Optimization for Data Science

Frank-Wolfe Methods for Adversarial Attacks

Academic Year 2024/2025

Elia Carta	2156440
Jianan E	2144167
Gianfranco Mauro	2160477
Davide Volpi	2140728

Contents

1 Introduction

This project investigates the application of various optimization methods to generate white-box adversarial attacks on machine learning models. More specifically, building on the theoretical foundations of the Frank-Wolfe (FW) algorithm and its variants, alongside Projected Gradient Descent, we implement these algorithms and evaluate their performance in crafting adversarial examples on real-world datasets. Our approach for generating attacks follows the methodologies outlined in Section 3.5 of [1] and Section 5 of [2], drawing insights from the other four papers shared by the professors.

Adversarial examples are maliciously perturbed inputs designed to mislead a trained machine learning model at test time. An adversarial attack, therefore, consists of taking a correctly classified data point $\mathbf{x}_{\text{ori}}$ and slightly modifying it to create a new data point that the model misclassifies.

Attacks can be categorized by their purpose into two main types: *untargeted* and *targeted*. Untargeted attacks aim to change the prediction to any incorrect label, whereas targeted attacks aim to misclassify an input as a specific target class. In this report, we focus on the more challenging targeted attacks, noting that the presented algorithms can be easily adapted for untargeted scenarios.

2 Methodology

2.1 Notation

Throughout this report, scalars are denoted by lowercase letters (e.g., ϵ), vectors by lowercase bold letters (e.g., $\mathbf{x}$), and sets by calligraphic uppercase letters (e.g., $\mathcal{D}$).

For a vector $\mathbf{x} \in \mathbb{R}^d$, the L_p-norm is defined as $\|\mathbf{x}\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$. In particular, the L_∞-norm is given by $\|\mathbf{x}\|_\infty = \max_{i=1,\dots,d} |x_i|$. We denote the projection of a vector $\mathbf{x}$ onto a set $\mathcal{D}$ as $\mathcal{P}_\mathcal{D}(\mathbf{x})$.

2.2 Problem Formulation

Let $\ell(\mathbf{x}, y_{\text{tar}})$ be the classification loss function of a model for an input $\mathbf{x} \in \mathbb{R}^d$ and a target label y_{tar}. To generate a targeted adversarial attack, we aim to find an adversarial example $\mathbf{x}$ by minimizing this loss function with respect to the target class. The problem can be formulated as the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & f(\mathbf{x}) = \ell(\mathbf{x}, y_{\text{tar}}) \\ \text{subject to} \quad & \|\mathbf{x} - \mathbf{x}_{\text{ori}}\|_p \leq \epsilon \end{aligned} \tag{1}$$

The constraint set $\mathcal{D} := \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{x}_{\text{ori}}\|_p \leq \epsilon\}$ is a compact and convex set for $p \geq 1$.

While Projected Gradient Descent (PGD) can achieve a high attack success rate, its update rule requires a projection step at each iteration to ensure the iterates remain within the feasible set $\mathcal{D}$. This can lead to adversarial examples that lie on the boundary of the constraint set, resulting in unnecessarily large distortion, while Frank-Wolfe methods offer a projection-free alternative.

While line search strategies can be effective for FW algorithms, they may become computationally expensive for high-dimensional inputs, as they require multiple forward/backward passes. In our setting, a predefined step-size rule, such as $\gamma_k = \frac{2}{k+2}$, is often preferred to balance the cost per iteration against the overall convergence speed [5].

2.3 Projected Gradient Descent (PGD)

The Projected Gradient Descent (PGD) is an iterative gradient descent method widely used to perform adversarial attacks. In the context of constrained problems, the standard gradient method cannot be applied because the new point

$$\mathbf{x}_{k+1} = \mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)$$

might not belong to the closed and convex set $\mathcal{D}$. For this reason, we search for the point in $\mathcal{D}$ nearest to the one obtained with a projection, which is defined as:

$$\mathcal{P}_\mathcal{D}(\bar{\mathbf{x}}) = \min_{\mathbf{x} \in \mathcal{D}} \frac{1}{2} \|\mathbf{x} - \bar{\mathbf{x}}\|$$

There exist many types of algorithms to implement the projection; in this case, we applied the one proposed in [4] that has been observed in practice to have a complexity of $O(N)$. With the initial point $\mathbf{x}_1 \in \mathcal{D}$, for each iteration k, the algorithm performs the following steps:

$$\hat{\mathbf{x}}_k := \mathcal{P}_\mathcal{D}(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)), \quad s_k > 0 \tag{2}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(\hat{\mathbf{x}}_k - \mathbf{x}_k), \quad \alpha_k \in (0, 1] \tag{3}$$

When the difference between the previous point and the new point is less than a default tolerance, the algorithm stops.

This method allows to achieve high attack success rates in a few iterations. On the other hand, it usually generates adversarial examples near the boundary of the perturbation set, and this causes a large distortion in the resulting adversarial example.

2.4 Frank-Wolfe (FW)

The foundational problem that Frank-Wolfe methods address is the general constrained optimization problem:

$$\min_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x}) \tag{4}$$

where:

- $f : \mathbb{R}^d \to \mathbb{R}$ is a convex and continuously differentiable objective function. Its gradient, $\nabla f(\mathbf{x})$, is assumed to be L-Lipschitz continuous, meaning there exists a constant $L > 0$ such that for all $\mathbf{x}, \mathbf{y} \in \mathcal{D}$:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\| \tag{5}$$

  This property is fundamental for the convergence analysis of first-order methods [1].

- $\mathcal{D} \subset \mathbb{R}^d$ is a compact (i.e., closed and bounded) and convex feasible set. This ensures that a minimizer exists and that linear functions achieve their minimum over $\mathcal{D}$ at an extreme point (vertex).

The Frank-Wolfe algorithm is an iterative method that avoids projections. At each iteration k, given the current iterate $\mathbf{x}^{(k)}$, the algorithm proceeds in two main steps:

1. **Find the FW Vertex:** A linear approximation of f around $\mathbf{x}^{(k)}$ is minimized over the feasible set $\mathcal{D}$. This is equivalent to finding the vertex $\mathbf{s}^{(k)}$ that minimizes the inner product with the gradient:

$$\mathbf{s}^{(k)} := \arg\min_{\mathbf{s}\in\mathcal{D}} \langle \mathbf{s}, \nabla f(\mathbf{x}^{(k)})\rangle \tag{6}$$

   This subproblem is solved by a Linear Minimization Oracle (LMO), a routine designed to solve linear programs over the set $\mathcal{D}$ [5].

2. **Update the Iterate:** The next iterate $\mathbf{x}^{(k+1)}$ is computed by taking a step from $\mathbf{x}^{(k)}$ towards the FW vertex $\mathbf{s}^{(k)}$ via a convex combination:

$$\mathbf{x}^{(k+1)} := (1 - \gamma_k)\mathbf{x}^{(k)} + \gamma_k \mathbf{s}^{(k)} \tag{7}$$

   where $\gamma_k \in [0, 1]$ is the step size.

2.5 Away-Step Frank-Wolfe (AFW)

The key idea behind the Away-steps Frank-Wolfe algorithm is to fix the "zig-zagging" problem and the slow speed at which it finds solutions, especially when the best answer is located at the edge of the possible solution area [6]. AFW achieves this by allowing "away steps" in addition to the usual "Frank-Wolfe steps."

With the initial point $\mathbf{x}^{(0)} \in \mathcal{D}$, the initial active set $S^{(0)} := \{\mathbf{x}^{(0)}\}$ (with $\alpha^{(0)}_{\mathbf{x}^{(0)}} = 1$ and 0 otherwise). For each iteration k, the algorithm performs the following steps:

1. **Determine FW Atom, Direction and Gap**

$$\mathbf{s}_k := \mathrm{LMO}_{\mathcal{D}}(\nabla f(\mathbf{x}^{(k)}))$$
$$\mathbf{d}_k^{\mathrm{FW}} := \mathbf{s}_k - \mathbf{x}^{(k)}$$
$$g_k^{\mathrm{FW}} := -\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\mathrm{FW}}\rangle$$

2. **Determine Away Atom, Direction and Gap**

$$\mathbf{v}_k := \arg\max_{\mathbf{v}\in S^{(k)}} \langle \nabla f(\mathbf{x}^{(k)}), \mathbf{v}\rangle$$
$$\mathbf{d}_k^{\mathrm{A}} := \mathbf{x}^{(k)} - \mathbf{v}_k$$
$$g_k^{\mathrm{A}} := -\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\mathrm{A}}\rangle$$

3. **Examine FW Gap and Away Gap, determine Direction and Max Step Size**
   If $g_k^{\text{FW}} \leq \epsilon$ (where ϵ is a small tolerance), then return $\mathbf{x}^{(k)}$ and terminate.
   If $-\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\text{FW}} \rangle \geq -\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\text{A}} \rangle$ Choose the FW direction:
$$\mathbf{d}_k := \mathbf{d}_k^{\text{FW}}$$
$$\gamma_{\max} := 1$$
   If $-\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\text{FW}} \rangle < -\langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d}_k^{\text{A}} \rangle$ Choose the away direction:
$$\mathbf{d}_k := \mathbf{d}_k^{\text{A}}$$
$$\gamma_{\max} := \alpha_{\mathbf{v}_k}^{(k)} / (1 - \alpha_{\mathbf{v}_k}^{(k)})$$
   (the maximum feasible step-size ensuring the new iterate stays within the convex hull of $S^{(k)}$)

4. **Line-Search Find Step Size:** Find the optimal step size $\gamma_k \in [0, \gamma_{\max}]$ that minimizes the objective function along the chosen direction:
$$\gamma_k \in \arg \min_{\gamma \in [0, \gamma_{\max}]} f(\mathbf{x}^{(k)} + \gamma \mathbf{d}_k)$$

5. **Update X based on Optimized Step Size:**
$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \gamma_k \mathbf{d}_k$$

6. **Update Active Set ($S^{(k+1)}$):** The active set $S^{(k+1)}$ is updated to contain all atoms $\mathbf{v} \in \mathcal{D}$ for which $\alpha_{\mathbf{v}}^{(k+1)} > 0$.
   - If FW step and $\gamma_k = 1$: $S^{(k+1)} = \{\mathbf{s}_k\}$. Otherwise: $S^{(k+1)} = S^{(k)} \cup \{\mathbf{s}_k\}$.
   - If away step and $\gamma_k = \gamma_{\max}$ (a "drop step"): $S^{(k+1)} = S^{(k)} \setminus \{\mathbf{v}_k\}$. Otherwise: $S^{(k+1)} = S^{(k)}$.

7. **Update Weights ($\alpha^{(k+1)}$):** The weights $\alpha_{\mathbf{v}}^{(k+1)}$ for $\mathbf{v} \in \mathcal{D}$ are updated based on the type of step:
   - **If FW step was chosen ($\mathbf{d}_k = \mathbf{d}_k^{\text{FW}}$):**
$$\alpha_{\mathbf{s}_k}^{(k+1)} := (1 - \gamma_k) \alpha_{\mathbf{s}_k}^{(k)} + \gamma_k$$
     $\alpha_{\mathbf{v}}^{(k+1)} := (1 - \gamma_k) \alpha_{\mathbf{v}}^{(k)}$ for $\mathbf{v} \in S^{(k)} \setminus \{\mathbf{s}_k\}$
   - **If away step was chosen ($\mathbf{d}_k = \mathbf{d}_k^{\text{A}}$):**
$$\alpha_{\mathbf{v}_k}^{(k+1)} := (1 + \gamma_k) \alpha_{\mathbf{v}_k}^{(k)} - \gamma_k$$
     $\alpha_{\mathbf{v}}^{(k+1)} := (1 + \gamma_k) \alpha_{\mathbf{v}}^{(k)}$ for $\mathbf{v} \in S^{(k)} \setminus \{\mathbf{v}_k\}$

These "away steps" are crucial because they allow the search to not only find the best direction but also to move away from less effective or "worst" directions. By comparing which step offers the biggest improvement (how much they can reduce the 'dual gap' or difference from the ideal solution), the algorithm becomes more efficient in its search.

2.6 Pairwise Frank-Wolfe (PFW)

The key idea of the Pairwise Frank-Wolfe (PFW) algorithm is to move weight mass between two atoms at each step. The current iterate $\mathbf{x}^{(k)}$ is represented as a convex combination of atoms (vertices) in its active set $\mathcal{A}_k \subseteq \mathcal{D}$:
$$\mathbf{x}^{(k)} = \sum_{\mathbf{v} \in \mathcal{A}_k} \alpha_{\mathbf{v}}^{(k)} \mathbf{v}, \quad \text{with} \quad \sum_{\mathbf{v} \in \mathcal{A}_k} \alpha_{\mathbf{v}}^{(k)} = 1 \text{ and } \alpha_{\mathbf{v}}^{(k)} > 0. \tag{8}$$

At each iteration k, the algorithm performs the following steps:

1. **Select FW Atom:** Find the standard FW vertex $\mathbf{s}^{(k)}$ by minimizing the linear objective over the entire feasible set $\mathcal{D}$.
$$\mathbf{s}^{(k)} = \text{LMO}_{\mathcal{D}}(\nabla f(\mathbf{x}^{(k)})) := \arg \min_{\mathbf{s} \in \mathcal{D}} \langle \nabla f(\mathbf{x}^{(k)}), \mathbf{s} \rangle \tag{9}$$
   The FW direction is $\mathbf{d}_{\text{FW}}^{(k)} = \mathbf{s}^{(k)} - \mathbf{x}^{(k)}$.

2. **Select Away Atom:** Find the "worst" atom $\mathbf{v}^{(k)}$ in the current active set $\mathcal{A}_k$ by maximizing the linear objective.
$$\mathbf{v}^{(k)} = \arg\max_{\mathbf{v}\in\mathcal{A}_k} \langle \nabla f(\mathbf{x}^{(k)}), \mathbf{v}\rangle \tag{10}$$
The away direction is $\mathbf{d}_{\mathrm{A}}^{(k)} = \mathbf{x}^{(k)} - \mathbf{v}^{(k)}$.

3. **Form Pairwise Direction:** The pairwise direction is formed by combining the FW and away directions, which moves mass from the away atom $\mathbf{v}^{(k)}$ to the FW atom $\mathbf{s}^{(k)}$.
$$\mathbf{d}_{\mathrm{PFW}}^{(k)} = \mathbf{s}^{(k)} - \mathbf{v}^{(k)} = \mathbf{d}_{\mathrm{FW}}^{(k)} + \mathbf{d}_{\mathrm{A}}^{(k)} \tag{11}$$

4. **Step-Size and Update:** The maximum feasible step size is the weight of the away atom, $\gamma_{\max} = \alpha_{\mathbf{v}^{(k)}}^{(k)}$. The step size γ_k is chosen to minimize the objective along the pairwise direction, typically via line search:
$$\gamma_k = \arg\min_{0\le\gamma\le\gamma_{\max}} f\left(\mathbf{x}^{(k)} + \gamma \mathbf{d}_{\mathrm{PFW}}^{(k)}\right) \tag{12}$$
Finally, the next iterate is calculated as:
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma_k \mathbf{d}_{\mathrm{PFW}}^{(k)} \tag{13}$$
The active set $\mathcal{A}_{k+1}$ is then updated based on the new non-zero weights $\alpha_{\mathbf{v}}^{(k+1)}$.

Unlike the classic FW algorithm, PFW can remove atoms from the active set, which helps enforce sparsity and can lead to faster convergence. Finding $\mathbf{s}^{(k)}$ requires one LMO call over $\mathcal{D}$, while finding $\mathbf{v}^{(k)}$ only requires a search over the typically much smaller active set $\mathcal{A}_k$, making it computationally cheaper [6].

2.7 White-Box Framework

The proposed Projected Gradient and Frank-Wolfe based white-box attack algorithms are built upon the classic Projected Gradient and Frank-Wolfe variants algorithms. The key difference between the proposed white-box algorithms and the classic ones is the adaptation to the adversarial attack domain, which fundamentally transforms the optimization method to handle neural network manipulation.

The most significant modification lies in the objective function specialization, where the classic algorithm's general convex optimization is replaced with the specific goal of maximizing the negative cross-entropy loss $-\ell(\mathbf{x} + \boldsymbol{\delta}, y_{\text{target}})$. This shift necessitates a fundamental change in the optimization space itself, as the white-box algorithm operates in the perturbation space $\boldsymbol{\delta}_k$ rather than directly in the solution space $\mathbf{x}_k$, constructing adversarial examples as $\mathbf{x}_{\text{adv}} = \mathbf{x}_{\text{clean}} + \boldsymbol{\delta}_k$.

The Linear Minimization Oracle is correspondingly specialized for L_∞-ball constraints. Since the LMO must solve $\arg\min_{\mathbf{s}} \langle \mathbf{s}, \nabla f(\mathbf{x}^{(k)})\rangle$, the solution lies in the opposite direction of the gradient. For an L_∞-ball of radius ϵ, this is computed as $\text{LMO}(\nabla f(\mathbf{x}^{(k)})) = -\epsilon \cdot \text{sign}(\nabla f(\mathbf{x}^{(k)}))$, enabling attacks on arbitrary differentiable models.

The convergence criteria incorporate an early stopping mechanism that terminates the algorithm upon attack success (i.e., when the model's predicted class for $\mathbf{x}_{\text{adv}}$ equals y_{target}), complementing the traditional duality gap-based stopping condition.

In the PGD and FW algorithm, an additional momentum term, initialized with the original image $\mathbf{x}_0$ as $\mathbf{m}_{-1} = \nabla f(\mathbf{x}_0)$ and then updated during the iteration as $\mathbf{m}_k = \beta \cdot \mathbf{m}_{k-1} + (1-\beta)\cdot \nabla f(\mathbf{x}_k)$ used to compute the vertex $\mathbf{s}^{(k)}$ instead of using the gradient, is introduced to help smooth noisy gradients, achieve faster convergence, improve success rates by helping escape local minima, and enhance stability in the optimization direction.

3 Experiments

In this section, we present the experimental results for our Frank-Wolfe algorithm variants and the Projected Gradient for white-box adversarial attacks.

3.1 Evaluation Setup

We compare the performance of all attack algorithms by evaluating on the CIFAR-10 [7] dataset and the CIFAKE [8] dataset, a dataset that contains 60,000 synthetically-generated images and 60,000 real images (collected from CIFAR-10), which contains two classes: REAL and FAKE. For CIFAR-10, we attacked a pre-trained ResNet50 model, pre-trained on the ImageNet dataset, and mapped each class of the CIFAR-10 dataset to similar classes of the ImageNet dataset. For CIFAKE, we used a pre-trained ViT based on the Google's "vit-base-patch16-224-in21k" model, which was fine-tuned on CIFAKE, and is reported to have a 98.25% accuracy.

After performing the prediction step, we randomly chose 100 images that were verified to be correctly classified by the pre-trained models (to ensure the initial correct classification we add also further controls in the experiments).

3.2 White-Box Attack Experiments

In this subsection, we present the white-box attack experiments on both CIFAR-10 and CIFAKE datasets. We choose $\epsilon = 8/255$ for both datasets. For comparison, we report the average success rate of the attack (ASR), the average number of iterations, and the average distortion for each method.

Tables 1 and 2 present our experimental results for the white-box attack experiments.

On the CIFAR-10 dataset (Table 1), Momentum based PGD (PGD-M) achieves a 100% attack success rate (ASR) with the least iterations (5.32), making it the most efficient method in this scenario, outperforming the standard PGD (PGD-S) algorithm. Momentum based FW (FW-M) also reaches 100% ASR but requires slightly more iterations. AFW is slightly less successful (98% ASR), while PFW performs significantly worse, with 85% ASR and the highest number of iterations. The distortion of both FW and AFW algorithms are very close to the perturbation limit ϵ, which indicates that their generated adversarial examples are near or upon the boundary of the constraint set. On the other hand, the momentum based PGD whitebox attack algorithm achieves not only the smallest average number of iterations per attack, but also the smallest distortion among the algorithms.

Methods	**ASR(%)**	**#Iterations**	**Distortion**
PGD-S	**100**	6.52	0.0277
PGD-M	**100**	**5.32**	**0.0260**
FW-M	**100**	6.23	0.0290
AFW	98	7.20	0.0292
PFW	85	9.42	0.0266

Table 1: Comparison of targeted L_∞ norm based white-box attacks on ResNet50 model on CIFAR-10 dataset with $\epsilon = 8/255$.

On the CIFAKE dataset (Table 2), all the algorithms achieve a 100% ASR, with the momentum based FW algorithm achieving the least average number of iterations (2.39), making it the most efficient method in this scenario, outperforming the momentum based PGD algorithm and the other FW variants. On the other hand, momentum based PGD reaches the smallest average distortion (0.0125). All the algorithms have nearly the same number of iterations (between 2 and 3), making them all suitable for this task. The distortions of all the algorithms are very far to the perturbation limit ϵ, which indicates that their generated adversarial examples are not near or upon the boundary of the constraint set.

Methods	**ASR(%)**	**#Iterations**	**Distortion**
PGD-S	**100**	2.61	0.0126
PGD-M	**100**	2.61	**0.0125**
FW-M	**100**	**2.39**	0.0226
AFW	**100**	2.42	0.0227
PFW	**100**	2.44	0.0234

Table 2: Comparison of targeted L_∞ norm based white-box attacks on ViT model on CIFAKE dataset with $\epsilon = 8/255$.

3.3 Experiments on Adversarially Trained Models

In this subsection, we test our attack algorithms on a more challenging adversarially trained model to evaluate their robustness. We apply the attacks to the adversarially trained Inception_v3 model [9] on the ImageNet1k dataset [10], with $\epsilon = 8/255$.

Table 3 presents the results on the ImageNet1k dataset. Momentum based FW achieves a 65% ASR with the least average number of iterations (15), making it the most efficient method in this settings, outperforming the other algorithms under these metrics. On the other hand, PFW reaches the smallest average distortion (0.0294), but the distortion of all the algorithms is very close to the perturbation limit ϵ, which indicates that their generated adversarial examples are near or upon the boundary of the constraint set.

Methods	**ASR(%)**	**#Iterations**	**Distortion**
PGD-S	57	15.95	0.0312
PGD-M	61	15.25	0.0313
FW-M	**65**	**15**	0.0309
AFW	47	17.18	0.0311
PFW	19	18.49	**0.0294**

Table 3: Comparison of targeted L_∞ norm based white-box attacks on adversarially trained inception_v3 on ImageNet1k dataset with $\epsilon = 8/255$.

4 Conclusions

In this project, we implemented and evaluated several first-order optimization methods for generating targeted white-box adversarial attacks. Our experiments on standard and adversarially trained models give several key insights.

For standard image classification models, all tested algorithms proved highly effective, achieving near-perfect attack success rates. Projected Gradient Descent with momentum was most efficient on the ResNet50 on CIFAR-10 model, while Frank-Wolfe with momentum got better results on ViT on CIFAKE, suggesting that the best choice of algorithm can be model (and data) dependent.

When attacking a robust, adversarially trained Inception_v3 model, the task became significantly more challenging. The FW with momentum algorithm variant demonstrated the best performance, achieving the highest success rate with the fewest iterations. This is an insightful result, as it positions the simplest projection-free method as a strong candidate for attacking adversarially trained models. On the other hand, the more complex variants like AFW and especially PFW showed a significant decline in effectiveness.

Overall, our findings confirm that while PGD is a powerful and reliable baseline for adversarial attacks, Frank-Wolfe methods, particularly the momentum variant, offer a compelling, projection-free alternative that can be both efficient and highly effective, especially in challenging scenarios where models are specifically trained to be less subject to attacks of this sort.

References

[1] I. M. Bomze, F. Rinaldi, and D. Zeffiro. Frank-Wolfe and friends: a journey into projection-free first-order optimization methods. *4OR*, 19:313–345, 2021.

[2] J. Chen, D. Zhou, J. Yi, and Q. Gu. A Frank-Wolfe framework for efficient and effective adversarial attacks. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 3486–3494, 2020.

[3] C. W. Combettes and S. Pokutta. Complexity of linear minimization and projection on some sets. *arXiv preprint arXiv:2101.10040v2*, 2021.

[4] L. Condat. Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1):575–585, 2016.

[5] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[6] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Advances in Neural Information Processing Systems*, 28, 2015.

[7] Krizhevsky, A. & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[8] Bird, J.J. and Lotfi, A., 2024. CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. IEEE Access.

[9] Ross Wightman. *inception_v3.tf_adv_in1k*, Hugging Face, 2022. Available at: https://huggingface.co/timm/inception_v3.tf_adv_in1k

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. *ImageNet: A Large-Scale Hierarchical Image Database*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. Available at: https://www.image-net.org/download.php.