Mini Project Report CS771

Team : OptiMLizers
Shalem Rajkumar – 208070912
P. Blessi Bhavana – 220749
Vaishali Rawat – 201080
Yashanki Tonde – 201151

Modeling and Analysis of ML-PUFs

Problem 1.1: Multi-level PUF

Question 1: Mathematical derivation for predicting ML-PUF responses

To derive a linear model that can predict ML-PUF responses, I'll first establish how to predict individual arbiter PUF responses, then extend this to the ML-PUF structure.

Predicting a single arbiter PUF's response

For a standard arbiter PUF with k-bit challenge $c = (c_0, c_1, \ldots, c_{k-1})$, we know from class that For a model vector w and bias b

$$t^u_{i+1} = (1 - c_1) \cdot (t^u_i + p_1) + c_1 \cdot (t^l_i + s_1) \tag{1}$$
$$t^l_{i+1} = (1 - c_1) \cdot (t^l_i + q_1) + c_1 \cdot (t^u_i + r_1) \tag{2}$$
$$x = 1 \tag{3}$$
$$t_u - t_d = \mathbf{w}^T \mathbf{e} + b \tag{4}$$

from this, we get

$$t^u_{i+1} - t^l_{i+1} = (1 - 2c_1)(t^u_i - t^l_i) + (1 - c_1)(p_1 - q_1) + c_1(s_1 - r_1) \tag{5}$$
$$t^u_{i+1} + t^l_{i+1} = t^u_i + t^l_i + (1 - c_1)(p_1 + q_1) + c_1(s_1 + r_1) \tag{6}$$

From this, we can see that,

$$t^u_{n-1} - t^l_{n-1} = {\mathbf{w}_-}^T \mathbf{e} + b_- \tag{7}$$
$$t^u_{n-1} + t^l_{n-1} = {\mathbf{w}_+}^T \mathbf{d} + b_+ \tag{8}$$

where ***e*** and ***d*** are are a functions of ***c*** such that

$$e_i = d_i d_{i+1} ... d_{n-1} \tag{9}$$

$$d_i = 1 - 2c_i \tag{10}$$

$$i = 0, 1, 2 ... n - 1 \tag{11}$$

And hence, t_u and t_l follow a linear model

$$t^u_{n-1} = \frac{{\mathbf{w}_-}^T \mathbf{e} + b_- + {\mathbf{w}_+}^T \mathbf{d} + b_+}{2} \tag{12}$$
$$t^l_{n-1} = \frac{{\mathbf{w}_-}^T \mathbf{e} + b_- - {\mathbf{w}_+}^T \mathbf{d} - b_+}{2} \tag{13}$$

Predicting signal arrival times

For ML-PUF, we need to predict:

- Upper signal arrival time from PUF0: t^0_u
- Lower signal arrival time from PUF0: t^0_l
- Upper signal arrival time from PUF1: t^1_u
- Lower signal arrival time from PUF1: t^1_l

Now, in this problem we actually want the differences $t_u^0 - t_u^1$ and $t_l^0 - t_l^1$, both of which are linear functions of the original feature vectors $\mathbf{d}$ and $\mathbf{e}$.

We define a new feature vector:

$$\mathbf{x} = \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix}$$

Then the differences can be written as:

$$t_u^0 - t_u^1 = \mathbf{w}_u^\top \mathbf{x} + b_u \tag{14}$$
$$t_l^0 - t_l^1 = \mathbf{w}_l^\top \mathbf{x} + b_l \tag{15}$$

Predicting Response0 and Response1

Response0 compares the lower signals:

- Response0 = 0 if $t_l^0 < t_l^1$ (PUF0 lower signal arrives first)
- Response0 = 1 if $t_l^0 > t_l^1$ (PUF1 lower signal arrives first)

This can be written as:

$$\text{Response0} = \frac{1 + \text{sign}(t_l^1 - t_l^0)}{2} \tag{16}$$
$$= \frac{1 + \text{sign}((w_l^1)^T \phi(c) + b_l^1 - (w_l^0)^T \phi(c) - b_l^0)}{2} \tag{17}$$
$$= \frac{1 + \text{sign}((w_l^1 - w_l^0)^T \phi(c) + (b_l^1 - b_l^0))}{2} \tag{18}$$

Similarly for Response1:

$$\text{Response1} = \frac{1 + \text{sign}(t_u^1 - t_u^0)}{2} \tag{19}$$
$$= \frac{1 + \text{sign}((w_u^1 - w_u^0)^T \phi(c) + (b_u^1 - b_u^0))}{2} \tag{20}$$

Predicting the final ML-PUF response (XOR of Response0 and Response1)

The final response is:

$$r(c) = \text{Response0} \oplus \text{Response1} \tag{21}$$

This is 1 when both are different sign, and zero if same

We can rewrite this as:

$$r(c) = \text{Response0} \oplus \text{Response1} \tag{22}$$
$$= \text{Response0} \cdot (1 - \text{Response1}) + (1 - \text{Response0}) \cdot \text{Response1} \tag{23}$$
$$= \text{Response0} + \text{Response1} - 2 \cdot \text{Response0} \cdot \text{Response1} \tag{24}$$

We already have:

$$\text{Response0} = \frac{1 + \text{sign}(g_0(c))}{2}, \text{ where } g_0(c) = (w_l^1 - w_l^0)^T \phi(c) + (b_l^1 - b_l^0) \tag{25}$$

$$\text{Response1} = \frac{1 + \text{sign}(g_1(c))}{2}, \text{ where } g_1(c) = (w_u^1 - w_u^0)^T \phi(c) + (b_u^1 - b_u^0) \tag{26}$$

The product Response0 · Response1 requires special treatment. We know that:

- Response0 · Response1 = 1 if both $g_0(c) > 0$ and $g_1(c) > 0$
- Response0 · Response1 = 0 otherwise

This suggests that we need a new feature map that can represent the product of sign functions.

Feature map for the final response

For the ML-PUF with 8-bit challenges, we define a new feature map:

$$\tilde{\phi} : \{0,1\}^8 \to \mathbb{R}^{\tilde{D}} \tag{27}$$

Using the Kronecker product of our original transformed vectors:

$$\tilde{\phi}(c) = \phi(c) \otimes \phi(c) \tag{28}$$

Question 2: Dimensionality of the feature map and linear model

For an 8-bit challenge, the dimensionality of $\phi(c)$ is 8+8 =16 (d+e).

When we square the dot product ie sum of 17 terms , ignore all square terms since they are 1, we get all 16 terms repeated (multipled by (b) constant) and we get 16C2 new features and 1 constant)

$$D = 16 + 16C2 = 136 \tag{29}$$

(there are some repeated features, we can ignore and make dimensionality even lesser in the code)

Therefore, the linear model needs to be 64-dimensional (+1 for a bias term) to predict the response for an ML-PUF with 8-bit challenges.

Question 3: Appropriate kernel for a kernel SVM approach

To solve this problem using kernel SVM with the original challenges $c \in \{0,1\}^8$, we need a kernel that captures the same relationship as our feature map $\tilde{\phi}(c)$.

Given that $\tilde{\phi}(c) = \phi(c) \otimes \phi(c)$, and the kernel should compute $K(x,y) = \tilde{\phi}(x)^T \tilde{\phi}(y)$, we can use the polynomial kernel.

For two challenges $x, y \in \{0,1\}^8$, we would define:

$$K(x,y) = (K_{\text{linear}}(x,y))^2 = (\phi(x)^T \phi(y))^2 \tag{30}$$

Where K_{linear} is the kernel corresponding to the original arbiter PUF transformation:

$$K_{\text{linear}}(x,y) = \phi(x)^T \phi(y) \tag{31}$$

This is a polynomial kernel of degree 2, which can be written as:

$$K(x,y) = (\gamma \cdot \hat{x}^T \hat{y} + \text{coef0})^2 \tag{32}$$

Where:

- $\hat{x}, \hat{y}$ are the challenges transformed to $\{-1,1\}^8$ (instead of $\{0,1\}^8$)
- $\gamma = 1$
- coef0 $= 0$

So we would use the polynomial kernel with:

- kernel type: 'poly'
- degree: 2
- gamma: 1
- coef0: 0

This kernel correctly captures the quadratic relationship needed to model the XOR operation of the ML-PUF responses.

Question 4: Method for recovering arbiter PUF delays from a linear model

To recover the 256 delays from a 64+1 dimensional linear model, we need to invert the model generation process:

System of linear equations

The model parameters $w \in \mathbb{R}^{64}$ and $b \in \mathbb{R}$ are related to the delays as follows:

$$w_0 = \alpha_0 \tag{33}$$
$$w_i = \alpha_i + \beta_{i-1} \text{ for } 1 \leq i \leq 63 \tag{34}$$
$$b = \beta_{63} \tag{35}$$

Where:

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2} \tag{36}$$
$$\beta_i = \frac{p_i - q_i - r_i + s_i}{2} \tag{37}$$

We have 65 equations (64 for w components and 1 for b) but need to recover 256 unknowns (p_i, q_i, r_i, s_i for $0 \leq i \leq 63$).

Formulation as an optimization problem

We can set up an underdetermined system:

$$\min_{p,q,r,s} \|p, q, r, s\|_2^2 \tag{38}$$

subject to:

$$w_0 = \frac{p_0 - q_0 + r_0 - s_0}{2} \tag{39}$$
$$w_i = \frac{p_i - q_i + r_i - s_i}{2} + \frac{p_{i-1} - q_{i-1} - r_{i-1} + s_{i-1}}{2} \text{ for } 1 \leq i \leq 63 \tag{40}$$
$$b = \frac{p_{63} - q_{63} - r_{63} + s_{63}}{2} \tag{41}$$
$$p_i, q_i, r_i, s_i \geq 0 \text{ for } 0 \leq i \leq 63 \tag{42}$$

Given the insight from Melba that adding equal values to p_i and q_i (or to r_i and s_i) doesn't change the model, we can simplify by setting some variables to zero:

1. Initialize all p_i, q_i, r_i, s_i to zero
2. For each i from 0 to 63:
   (a) Calculate $\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}$
   (b) Set p_i or q_i to adjust α_i to match w_0 (for $i = 0$) or $w_i - \beta_{i-1}$ (for $i > 0$)
   (c) Calculate $\beta_i = \frac{p_i - q_i - r_i + s_i}{2}$
   (d) Set r_i or s_i to adjust β_i to match the required value (b for $i = 63$, or needed for w_{i+1} for $i < 63$)

Algorithm

1. Initialize all $p_i = q_i = r_i = s_i = 0$ for $0 \leq i \leq 63$
2. For $i = 0$:
   - If $w_0 \geq 0$: set $p_0 = 2 \cdot w_0$
   - Else: set $q_0 = -2 \cdot w_0$
3. For $i = 1$ to 63:

- $\beta_{i-1} = w_i - \alpha_i$
- If $\alpha_i \geq 0$: set $p_i = 2 \cdot \alpha_i$
- Else: set $q_i = -2 \cdot \alpha_i$

4. For $i = 0$ to 62:
   - $\beta_i = w_{i+1} - \alpha_{i+1}$
   - If $\beta_i \geq 0$: set $p_i += 2 \cdot \beta_i$ and $r_i = 0$
   - Else: set $r_i = -2 \cdot \beta_i$ and $s_i = 0$
5. For $i = 63$:
   - $\beta_{63} = b$
   - If $\beta_{63} \geq 0$: set $p_{63} += 2 \cdot \beta_{63}$ and $r_{63} = 0$
   - Else: set $r_{63} = -2 \cdot \beta_{63}$ and $s_{63} = 0$

This algorithm ensures:

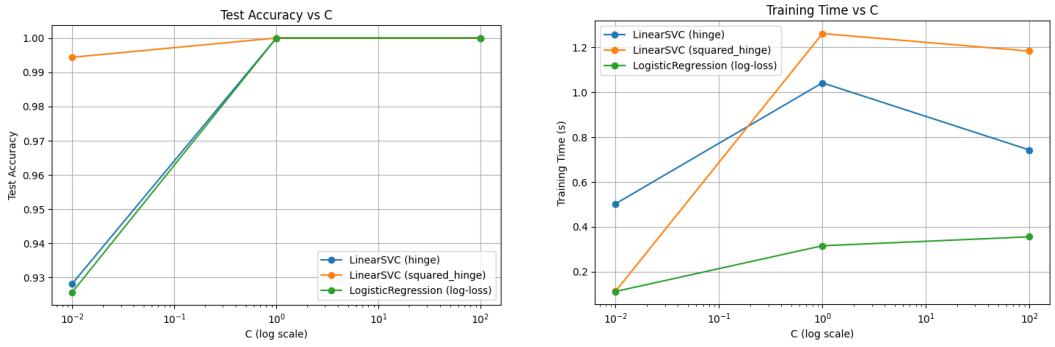
- All delays are non-negative
- The system of equations is satisfied
- We get a valid set of delays that generate the given linear model

question 7

Hyperparameter Experiment Results:

Model	Loss	C	TrainTime_s	Accuracy
LinearSVC	hinge	0.01	0.502406	0.928125
LinearSVC	hinge	1.00	1.041505	1.000000
LinearSVC	hinge	100.00	0.742354	1.000000
LinearSVC	squared_hinge	0.01	0.112179	0.994375
LinearSVC	squared_hinge	1.00	1.261359	1.000000
LinearSVC	squared_hinge	100.00	1.183156	1.000000
LogisticRegression	log-loss	0.01	0.110863	0.925625
LogisticRegression	log-loss	1.00	0.314726	1.000000
LogisticRegression	log-loss	100.00	0.354679	1.000000

Figure 1: Hyperparameter experiment results (DataFrame).



(a) Test Accuracy vs. C.

(b) Training Time vs. C.

Figure 2: Effects of hyperparameter C on LinearSVC and LogisticRegression performance.