TML ASSIGNMENT 2

MODEL STEALING ATTACK

TEAM 16

AHRAR BIN ASLAM

MUHAMMAD MUBEEN SIDDIQUI

In this assignment, we implemented a model stealing attack targeting a remote encoder API. The goal was to replicate the functionality of a black-box encoder by querying it with carefully selected inputs and then training a surrogate model on the returned outputs. Our final solution is based on training a ResNet18 encoder to mimic the representations returned by the victim model.

As an attacker, we had access to:

1. Trained encoder of unknown architecture hidden behind an API and protected using B4B noising.
2. Subset of encoder's training data. The structure of the dataset was similar to the previous assignment.
3. An API to send queries to.

Important Parameters for this Assignment:

TOKEN = "93145372"

SEED = "53027020"

PORT = "9935"

Our Approach: You can find our full implementation in Model Stealing TML25_A2_16.ipynb. Here's a summary of the process we followed:

1. Dataset Preparation

We started by loading a public dataset of images (ModelStealingPub.pt) that would serve as inputs to the victim model. We randomly selected 1000 images and queried the victim's API to obtain their embeddings. This step was necessary because we only have access to the victim model's output embeddings, not its internal parameters. The stolen embeddings were saved locally so we could efficiently train our model without repeated API calls.

2. Data Augmentation
To improve our encoder's generalization and robustness, we applied data augmentations such as random horizontal flips, cropping, color jittering, and grayscale conversion during training. For validation, we only normalized images to maintain consistent evaluation. These augmentations help our model better learn the victim's embedding patterns under varied input conditions.

3. Model Architecture
We experimented with two encoder designs:

- A custom ResNet-like encoder built from scratch, allowing us to tailor the architecture for the task and dataset.
- A modified ResNet18 encoder without pretraining, adapted for smaller image inputs (32x32).
  Both encoders project input images into a 1024-dimensional embedding space to align with the victim's output size. This flexibility allowed us to assess which architecture better captures the victim's embedding space.

4. Loss Function
We combined cosine similarity and mean squared error (MSE) losses to train our encoder. This hybrid loss ensures that not only are the predicted embeddings close in magnitude to the victim's, but they also maintain the correct directional relationships. This was essential for effectively mimicking the victim model's behavior.

5. Training Setup
We trained the encoder using the Adam optimizer with learning rate scheduling for stable convergence. Training was done on GPU where available to accelerate computation. We ran 20–40 epochs with a batch size of 32, monitoring the loss to track improvement. This setup balanced training speed and performance.

6. Evaluation
Using this approach, we achieved an L2 distance of **6.19** on the actual encoder on the server, indicating a close replication of the victim's embedding space and validating the effectiveness of our approach.

Approach 2:

We also tried a shadow encoder-based framework to simulate the behavior of the target encoder under various defense mechanisms [1]. Specifically, we trained multiple shadow encoders on a range of publicly available datasets, each shadow encoder corresponding to a particular defense strategy applied to the feature vectors. This setup allowed us to mimic the perturbations introduced by defenses such as adversarial training, feature squeezing, or randomized transformations.

The goal was first, to train meta-classifiers capable of accurately detecting the presence and type of defense applied by analyzing the perturbed feature vectors; second, to develop generative models that could recover the original, unperturbed feature representations from the defended outputs. By aggregating knowledge across diverse datasets and defense methods, the model aimed to generalize well in identifying and reversing defense-induced perturbations.

However, this methodology requires a substantial computational burden. Training multiple shadow encoders individually for each dataset-defense pair led to significant increases in training time and GPU memory consumption. Each shadow encoder required hyperparameter tuning and convergence checks, multiplying the overall resource requirements. Additionally, the meta-classifier and generative model training phases further compounded the computational costs due to the increased complexity and the volume of training samples generated from shadow models.

Additional Information: Due to resource constraints, all experiments and model training were conducted on Google Colab utilizing its free GPU runtime, rather than running locally via VS Code or any desktop environment. We have provided the complete implementation in .ipynb (Notebook). Please note that the full pipeline execution takes approximately 15 minutes to complete on Colab.

Github Repository: https://github.com/mubeensiddiqui99/TML25_A2_16

Embeddings Folder: Embeddings

References:

[1] X. Ren, H. Liang, Y. Wang, C. Zhang, Z. Xiong, and L. Zhu, "BESA: Boosting Encoder Stealing Attack with Perturbation Recovery," *arXiv preprint arXiv:2506.04556*, 2025. https://arxiv.org/abs/2506.04556