# Model Stealing Attack Against B4B-Protected Encoder

Team: 15
Github Link: https://github.com/javedakhtar0129/TML25_A2_15

## Problem Statement

This assignment implements a model stealing attack against a victim encoder protected by the Bucks for Buckets (B4B) defense mechanism. The B4B defense actively protects encoders by tracking the coverage of embedding space used by each client and applying adaptive penalties through representation transformations and noise addition. Our goal was to steal the functionality of this protected encoder while minimizing the L2 distance between the victim's and stolen model's representations, despite the active defense measures in place.

## Constrains

- B4B Defense applies random augmentations & noise to representations
- Budget for queries is limited i.e 100000K.
- We only have access to 30% of the encoder's training data.

## File Structure

- test.ipynb: Baseline implementation of model stealing.
- test_improved.ipynb: Enhanced/optimized implementation.
- ModelStealingPub.pt: Provided Subset of the Encoder's training data. (Not included in the project folder)

## Approach Used

The attack follows a two-phase approach: initial data collection followed by strategic query expansion, all while maintaining coverage tracking to optimize our query budget of 100,000 requests.

### Why use Bucketing ?

- To track explored embedding space and avoid redundant queries. Bucket size = 0.15 (typical for B4B defenses [1] )

### Why capture only the first 5 dimensions of the embedded space ?

- With the assumption that the first 5 dimensions capture most of the embedding space's important variations, this way we were able to use fewer queries to map the space.
- Now our query strategy was to first explore the embedding space with minimal assumption and them exploit it using periodic model training to refine the results and

giude our queries [2]. This way we saved ourselves from spending the entire budget on incorrect initial assumptions.

**Phase 1**
We performed random subsampling because of no prior assumption. By allocating only 5% of our budget we ensured that we can spend the rest 95% once we'll get a better understanding of the embedding space. Our results showed that it explored 3 to 7 percent of the possible buckets in the coverage tracker.

**Phase 2**
Testing showed that reaching approximately 30% coverage of the embedding space buckets is sufficient to train an accurate model while staying under the query limit.

**Model**
Input (3×32×32) →
[Conv2D (64, 3×3, stride 1, pad 1) + BatchNorm + ReLU] →
[Conv2D (128, 3×3, stride 1, pad 1) + BatchNorm + ReLU] →
[Conv2D (256, 3×3, stride 1, pad 1) + BatchNorm + ReLU] →
AdaptiveAvgPool2D (1×1) → FC (512) + ReLU →
FC (1024, output)

**Training approach**
Since we needed to calculate L2 loss, we went with MSE for the loss function.
5 epochs(With assumption, not tested on other epochs) and AdamW optimizer to prevent overfitting.

## Results

The above baseline approach resulted in a L2 distance of 5.7932024002075195.

## Limitations and other observations

- Baseline model performed fairly well with symmetric normalization as well.
- Mentioned are the features we added on top of Baseline implementation to improve the results in test_improved.ipynb, jittered and variable delays while querying B4B protected encoder, PCA-guided sampling, Deeper CNN with regularization and gradient clipping, but were not able to fetch results due to server issues.

## References:

[1] "Bucks for Buckets: Model Stealing Defense via Distribution Transformation"
*NeurIPS 2023*
[2] "Stealing Machine Learning Models via Prediction APIs"
*USENIX Security Symposium 2016*