# TML Assignment # 2 - Model Stealing


| Name | ID |
|---|---|
| Rithvika Pervala | 7072443 |
| Bushra Ashfaque | 7070555 |


This report details the technical approach to stealing a B4B-protected SSL encoder, focusing on **noise mitigation** , **model design** , and **training strategy.**

## Overcoming B4B noise:

- **Challenge**: B4B adds **Gaussian noise** to outputs and applies **per-user transformations** to hinder stealing.
- **Solution**:

1. Noise Averaging : Querying the same image multiple times and averaging outputs to reduce B4B noise.
2. Coverage Control : Using a narrow subset of the public dataset (e.g., 10k images) to avoid triggering high noise penalties (B4B increases noise with coverage).
3. Per-User Transformations : Training our model to learn the transformed representation space (B4B applies fixed transformations per user; our model adapts to these implicitly).

## Model Architecture:

The stolen model mimics the victim encoder's behavior using a CNN-based SSL encoder :

```
model = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=3, padding=1),
    nn.MaxPool2d(2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Flatten(),
    nn.Linear(4096, 1024)
)
```

### Design Rationale:

- Matches the input/output dimensions of the victim encoder (3×32×32 → 1024D).
- Uses ReLU + MaxPool for nonlinearity and downsampling, common in SSL architectures

## Training Strategy:

**Objective :** Minimize the MSE (Mean Squared Error) between stolen model outputs and victim's noisy representations .

### Data Preparation :

- Use the public dataset ( `ModelStealingPub.pt` ) with RGB conversion and SSL normalization ( `mean = [0.2980, 0.2962, 0.2987], std = [0.2886, 0.2875, 0.2889]` ).
- Apply data augmentation (ToTensor, Normalize) to improve generalization under B4B noise.

### Training Loop :

```
for epoch in range(20):
    for images, targets in train_loader:
        stolen_reps = model(images.float())
        loss = criterion(stolen_reps, targets.float())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

- **Loss Function** : MSE Loss directly compares stolen and victim outputs.
- **Optimization** : Adam optimizer with `lr=1e-3` updates weights to reduce error.
- **Batch Size** : 32 (smaller than API's 1000-image requirement for efficient training).

---

## Pseudocode:

```
# Noise-Averaged Data Collection
for batch in public_dataset:
    avg_reps = average(model_stealing(batch, num_samples=2))
    store(avg_reps)

# Training
for epoch in range(20):
    for images, targets in DataLoader:
        loss = MSE(model(images), targets)
        update_weights(optimizer, loss)
```

---

## Implementation Challenges :

- **Rate Limit Compliance** : Managing `time.sleep(60)` across queries to avoid API throttling.
- **Dtype Consistency** : Fixing Double vs. Float errors by enforcing `float32` for inputs/outputs.
- **API Instability** : Handling server failures by limiting query volume and using noise-averaged batches.

## Key Highlights:

- **Adaptive Noise Reduction** : Combines B4B's noise-averaging with SSL-inspired data augmentation.

- **Efficient Query Budget Use** : Limits queries to 10 batches (10k images) while respecting rate limits.
- **Simple yet Effective Architecture** : Mimics SSL encoder dynamics without overcomplicating the model.
- L2 distance achieved `{'L2': 5.882108211517334}`

---

## Questions

**Q1 : How did you determine the model is SSL-based?**

The victim encoder outputs high-dimensional representations (1024D), typical of SSL frameworks like SimSiam or DINO. B4B defense targeting SSL encoders [1] and the public dataset's unlabeled nature further confirm it's an SSL model.

**Q2: What attack types are possible?**

In this paper [2] , two methods are mentioned: (1) Direct Extraction (Algorithm 1) trains a stolen model via MSE loss on victim outputs; (2) Projection Head Recreation (Algorithm 2) mimics the victim's nonlinear transformations by aligning augmented views through a learned head. We implemented the first, which is simpler and less query-intensive. Computational and server constraints restricted us from trying second one.

**Q3: Why batch size 32?**

A balance between GPU memory constraints and efficient training. Larger batches (e.g., 128) risk exceeding API query limits (100k queries), while smaller batches (e.g., 8) slow training. 32 strikes a middle ground for speed and resource usage.

**Q4: Why not use InfoNCE (contrastive loss)?**

MSE directly minimizes representation differences under B4B noise, avoiding complexities like paired augmentations and head recreation. InfoNCE requires aligned augmentations and extra training steps, which are impractical given query limits and server instability.

---

## References:

[1] Dubinski, Jan, et al. "Bucks for Buckets (B4B): Active Defenses Against Stealing Encoders." *NeurIPS*. 2023.

[2] Dziedzic, Adam, et al. "On the difficulty of defending self-supervised learning against model extraction." *International Conference on Machine Learning*. PMLR, 2022