THE *Lazy* ADVERSARY: DATA-EFFICIENT UNIVERSAL PERTURBATIONS FOR TABULAR CLASSIFIERS VIA CORESETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Universal Adversarial Perturbations (UAPs) are single, input-agnostic perturbations that can induce misclassification in a frozen classifier. Gradient-based UAP methods, which frame the problem as loss maximization and solve it with the Fast Gradient Sign Method (FGSM) or Projected Gradient Descent (PGD) over the training set, have become the dominant approach due to their simplicity and effectiveness. However, these methods require aggregating gradients over the entire training set at every optimization epoch, making the per-epoch cost scale linearly with the dataset size. We observe that because the victim classifier is frozen, its penultimate feature embedding never changes. We propose UAPCore, which uses this frozen feature geometry as a practical proxy for gradient redundancy: before any gradient update, a coreset is selected in the feature space of the frozen victim model, and all subsequent UAP optimization is performed exclusively on that fixed subset. Whether this proxy preserves UAP quality is evaluated thoroughly with experiments across 44 tabular classification datasets from OpenML, which show that UAPCore with k-Center selection at only 10% of the training set recovers full-dataset fooling rates to within 2 percentage points on 72-91% of the datasets, while achieving a median wall-clock speedup of $8.5\times$ under PGD. We further show that MLP-crafted coreset perturbations transfer to unseen tree-based and attention-based tabular architectures with fidelity comparable to full-data crafting.

1 INTRODUCTION

Deep learning models are vulnerable to adversarial examples: small, carefully crafted perturbations that cause a trained classifier to misclassify while leaving semantics unchanged to a human observer (Szegedy et al., 2013). A particularly consequential variant is the *Universal Adversarial Perturbation* (UAP) (Moosavi-Dezfooli et al., 2017; Zhang et al., 2021): a single, input-agnostic perturbation that, when added to *any* input from the data distribution, induces misclassification with high probability. Unlike instance-specific attacks, UAPs expose systemic vulnerabilities that persist across the entire input space. An adversary can exploit them without access to individual samples at deployment time, making UAPs a practically important threat model for auditing robustness in security-critical applications (Hirano et al., 2021; Gajjar et al., 2025; Goldblum et al., 2021; Lu et al., 2026).

Gradient-based UAP methods, as formalized by (Shafahi et al., 2020), recast the problem as maximising the classifier's loss over the training set and solve it with FGSM (Goodfellow et al., 2014) or PGD (Madry et al., 2017). This has become the standard practice for UAP generation (Nicolae et al., 2019; Gajjar et al., 2025). However, each epoch of gradient-based UAP optimization requires a full pass over the entire training set to accumulate gradients, causing the per-epoch cost to scale linearly with the number of training samples and the total runtime to grow multiplicatively with the number of epochs and inner optimization steps. For image classifiers, this burden has motivated data-free attacks that maximise CNN layer activations without any training data (Mopuri et al., 2017) and generative methods that learn perturbation distributions (Poursaeed et al., 2018; Hayes & Danezis, 2018). These advances are effective precisely because they leverage properties of the image modality: spatial correlation in pixel space, translation invariance in convolutional features, and the ability to synthesise plausible class prototypes from model parameters. Tabular data, by contrast, possess none of these

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

properties: the features are heterogeneous, and there is no spatial or sequential structure to exploit. Models range from multilayer perceptrons to tree-based ensembles that lack the stationary feature embeddings on which data-free methods rely. Consequently, the image-centric UAP acceleration literature offers no practical recipe for tabular classifiers, and the handful of tabular adversarial studies that exist focus on instance-specific attacks or feature-restricted perturbations rather than universal ones (Ballet et al., 2019; Kireev et al., 2022). Hence, on tabular data, which dominates enterprise machine learning in finance, healthcare, and operations (Borisov et al., 2022), standard full-dataset gradient estimation remains the de facto approach for UAPs, representing a fundamental inefficiency.

We believe that the inefficiency of full-dataset UAP estimation is not merely an algorithmic inconvenience; it is *structural*. Because the victim classifier f_θ is held fixed throughout the UAP generation process, its penultimate feature embedding ϕ_θ never changes, so the feature-space representation used for sample selection is stationary throughout optimization. We treat distances in this frozen feature space as a *proxy* for gradient redundancy, and we test empirically whether this proxy identifies a subset that preserves UAP optimization. This observation leads to the following question:

> If a subset of samples captures a similar overall gradient and decision boundary to the full dataset $\mathcal{D}$, optimising over this subset should yield the same universal perturbation. *Can we efficiently find such an optimal subset to drastically reduce computation without sacrificing the UAP's fooling rate?*

We answer this question by framing universal adversarial optimization through the lens of *coresets* (Mirzasoleiman et al., 2020): small, mathematically curated subsets designed to approximate the geometric or gradient properties of the full dataset. Coresets have traditionally been leveraged to accelerate supervised model training; here we show that they are also well-suited for accelerating UAP optimization. In standard training, a model's evolving weights require expensive, iterative subset updates; in UAP generation, the victim model is fixed. This stationarity means a single, one-time coreset selection can guide the entire optimization trajectory, reducing per-epoch computational cost by a factor of $1/\rho$.

Contributions. We propose **UAPCore**, a two-phase framework for *Core*set-accelerated *UAP* computation. In the selection phase, UAPCore extracts penultimate features from the frozen f_θ for all N training points and selects a geometrically representative subset $\mathcal{C} \subset \mathcal{D}$ of size $k = \lfloor \rho |\mathcal{D}| \rfloor$. In the attack phase, all gradient updates are computed exclusively on this fixed subset, reducing per-epoch cost from $\mathcal{O}(N)$ to $\mathcal{O}(\rho N)$. We instantiate UAPCore with two standard gradient-based attack primitives, FGSM (Goodfellow et al., 2014) and PGD (Madry et al., 2017), and evaluate several coreset strategies. Experiments on 44 tabular classification datasets from OpenML (Vanschoren et al., 2014) show that UAPCore with k-Center selection at $\rho = 0.10$ matches full-dataset fooling rates to within 2 percentage points on up to 91% of datasets, and achieves a median $8.5\times$ wall-clock speedup under PGD. We further show that MLP-crafted coreset perturbations transfer to XGBoost, TabNet, and FT-Transformer with fidelity comparable to full-data crafting.

2 Background

Universal Adversarial Perturbations. Let $f_\theta : \mathbb{R}^d \to \{1, \ldots, C\}$ be a classifier with parameters θ, and let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ be a labelled dataset. A *Universal Adversarial Perturbation* (UAP) is a single, input-agnostic perturbation vector $\boldsymbol{\delta} \in \mathbb{R}^d$ satisfying $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ whose addition causes misclassification across the majority of inputs (Moosavi-Dezfooli et al., 2017). The quality of $\boldsymbol{\delta}$ is measured by its *fooling rate*,

$$\mathrm{FR}(\boldsymbol{\delta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[f_\theta(\mathbf{x}_i + \boldsymbol{\delta}) \neq y_i], \tag{1}$$

and standard algorithms find $\boldsymbol{\delta}$ by approximately solving

$$\max_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathcal{L}(\boldsymbol{\delta}) := \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\mathbf{x}_i + \boldsymbol{\delta}), y_i), \tag{2}$$

where ℓ is the cross-entropy loss. The original UAP algorithm of (Moosavi-Dezfooli et al., 2017) used an iterative DeepFool (Moosavi-Dezfooli et al., 2016) procedure to accumulate per-instance

adversarial directions over the full dataset. Subsequent work, notably (Shafahi et al., 2020), recast UAP generation as the loss-maximization problem in equation 2 and solved it with stochastic gradient methods. In this formulation, the victim classifier f_θ is held fixed; only $\boldsymbol{\delta}$ is updated. Gradient estimates for equation 2 are obtained by iterating over the full dataset $\mathcal{D}$ at each epoch, making the per-epoch cost $O(N)$ and the total runtime $O(NTE)$ for T inner steps and E epochs. In the tabular setting, (Ballet et al., 2019) studies feature-restricted perturbations, and adversarial robustness of tabular neural networks has been examined by (Kireev et al., 2022); neither addresses UAPs or the computational scalability of UAP computation.

Gradient-Based Attack Primitives. Two gradient-based methods serve as standard inner-loop primitives for solving equation 2. The *Fast Gradient Sign Method (FGSM)* (Goodfellow et al., 2014) takes a single projected gradient step on equation 2, $\boldsymbol{\delta} \leftarrow \Pi_\epsilon[\boldsymbol{\delta} + \alpha \cdot \mathrm{sign}(\nabla_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{\delta}))]$, where $\Pi_\epsilon[\cdot]$ denotes projection onto the ℓ_∞ ball of radius ϵ and α is the step size. *Projected Gradient Descent (PGD)* (Madry et al., 2017) applies this update iteratively for T inner steps with step size $\alpha = \epsilon/T$, yielding a stronger attack at the cost of $T\times$ more gradient evaluations per outer epoch. Both methods share a common structure: a gradient accumulation pass over the training set followed by a sign step and an ℓ_∞ projection.

Coreset Selection. A *coreset* is a small subset $\mathcal{C} \subseteq \mathcal{D}$ that provably approximates key statistics of the full dataset (Feldman & Langberg, 2011). In deep learning, coresets have been used to accelerate supervised training (Killamsetty et al., 2021b; Mirzasoleiman et al., 2020) and episodic memory selection in continual learning (Rebuffi et al., 2017). The following notion of gradient approximation quality is central to coreset-based optimization.

Definition 1 ((k,γ)-coreset)**.** *A subset $\mathcal{C} \subseteq \mathcal{D}$ of size k is a* (k,γ)-coreset *if*

$$\left\| \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{D}} \nabla_{\boldsymbol{\delta}} \ell(\mathbf{x}) - \frac{1}{k} \sum_{\mathbf{x} \in \mathcal{C}} \nabla_{\boldsymbol{\delta}} \ell(\mathbf{x}) \right\|_2 \leq \gamma. \tag{3}$$

A small γ guarantees that gradient steps computed on $\mathcal{C}$ closely track those of the full dataset. UAPCore does not optimise γ directly; instead, it selects samples by geometric coverage in the frozen penultimate feature space (§3.2) and evaluates empirically whether this proxy preserves UAP performance.

3 UAPCore

We now develop UAPCore, a coreset-accelerated framework for UAP computation. The central observation is a structural property of UAP optimization that makes a one-time coreset selection possible: the victim classifier f_θ is *entirely frozen* throughout. We describe how this enables a one-time feature-space proxy for gradient redundancy, and we present the resulting algorithm and its complexity.

3.1 Frozen Features as a Gradient-Redundancy Proxy

Let $f_\theta = h \circ \phi_\theta$, where $\phi_\theta : \mathbb{R}^d \to \mathbb{R}^m$ is the penultimate feature map and $h : \mathbb{R}^m \to \mathbb{R}^C$ is the final linear head. For a fixed perturbation $\boldsymbol{\delta}$, the per-sample gradient decomposes via the chain rule as:

$$\nabla_{\boldsymbol{\delta}}\, \ell(f_\theta(\mathbf{x}_i + \boldsymbol{\delta}), y_i) = J_{\phi_\theta}(\mathbf{x}_i + \boldsymbol{\delta})^\top \nabla_\phi\, \ell(h(\phi_\theta(\mathbf{x}_i + \boldsymbol{\delta})), y_i), \tag{4}$$

where $J_{\phi_\theta}(\mathbf{x}_i + \boldsymbol{\delta}) \in \mathbb{R}^{m \times d}$ is the Jacobian of ϕ_θ at $\mathbf{x}_i + \boldsymbol{\delta}$. This expression shows that gradient contributions depend on the victim's local Jacobian, the perturbed feature representation, the label, and the current perturbation. We therefore do not assume that Euclidean proximity in the unperturbed feature space directly guarantees gradient similarity. Instead, UAPCore uses the frozen penultimate representation as a geometry-aware proxy for sample redundancy. Because the victim parameters θ remain fixed, the feature mapping $\phi_\theta(\cdot)$, and hence the selection metric, is stationary throughout optimization. The selected subset consequently remains fixed under the selection criterion and does not need to be recomputed as the model evolves. However, gradients themselves depend on the perturbed inputs $\mathbf{x}_i + \boldsymbol{\delta}_t$ and may change throughout the attack trajectory; the continued effectiveness of the fixed subset is therefore an empirical property rather than a direct consequence of feature-space stationarity.

This is in sharp contrast to standard coreset methods for model training, such as CRAIG (Mirzasoleiman et al., 2020) and GradMatch (Killamsetty et al., 2021a), where the feature extractor itself changes, and subsets must be periodically recomputed to remain valid. In our setting, a *single, one-time* coreset selection is possible because the selection representation is frozen; whether that fixed subset preserves UAP quality is tested empirically.

3.2 Coreset Selection in Feature Space

The analysis above motivates selecting, *once* and before any gradient update, a geometrically representative subset $\mathcal{C} \subset \mathcal{D}$ of size $k = \lfloor \rho |\mathcal{D}| \rfloor$ such that every point in $\mathcal{D}$ has a close representative in $\mathcal{C}$ under the metric induced by ϕ_θ. We select $\mathcal{C}$ by applying a geometry-aware criterion directly to the penultimate feature matrix $\Phi = \{\phi_\theta(\mathbf{x}_i)\}_{i=1}^{N} \in \mathbb{R}^{N \times m}$. Operating in $\phi_\theta(\mathcal{D})$ rather than the raw input space has two practical advantages: (i) it places the selection metric in the same space that governs per-sample gradient information via equation 4; and (ii) it collapses heterogeneous tabular inputs into a common Euclidean space where distance meaningfully reflects model-relevant similarity. In this work, we evaluate the following selection strategies.

(C1) k-Center Greedy. A farthest-first traversal for the k-center objective, which seeks the subset minimising the largest distance from any point in $\mathcal{D}$ to its nearest selected center in ϕ_θ-space. The greedy solution achieves a $2\times$ approximation to the optimal k-center objective (Sener & Savarese, 2017), producing a subset that covers the feature space as uniformly as possible. This guarantee concerns the feature-space coverage objective and does not directly bound the gradient approximation error in Definition 1.

(C2) Facility Location. Rather than minimising the worst-case coverage gap, this strategy maximises total similarity-weighted coverage via the submodular objective $F(\mathcal{C}) = \sum_{\mathbf{x} \in \mathcal{D}} \max_{\mathbf{x}' \in \mathcal{C}} \kappa(\mathbf{x}, \mathbf{x}')$ with RBF kernel $\kappa(\mathbf{x}, \mathbf{x}') = \exp(-\|\phi_\theta(\mathbf{x}) - \phi_\theta(\mathbf{x}')\|_2^2/(2\sigma^2))$, where σ is the median pairwise distance in Φ. Greedy maximization of F carries a $(1 - 1/e)$ approximation guarantee for the monotone submodular objective under standard assumptions (Mirzasoleiman et al., 2020). As with k-center, this guarantee concerns the selection objective itself and does not imply a corresponding bound on the gradient approximation error in Definition 1. The RBF kernel κ down-weights outlying regions, concentrating the subset on high-density areas of the feature space.

(C3) Random Sampling. Points are drawn uniformly at random with no reference to the geometry of Φ. This requires no selection overhead and serves as a baseline that quantifies how much is gained by geometric awareness over simple data reduction. Together, C1 and C2 represent two distinct geometric objectives, minimax coverage and density-weighted coverage, while C3 provides an unstructured reference.

3.3 The UAPCore Algorithm

UAPCore proceeds in two phases. In the **selection phase**, executed once before any gradient update, we extract Φ from the frozen f_θ and apply one of the strategies described above to obtain $\mathcal{C}$ of size $k = \lfloor \rho |\mathcal{D}| \rfloor$. In the **attack phase**, we run E outer epochs of standard gradient-based UAP optimization *exclusively over* $\mathcal{C}$. Within each epoch, $\boldsymbol{\delta}$ is updated by one of two standard primitives. *FGSM epoch.* Accumulate the mean gradient over $\mathcal{C}$ and apply a single projected sign step. *PGD epoch.* Replace the single sign step with T iterative projected gradient descent steps at step size $\alpha = \epsilon/T$, initialised from the current $\boldsymbol{\delta}$. This yields a stronger inner-loop optimization at $T\times$ the per-epoch cost of FGSM. Both primitives share the same outer loop and differ only in their inner optimization; the coreset $\mathcal{C}$ is fixed and identical for both. The full procedure is given in Algorithm 1.

Computational cost. With exact k-center, UAPCore incurs a one-time selection cost $O(Nc_f + \rho N^2 D)$ and then gradient cost $O(\rho EPNc_g)$. When the gradient term dominates, the speedup over full-data UAP approaches the data-reduction factor $1/\rho$; otherwise, exact selection becomes as expensive as the entire subsequent coreset-based gradient optimization at dataset size $N_{\text{sel}} = EPc_g/D$. A complete derivation, including the runtime break-even point at which UAPCore and full UAP have equal total cost, is given in Appendix B; a linear-time approximation to the selection step is described in Appendix C.

4 EXPERIMENTAL SETUP

Compute. All victim-model training and attack experiments are executed in PyTorch 2.6 with CUDA 12.4 on a single NVIDIA GeForce RTX 4090 GPU. Software versions are numpy 1.26.4, SciPy 1.15.2, pandas 2.2.3, and scikit-learn 1.7.1. Gradients are accumulated in minibatches of size 32. The following experimental configurations are implemented with respect to the available compute setup.

Datasets. We evaluate UAPCore on tabular classification datasets collected from OpenML. We retain only supervised classification tasks with no missing values, instance counts $500 \leq N \leq 2{,}500$, and feature dimensionality $5 \leq d \leq 70$. Each dataset is split 80/20 into stratified training and test sets with a fixed `random state` and is standardized with zero mean and unit variance, using statistics fit on the training split only. Before attack experiments, we further require that the victim MLP achieves macro-F1 ≥ 0.90 under stratified 5-fold cross-validation. This filtering yields **44 datasets**. The retained sets span $N \in [500, 2{,}500]$ (median 2000), $d \in [5, 70]$ (median 20), and 2–10 classes, with 32 binary and 12 multiclass problems. Appendix A Figure 3 further summarizes the distribution of N and d.

Algorithm 1 UAPCore

Require: Data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$; frozen $f_\theta = h \circ \phi_\theta$; budget ϵ; coreset ratio $\rho \in (0, 1]$; epochs E; batch size B; step α; ATK $\in \{\text{FGSM}, \text{PGD}\}$; PGD steps T

Ensure: UAP $\boldsymbol{\delta} \in \mathbb{R}^d$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$

$\boldsymbol{\delta} \leftarrow \mathbf{0}^d$; $k \leftarrow \lfloor \rho N \rfloor$
Selection phase (once)
$\Phi \leftarrow \{\phi_\theta(\mathbf{x}_i)\}_{i=1}^{N}$ ▷ penult. features
$\mathcal{C} \leftarrow \text{SELECTCORESET}(\Phi, k)$ ▷ (C1)–(C3)
Attack phase
for $e = 1, \ldots, E$ **do**
  if ATK = FGSM **then**
    $\leftarrow \mathbf{0}^d$
    for mini-batch $\subseteq \mathcal{C}$, $|| = B$ **do**
      $\leftarrow + \sum_{i \in} \nabla_{\boldsymbol{\delta}} \ell(f_\theta(\mathbf{x}_i + \boldsymbol{\delta}), y_i)$
    end for
    $\boldsymbol{\delta} \leftarrow \Pi_\epsilon[\boldsymbol{\delta} + \alpha \operatorname{sign}(/|\mathcal{C}|)]$
  else if ATK = PGD **then**
    $\boldsymbol{\delta}^{(0)} \leftarrow \boldsymbol{\delta}$
    for $t = 1, \ldots, T$ **do**
      $\leftarrow \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \nabla_{\boldsymbol{\delta}} \ell(f_\theta(\mathbf{x}_i + \boldsymbol{\delta}^{(t-1)}), y_i)$
      $\boldsymbol{\delta}^{(t)} \leftarrow \Pi_\epsilon[\boldsymbol{\delta}^{(t-1)} + \frac{\epsilon}{T} \operatorname{sign}()]$
    end for
    $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta}^{(T)}$
  end if
end for
return $\boldsymbol{\delta}$

Victim Model. The victim is a fully connected multilayer perceptron (MLP) trained independently for each dataset. The architecture has three hidden layers of size $256 \to 128 \to 64$, ReLU activations, dropout $p = 0.2$, and a linear output layer for C classes. Let $\phi(\mathbf{x}) \in \mathbb{R}^{64}$ denote the penultimate-layer representation used for coreset selection. The model is trained with Adam ($\eta = 10^{-3}$), minibatch size 256, and 50 epochs on the training split; 10% of the training data is held out for validation during training. After training, the victim is frozen and used as the target for universal adversarial perturbation optimization.

Attack Configuration. We study universal adversarial perturbations $\boldsymbol{\delta} \in \mathbb{R}^d$ under an ℓ_∞ budget $\|\boldsymbol{\delta}\|_\infty \leq \varepsilon$ with $\varepsilon \in \{0.05, 0.10, 0.20\}$. Perturbed inputs are $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}$, with $\boldsymbol{\delta}$ clamped to $[-\varepsilon, \varepsilon]$ after each update. As explained in §3.3, Full-UAP uses the entire training set ($\rho = 1$). UAPCore subsamples a coreset of size $k = \min(\max(\lfloor \rho N \rfloor, C), N)$, where C is the number of classes and $\rho = 0.10$ by default. We compare three selection strategies on penultimate features $\{\phi(\mathbf{x}_i)\}_{i=1}^{N}$: k-center greedy, facility-location (FacLoc), and random sampling. In the final pipeline, the coreset is selected once at epoch 0 and held fixed for all $E = 50$ optimization epochs.

As explained in §3.3, we use two gradient-based universal perturbation primitives: (i) **FGSM**, a single signed-gradient step with step size $\alpha = \varepsilon$; and (ii) **PGD**, an iterative signed-gradient attack with 5 inner steps, and no random restarts. At each epoch, the primitive updates $\boldsymbol{\delta}$ using the mean cross-entropy gradient over the current coreset.

On the held-out test set $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{N_{\text{test}}}$, with predicted labels $\hat{y}_j = \arg\max_c f_\theta(\mathbf{x}_j + \boldsymbol{\delta})_c$, we report the test fooling rate $\text{FR}_{\text{test}}(\boldsymbol{\delta}) = \text{FR}(\boldsymbol{\delta}; \mathcal{D}_{\text{test}}) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \mathbb{1}[\hat{y}_j \neq y_j]$ from equation 1, i.e., the fraction of test samples misclassified under the universal perturbation. To measure how closely a coreset-based attack matches full-data optimization, we define the fooling-rate gap relative

Table 1: Coreset fidelity. The Facility Location (C2) is denoted as FacLoc. Random-search baseline: uniform random coreset selection with 30 independent seeds per dataset; we report the best and worst ΔFR observed (oracle upper bound and pessimistic lower bound). Med./Mean ΔFR vs. full-UAP; % w/in 2%: $\Delta\mathrm{FR} > -0.02$.

FGSM					PGD				
Strategy	ε	**Med. ΔFR**	**Mean ΔFR**	**% w/in 2%**	**Strategy**	ε	**Med. ΔFR**	**Mean ΔFR**	**% w/in 2%**
FacLoc	0.050	0.000	−0.003	90.9	FacLoc	0.050	0.000	−0.003	90.9
	0.100	0.000	−0.010	84.1		0.100	0.000	−0.010	81.8
	0.200	0.000	−0.023	77.3		0.200	0.000	−0.024	75.0
K-Center	0.050	0.000	−0.001	90.9	K-Center	0.050	0.000	−0.001	90.9
	0.100	0.000	−0.006	86.4		0.100	0.000	−0.006	86.4
	0.200	0.000	−0.030	72.7		0.200	0.000	−0.031	72.7
Rand. `Best`	0.050	0.005	0.010	100.0	Rand. `Best`	0.050	0.005	0.010	100.0
	0.100	0.008	0.027	100.0		0.100	0.008	0.027	100.0
	0.200	0.019	0.048	100.0		0.200	0.014	0.047	100.0
Rand. `Worst`	0.050	−0.008	−0.013	75.0	Rand. `Worst`	0.050	−0.008	−0.013	75.0
	0.100	−0.016	−0.032	56.8		0.100	−0.014	−0.031	59.1
	0.200	−0.038	−0.073	36.4		0.200	−0.036	−0.074	38.6

to full-UAP, $\Delta\mathrm{FR} = \mathrm{FR}_{\mathrm{test}}^{\mathrm{UAPCore}} - \mathrm{FR}_{\mathrm{test}}^{\mathrm{full}}$; values near zero indicate that the coreset attack achieves nearly the same test fooling rate as optimizing on the full training set. We also report the fraction of datasets with $\Delta\mathrm{FR} > -0.02$, i.e., within 2 percentage points of full-UAP. Finally, we report the end-to-end *speedup* $S = T_{\mathrm{full}}/T_{\mathrm{core}}$, where T_{full} and T_{core} denote the total wall-clock time for full-UAP and UAPCore, respectively, under matched primitive, ε, seed, and epoch count.

By leveraging random sampling in Algorithm 1, we generate 30 independent random coresets for each dataset. We report the best and worst $\Delta\mathrm{FR}$, providing an empirical bound for coreset performance and validating whether the samples chosen by a smarter coreset strategy are actually important.

5 Results

Table 1 summarizes aggregate performance across all 44 datasets, for each combination of attack primitive, coreset strategy, and perturbation budget ε.

Pooled over both geometric strategies, both primitives, and all three budgets, the mean $\Delta\mathrm{FR}$ is -0.012, and 83.3% of runs satisfy $\Delta\mathrm{FR} > -0.02$. At the default budget $\varepsilon = 0.10$, the median $\Delta\mathrm{FR}$ is exactly zero for every geometric primitive-strategy pair, meaning UAPCore matches the full-UAP fooling rate on the typical dataset; the negative mean is driven by a small number of datasets that underperform at larger ε, where the fraction within 2% drops as low as 72.7%.

Table 2: Wall-clock speedup over full-UAP ($T_{\mathrm{full}}/T_{\mathrm{core}}$).

Primitive	**Strategy**	**Median**	**Mean**
FGSM	FacLoc	0.956	0.932
	K-Center	2.770	3.988
	Random	3.115	4.467
PGD	FacLoc	4.011	3.788
	K-Center	8.548	8.169
	Random	8.856	8.802

Table 1 contextualizes these results against the random-search bounds. Even the `Best` random oracle, the most favorable of 30 independent seeds, achieves only slightly positive mean $\Delta\mathrm{FR}$ (e.g., $+0.027$ for FGSM at $\varepsilon = 0.10$), while the `Worst` bound plunges to -0.032, with only 56.8% of datasets within 2% of full-UAP. By contrast, k-center sits between these extremes with a mean $\Delta\mathrm{FR}$ of -0.006 and 86.4% within the 2% band, delivering *reliable* near-full-UAP fidelity without requiring an expensive oracle search. Facility Location is equally accurate in the median (zero throughout), but its greedy submodular selection is itself expensive: under FGSM it is marginally slower than full-UAP (median $0.96\times$), and under PGD it reaches only $\sim$3–4$\times$ (Table 2). Uniform random sampling incurs virtually no selection overhead and therefore yields the highest raw speedups (median $3.1\times$ under FGSM and $8.9\times$ under PGD), but k-center comes close ($2.8\times$ and $8.5\times$), retaining roughly 89% and 97% of random sampling's speedup. Exact k-center selection is therefore cheap at this scale, while random sampling's high variance between the best- and worst-case seeds makes it unsuitable when attack fidelity must be guaranteed. Together, these results indicate two things. First, the speed difference between the two geometric strategies comes from selection cost rather than from the

geometric objective: k-center and Facility Location reach similar fidelity (Facility Location is slightly more reliable at $\varepsilon = 0.20$), but k-center is far cheaper to select. Second, the benefit of geometric selection is reliability: the `Best` random bound is an oracle chosen in hindsight from 30 seeds using the test fooling rate itself and is not available in practice, whereas k-center attains near-full-UAP fidelity from a single, one-time selection with no seed search and avoids the severe tail risk of the `Worst` random seeds. Obtaining the `Best` random coreset would moreover require all 30 seeded runs, roughly $30/8.9 \approx 3.4\times$ the cost of a single full-UAP run under PGD. In absolute terms, the coreset UAPs closely track full-data optimization. Under PGD, the full-UAP reaches a mean test fooling rate of 0.077, 0.121, and 0.216 at $\varepsilon = 0.05$, 0.10, and 0.20, while k-center reaches 0.076, 0.115, and 0.184 and Facility Location reaches 0.074, 0.111, and 0.192, using only 10% of the training data. These rates include test points the victim already misclassifies without perturbation (mean clean error 0.055, Table 5). The attack's effect therefore grows substantially with the budget, and at the default $\varepsilon = 0.10$ the coreset UAPs reproduce most of the full-UAP's effect at a fraction of its cost. Mean fooling rates exceed medians throughout (e.g., 0.216 vs. 0.164 for full-UAP at $\varepsilon = 0.20$), indicating that a subset of datasets is considerably more vulnerable than the typical one.

5.1 Ablation: Speedup

Figure 1 asks whether UAPCore's advantage grows with dataset size, comparing full-UAP against UAPCore ($\rho = 0.10$, PGD, k-center, $\varepsilon = 0.10$) on all 44 datasets. The overall median speedup is roughly $8.5\times$ and the relationship with N is positive but sub-linear: cluster medians rise from $6.5\times$ for small datasets ($N \le 650$, 11% of the benchmark) to $7.2\times$ for medium datasets ($650 < N \le 1200$, 16%) to $8.8\times$ for large datasets ($N > 1200$, 73%). A log-log fit gives speedup $\approx 1.15\, N^{0.27}$, consistent with feature extraction being amortized once under the fixed-coreset protocol while gradient cost continues to scale with the coreset size $\lfloor \rho N \rfloor$. This fit describes the benchmark range only and cannot be extrapolated: speedup is bounded above by $1/\rho = 10\times$ (equation 9), and for exact k-center it eventually degrades as $O(N^{-1})$ once the quadratic selection term dominates (Appendix B). Speedup shows little dependence on feature dimension d, indicating that the benefit is driven by N rather than input width. Practically, this means UAPCore's advantage is largest precisely in the regime the benchmark emphasizes (median $N = 2000$), and even the smallest retained datasets see better than $5\times$ speedup.

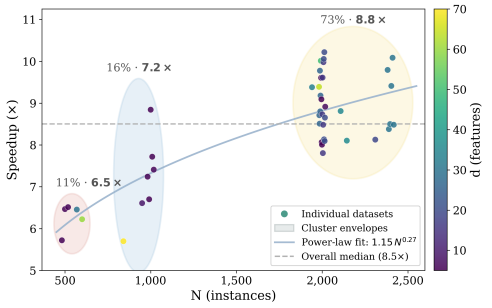

Figure 1: Speedup vs. training-set size N at $\rho = 0.10$, PGD, k-center, $\varepsilon = 0.10$.

5.2 Ablation: ρ

Figure 2 ablates the coreset fraction $\rho \in \{0.05, 0.10, 0.15, 0.20, 0.30, 1.0\}$ with k-center selection and PGD at $\varepsilon = 0.10$, and two trends emerge. Fidelity is remarkably stable: the median ΔFR stays at zero across the entire range, and only at the most aggressive setting, $\rho = 0.05$, does the lower quartile drop to roughly -0.019, indicating that a minority of datasets lose up to about two fooling-rate points; by $\rho \ge 0.15$ the interquartile range has collapsed to essentially zero. Speedup, in contrast, trades off smoothly and roughly inversely with ρ, falling from $14.4\times$ at $\rho = 0.05$ to $8.5\times$ at our default $\rho = 0.10$, $4.6\times$ at $\rho = 0.20$, $3.1\times$ at $\rho = 0.30$, and $1\times$ at $\rho = 1$. This places $\rho = 0.10$ as a practical sweet spot, recovering most of the available speedup while keeping the fidelity tail tight.

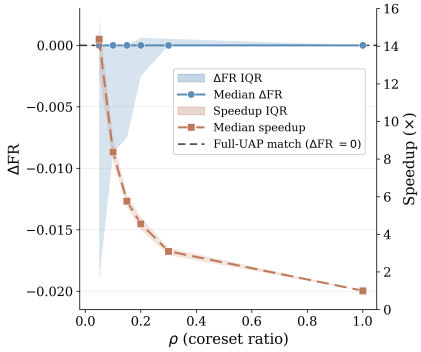

Figure 2: Median ΔFR (left axis) and speedup (right axis) vs. coreset ratio ρ, aggregated over 44 datasets (PGD, k-center, $\varepsilon = 0.10$). Shaded bands show interquartile ranges; $\rho = 1$ is full-UAP.

Table 4: Transfer fidelity of MLP-crafted K-Center UAPs. ΔFR is evaluated on each recipient relative to the MLP full-UAP transferred to that same recipient.

FGSM			
Recipient	ε	**Mean ΔFR**	**% Within 2%**
XGBoost	0.05	+0.004	93.2
	0.10	+0.003	86.4
	0.20	−0.018	79.5
TabNet	0.05	−0.000	95.5
	0.10	−0.003	88.6
	0.20	−0.013	84.1
FT-Transformer	0.05	+0.002	97.7
	0.10	+0.001	93.2
	0.20	−0.019	79.5

PGD			
Recipient	ε	**Mean ΔFR**	**% Within 2%**
XGBoost	0.05	+0.004	93.2
	0.10	+0.003	86.4
	0.20	−0.017	79.5
TabNet	0.05	+0.000	95.5
	0.10	−0.003	88.6
	0.20	−0.014	84.1
FT-Transformer	0.05	+0.001	97.7
	0.10	+0.000	93.2
	0.20	−0.019	75.0

5.3 Cross-Architecture Transfer of MLP-Crafted UAPs

All results so far craft and evaluate UAPs on the same frozen MLP victim used for coreset selection. To test whether coreset fidelity to full-data optimization persists beyond the architecture used for selection, we train three additional classifiers on the same 44 OpenML tasks and the same train/test splits: XGBoost (Chen & Guestrin, 2016), TabNet (Arik & Pfister, 2021), and an FT-Transformer (Gorishniy et al., 2021). We then craft each perturbation δ only on the MLP, using the original full-UAP, Facility Location, and exact k-center coreset procedures; no gradients from the recipient model are used at any point. For a recipient model g, we report $\mathrm{FR}_g(\delta) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_{\text{test}}} \mathbb{I}[g(x+\delta) \neq y]$, i.e., equation 1 on $\mathcal{D}_{\text{test}}$ with g in place of f_θ. To isolate the effect of coreset selection under transfer, the transfer gap $\Delta\mathrm{FR}_g = \mathrm{FR}_g(\delta_{\text{core}}) - \mathrm{FR}_g(\delta_{\text{MLP-full}})$ is defined relative to the *MLP-crafted full-UAP evaluated on the same recipient*.

Across all three recipients, coreset-crafted perturbations transfer with fidelity comparable to the MLP full-UAP: mean $\Delta\mathrm{FR}$ relative to the transferred full-UAP stays within ± 0.01 for k-center and Facility Location at $\varepsilon \leq 0.10$, and the great majority of datasets remain within two percentage points at the default budget across all three architectures (Table 3). This is consistent with the broader finding that surrogate-crafted adversarial perturbations transfer across architectures that share similar decision boundaries (Demontis et al., 2019); our results extend this observation to the coreset setting, showing that restricting the surrogate's optimization to a geometrically representative subset does not measurably degrade transferability relative to full-dataset surrogate crafting.

Table 3: Transfer of MLP-crafted PGD UAPs to unseen architectures ($\epsilon = 0.1$, 44 datasets). ΔFR is relative to the MLP full-UAP on the same recipient; positive means the coreset UAP transfers at least as well.

Recipient	**MLP coreset**	**Med. ΔFR**	**Mean ΔFR**	**% within 2%**
XGBoost	FacLoc	0.000	−0.005	84.1
	K-Center	0.000	+0.003	86.4
TabNet	FacLoc	0.000	+0.000	93.2
	K-Center	0.000	−0.003	88.6
FT-Transformer	FacLoc	0.000	+0.002	88.6
	K-Center	0.000	+0.000	93.2

Fidelity degrades somewhat at the larger $\varepsilon = 0.20$ budget (Table 4), mirroring the same-architecture pattern in Table 1, where larger perturbation budgets amplify the gap between coreset and full-data optimization. The absolute transferred fooling rates are of the same order as the white-box rates on the MLP itself. At $\varepsilon = 0.10$ under PGD, the transferred full-UAP reaches a mean fooling rate of 0.116 on XGBoost, 0.135 on TabNet, and 0.105 on FT-Transformer, compared with 0.119, 0.131, and 0.106 for the transferred k-center UAP. At this budget and at $\varepsilon = 0.05$, k-center matches or marginally exceeds the transferred full-UAP on XGBoost and FT-Transformer. At $\varepsilon = 0.20$, the transferred full-UAP reaches 0.202, 0.192, and 0.184 on the three recipients, and k-center remains close at 0.185, 0.179, and 0.165. Perturbations crafted on a 10% coreset of a single MLP thus transfer

to unseen tree-based and attention-based models about as well as full-data MLP perturbations do, including to a non-differentiable ensemble whose gradients are never used.

6 DISCUSSION

Once $\boldsymbol{\delta}$ has partially converged, many training examples may provide redundant optimization signals. UAPCore selects a geometry-aware subset in the frozen victim's feature space and tests whether this subset preserves the UAP optimization trajectory at substantially lower per-step cost. Crucially, because f_θ is frozen throughout UAP optimization, the feature embedding ϕ_θ is stationary, so the coreset selected once upfront does not need to be recomputed; this stationarity argument is what licenses the one-shot design and removes the periodic coreset recomputation required by coreset methods for standard model training such as CRAIG (Mirzasoleiman et al., 2020) and GradMatch (Killamsetty et al., 2021a).

Limitations. UAPCore has been primarily evaluated in a white-box setting: it requires access to the victim's gradients and penultimate-layer representations ϕ_θ, and, as stated, does not extend to gradient-free or score-based threat models. Our evaluation is restricted to 44 tabular datasets with N in the 500–2,500 range, and we have not evaluated image or text modalities where UAPs originated. We do not construct coresets natively in tree-ensemble feature space, since gradient-boosted trees lack a differentiable penultimate embedding; §5.3 shows this can be partially sidestepped via MLP-surrogate transfer, which preserves coreset fidelity under transfer about as well as it preserves same-architecture fidelity, but a native tree-ensemble coreset method remains open.

The complexity analysis in Appendix B surfaces a further structural limitation of exact selection: because k-center selection costs $O(\rho N^2 D)$ while the amortised gradient work scales only as $O(\rho EPNc_g)$, there exists a selection crossover $N_{\text{sel}} = \frac{EPc_g}{D}$, at which the one-time selection cost equals the entire subsequent coreset-based gradient optimization. For $N \gg N_{\text{sel}}$, the realized speedup over full UAP degrades as $O(N^{-1})$ rather than the ideal $1/\rho$; the runtime break-even N_{break} is derived in Appendix B. Appendix C describes and evaluates a block-local approximation that removes this quadratic term by construction, though without the exact k-center's global $2\times$ approximation guarantee.

7 CONCLUSION

We introduced UAPCore, a framework for coreset-accelerated optimization of universal adversarial perturbations that exploits a property specific to the UAP setting: because the victim classifier f_θ is frozen throughout optimization, the feature representation used for selection is stationary, so a single coreset selected at epoch 0 remains fixed under the same feature-space selection criterion and does not require recomputation; whether this fixed subset preserves UAP performance is then evaluated empirically. Selecting a geometrically representative 10% subset with k-center greedy and holding it fixed for all $E = 50$ optimization epochs yields a median $8.5\times$ wall-clock speedup over full-dataset PGD, with a median fooling-rate gap of exactly zero across 44 OpenML tabular datasets and agreement within two percentage points of full-UAP on up to $\approx 91\%$ of datasets. The advantage is not narrowly tuned: fidelity stays essentially flat across $\rho \in [0.10, 0.30]$, and speedup grows with N within the range studied ($N \leq 2{,}500$), although exact k-center selection is expected to erode this advantage at much larger N (Appendix B). We further showed that MLP-crafted coreset perturbations transfer to unseen tree-based and attention-based tabular architectures (§5.3) with fidelity comparable to full-data surrogate crafting. In our experiments, k-center and Facility Location achieved similar fidelity, with Facility Location slightly more reliable at $\varepsilon = 0.20$, but k-center selection is substantially cheaper and therefore delivers the larger end-to-end speedup.

Future Work. Natural next steps are: (i) evaluating UAPCore at $N \gg 2{,}500$, closer to the enterprise-scale tabular datasets that motivate this work; (ii) a native tree-ensemble coreset method that does not rely on MLP-surrogate transfer, for instance built on leaf-index or leaf-embedding representations; and (iii) directly operationalising gradient alignment as a selection objective and testing whether it predicts coreset fidelity more reliably than geometric coverage alone.

AI USE STATEMENT

We used AI to edit author-written text for readability and to generate supporting code for plotting and result aggregation. We also used these tools to improve productivity during code development. An initial version of the UAPCore codebase was written manually by the authors. AI coding assistants were then used to help extend it for running the ablation studies and the cross-architecture evaluation on XGBoost, TabNet, and FT-Transformer (§5.3). All AI-edited text was reviewed by the authors, and all AI-assisted code was manually reviewed and validated before being used to produce any reported result. The authors take full responsibility for the content of this paper.

ETHICS STATEMENT

This work makes universal adversarial perturbations cheaper to compute and could in principle lower the cost of attacking deployed tabular classifiers. We believe the added risk is limited. UAPCore introduces no new attack capability; it accelerates existing gradient-based UAP methods, and crafting requires white-box access to a model's gradients and internal representations. Although §5.3 shows that surrogate-crafted perturbations transfer to other architectures, this transferability is a property of UAPs in general: coreset crafting matches, but does not exceed, full-data transfer. The intended use is robustness auditing, where cheaper UAP computation makes it practical to evaluate many models and datasets. All experiments use publicly available OpenML benchmark datasets; we collect no new data and involve no human subjects.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Section 3 specify UAPCore, including the definition and hyperparameters of each coreset strategy. Appendix C specifies the Block-k-Center variant. All 44 datasets are public OpenML tasks, listed with exact task IDs and metadata in Appendix A. Section 4 gives the data filtering and preprocessing steps, the victim model and its training configuration, the attack hyperparameters, the metric definitions, and the hardware and software versions. The ablation grids and the transfer protocol are stated in Sections 5.2 and 5.3 and Appendix C.3. Appendices B and C give the complexity derivations, which complement the hardware-dependent wall-clock measurements.

REFERENCES

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6679–6687, 2021.

Vincent Ballet, Xavier Renard, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, and Marcin Detyniecki. Imperceptible adversarial attacks on tabular data. *arXiv preprint arXiv:1911.03274*, 2019.

MohammadHossein Bateni, Hossein Esfandiari, Manuela Fischer, and Vahab Mirrokni. Extreme k-center clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3941–3949, 2021.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 35(6):7499–7519, 2022.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium (USENIX security 19)*, pp. 321–338, 2019.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 569–578, 2011.

Pranshav Gajjar, Molham Khoja, Abiodun Ganiyu, Marc Juarez, Mahesh K Marina, Andrew Lehane, and Vijay K Shah. Black-box evasion attacks on data-driven open ran apps: Tailored design and experimental evaluation. *Proceedings of the ACM on Networking*, 3(CoNEXT4):1–26, 2025.

Micah Goldblum, Avi Schwarzschild, Ankit Patel, and Tom Goldstein. Adversarial attacks on machine learning systems for high-frequency trading. In *Proceedings of the Second ACM International Conference on AI in Finance*, pp. 1–9, 2021.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34:18932–18943, 2021.

Jamie Hayes and George Danezis. Learning universal adversarial perturbations with generative models. In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 43–49. IEEE, 2018.

Hokuto Hirano, Akinori Minagi, and Kazuhiro Takemoto. Universal adversarial attacks on deep neural networks for medical image classification. *BMC medical imaging*, 21(1):9, 2021.

Krishnateja Killamsetty, Sivasubramanian Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pp. 5464–5474. PMLR, 2021a.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 8110–8118, 2021b.

Klim Kireev, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. *arXiv preprint arXiv:2208.13058*, 2022.

Hui Lu, Yi Yu, Yiming Yang, Chenyu Yi, Qixin Zhang, Bingquan Shen, Alex C Kot, and Xudong Jiang. When robots obey the patch: Universal transferable patch attacks on vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22867–22878, 2026.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

Konda Reddy Mopuri, Utsav Garg, and R Venkatesh Babu. Fast feature fool: A data independent approach to universal adversarial perturbations. *arXiv preprint arXiv:1707.05572*, 2017.

Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, and Ben Edwards. Adversarial robustness toolbox v1.0.0, 2019. URL `https://arxiv.org/abs/1807.01069`.

Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4422–4431, 2018.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5533–5542. IEEE, 2017.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5636–5643, 2020.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Chaoning Zhang, Philipp Benz, Chenguo Lin, Adil Karjauv, Jing Wu, and In So Kweon. A survey on universal adversarial attack. *arXiv preprint arXiv:2103.01498*, 2021.

A Full Dataset List and Metadata

Table 5 lists all 44 retained OpenML datasets, including dataset name, OpenML task ID, number of samples N, number of features d, number of classes C, task type (binary/multi-class), baseline macro-F1 of the victim MLP under stratified 5-fold cross-validation, and its test accuracy on the held-out split. Figure 3 summarizes the resulting distribution of N and d.

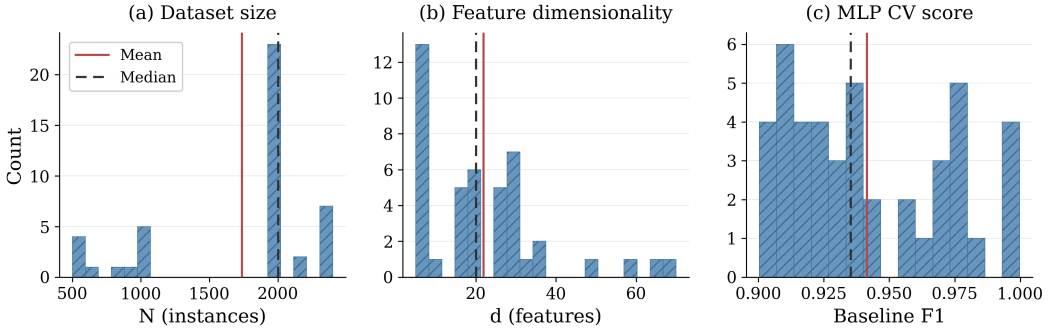

Figure 3: Distribution of the 44 retained OpenML datasets.

Table 5: Baseline F1 is the macro average from stratified 5-fold cross-validation on the victim MLP; test accuracy is on the held-out 20% split after training.

#	Dataset	Task ID	N	d	C	Type	F1	Test acc.
1	mfeat-karhunen	16	2000	64	10	Multi-class	0.969 ± 0.007	0.965
2	segment	36	2310	19	7	Multi-class	0.966 ± 0.005	0.957
3	synthetic_control	3512	600	60	6	Multi-class	0.993 ± 0.006	0.992
4	analcatdata_authorship	3549	841	70	4	Multi-class	0.995 ± 0.006	0.988
5	fri_c1_1000_5	3609	1000	5	2	Binary	0.928 ± 0.020	0.950
6	fri_c3_500_5	3615	500	5	2	Binary	0.916 ± 0.010	0.880
7	fri_c2_500_5	3657	500	5	2	Binary	0.900 ± 0.022	0.910
8	fri_c0_1000_5	3664	1000	5	2	Binary	0.904 ± 0.010	0.910

continued on next page

(continued from previous page)

#	Dataset	Task ID	N	d	C	Type	F1	Test acc.
9	fri_c3_1000_5	3678	1000	5	2	Binary	0.932 ± 0.016	0.935
10	stock	3706	950	9	2	Binary	0.960 ± 0.016	0.942
11	fri_c1_500_5	3734	500	5	2	Binary	0.906 ± 0.031	0.910
12	fri_c2_1000_5	3775	1000	5	2	Binary	0.938 ± 0.007	0.935
13	cardiotocography	9898	2126	35	3	Multi-class	0.982 ± 0.007	0.988
14	wdbc	9946	569	30	2	Binary	0.977 ± 0.010	0.974
15	steel-plates-fault	9967	1941	33	2	Binary	0.999 ± 0.001	1.000
16	pol_seed_1_nr...	362627	2000	26	2	Binary	0.976 ± 0.011	0.967
17	pol_seed_0_nr...	362628	2000	26	2	Binary	0.977 ± 0.004	0.967
18	pol_seed_2_nr...	362629	2000	26	2	Binary	0.970 ± 0.005	0.980
19	pol_seed_3_nr...	362630	2000	26	2	Binary	0.967 ± 0.004	0.970
20	pol_seed_4_nr...	362631	2000	26	2	Binary	0.973 ± 0.005	0.972
21	MiniBooNE_seed_0...	362657	2000	50	2	Binary	0.900 ± 0.013	0.880
22	wilt_seed_1...	362732	2000	5	2	Binary	0.911 ± 0.034	0.985
23	wilt_seed_2...	362735	2000	5	2	Binary	0.936 ± 0.025	0.985
24	wilt_seed_3...	362736	2000	5	2	Binary	0.925 ± 0.022	0.990
25	wilt_seed_4...	362737	2000	5	2	Binary	0.915 ± 0.026	0.975
26	segment_seed_0_nr...	362878	2000	16	7	Multi-class	0.908 ± 0.012	0.930
27	segment_seed_1_nr...	362879	2000	16	7	Multi-class	0.910 ± 0.019	0.897
28	segment_seed_2_nr...	362880	2000	16	7	Multi-class	0.914 ± 0.010	0.925
29	segment_seed_3_nr...	362881	2000	16	7	Multi-class	0.912 ± 0.012	0.912
30	segment_seed_4_nr...	362882	2000	16	7	Multi-class	0.909 ± 0.010	0.907
31	sylvine_seed_0_nr...	362923	2000	20	2	Binary	0.908 ± 0.013	0.895
32	sylvine_seed_1_nr...	362924	2000	20	2	Binary	0.919 ± 0.014	0.933
33	sylvine_seed_2_nr...	362925	2000	20	2	Binary	0.923 ± 0.015	0.940
34	sylvine_seed_3_nr...	362926	2000	20	2	Binary	0.922 ± 0.008	0.922
35	sylvine_seed_4_nr...	362927	2000	20	2	Binary	0.921 ± 0.017	0.915
36	dummy	363108	1000	6	2	Binary	0.977 ± 0.012	0.985
37	Contaminant-det...	363324	2400	30	2	Binary	0.945 ± 0.007	0.938
38	Contaminant-det...	363325	2400	30	2	Binary	0.936 ± 0.002	0.931
39	Contaminant-det...	363326	2400	30	2	Binary	0.935 ± 0.013	0.927
40	Contaminant-det...	363327	2400	30	2	Binary	0.936 ± 0.013	0.929
41	Contaminant-det...	363328	2400	30	2	Binary	0.929 ± 0.010	0.906
42	student_lifestyle_dataset	363450	2000	7	3	Multi-class	0.958 ± 0.009	0.957
43	Fetal_cardioto...	363573	2126	36	10	Multi-class	1.000 ± 0.000	1.000
44	hazelnut-spread...	363637	2400	30	2	Binary	0.944 ± 0.012	0.944

B Complexity and Amortization

Let N be the training-set size, d the input dimension, and D the penultimate feature dimension used for coreset selection. Let c_f denote the cost of a single forward evaluation of the victim network, and let c_g denote the cost of a single per-sample forward-backward pass used to compute the gradient with respect to the universal perturbation. Let the universal perturbation be optimised for E epochs, each consisting of P projected-gradient steps. For FGSM, $P = 1$; for PGD, P equals the number of inner steps.

Full-data UAP. Full UAP evaluates the gradient over all N points at every projected-gradient step. Its total cost is therefore linear in N:

$$T_{\text{full}} = EPNc_g = O(EPNc_g). \tag{5}$$

Exact UAPCore. UAPCore incurs a one-time feature-extraction cost Nc_f, followed by exact greedy k-center selection on D-dimensional features, which costs $O(\rho N^2 D)$, and then restricts every gradient step to the ρN coreset points:

$$T_{\text{UAPCore}} = \underbrace{Nc_f + \rho N^2 D}_{\text{one-time selection}} + \underbrace{\rho EPNc_g}_{\text{gradient updates}}. \tag{6}$$

Speedup. The speedup over full-data UAP is

$$S = \frac{T_{\text{full}}}{T_{\text{UAPCore}}} = \frac{EPNc_g}{Nc_f + \rho N^2 D + \rho EPNc_g}. \tag{7}$$

Dividing numerator and denominator by Nc_g gives

$$S = \frac{EP}{\frac{c_f}{c_g} + \frac{\rho ND}{c_g} + \rho EP}. \tag{8}$$

When the one-time feature-extraction and coreset-selection costs are negligible relative to the accumulated gradient work, i.e.

$$\frac{c_f}{c_g} + \frac{\rho ND}{c_g} \ll \rho EP,$$

The speedup approaches the data-reduction factor:

$$S \longrightarrow \frac{EP}{\rho EP} = \frac{1}{\rho}. \tag{9}$$

At $\rho = 0.10$, this ideal limit is $10\times$. Under PGD, k-center reaches a median of $8.5\times$, while random selection, which has no selection cost, reaches $8.9\times$; exact selection therefore accounts for only a small part of the gap to $10\times$, and the remainder comes from fixed costs outside the coreset gradient loop. The model above omits fixed per-run overheads (e.g., data transfer, evaluation, and per-epoch loop costs) that do not scale with ρ. These are small relative to PGD runs but not to the much shorter FGSM runs at $N \leq 2{,}500$, which is why even random selection reaches only $3.1\times$ under FGSM (Table 2).

Selection crossover. We define N_{sel} as the dataset size at which the one-time exact k-center selection cost equals the cumulative gradient-update cost on the coreset. Setting

$$\rho N^2 D = \rho EPNc_g$$

and solving yields

$$N_{\text{sel}} = \frac{EPc_g}{D}. \tag{10}$$

For $N \ll N_{\text{sel}}$, the gradient term dominates T_{UAPCore}, and equation 9 holds. For $N \gg N_{\text{sel}}$, the $\rho N^2 D$ selection term dominates T_{UAPCore}. This does *not* by itself imply that UAPCore is slower than full UAP; it characterises the internal cost structure of UAPCore.

Runtime break-even with full UAP. The dataset size at which UAPCore and full UAP have equal total runtime, N_{break}, is obtained by setting

$$Nc_f + \rho N^2 D + \rho EPNc_g = EPNc_g.$$

Rearranging,

$$Nc_f + \rho N^2 D = (1-\rho)EPNc_g,$$

and dividing by N gives

$$c_f + \rho ND = (1-\rho)EPc_g.$$

Thus,

$$N_{\text{break}} = \frac{(1-\rho)EPc_g - c_f}{\rho D}, \tag{11}$$

Provided the numerator is positive. For $N > N_{\text{break}}$, UAPCore with exact k-center becomes slower than full-data UAP.

The selection crossover N_{sel} and the runtime break-even N_{break} capture different phenomena. N_{sel} is the dataset size at which exact k-center selection becomes as expensive as the subsequent coreset-based gradient optimization. In contrast, N_{break} is the dataset size at which total UAPCore runtime equals full-data UAP runtime. The former characterises the internal cost structure of UAPCore, whereas the latter determines whether UAPCore remains faster than full UAP. In the large-N regime where exact selection dominates, $S = O(N^{-1})$.

C BLOCK-k-CENTER: A LINEAR-TIME APPROXIMATION TO EXACT SELECTION

The quadratic selection term identified in Appendix B motivates a cheaper approximation to exact k-center selection. This appendix describes the approximation (C.1), analyses its cost (C.2) and evaluates it empirically against base-k-Center selection (C.3).

C.1 METHOD

Exact k-center selection, as used in C1 (§3.2), has cost $O(\rho N^2 D)$; this quadratic term can itself dominate total runtime when N grows large, as formalised in Appendix B. To remove this quadratic selection term while preserving the spirit of minimax coverage, we introduce a simple block-local approximation. Inspired by distributed k-center algorithms that localize greedy selection to data blocks (Bateni et al., 2021), Block-k-Center replaces their costly nearest-neighbor hub assignment with a random permutation into equal-sized blocks and a fixed local budget b, yielding a single-machine linear-time heuristic with no nearest-neighbor overhead.

Given feature representations $\Phi = \{\phi(\mathbf{x}_i)\}_{i=1}^N$, a target coreset size k, and a constant local center budget b, we randomly permute the data and partition it into

$$B = \left\lceil \frac{k}{b} \right\rceil$$

approximately equal-sized blocks. In each block, we run the base-k-Center farthest-first traversal locally to select approximately b centers. The union of all local selections forms the coreset; if required, it is trimmed or padded to exactly k points. Block-k-Center is a drop-in replacement for SELECTCORESET in Algorithm 1; the attack phase is unchanged. Because farthest-first traversal is performed only within blocks, Block-k-Center is a heuristic approximation and does *not* retain the global 2-approximation guarantee of base-k-Center selection.

C.2 COMPLEXITY

Block-k-Center replaces the base-k-Center traversal's $O(\rho N^2 D)$ selection cost with a block-local variant. Partitioning $\mathcal{D}$ into $B = \lceil k/b \rceil$ blocks of roughly N/B points each and selecting up to b centers per block via local Gonzalez traversal costs

$$B \cdot O\left(b \cdot \frac{N}{B} \cdot D\right) = O(bND),$$

which for a fixed local budget b is linear in N, in contrast to the quadratic $O(\rho N^2 D)$ cost of exact global selection. The resulting end-to-end runtime is

$$T_{\text{block}} = \underbrace{Nc_f + NbD}_{\text{feature extraction and block-local selection}} + \underbrace{\rho EPNc_g}_{\text{gradient updates}} = O(N(c_f + bD + \rho EPc_g)). \tag{12}$$

Normalising by Nc_g as in equation 8 gives

$$S_{\text{block}} = \frac{T_{\text{full}}}{T_{\text{block}}} = \frac{EP}{\frac{c_f}{c_g} + \frac{bD}{c_g} + \rho EP}. \tag{13}$$

Unlike equation 8, this expression is independent of N: because every term in equation 12 is linear in N, the speedup approaches a constant plateau as $N \to \infty$ rather than collapsing toward $O(N^{-1})$. This removes the selection crossover entirely, provided the local budget b is held fixed rather than scaled with N, which is the protocol used in Appendix C.3, where the same b values are applied across all 44 datasets regardless of their individual N. Enterprise-scale tabular tasks routinely exceed $N = 10^5$ (Borisov et al., 2022), well beyond the crossover regime where exact selection would otherwise dominate; equation 13 indicates that, at fixed b, Block-k-Center's relative advantage over full-UAP does not degrade in this regime. We reiterate, however, that this is an analytic prediction, and we have not empirically probed N beyond the 2,500-sample regime of our benchmark.

Table 6: Block-k-Center budget sweep on 44 datasets (PGD, $\varepsilon = 0.10$, $\rho = 0.10$). b: local center budget per block. Speedup is selection time only, $T_{\text{sel}}^{\text{exact}}/T_{\text{sel}}^{\text{block}}$. ΔFR is the test fooling rate of the Block-k-Center UAP minus that of the base-k-Center UAP; the last column counts datasets with $|\Delta\text{FR}| \leq 0.02$.

Method	**Med. sel. speedup**	**Mean ΔFR**	**% $\lvert\Delta\text{FR}\rvert \leq 2\%$**
Base-k-Center (exact)	1.00×	0.000	100.0
$b = 8$	4.18×	+0.003	81.8
$b = 16$	3.92×	+0.006	79.5
$b = 32$	3.24×	+0.000	72.7
$b = 64$	2.41×	+0.006	79.5
$b = 128$	1.78×	+0.009	79.5

C.3 Empirical Evaluation

The main-text results (§5) compare selection *objectives* and are deliberately restricted to exact selection algorithms, so that approximation error in the selection step is not conflated with the choice of geometric criterion. Here we ask an orthogonal question: *how cheaply can the k-center objective itself be approximated without changing the resulting UAP?* Table 6 sweeps the local center budget b on all 44 datasets under the default attack configuration (PGD, $\varepsilon = 0.10$, $\rho = 0.10$). Because the goal is to match exact selection, ΔFR is reported relative to base-k-Center rather than to full-UAP, and agreement is measured two-sidedly as $|\Delta\text{FR}| \leq 0.02$.

Block-k-Center UAPs are, on average, at least as strong as base-k-Center UAPs: mean ΔFR is non-negative for every budget. Per-dataset agreement is lower, with 72.7–81.8% of datasets within two points of exact selection, and it does not vary monotonically with b. For example, $b = 32$ has the smallest mean deviation (0.000) but the lowest agreement (72.7%), whereas $b = 8$ is both the fastest variant and the one with the highest agreement (81.8%). This is unlike the smooth trade-off observed for ρ in §5.2, and likely reflects the interaction between the random block partition and heterogeneity across the benchmark. As a practical default we recommend $b = 8$–16, which gives the largest reduction in selection time at agreement comparable to larger budgets. Larger b produces fewer, larger blocks and moves the selection closer to exact global traversal, with diminishing returns once selection cost is no longer the bottleneck.

The selection crossover $N_{\text{sel}} = EPc_g/D$ derived in Appendix B is the dataset size at which exact selection becomes as expensive as the subsequent coreset-based gradient optimization. Table 6 is measured in the same $N \leq 2{,}500$ regime as the rest of the benchmark, so we have *not* empirically traced this crossover. What Appendix C.2 establishes analytically is that Block-k-Center's selection cost is $O(NbD)$ rather than $O(\rho N^2 D)$, which removes the quadratic term responsible for the crossover. A direct empirical trace at larger, synthetically extended N is left to future work.