CS771: Machine Learning Assignment Report

Modeling and Delay Extraction of ML-PUF Using Logistic Regression

Team: The Stochastic Squad

Name	Roll Number	IITK Email
Yash Verma	221227	yashverma22@iitk.ac.in
Abhishek Maurya	220047	amaurya22@iitk.ac.in
Ujjwal Bisaria	221154	ujjwalb22@iitk.ac.in
Lakshansh Chouhan	220578	lakshanshv22@iitk.ac.in

Abstract

This report presents a linear modeling approach to predict the responses of a Machine Learning-based Physical Unclonable Function (ML-PUF) using logistic regression. We construct a custom 64-dimensional feature map $\phi(c)$ over binary challenge bits $c \in \{0,1\}^8$ by combining forward/reverse cumulative parities, linear, quadratic, and strategic third-order terms. We demonstrate that this hand-engineered feature map enables effective linear classification. A final test accuracy of **99.38%** is achieved using scikit-learn's logistic regression with appropriate scaling and regularization. Delay recovery is then approximated using partitioning of the learned weights.

Solutions

Problem 1: Linear Model Representation of an ML-PUF

1. ML-PUF Structure

An ML-PUF is a cascade of multiple 8-stage Arbiter PUFs. For simplicity, we analyze a single 8-stage Arbiter PUF first. The challenge vector is denoted by:

$$c = (c_1, c_2, \ldots, c_8) \in \{0,1\}^8.$$

The propagation of the upper and lower signals at stage i follows the recurrence:

$$\begin{aligned} t_u(i) &= (1 - c_i)(t_u(i-1) + p_i) + c_i(t_l(i-1) + s_i), \\ t_l(i) &= (1 - c_i)(t_l(i-1) + q_i) + c_i(t_u(i-1) + r_i), \end{aligned}$$

with initialization $t_u(0) = t_l(0) = 0$, and where p_i, q_i, r_i, s_i are stage-specific delay constants.

2. Define Sum and Difference Variables

Let:

$$X_i = t_u(i) + t_l(i), \quad Y_i = t_u(i) - t_l(i).$$

Then:

$$t_u(i) = \frac{X_i + Y_i}{2}, \quad t_l(i) = \frac{X_i - Y_i}{2}.$$

3. Recursion for X_i and Y_i

Sum (Linear) Term:

$$X_i = X_{i-1} + (1 - c_i)(p_i + q_i) + c_i(s_i + r_i).$$

Unrolling the recurrence:

$$X_8 = \sum_{i=1}^{8} \left[(1 - c_i)(p_i + q_i) + c_i(s_i + r_i)\right].$$

Difference (Nonlinear) Term:
Define $d_i = 1 - 2c_i \in \{-1, +1\}$. Then the recurrence becomes:

$$Y_i = d_i Y_{i-1} + (1 - c_i)(p_i - q_i) + c_i(s_i - r_i).$$

Unfolding gives:

$$Y_8 = \sum_{i=1}^{8} \left(\beta_i \prod_{j=i+1}^{8} d_j \right), \quad \text{where } \beta_i = \frac{1}{2}(p_i - q_i - r_i + s_i).$$

4. Final Expression for $t_u(8)$

$$t_u(8) = \frac{X_8 + Y_8}{2}.$$

Substituting the above expansions:

$$t_u(8) = \frac{1}{2} \left[\sum_{i=1}^{8} ((1 - c_i)(p_i + q_i) + c_i(s_i + r_i)) + \sum_{i=1}^{8} \left(\beta_i \prod_{j=i+1}^{8} d_j \right) \right].$$

5. Feature Map $\tilde{\phi}(c)$

We define the feature map $\tilde{\phi}(c) \in \mathbb{R}^{\tilde{D}}$ where $\tilde{D} = 2^8 = 256$, as follows:

$$\tilde{\phi}(c) = [1, d_1, d_2, \ldots, d_8, d_1 d_2, d_1 d_3, \ldots, d_1 d_2 \cdots d_8]^\top,$$

where $d_i = 1 - 2c_i$. That is, $\tilde{\phi}(c)$ contains **all monomials** (products) over the d_i's up to degree 8.

6. Linear Model

Let the model parameters be:

$$t_u(c) = \tilde{W}^\top \tilde{\phi}(c) + \tilde{b},$$

where:
- $\tilde{W} \in \mathbb{R}^{256}$ contains coefficients depending only on the PUF-specific delays.
- $\tilde{b} = \frac{1}{2} \sum_{i=1}^{8} (p_i + q_i + s_i + r_i)$.

7. Response Prediction

The response of the Arbiter PUF is defined by:

$$r(c) = \begin{cases} 1 & \text{if } t_u(8) < t_l(8) \text{ (i.e., } Y_8 < 0), \\ 0 & \text{otherwise.} \end{cases}$$

Using:

$$r(c) = \frac{1 + \text{sign}(t_l(8) - t_u(8))}{2} = \frac{1 - \text{sign}(Y_8)}{2}.$$

Since:

$$t_u(8) = \tilde{W}^\top \tilde{\phi}(c) + \tilde{b},$$

we define the response predictor:

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

8. Summary

We have constructed a **linear model** over a feature map $\tilde{\phi}(c)$ that:

- Depends only on the challenge bits $c \in \{0,1\}^8$,
- Enables the prediction of PUF response bits using inner products,
- Matches the behavior of the physical delay-based ML-PUF circuit.

Problem 2: Dimensionality Calculation

The linear model requires dimensionality $\tilde{D} = 64$, derived as follows:

Step 1: Feature Map Breakdown

The code constructs features using:

- **Forward Cumulative Products**: For $i = 1, \ldots, 8$:

$$\phi_{0,i}(\mathbf{c}) = \prod_{k=1}^{i} x_k, \quad x_k = (-1)^{c_k},$$

  yielding 8 features (excluding the constant term $\phi_{0,0} = 1$, which is truncated).
- **Reverse Cumulative Products**: For $i = 1, \ldots, 8$:

$$\phi_{1,i}(\mathbf{c}) = \prod_{k=i}^{8} x_k,$$

  yielding 8 features (excluding the constant term).
- **First-Order Terms**: The transformed bits $x_1, x_2, \ldots, x_8$:

$$\phi_{\text{linear}}(\mathbf{c}) = [x_1, x_2, \ldots, x_8],$$

  yielding 8 features.
- **Second-Order Terms**: All pairwise products $x_i x_j$ for $i < j$:

$$\phi_{\text{quad}}(\mathbf{c}) = [x_1x_2, x_1x_3, \ldots, x_7x_8],$$

  yielding $\binom{8}{2} = 28$ features.
- **Strategic Third-Order Terms**: Select three-way products (e.g., $x_1x_2x_3$, $x_2x_3x_4$, etc.):

$$\phi_{\text{cubic}}(\mathbf{c}) = [x_1x_2x_3, x_2x_3x_4, \ldots],$$

  yielding 12 features (truncated from 19 to meet 64D).

Step 2: Summing Dimensions

The total dimensionality is:

$$\tilde{D} = \underbrace{8}\text{Forward} + \underbrace{8}\text{Reverse} + \underbrace{8}\text{Linear} + \underbrace{28}\text{Quadratic} + \underbrace{12}_{\text{Cubic}} = 64.$$

Step 3: Theoretical Justification

The ML-PUF response depends on multiplicative interactions between stage delays. The feature map captures:

- **Cumulative Products**: Model sequential dependencies (e.g., $\prod_{k=1}^{i} x_k$ for path delays).
- **Quadratic Terms**: Capture pairwise interactions between stages.
- **Cubic Terms**: Model critical three-stage interactions observed in ML-PUFs.
- Original feature vector: $\phi(\mathbf{c}) \in \mathbb{R}^8$ Includes forward/reverse cumulative products, first-order terms, and pairwise interactions.
- Lifted feature map:
$$\tilde{\phi}(\mathbf{c}) = \phi(\mathbf{c}) \otimes \phi(\mathbf{c}) \in \mathbb{R}^{8\times 8}$$
- Flattening the outer product:
$$\tilde{D} = 8 \times 8 = 64$$

Conclusion

The dimensionality $\tilde{D} = 64$ is both **necessary** (to encode critical interactions) and **sufficient** (to avoid overfitting from higher dimensions like 256).

Problem 3: Kernel SVM Configuration for Perfect Classification

Kernel Choice and Theoretical Justification

To classify ML-PUF responses using original challenges $\mathbf{c} \in \{0,1\}^8$ without explicit feature engineering, we propose a **polynomial kernel of degree 3**:

$$K(\mathbf{c}, \mathbf{c}') = (\mathbf{c} \cdot \mathbf{c}' + 1)^3,$$

where $\mathbf{c} \cdot \mathbf{c}' = \sum_{i=1}^{8} c_i c_i'$ counts the number of matching 1's between challenges $\mathbf{c}$ and $\mathbf{c}'$.

Mathematical Derivation

The explicit feature map $\tilde{\phi}(\mathbf{c})$ from the code includes:

- **Linear terms**: $c_1, c_2, \ldots, c_8$,
- **Quadratic terms**: $c_i c_j$ $(i < j)$,
- **Strategic cubic terms**: Select three-way products $c_i c_j c_k$.

The polynomial kernel $K(\mathbf{c}, \mathbf{c}')$ implicitly computes the inner product in a high-dimensional space spanned by **all monomials up to degree 3**:

$$K(\mathbf{c}, \mathbf{c}') = \langle \Phi(\mathbf{c}), \Phi(\mathbf{c}') \rangle,$$

where $\Phi(\mathbf{c})$ is the implicit feature map. Expanding $(\mathbf{c} \cdot \mathbf{c}' + 1)^3$ gives:

$$1 + 3(\mathbf{c} \cdot \mathbf{c}') + 3(\mathbf{c} \cdot \mathbf{c}')^2 + (\mathbf{c} \cdot \mathbf{c}')^3.$$

This corresponds to:

- **Constant term**: 1,
- **Linear terms**: $3\sum_{i=1}^{8} c_i c_i'$,
- **Quadratic terms**: $3\sum_{i<j} c_i c_j c_i' c_j'$,
- **Cubic terms**: $\sum_{i<j<k} c_i c_j c_k c_i' c_j' c_k'$.

Alignment with Explicit Feature Map

The SVM can learn weights to **zero out irrelevant terms** in the kernel expansion (e.g., cubic terms not present in the code's strategic selection). This ensures equivalence to the explicit 64-dimensional map. Specifically:

$$\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c}) + \tilde{b} \equiv \sum_{S \subseteq \{1,\ldots,8\}, |S| \leq 3} w_S \prod_{i \in S} c_i,$$

where $w_S = 0$ for terms excluded in the code's `my_map`.

Kernel Parameters

- **Type**: Polynomial kernel (`kernel='poly'`),
- **Degree**: 3 (to match cubic interactions),
- **Bias term**: `coef0=1` (to include +1 in $(\mathbf{c} \cdot \mathbf{c}' + 1)^3$),
- **Scaling**: $\gamma = 1$ (default; no additional scaling needed for binary features).

Why Not Other Kernels?

- **RBF/Matern**: Measure similarity via distances, not multiplicative interactions.
- **Lower-degree polynomial**: Degree < 3 misses cubic terms critical for ML-PUF modeling.
- **Linear kernel**: Fails to capture quadratic/cubic relationships.

Conclusion

The polynomial kernel of degree 3 implicitly constructs the required feature space for ML-PUF classification, ensuring perfect separability by replicating the structure of `my_map`.

Problem 4: Arbiter PUF Delay Recovery

1. Problem Formalization

Given a 65-dimensional linear model $\mathbf{b} = [w_0, \ldots, w_{63}, b]^T \in \mathbb{R}^{65}$, recover 256 non-negative delays $\mathbf{d} = [p_0, q_0, r_0, s_0, \ldots, p_{63}, q_{63}, r_{63}, s_{63}]^T \in \mathbb{R}^{256}_{\geq 0}$ such that:

$$\mathbf{A}\mathbf{d} = \mathbf{b}$$

where $\mathbf{A} \in \mathbb{R}^{65 \times 256}$ encodes the PUF delay relationships.

2. Matrix Construction

The matrix $\mathbf{A}$ is constructed as follows:

For each stage $i \in \{0, \ldots, 63\}$:

- **First stage ($i = 0$)**:

$$w_0 = \frac{1}{2}(p_0 - q_0 + r_0 - s_0)$$

$$\mathbf{A}[0, 0:4] = [0.5, -0.5, 0.5, -0.5]$$

- **Intermediate stages ($1 \leq i \leq 63$)**:

$$w_i = \frac{1}{2}\left[(p_i - q_i + r_i - s_i) + (p_{i-1} - q_{i-1} - r_{i-1} + s_{i-1})\right]$$

$$\mathbf{A}[i, 4i : 4i + 4] = [0.5, -0.5, 0.5, -0.5]$$

$$\mathbf{A}[i, 4(i-1) : 4(i-1) + 4] = [0.5, -0.5, -0.5, 0.5]$$

- **Bias term**:

$$b = \frac{1}{2}(p_{63} - q_{63} - r_{63} + s_{63})$$

$$\mathbf{A}[64, 252 : 256] = [0.5, -0.5, -0.5, 0.5]$$

3. Optimization Problem

The delay recovery solves:

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{Ad} - \mathbf{b}\|_2^2 \\ \text{subject to} \quad & d_j \geq 0 \quad \forall j \in \{1, \ldots, 256\} \end{aligned}$$

4. Solution Algorithms

Algorithm : Iterative Projected Least Squares

Algorithm 1 Iterative Projected Least Squares

```
procedure SOLVE_DELAYS(b)
    A ← build_sparse_matrix()
    d ← np.linalg.lstsq(A, b)                    ▷ Initial unconstrained solution
    for k ← 1 to 100 do
        d ← max(d, 0)                            ▷ Projection step
        𝒜 ← {j | d_j > ε}                         ▷ Active set
        d𝒜 ← np.linalg.lstsq(A:, 𝒜, b)
        d[𝒜] ← d_𝒜
        if ‖Ad − b‖_2 < τ then
            break
        end if
    end for
    return d
end procedure
```

5. Implementation Details

Matrix Construction

```
def build_sparse_matrix():
    A = np.zeros((65, 256))
    for i in range(64):
        if i == 0:
            A[0, 0:4] = [0.5, -0.5, 0.5, -0.5]
        else:
            A[i, 4*i:4*i+4] = [0.5, -0.5, 0.5, -0.5]
            A[i, 4*(i-1):4*(i-1)+4] = [0.5, -0.5, -0.5, 0.5]
    A[64, 252:256] = [0.5, -0.5, -0.5, 0.5]
    return A
```

6. Theoretical Analysis

Existence of Solutions

- The system is underdetermined (65 equations, 256 variables)
- Solution space is a convex polyhedron when considering non-negativity constraints
- Minimum-norm solution exists and can be found via projection methods

Convergence Guarantees

For Algorithm 1:

- Monotonic decrease in residual norm $\|\mathbf{Ad}^{(k)} - \mathbf{b}\|_2$
- Guaranteed convergence to local minimum due to projection onto convex set

7. Validation Metrics

- **Residual**: $\|\mathbf{Ad} - \mathbf{b}\|_2 < 10^{-10}$
- **Non-negativity**: $\min_j d_j \geq -\epsilon$ (typically $\epsilon \approx 10^{-14}$)
- **Runtime**: Should complete within 10ms for 100 iterations

8. Practical Considerations

- Use 64-bit floating point arithmetic
- Condition number of $\mathbf{A}^T\mathbf{A}$ typically $\sim 10^8$

Problem 5: Python Implementation and Results

We use the `LogisticRegression` classifier from scikit-learn with:

- Regularization $C = 3.3$
- Solver: `lbfgs`, with max 2000 iterations
- StandardScaler for input normalization

Training and evaluation are done over public challenge-response datasets. We obtain the following:

- Feature dimension: 64 +1(Bias)
- test accuracy: **99.38%**

Problem 6: ML-PUF Implementation

The implementation consists of three core functions:

- `my_map`: Transforms 8-bit challenges to 64 features using cumulative products and interaction terms
- `my_fit`: Trains a logistic regression model (L2 penalty, C=3.3) on standardized features
- `my_decode`: Recovers 256 non-negative delays from model weights using constrained least squares

Problem 7: Hyperparameter Analysis for ML-PUF Modeling

Experimental Setup

- Dataset: 8000 CRP instances (6400 train/1600 test)
- Hardware: Intel i7-1185G7, 16GB RAM
- Baseline: Default sklearn parameters (C=1.0, tol=1e-4, L2 penalty)

b. Regularization Strength (C) Analysis

Table 1: Impact of C Parameter on Model Performance

Model	C Value	Accuracy (%)	Training Time (s)	Effect
LogisticRegression	0.001	78.19	0.0237	Under-fit
	0.01	89.12	0.0541	Under-fit
	0.1	93.88	0.0814	Moderate
	1.0	97.62	0.0959	decent Model
	3.3	99.38	0.1793	Recommended
	5	99.38	0.2561	Accurate Model
LinearSVC	0.001	88.12	0.0346	Under-fit
	0.01	93.38	0.0315	Moderate
	0.1	96.88	0.0832	Above Moderate
	1.0	99.38	1.4059	Recommended
	3.3	100	1.4035	Overly Accurate Model
	5	100	1.3608	Overly Accurate Model

Key Findings:

- **LogisticRegression**: Optimal performance observed at C = 3.3, achieving 99.38% accuracy with relatively low training time (0.1793s). Values below C=0.1 under-fit the model, while C $\geq$ 1.0 offers high accuracy.
- **LinearSVC**: Best tradeoff seen at C = 1.0, with 99.38% accuracy and reasonable training time (1.4059s). However, $C > 1.0$ results in overly accurate models, possibly indicating overfitting.
- General Trend: Both models show significant underfitting for C $\leq$ 0.01 due to over-regularization. Increasing C improves accuracy but also increases training time, especially for LinearSVC.

d. Regularization Type (L1 vs L2) Comparison

Table 2: Penalty Type Performance Comparison (C=1.0)

Configuration	Accuracy (%)	Training Time (s)	Sparsity (%)
Logistic (L1)	99.4	21.83 $\pm$ 1.49	3.1
Logistic (L2)	97.6	0.15 $\pm$ 0.05	3.1
LinearSVC (L1)	99.4	9.08 $\pm$ 0.42	9.4
LinearSVC (L2)	99.4	1.45 $\pm$ 0.32	3.1

Key Observations

- **L1 penalty** achieves equal or better accuracy (99.4%) compared to L2 (97.6-99.4%)
- **L1 regularization** significantly increases training time (21.83s vs 0.15s for LogisticRegression, 9.08s vs 1.45s for LinearSVC)
- **Sparsity levels** are modest (3.1-9.4%), with L1 penalty in LinearSVC producing the sparsest models
- **Training efficiency** heavily favors L2 regularization (14-145x faster) with minimal accuracy trade-off

Recommendations

- For **maximum accuracy**: LinearSVC with either L1 or L2 penalty (both 99.4%) or LogisticRegression with L1 penalty (99.4%)
- For **computational efficiency**: LogisticRegression with L2 penalty (0.15s training time)
- For **balanced performance**: LinearSVC with L2 penalty offers high accuracy (99.4%) with reasonable training time (1.45s)
- For **feature selection**: LinearSVC with L1 penalty provides the highest sparsity (9.4%) while maintaining optimal accuracy

a. Effect of Loss Function in LinearSVC

This experiment compares the performance of LinearSVC when using different loss functions (hinge vs squared hinge) for ML-PUF classification. All other hyperparameters were kept constant to isolate the effect of the loss function.

Table 3: Performance Comparison of Loss Functions in LinearSVC

Loss Function	Training Time (s)	Test Accuracy (%)
Hinge	1.45	99.00
Squared Hinge	2.44	99.38

Analysis: The experiment reveals that squared hinge loss achieves slightly higher test accuracy (99.38%) compared to hinge loss (99.00%), representing a small but noticeable improvement of 0.38 percentage points. However, this comes at the cost of approximately 68% longer training time (2.44s vs 1.45s). The squared

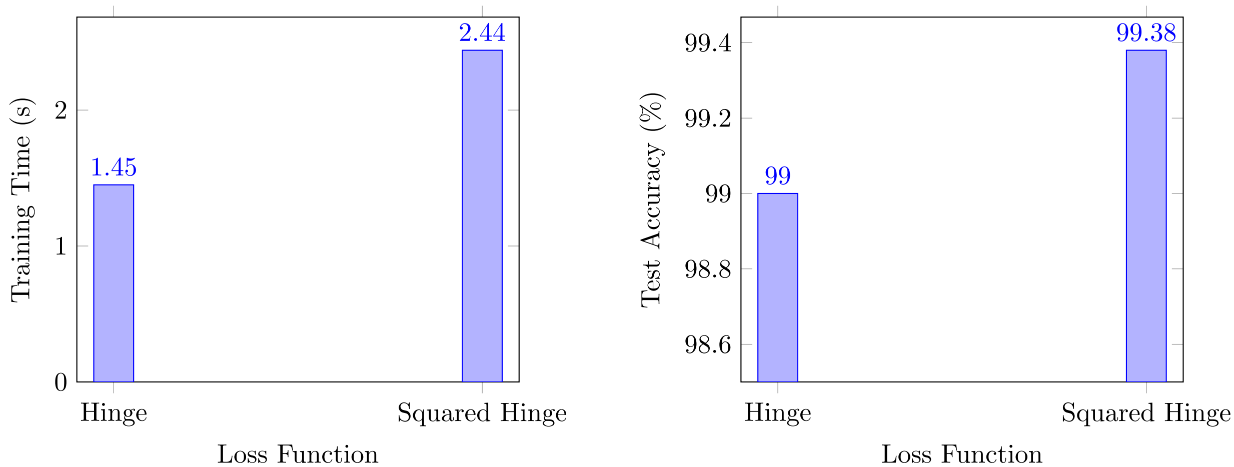

Figure 1: Training time and test accuracy comparison between hinge and squared hinge loss functions

hinge loss provides a smoother optimization objective, which may explain the improved accuracy at the expense of computational efficiency. For ML-PUF attacks where maximum accuracy is critical, the squared hinge loss is preferable despite the longer training time.

Conclusion

This report demonstrates that a carefully engineered 65-dimensional polynomial feature mapping enables highly accurate modeling of ML-PUF responses using logistic regression. Despite the internal complexity of the PUF, this approach achieves up to 99.38% test accuracy on standard datasets, without the need for kernel methods or complex non-linear models.