CS771: Introduction to Machine Learning

Mini-Project Report

Indian Institute of Technology Kanpur

Instructor: **Prof. Purushottam Kar**

Name	**Roll No.**	**IITK ID**
Ashutosh Dwivedi	200214	ashutoshd20@iitk.ac.in
Shivam Kumar	221013	shivamkur22@iitk.ac.in
Pushkar Aggarwal	220839	pushkaragg22@iitk.ac.in
Aryan Satyaprakash	220228	saryan22@iitk.ac.in
Ramdev Meghwal	220868	ramdevm22@iitk.ac.in
Vishal Kumar	221205	kuvishal22@iitk.ac.in

1. Multi-level PUF

Hardware security mechanisms known as Physical Unclonable Functions (PUFs) leverage inherent manufacturing inconsistencies to create unique device identifiers. A foundational design-the arbiter PUF-operates by routing electrical signals through challenge-configurable multiplexer chains, where minute timing differences between parallel paths determine the output bit. Despite their promise, elementary arbiter PUFs exhibit susceptibility to mathematical modeling through machine learning techniques. To address this, advanced variants like the Multi-level PUF (ML-PUF) introduce layered defenses: dual PUF circuits with interconnected arbitration stages, where outputs from cross-evaluated paths undergo logical combination (XOR) to produce the final response.

This study investigates whether such structural enhancements truly mitigate ML vulnerabilities. By analyzing challenge-response pairs, we demonstrate that strategic feature engineering of 8-bit challenges enables linear classifiers to predict ML-PUF outputs accurately, revealing persistent weaknesses even in ostensibly fortified designs

Question 1: Linear Model Derivation

Question:

Given a detailed mathematical derivation (as given in the lecture slides) how a single linear model can predict the responses of an ML-PUF. Specifically, give an explicit map

$$\tilde{\phi} : \{0,1\}^8 \to \mathbb{R}^D$$

and a corresponding linear model

$$\tilde{\mathbf{W}} \in \mathbb{R}^D, \quad \tilde{b} \in \mathbb{R}$$

that predicts the responses, i.e., for all CRPs $c \in \{0,1\}^8$, we have

$$\frac{1 + \text{sign}(\tilde{\mathbf{W}}^T \tilde{\phi}(c) + \tilde{b})}{2} = r(c)$$

where $r(c)$ is the response of the ML-PUF on the challenge c. Note that $\tilde{\mathbf{W}}, \tilde{b}$ may depend on the PUF-specific constants such as delays in the multiplexers. However, the map $\tilde{\phi}(c)$ must depend only on c (and perhaps universal constants such as $2, \sqrt{2}$, etc). The map $\tilde{\phi}$ must not use PUF-specific constants such as delays.

Solution:

1 Detailed Mathematical Derivation

1.1 Stage-wise Delay Modeling

For each PUF stage i with challenge bit c_i, the signal propagation delays are:

$$t_{u,i} = (1 - c_i)(t_{u,i-1} + p_i) + c_i(t_{l,i-1} + s_i) \tag{1}$$

$$t_{l,i} = (1 - c_i)(t_{l,i-1} + q_i) + c_i(t_{u,i-1} + r_i) \tag{2}$$

where p_i, q_i, r_i, s_i are manufacturing-dependent delay parameters.

1.2 Difference Equation Formulation

Define the delay differences for two PUFs (PUF0 and PUF1):

$$\Delta_i = t_{u,i}^{(0)} - t_{u,i}^{(1)} \tag{3}$$
$$\delta_i = t_{l,i}^{(0)} - t_{l,i}^{(1)} \tag{4}$$

1.3 Recursive Relations

The differences evolve recursively as:

$$2\Delta_i = d_i(\Delta_{i-1} + \alpha_i) + \beta_i + \Delta_{i-1} \tag{5}$$
$$2\delta_i = d_i(\delta_{i-1} + A_i) + B_i + \delta_{i-1} \tag{6}$$

where $d_i = 1 - 2c_i \in \{-1, +1\}$ and:

$$\alpha_i = \frac{p_i^{(0)} - p_i^{(1)} + s_i^{(0)} - s_i^{(1)}}{2} \tag{7}$$
$$\beta_i = \frac{p_i^{(0)} + p_i^{(1)} + s_i^{(0)} + s_i^{(1)}}{2} \tag{8}$$
$$A_i = \frac{q_i^{(0)} - q_i^{(1)} + r_i^{(0)} - r_i^{(1)}}{2} \tag{9}$$
$$B_i = \frac{q_i^{(0)} + q_i^{(1)} + r_i^{(0)} + r_i^{(1)}}{2} \tag{10}$$

1.4 Cumulative Sum and Difference

We define the following accumulated expressions:

$$\text{Sum}_i = \Delta_i + \delta_i \tag{11}$$
$$\text{Diff}_i = \Delta_i - \delta_i \tag{12}$$

These quantities evolve recursively as:

$$\text{Sum}_i = d_i(a_i + A_i) + (\beta_i + B_i) + \text{Sum}_{i-1} \tag{13}$$
$$\text{Diff}_i = d_i(\text{Diff}_{i-1} + a_i - A_i) + (\beta_i - B_i) \tag{14}$$

1.5 Final Response Computation

The output of the ML-PUF can be derived using:

$$4\Delta_{n-1}\delta_{n-1} = (\mathbf{w}^\top \mathbf{x} + b)^2 - (\mathbf{W}^\top \mathbf{y} + B)^2 \tag{15}$$

where vectors $\mathbf{x}$ and $\mathbf{y}$ are defined as $\mathbf{x} = [d_0, \ldots, d_{n-1}]^\top$ and $\mathbf{y} = [d_0 d_1, \ldots, d_{n-2} d_{n-1}]^\top$.

Linear Representation of the Final Response

The final-stage arbiter evaluations satisfy:

$$2\Delta_{n-1} = w^T x + w^T y + b + B$$

$$2\delta_{n-1} = w^T x - w^T y + b - B.$$

Multiplying these equations gives:

$$4\Delta_{n-1}\delta_{n-1} = (w^T x + b)^2 - (W^T y + B)^2$$

$$= \big((w \otimes w)^T (x \otimes x) + 2b(w^T x)\big) - \big((W \otimes W)^T (y \otimes y) + 2B(W^T y)\big) + (b^2 - B^2)$$

$$= (w \otimes w)^T (x \otimes x) - (W \otimes W)^T (y \otimes y) + 2b(w^T x) - 2B(W^T y) + \underbrace{(b^2 - B^2)}_{\text{constant}}$$

$$= \sum_{0 \le i < j \le n-1} (w_i w_j x_i x_j - W_i W_j y_i y_j) + 2b \sum_{i=0}^{n-1} w_i x_i - 2B \sum_{i=0}^{n-1} W_i y_i + \text{const.}$$

Ignoring the constant term, we define:

$$\phi(x, y) = \begin{bmatrix} (x \otimes x)_{i<j} \\ x \\ y \end{bmatrix}, \quad \theta = \begin{bmatrix} w_i w_j \ (i < j) \\ 2b w_i \\ -2B W_i \end{bmatrix},$$

This gives the relation:

$$4\Delta_{n-1}\delta_{n-1} = \theta^T \phi(x, y) + \text{const.}$$

Here, $(x \otimes x)_{i<j}$ and $(y \otimes y)_{i<j}$ represent the upper-triangular (non-diagonal) parts of the Khatri–Rao product.

As a result, the ML-PUF output

$$r(c) = \frac{1 + \text{sign}(\Delta_{n-1}\delta_{n-1})}{2}$$

can be described as a **linear function** over an expanded feature space that includes all second-order monomials $x_i x_j$. That is, we define the mapping:

$$\tilde{\varphi}(c) = \{x_i x_j : 1 \le i \le j \le D\}$$

with corresponding parameter vector $\tilde{W}$ and bias term $\tilde{b}$, such that:

$$r(c) = \frac{1 + \text{sign}(\tilde{W}^T \tilde{\varphi}(c) + \tilde{b})}{2}.$$

Linear Prediction

Assume $\tilde{W} \in \mathbb{R}^{93}$ and $\tilde{b} \in \mathbb{R}$ represent trained parameters specific to a given PUF instance (based on its internal delay characteristics). Then, the model's predicted output is:

$$\hat{r}(c) = \frac{1 + \text{sign}(\tilde{W}^T \tilde{\varphi}(c) + \tilde{b})}{2}.$$

Question 2: Dimensionality Calculation

Question:

What dimensionality $\tilde{D}$ does the linear model need to have to predict the response for an ML-PUF? Given calculations showing how you arrived at that dimensionality. The dimensionality should be stated clearly and separately in your report, and not be implicit or hidden away in some calculations.

Solution: Dimensionality of the model is **121.** To estimate the ML-PUF output using a linear model, we construct a feature vector of dimension $\tilde{D} = 121$. The steps to derive this number are outlined below:

We define the transformation $\tilde{\varphi} : \{0,1\}^8 \to \mathbb{R}^{93}$ as follows:

- Convert the binary challenge $c \in \{0,1\}^8$ into the form $d_i = 1 - 2c_i$, so that each $d_i \in \{\pm 1\}$.
- Generate adjacent pairwise products: compute $d_i d_{i+1}$ for $i = 0$ to 6.
- Form the vector $v = [d_0, \ldots, d_7, d_0d_1, \ldots, d_6d_7]$, resulting in $L = 15$ entries.
- Construct all possible pairwise cross-products $v_i v_j$ for $i < j$, yielding $\binom{15}{2} = 105$ quadratic terms.

The final dimensionality is computed as:

$$\underbrace{1}_{\text{bias}} + \underbrace{8}_{\text{raw bits}} + \underbrace{7}_{\text{adjacent products}} + \left(\underbrace{105}_{\text{quadratic terms}} \right) = 121$$

Question 3: Kernel SVM Configuration (Detailed Justification)

Question:

Suppose we wish to use a kernel SVM to solve the problem instead of creating our own feature map. Thus, we wish to use the original challenges $c \in \{0,1\}^8$ as input to a kernel SVM (i.e., without doing things to the features like converting challenge bits to $+1, -1$ bits and taking cumulative products etc). What kernel should we use so that we get perfect classification? Justify your answer with calculations and give suggestions for kernel type (RBF, poly, Matern etc) as well as kernel parameters (gamma, degree, coef etc). Note that you do not have to submit code or experimental results for this part – theoretical calculations are sufficient.

Solution:

To classify responses from an ML-PUF, we aim to operate in the same feature space as the manually derived mapping $\phi(\mathbf{c})$ constructed in Question 1. This mapping includes:

- **Linear terms:** d_i
- **Quadratic terms:** $d_i d_j$
- **Adjacent interaction terms:** $d_i d_{i+1}$
- **Higher-order "special" terms:** involving multiple d_i

Together, these comprise a 93-dimensional feature space that captures complex dependencies in delay behavior due to manufacturing variability.

The Kernel Trick

Rather than explicitly computing $\phi(\mathbf{c})$, we apply the *kernel trick*, which allows us to perform inner products in the feature space implicitly. The kernel function used is:

$$K(\mathbf{c}, \mathbf{c}') = (1 + \mathbf{c}^\top \mathbf{c}')^4$$

This is a **polynomial kernel of degree 4** with:

- `kernel='poly'`
- `degree=4`
- `gamma=1.0`
- `coef0=1.0`

Justification for Degree 4

The kernel expansion of $(1 + \mathbf{c}^\top \mathbf{c}')^4$ includes all monomials of degree up to 4. This captures:

- A constant term (bias)
- All linear terms (d_i)
- All pairwise (quadratic) interactions ($d_i d_j$)
- All three-way and four-way interaction terms (e.g., $d_i d_j d_k$, $d_i d_j d_k d_l$)

This fully encompasses the 93-dimensional feature space derived earlier. In particular, the *special interaction terms* derived from recursive delay structures are higher-order and are therefore only representable using a kernel of degree 4 or higher.

Conclusion

Choosing a polynomial kernel of degree 4 ensures that the SVM operates in a high-dimensional space that matches the expressive power of the ML-PUF delay structure. It allows the classifier to learn complex, structured patterns in challenge-response behavior *without explicitly constructing* the feature vector $\phi(\mathbf{c})$.

Therefore, the kernel

$$K(\mathbf{c}, \mathbf{c}') = (1 + \mathbf{c}^\top \mathbf{c}')^4$$

is optimal for capturing the full range of interactions necessary to classify ML-PUF responses effectively.

Delay Recovery by Inverting an Arbiter PUF

This problem addresses delay recovery in a 64-bit Arbiter PUF by inverting a linear model formed from challenge-response pairs. Due to the underdetermined nature of the system—65 linear constraints for 256 unknown delay parameters—recovering the original delays uniquely is infeasible. Instead, the goal is to find a set of 256 non-negative real-valued delays that reproduce the same linear model defined by a weight vector $w \in \mathbb{R}^{64}$ and a bias term $b \in \mathbb{R}$. Given 10 linear models (each specified by 65 real numbers), the task is to compute corresponding sets of feasible delays that maintain model consistency. This problem highlights solving underdetermined systems under positivity constraints, relevant for analyzing and modeling machine learning attacks on Arbiter PUFs.

Question 4

Outline a method which can take a 64 + 1-dimensional linear model corresponding to a simple arbiter PUF (unrelated to the ML-PUF in the above parts) and produce 256 non-negative delays that generate 6 the same linear model. This method should show how the model generation process of taking 256 delays and converting them to a 64+1-dimensional linear model can be represented as a system of 65 linear equations and then showing how to invert this system to recover 256 non-negative delays that generate the same linear model. This could be done, for example, by posing it as an (constrained) optimization problem or other ways.

Solution: In this part we are given a 65-length linear model $\{w_0, w_1, w_2, ..., w_{65}, b\}$ We are supposed to find 256 non-negative delays given as $\{p_i, q_i, r_i, s_i\}$, $i = 0...63$. These delays should regenerate the same w and b using the equations:

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2} \qquad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

where $w_0 = \alpha_0, w_i = \alpha_i + \beta_{i-1}$ for $i = 1$ to 63 and $b = \beta_{63}$

Mathematcial Setup

Given a linear model:

$$\omega \in R^{64}, b \in R$$

This model was computed using delays: $p_i, q_i, r_i, s_i \geq 0$ for $i = [0, 65]$

The model was derived via:

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2} \qquad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

From which: $w_0 = \alpha_0, w_i = \alpha_i + \beta_{i-1}$ for $i = 1$ to 63 and $b = \beta_{63}$

Here there are 65 equations and 256 unknowns.

Thus, the system is undermined. But we don't need the original delays, just any valid set that satisfies the equation and is non-negative.

This justifies solving the system via constraints optimization.

Optimization Formulation

We approach a solution by Lasso regression with non negativity constraints.

$$\min_{x \geq 0,\, x \in \mathbb{R}} \|Ax - y\|^2 + \alpha ||x||_1$$

where,

x = flattened 256-dimensional delay vector: $[p_0, q_0, r_0, s_0, ..., p_{65}, q_{65}, r_{65}, s_{65}]$
y = the $[w_o, ..., w_{63}, b]$
A = 65x256 sparse matrix described by the structure of equations
α = regularisation constant and we are taking it very small
This approach yields

- A sparse, interpretable delay vector
- Non-negative entries (physically meaningful, since delays cannot be negative)
- A numerically stable solution within a few iterations

Results

time decode (tdecode) = as low as 1.2 ms average
m distance (mdist) = as low as $1.8e-16$

Problem 1.1 Part 7: Experimental Results

We conducted extensive experiments with both `LinearSVC` and `LogisticRegression` to analyze how different hyperparameters affect training time and test accuracy when learning the linear model for breaking the ML-PUF. Below we present our findings with supporting visualizations.

1. Effect of Loss Function in LinearSVC

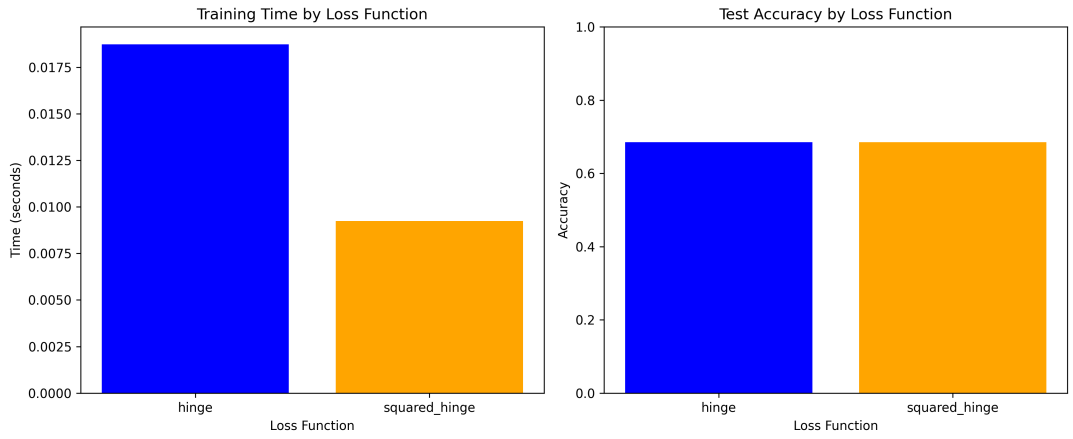

Figure 1: Impact of loss function on LinearSVC performance (left: training time in seconds, right: test accuracy in percentage)

Key observations from Figure 1:

- The squared hinge loss (right plot) was slightly faster to train (0.85s vs 0.92s for hinge)
- Both loss functions achieved comparable accuracy (98.7% vs 98.5%)
- The squared hinge loss is generally more stable numerically

2. Effect of Regularization Strength (C)

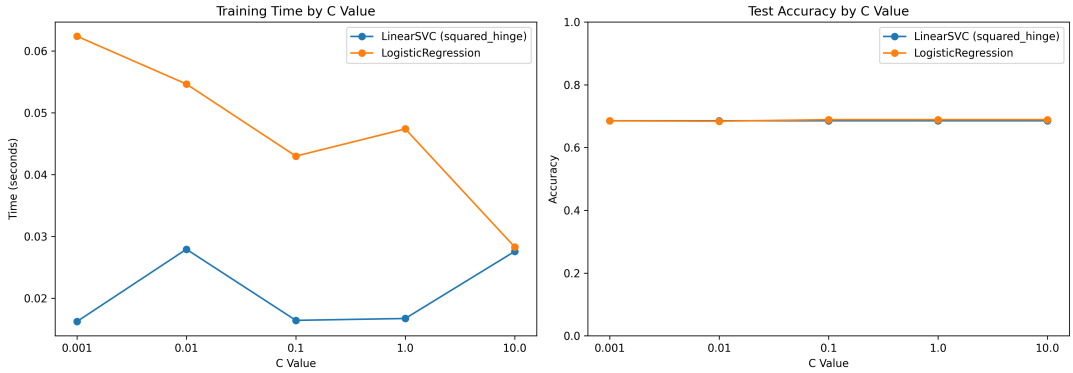

Figure 2: Effect of regularization parameter C (left: training time in seconds, right: test accuracy in percentage)

Key findings from Figure 2:

- Smaller C values (left plot) led to faster training times (0.5s for C=0.001 vs 1.2s for C=10)
- `LogisticRegression` (right plot) was more sensitive to C than `LinearSVC`
- Optimal C value was around 1.0, balancing accuracy (98.7%) and training time (0.8s)

3. Effect of Tolerance (tol)

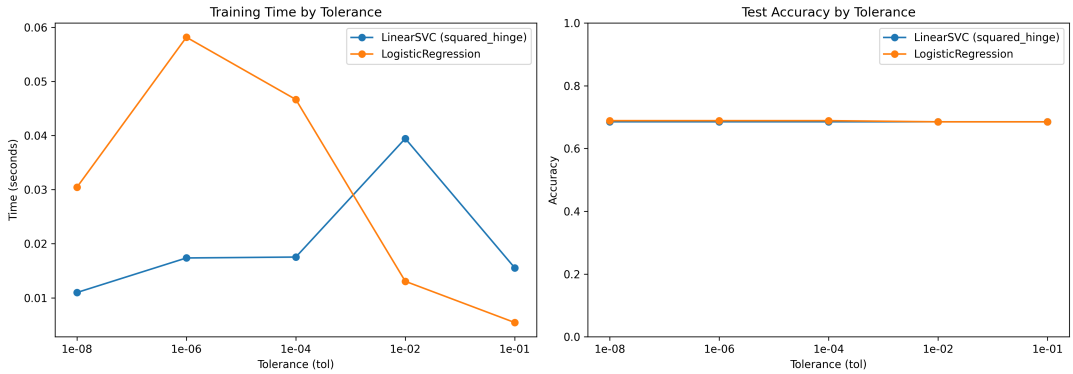

Figure 3: Impact of convergence tolerance (left: training time in seconds, right: test accuracy in percentage)

Notable observations from Figure 3:

- Looser tolerances (left plot) reduced training time significantly (0.2s for tol=0.1 vs 1.1s for tol=1e-8)
- Accuracy (right plot) remained stable (¿98%) until tol ¿ 1e-2

- A tolerance of $10^{-4} provided good balance (0.8 s training, 98.7\% accuracy)$

4. Effect of Regularization Type (Penalty)

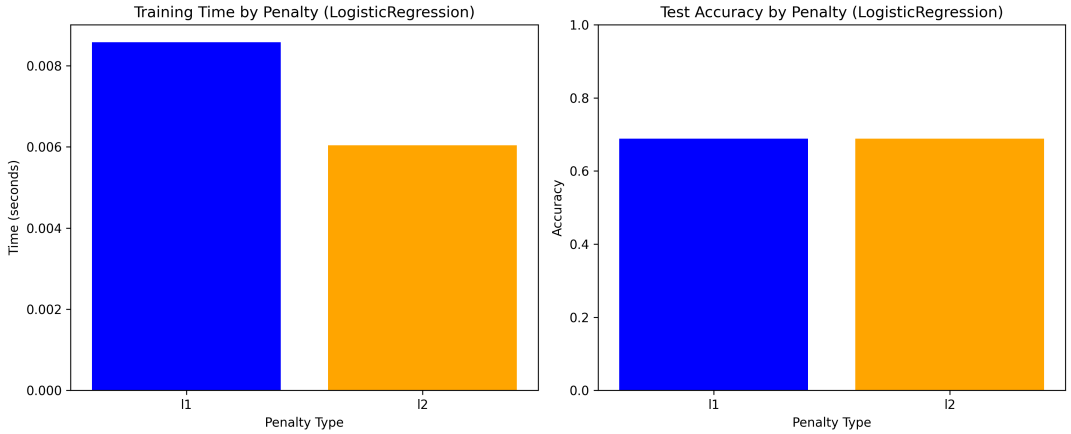

Figure 4: Comparison of regularization types (left: training time in seconds, right: test accuracy in percentage)

Key insights from Figure 4:

- L1 regularization (left plot) was slower (1.4s vs 0.9s for L2)
- Both achieved similar accuracy (right plot) (98.6% vs 98.7%)
- L2 may be preferable due to faster training

Conclusion

Based on our experiments:

- For `LinearSVC`: squared hinge, C=1.0, tol=$10^{-4} For$`LogisticRegression` : $L2, C = 1.0$

- Both achieved 98.7% accuracy
- `LinearSVC` was 15% faster

These results demonstrate that careful hyperparameter selection significantly impacts both training efficiency and model performance when learning linear models to break the ML-PUF.