINO in Figure 10. The impact of transformation on the representations from DINO is shown Table 7. Finally, the end to end experiment for DINO is presented in Table 6.

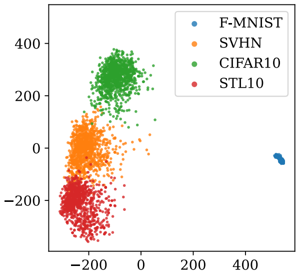


Figure 9: **Representations from Different Tasks Occupy Different Sub-Spaces of the Embedding Space. Presented for FashionMNIST, SVHN, CIFAR10, and STL10.** In this plot, we used the DINO ViT Small encoder trained on ImageNet.

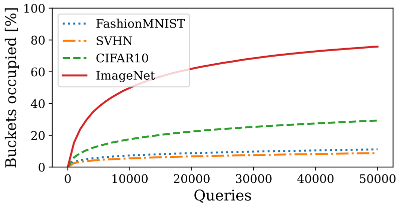


Figure 10: **Estimating Embedding Space Coverage through LSH on the DINO Encoder.** The number of buckets is set to 2^{12}. We also use the same notation and carry out our experiments in the same way as in Figure 7a.

Table 6: **Stealing and Using Encoders With and Without our Defense**. The model used in the experiments is DINO, with the following parameters for the cost function $\lambda = 10^{-6}$, $\alpha = 1000$, and $\beta = 60\%$, and the number of buckets equal to 2^{12}. We have to increase the value of parameter α by $\times 1000$ since the norms of the DINO representations are also around 10^3 higher than for SimSiam. We observe that B4B performs similarly on DINO as for SimSiam.

User	Defense	# Queries	Dataset	Type	CIFAR10	STL10	SVHN	F-MNIST
N/A	N/A	*Victim*	*ImageNet*	*Train*	94.51 ±0.08	97.98 ±0.04	70.66 ±0.16	89.98 ±0.03
Legit	B4B	50k	CIFAR10	Query	94.25 ±0.11	N/A	N/A	N/A
Legit	B4B	5k	STL10	Query	N/A	98.05 ±0.04	N/A	N/A
Legit	B4B	73k	SVHN	Query	N/A	N/A	69.66 ±0.14	N/A
Legit	B4B	60k	F-MNIST	Query	N/A	N/A	N/A	89.68 ±0.01
Attack	None	50K	ImageNet	Steal	67.92 ± 0.04	66.02 ± 0.22	61.30 ± 0.01	89.46 ± 0.01
Attack	B4B	50k	ImageNet	Steal	42.02±0.05	38.91±0.06	19.94±0.02	73.33±0.04
Attack	None	100K	ImageNet	Steal	75.07 ± 0.01	76.32 ± 0.02	71.79 ± 0.06	89.76 ± 0.01
Attack	B4B	100k	ImageNet	Steal	19.27±0.03	21.24±0.03	19.84±0.01	71.01±0.03
Sybil	B4B	50k+50k	ImageNet	Steal	45.56± 0.06	42.50±0.02	24.25±0.03	78.01± 0.08

F.6 Additional evaluation of transformations

Additionally, we show the impact of transformations on the performance of legitimate users in Table 7 (for both SimSiam and DINO).

Table 7: **Impact of Transformations on the Performance for Legitimate Users.** We show that the transformations applied per-account do not harm the performance of legitimate users on their downstream tasks. The victim encoders was trained on the ImageNet dataset using SimSiam and DINO frameworks.

Transformation	Encoder	CIFAR10	STL10	SVHN	F-MNIST
None	*Victim SimSiam*	90.41±0.02	95.08 ±0.13	75.47 ±0.04	91.22 ±0.11
Affine	SimSiam	90.24 ±0.11	95.05 ±0.1	74.96 ±0.18	91.42 ±0.15
Pad+Shuffle	SimSiam	90.4 ±0.05	95.34 ±0.06	75.47 ±0.01	91.38 ±0.15
Affine+Pad+Shuffle	SimSiam	90.18 ±0.06	95.03 ±0.05	74.86 ±0.1	91.35 ±0.1
Binary	SimSiam	88.78 ±0.2	94.72 ±0.02	68.42 ±0.16	88.91 ±0.34
None	*Victim DINO*	94.51 ±0.08	97.98 ±0.04	70.66 ±0.16	89.98 ±0.03
Affine	DINO	94.25 ±0.11	98.05 ±0.04	69.77 ±0.11	89.68 ±0.01
Pad+Shuffle	DINO	94.72 ±0.02	98.07 ±0.03	70.44 ±0.1	89.91 ±0.08
Affine+Pad+Shuffle	DINO	94.26 ±0.06	98.02 ±0.01	69.49 ±0.2	89.70 ±0.1
Binary	DINO	92.96 ±0.1	98.03 ±0.03	59.53 ±0.27	88.26 ±0.04

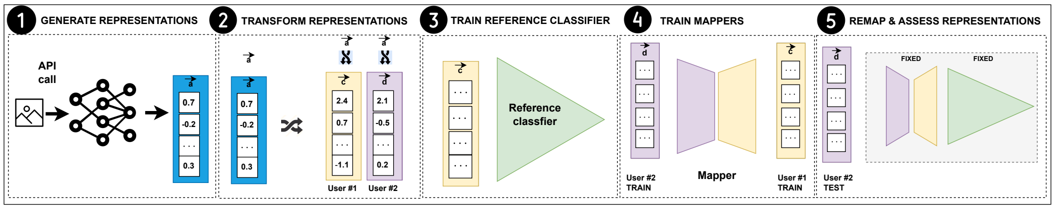


Figure 11: **Protocol to Evaluate the Mapping Between Representations.** We present the protocol of evaluating remappings for two sybil accounts. ❶ API receives inputs from two sybil accounts and generates corresponding representations. ❷ Representations are transformed on a per-user basis and returned. ❸ Adversary trains a reference classifier on representations from account one. ❹ Adversary trains a linear model to find mapping from representations of account two to representations of account one. ❺ To check the quality of obtained mapping representations from test set of account two are mapped using the fixed mapper (from step 4) to representation space of account one. This enables the calculation of cosine distance between representations from account one and their counterparts from account two shown in Figure 5. Additionally, the fixed reference classifier (from step 3) can be used to measure the accuracy drop caused by remapping .