Group 76: survivors_of_ml

Asit Biswas
241110014
MTech CSE, IIT Kanpur
asitbiswas24@iitk.ac.in

Jatin Jangir
241110031
MTech CSE, IIT Kanpur
jatinj24@iitk.ac.in

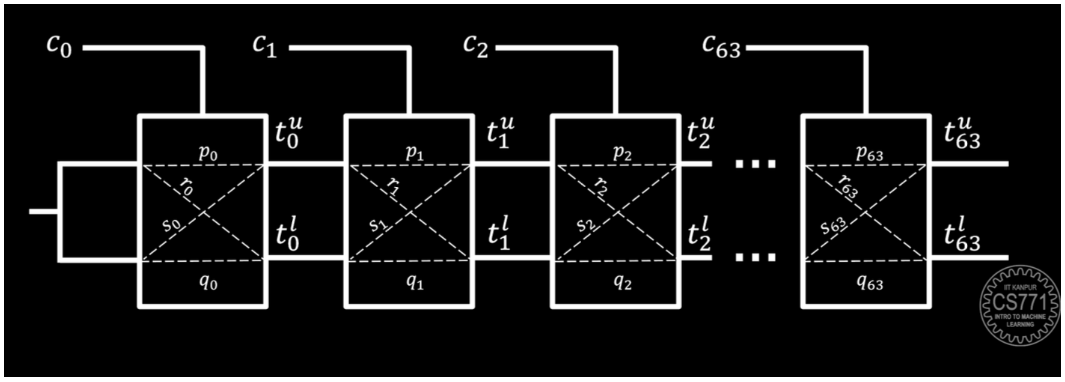
Kumari Ritika
241110039
MTech CSE, IIT Kanpur
ritika@iitk.ac.in

Milian Roy
241110042
MTech CSE, IIT Kanpur
milanr24@iitk.ac.in

Q1. Derivation of a Linear Model for Predicting ML-PUF

We will use a straightforward approach to solve this problem. The strategy involves leveraging the known differences in the time delays for the upper and lower signals, as well as the sum of their delays, to predict the upper delay. After getting both responses from arbiters, we perform XOR by multiplying both linear models to get the final response.

A Linear Model for Predicting the Upper Signal Traversal Time in an Arbiter PUF



Arbiter PUF structure as described in lectures

Lets prove that a linear model can predict the time taken for upper (or lower) signal to reach the finish line for simple arbiter PUF.

Initial Definitions

For each PUF, we have 8 PUFF stages numbered from 0 to 7. We define:

t_i^u as the time for the upper signal to be generated at stage i

t_i^l as the time for the lower signal to be generated at stage i

$\Delta_i = t_i^u - t_i^l$ as the delay difference between upper and lower signal generation at stage i

The initial conditions are:

$$t_{-1}^u = 0, \quad t_{-1}^l = 0, \quad \Delta_{-1} = 0 \tag{1}$$

Time Delay Recursion Relations

From the circuit analysis, we derive the following recursive relations:

$$t_i^u = (1 - c_i)\left(t_{i-1}^u + p_i\right) + c_i\left(t_{i-1}^l + s_i\right) \tag{2}$$

where the switching parameter c_i is defined as:

$$c_i = \frac{1 - d_i}{2} \tag{3}$$

Expanding the equation (2), we obtain:

$$\begin{aligned} t_i^u &= t_{i-1}^u + p_i - c_i t_{i-1}^u - c_i p_i + c_i t_{i-1}^l + c_i s_i \\ &= t_{i-1}^u + c_i\left[s_i - p_i + \Delta_{i-1}\right] + p_i \end{aligned} \tag{4}$$

Substituting equation (3) in equation (4) gives:

$$t_i^u = t_{i-1}^u + \left(\frac{1 - d_i}{2}\right)(s_i - p_i - \Delta_{i-1}) + p_i \tag{5}$$

Which simplifies to:

$$t_i^u = t_{i-1}^u + \left(\frac{p_i + s_i}{2}\right) + \frac{d_i}{2}(p_i - s_i) + \left(\frac{1 - d_i}{2}\right)\Delta_{i-1} \tag{6}$$

Let us define:

$$e_i = \frac{p_i + s_i}{2} \tag{7}$$

$$f_i = \frac{p_i - s_i}{2} \tag{8}$$

This allows us to write the final recursive relation in compact form:

$$t_i^u = t_{i-1}^u + e_i + d_i f_i - \left(\frac{1 - d_i}{2}\right)\Delta_{i-1} \tag{9}$$

Stage-wise Time Delay Formulation

For stage $i = 0$, we have the initial condition:

$$t_0^u = e_0 + d_0 f_0 \quad \text{with} \quad t_{-1}^u = 0 \text{ and } \Delta_{-1} = 0 \tag{10}$$

Recursive Relations for Subsequent Stages

For stage $i = 1$:

$$t_1^u = e_0 + d_0 f_0 + e_1 + d_1 f_1 + \frac{(1 - d_1)}{2}\Delta_0 \tag{11}$$

$$t_1^u = (e_0 + e_1) + (d_0 f_0 + d_1 f_1) - \frac{(1 - d_1)}{2}\Delta_0 \tag{12}$$

For stage $i = 2$:

$$t_2^u = (e_0 + e_1 + e_2) + (d_0 f_0 + d_1 f_1 + d_2 f_2) - \frac{1}{2}\left((1 - d_1)\Delta_0 + (1 - d_2)\Delta_1\right) \tag{13}$$

Continuing this pattern, we obtain the general solution:

$$t_7^u = \sum_{i=0}^{7}(e_i + d_i f_i) - \frac{1}{2}\left(\Delta_0 + \Delta_1 \cdots + \Delta_6 - d_1\Delta_0 - d_2\Delta_1 \cdots - d_7\Delta_6\right) \tag{14}$$

$$= \sum_{i=0}^{7}(e_i + d_i f_i) - \frac{1}{2}\left(\Delta_6 - d_6\Delta_5 + (\Delta_5 - d_5\Delta_4) + \cdots + (\Delta_1 - d_1\Delta_0) + \Delta_0 - d_7\Delta_6\right) \tag{15}$$

From the lecture slides, we have the following relations for the time differences:

$$\begin{aligned} \Delta_{-1} &= 0 && (16)\\ \Delta_i &= d_i\Delta_{i-1} + \alpha_i d_i + \beta_i && (17)\\ \Delta_i - d_i\Delta_{i-1} &= \alpha_i d_i + \beta_i && (18)\\ \Delta_0 &= \alpha_0 d_0 + \beta_0 && (19)\\ d_7\Delta_6 &= \Delta_7 - \alpha_7 d_7 - \beta_7 && (20) \end{aligned}$$

$$\Delta_{k-1} = \sum_{i=0}^{k-1}(w_i x_i) + b \tag{21}$$

Where

$$\begin{aligned} x_i &= d_i d_{i+1}\cdots d_{k-1} && (22)\\ w_0 &= \alpha_0 && (23)\\ w_i &= \alpha_i + \beta_{i-1} && (24)\\ b &= \beta_{k-1} && (25)\\ \alpha_i &= \frac{p_i - q_i + r_i - s_i}{2} && (26)\\ \beta_i &= \frac{p_i - q_i - r_i + s_i}{2} && (27) \end{aligned}$$

Putting above thing together

$$t_7^u = \sum_{i=0}^{7}(e_i + d_i f_i) - \frac{1}{2}\left(a_6 d_6 + \beta_6 + a_5 d_5 + \beta_5 + \cdots + a_1 d_1 + \beta_1 + a_0 d_0 + \beta_0 - \Delta_7 + \alpha_7 d_7 + \beta_7\right) \tag{28}$$

The expression can be rewritten in compact summation form as:

$$t_7^u = \sum_{i=0}^{7}(e_i + d_i f_i) - \frac{1}{2}\sum_{i=0}^{7}(\alpha_i d_i + \beta_i) + \frac{\Delta_7}{2} \tag{29}$$

Expanding Δ_7 and grouping terms gives:

$$t_7^u = \sum_{i=0}^{7}(e_i - \frac{\beta_i}{2}) + \sum_{i=0}^{7}(d_i f_i - \frac{\alpha_i d_i}{2}) + \frac{1}{2}\sum_{i=0}^{7} w_i x_i + \frac{\beta_7}{2} \tag{30}$$

By expanding the expressions for e_i, f_i, we obtain:

$$t_7^u = \sum_{i=0}^{7}\frac{(p_i + s_i - \beta_i)}{2} + \sum_{i=0}^{7}\left(d_i\frac{(p_i - s_i - \alpha_i)}{2}\right) + +\frac{1}{2}\sum_{i=0}^{7} w_i x_i + \frac{\beta_7}{2} \tag{31}$$

For simplification, we define:

$$\begin{aligned} g_i &= \frac{p_i + s_i - \beta_i}{2} && (32)\\ h_i &= \frac{p_i - s_i - \alpha_i}{2} && (33)\\ \hat{w}_i &= \frac{w_i}{2} && (34) \end{aligned}$$

The time delay for the final stage (t_7^u) is given by:

$$\boxed{t_7^u = \sum_{i=0}^{7} d_i h_i + \sum_{i=0}^{7}\hat{w}_i x_i + \sum_{i=0}^{7} g_i + \frac{\beta_7}{2}} \tag{35}$$

The above equation demonstrates that a linear model can be constructed to predict the time taken for the upper signal to reach the finish line.
At first glance, it may appear that 16 weights are required for this model: 8 corresponding to the terms h_i and 8 for the terms d_i.

But for $i = 7$

$$x_7 = d_7 \tag{36}$$

This allows us to combine weights for i = 7, reducing the effective dimensionality to 15. The final model becomes:

$$t_7^u = W^T \phi(C) + b \tag{37}$$

where:

$$W_{15\times 1} = \begin{cases} \dfrac{\alpha_0}{2} & \text{if } i = 0 \\ \dfrac{\alpha_i + \beta_{i-1}}{2} & \text{if } 1 <= i <= 6 \\ \dfrac{p_7 + \beta_6 - s_6}{2} & \text{if } i = 7 \\ \dfrac{p_{i-8} - \alpha_{i-8} - s_{i-8}}{2} & \text{if } 8 <= i <= 14 \end{cases} \tag{38}$$

$$b = \sum_{i=0}^{7} g_i + \beta_7 \tag{39}$$

$$\phi(C)_{15\times 1} = \begin{cases} x_0 = d_0 d_1 \cdots d_7 \\ x_1 = d_1 d_2 \cdots d_7 \\ \vdots \\ x_7 = d_7 \\ d_0 \\ d_1 \\ \vdots \\ d_6 \end{cases} \tag{40}$$

$$\phi(C) = (\phi_i)_{15\times 1} \tag{41}$$

$$\phi_i = \begin{cases} x_i = d_i d_{i+1} \cdots d_7 & \text{if } 0 <= i <= 7 \\ d_{i-8} & \text{if } 8 <= i <= 14 \end{cases} \tag{42}$$

Hence, we have demonstrated that a linear model can predict the time taken for upper signal to reach the finish line in a PUF. We can similiarly do the same for the lower signal.

A Linear Model Predicting ML-PUF output

The above proof demonstrated that the time taken by an arbiter PUF to produce the upper or lower signal can be modeled linearly. Since the ML-PUF in question consists of two independent PUFs, we can express their outputs using the following equations:

$$t_{PUF_0}^u = W_a^T \phi(C) + b_a \tag{43}$$

$$t_{PUF_1}^u = W_b^T \phi(C) + b_b \tag{44}$$

$$t_{PUF_0}^l = W_c^T \phi(C) + b_c \tag{45}$$

$$t_{PUF_1}^l = W_d^T \phi(C) + b_d \tag{46}$$

where:

W_a, W_b, W_c, W_d are corresponding model weight vectors.
b_a, b_b, b_c, b_d are corresponding bias terms.
$\phi(C)$ represents the feature mapping.

Response Function Formulation

$Response_0$ is defined to be 0, if signal from PUF_0 reaches first, else 1. If we define

$$\Delta_0(C) = t^l_{PUF_0} - t^l_{PUF_1} \tag{47}$$

Then we can define:

$$Response_0 = \begin{cases} 0 & \text{if } \Delta_0(C) < 0 \\ 1 & \text{if } \Delta_0(C) > 0 \end{cases} \tag{48}$$

Substituting equation (45) and equation (46) in equation (47) gives:

$$\Delta_0(C) = (W_c^T - W_d^T)\phi(C) + (b_c - b_d) \tag{49}$$

Which simplifies to the linear form:

$$\Delta_0(C) = W_0^T\phi(C) + b_0 \tag{50}$$

$Response_0$ can be compactly represented using the sign function as:

$$Response_0 = \frac{1 + \text{sign}(W_0^T\phi(C) + b_0)}{2} \tag{51}$$

We can similarly get an expression for $Response_1$ as:

$$Response_1 = \frac{1 + \text{sign}(W_1^T\phi(C) + b_1)}{2} \tag{52}$$

Where:

W_0, W_1 are the combined weight vectors
b_0, b_1 are the combined bias terms
$\phi(C)$ is the feature mapping function
$\text{sign}(\cdot)$ is the sign function

XOR PUF Formulation

In this section, for ease of representation, we are hidding the bias term inside the weight vector.
We know from the slides, that output of an XOR PUF is given as:

$$\frac{1 - (-1)^{n+1}\text{sign}(\Pi_i^n(W_i^T\phi(C)))}{2} \tag{53}$$

where n represents the number of parallel PUFs in the XOR configuration.
When $n = 2$, the response condition simplifies to:

$$\frac{1 - (-1)^3\text{sign}((W_0^T\phi(C))(W_1^T\phi(C))}{2} \tag{54}$$

The weight vectors are combined as:

$$W_0^T\phi(C) \cdot W_1^T\phi(C) = W^T\phi'(C) \tag{55}$$

where $\phi'(C)$ is the feature map. Equation (39) provides the expression for $\phi(C)$. As the bias term is incorporated into the weight vector, we need to append a 1 to it.

$$\phi(C)_{16\times 1} = \begin{cases} x_0 = d_0 d_1 \cdots d_7 \\ x_1 = d_1 d_2 \cdots d_7 \\ \vdots \\ x_7 = d_7 \\ d_0 \\ d_1 \\ \vdots \\ d_6 \\ 1 \end{cases} \qquad \text{where } d_i = 1 - 2c_i \tag{56}$$

The feature map $\phi'(C)$ is constructed by multiplying each element in the original feature map $\phi(C)$ with every other element. Hence, a very naive way of constructing $\phi'(C)$ would be:

$$\phi'(C)_{256\times 1} = \begin{cases} z_0 z_0 \\ z_0 z_1 \\ z_0 z_2 \\ \vdots \\ z_{15} z_{15} \end{cases} \tag{57}$$

here z_i is i^{th} term of $\phi(C)$, and $z_{15} = 1$
The next section explains how to substantially reduce the dimensionality.

Q2. Dimensionality Calculation

The linear model used to predict the output of an ML-PUF requires a dimensionality of 87.
Note: This count excludes the bias term, in line with the evaluation script provided in Google Colab. Including the bias term would result in a total dimensionality of 88.

$$\boxed{\mathbf{D = 87}}$$

Proof

This section explains how we reduce the dimensionality from 256 to 87. The current feature map treats different permutations of the same combination of d_i as distinct—for example, it considers $d_0 d_1 d_2$ different from $d_1 d_2 d_0$. By merging all such permutations into a single feature, we can effectively reduce the dimensionality. Additionally we know that

$$d_i \in \{-1, 1\}$$

Therefore:

$$d_i^2 = 1 \tag{58}$$

Also, since x_i is a product of d_i:

$$x_i^2 = 1 \tag{59}$$

Thus, we can eliminate repeating terms in features by performing the following substitutions:

$$d_i^2 = 1 \tag{60}$$

$$d_0^2 d_1 d_2^2 = d_1 \tag{61}$$

By once again merging features that are permutations of the same set of d_i terms into a single feature, we obtain the following feature map:

$$
\begin{aligned}
\phi'(C)_{87\times 1} = (&d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7,\\
&d_0d_1, d_0d_2, d_0d_3, d_0d_4, d_0d_5, d_0d_6, d_0d_7,\\
&d_1d_2, d_1d_3, d_1d_4, d_1d_5, d_1d_6, d_1d_7,\\
&d_2d_3, d_2d_4, d_2d_5, d_2d_6, d_2d_7,\\
&d_3d_4, d_3d_5, d_3d_6, d_3d_7,\\
&d_4d_5, d_4d_6, d_4d_7,\\
&d_5d_6, d_5d_7,\\
&d_6d_7,\\
&d_0d_1d_2, d_0d_6d_7, d_1d_2d_3, d_1d_6d_7, d_2d_3d_4, d_2d_6d_7,\\
&d_3d_4d_5, d_3d_6d_7, d_4d_5d_6, d_4d_5d_7, d_4d_6d_7, d_5d_6d_7,\\
&d_0d_1d_2d_3, d_0d_5d_6d_7, d_1d_2d_3d_4, d_1d_5d_6d_7,\\
&d_2d_3d_4d_5, d_2d_5d_6d_7, d_3d_4d_5d_6, d_3d_4d_5d_7,\\
&d_3d_4d_6d_7, d_3d_5d_6d_7, d_4d_5d_6d_7,\\
&d_0d_1d_2d_3d_4, d_0d_4d_5d_6d_7, d_1d_2d_3d_4d_5, d_1d_4d_5d_6d_7,\\
&d_2d_3d_4d_5d_6, d_2d_3d_4d_5d_7, d_2d_3d_4d_6d_7,\\
&d_2d_3d_5d_6d_7, d_2d_4d_5d_6d_7, d_3d_4d_5d_6d_7,\\
&d_0d_1d_2d_3d_4d_5, d_0d_3d_4d_5d_6d_7,\\
&d_1d_2d_3d_4d_5d_6, d_1d_2d_3d_4d_5d_7, d_1d_2d_3d_4d_6d_7,\\
&d_1d_2d_3d_5d_6d_7, d_1d_2d_4d_5d_6d_7, d_1d_3d_4d_5d_6d_7,\\
&d_2d_3d_4d_5d_6d_7,\\
&d_0d_1d_2d_3d_4d_5d_6, d_0d_1d_2d_3d_4d_5d_7,\\
&d_0d_1d_2d_3d_4d_6d_7, d_0d_1d_2d_3d_5d_6d_7,\\
&d_0d_1d_2d_4d_5d_6d_7, d_0d_1d_3d_4d_5d_6d_7,\\
&d_0d_2d_3d_4d_5d_6d_7, d_1d_2d_3d_4d_5d_6d_7,\\
&d_0d_1d_2d_3d_4d_5d_6d_7)
\end{aligned}
$$

where $d_i = 1 - 2c_i$

It should be noted that our feature mapping process actually yields an 88th term, which is a constant 1. However, we have excluded this term from the feature map, as its effect will be captured by the model's bias term. This also allows us to report a reduced dimensionality for the feature map and model.

Q3. Kernel SVM

A kernel computes the dot product between two data points in a high-dimensional space without explicitly mapping the data points to that space. Since the high-dimensional feature mapping described in the previous section is effective, we need a kernel that computes the dot product in the dimension of $\phi'(C)$.

That is, we need a kernel $K : \mathcal{C} \times \mathcal{C} \to R$ that satisfies:

$$K(C, C') = \langle \phi'(C), \phi'(C') \rangle \tag{62}$$

We know that $\phi'(C)$ includes features involving individual d_i terms as well as products of 2 to 8 such terms. However, not all possible combinations of these products are present in the feature map.

Suppose we construct a new feature map $\phi''(C)$ which contains all such possible products:

$$\phi''(C)_{256\times 1} = \begin{cases} 1 \\ \text{Single terms like } d_0, d_1, \dots \\ \text{Pairwise terms like } d_0 d_1, d_0 d_1, \dots \\ \vdots \\ \text{Product of all terms } d_0 d_1 \dots d_7 \end{cases} \tag{63}$$

Since $\phi''(C)$ produces a feature map that is a subset of the one produced by $\phi'(C)$, it retains at least the same representational power. Therefore, we can safely use $\phi''(C)$ to construct the kernel. Since an order-8 expansion would involve an excessive number of terms, lets try the calculations for order 2.

Given $D = [d_0, d_1]$, then:

$$\phi''(D) = [1, d_0, d_1, d_0 d_1]$$
$$\phi''(D)^T \phi''(D') = 1 + d_0 d_0' + d_1 d_1' + d_0 d_1 d_0' d_1' \tag{64}$$

We know that:

$$(1 + D \cdot D')^2 = (1 + d_0 d_0' + d_1 d_1')^2 = 1 + 2d_0 d_0' + 2d_1 d_1' + 2d_0 d_0' d_1 d_1' + \text{higher terms} \tag{65}$$

Although the terms do not match exactly in form, all the terms in Equation (64) are present in Equation (65). Therefore, we can conclude that:

$$\phi''(D)^T \phi''(D') = (1 + D \cdot D')^2 \tag{66}$$

The right-hand side of the above equation is, in fact, the **Polynomial Kernel** of degree 2. Extending this calculation to order 8, the kernel we require becomes:

$$K(D, D') = (1 + D \cdot D')^8 = \left(1 + \sum_{i=0}^{7} d_i d_i'\right)^8 \tag{67}$$

Substituting $d_i = 1 - 2c_i$

$$\boxed{K(C, C') = \left(1 + \sum_{i=0}^{7} (1 - 2c_i)(1 - 2c_i')\right)^8} \tag{68}$$

Kernel Type and Parameters:

Kernel Type: Polynomial Kernel

Degree: 8

Gamma: 1 (Default scaling of dot product)

Coeff: 1 (Includes all degrees upto 8)

Q4. Delay Recovery by Inverting an Arbiter PUF

Objective

Given a 65-dimensional linear model ($\mathbf{w} \in R^{64}, b \in R$) for a 64-bit arbiter PUF, we aim to compute 256 non-negative delays $\{\hat{p}_i, \hat{q}_i, \hat{r}_i, \hat{s}_i\}_{i=0}^{63} \in R_{\geq 0}$ that reproduce the same model.

Theoretical Background From Slides

For a $k = 64$-stage arbiter PUF, the delays p_i, q_i, r_i, s_i ($i = 0, \dots, 63$) are used to define intermediate parameters, which are then employed to construct a linear model that predicts whether the upper or lower signal reaches the finish line first. Feature mapping is also done on the challenge bits to make the model's response linearly seperable.

The linear model is:

$$\Delta_{k-1} = \sum_{i=0}^{k-1} (w_i x_i) + b \tag{69}$$

where

$$\begin{aligned} x_i &= d_i d_{i+1} \cdots d_{k-1} && (70)\\ d_i &= 1 - 2c_i && (71)\\ w_0 &= \alpha_0 && (72)\\ w_i &= \alpha_i + \beta_{i-1} && (73)\\ b &= \beta_{k-1} && (74)\\ \alpha_i &= \frac{p_i - q_i + r_i - s_i}{2} && (75)\\ \beta_i &= \frac{p_i - q_i - r_i + s_i}{2} && (76) \end{aligned}$$

Formulating a System of Linear Equations for Model Generation

The above model generation can be formluated as a system of linear equations as follows:

$$A\mathbf{x} = \mathbf{y} \tag{77}$$

where:

$\mathbf{x} \in R^{256\times 1} = [p_0, q_0, r_0, s_0, \ldots, p_{63}, q_{63}, r_{63}, s_{63}]^T$
$\mathbf{y} \in R^{65\times 1} = [w_0, w_1, \ldots, w_{63}, b]^T$
$A \in R^{65\times 256}$ is the coefficient matrix.

The i-th row of matrix $\mathbf{A}$ contains the coefficients for the weight w_i, which corresponds to the i-th row in vector $\mathbf{y}$. Mathematically, the weight w_i depends only on the i-th and/or $(i-1)$-th delay elements, while all other delays contribute a coefficient of zero. Therefore, only the columns corresponding to the above mentioned delays will have non zero entries in any row of $\mathbf{A}$.

Hence, we can construct the $\mathbf{A}$ matrix in the following manner:

Row 0: $w_0 = \alpha_0 \implies 0.5p_0 - 0.5q_0 + 0.5r_0 - 0.5s_0 = w_0$
Rows $i = 1, \ldots, 63$: $w_i = \alpha_i + \beta_{i-1}$

$$0.5(p_{i-1} - q_{i-1} - r_{i-1} + s_{i-1}) + 0.5(p_i - q_i + r_i - s_i) = w_i$$

Row 64: $b = \beta_{63} \implies 0.5p_{63} - 0.5q_{63} - 0.5r_{63} + 0.5s_{63} = b$

Matrix A:

Row 0:
$A[0, 0:4] = [0.5, -0.5, 0.5, -0.5]$
$A[0, 4:] = [0, 0, 0...]$
Row $i = 1, \ldots, 63$:
$A[i, 0:4(i-1)] = [0, 0, 0...]$
$A[i, 4(i-1):4i] = [0.5, -0.5, -0.5, 0.5]$
$A[i, 4i:4(i+4)] = [0.5, -0.5, 0.5, -0.5]$
$A[i, 4(i+4):] = [0, 0, 0...]$
Row 64:
$A[64, 0:252] = [0, 0, 0...]$
$A[64, 252:] = [0.5, -0.5, -0.5, 0.5]$

Therefore, given complete knowledge of the $4k$ delay parameters in a k-stage Arbiter PUF, we can formulate a linear prediction model using equation (78). The model parameters are directly derived from the vector $\mathbf{y}$, where the first k components determine the weight coefficients and the $(k+1)$-th component provides the bias term of the linear model.

Inverting the System to Generate the Delays

In this section, we describe how, given the weights and bias of a linear model designed to predict the responses of an arbiter PUF, we can determine the individual delays of the PUF. Our main challenges are twofold:

Our system of linear equations is underdetermined, meaning we have more unknowns (delays) than equations. Consequently, we cannot apply the direct solution formula:

$$\mathbf{x} = A^{-1}\mathbf{y} \tag{78}$$

The solution must be non-negative since PUF circuit elements cannot physically exhibit negative delays. This constraint prevents direct application of the standard solution equation:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \tag{79}$$

Given these physical constraints, we formulate the delay extraction problem as a constrained optimization problem:

$$\min_{\mathbf{x}\in R^{256}} \|A\mathbf{x} - \mathbf{y}\|_2^2 \tag{80}$$

$$\text{s.t. } \mathbf{x} \geq 0 \tag{81}$$

To solve this constrained optimization problem, we utilize the `LinearRegression` implementation from scikit-learn. Although the standard implementation solves an unconstrained least squares problem, we can enforce positivity constraints on the solution by setting the `positive=True` parameter.

Q5. Code to solve the ML-PUF Problem

Zipped Solution to $cse.iitk.ac.in/users/jatinj24/survivors\_of\_ml.zip$

Q6. Code to solve the Arbiter PUF inversion problem to recover delays

Zipped Solution to $cse.iitk.ac.in/users/jatinj24/survivors\_of\_ml.zip$

Q7: Outcomes of experiments with LinearSVC and LogisticRegression methods

LogisticRegression

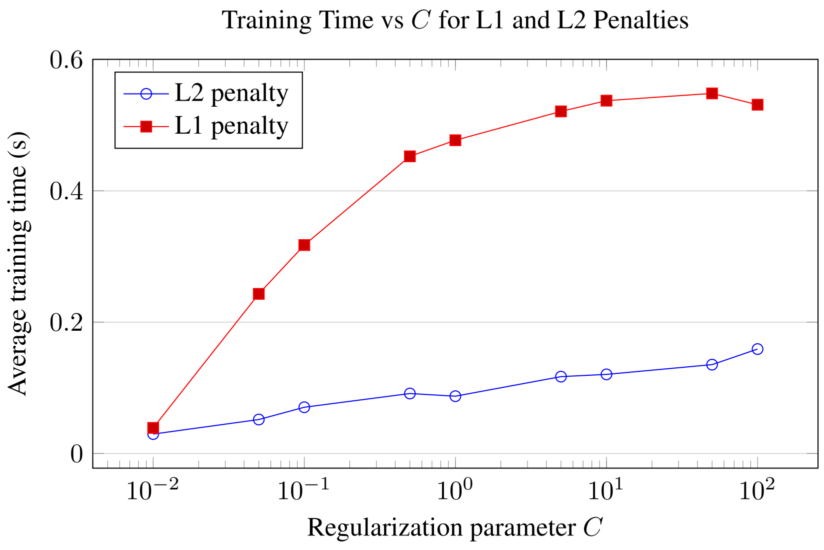
Effect of Penalty Type and Regularization Strength

Penalty	C	Training Time (s)	Misclass. Rate (%)	Accuracy (%)
L2	0.01	0.0295	8.06	91.94
	1.00	0.0872	0.00	100.00
	100.00	0.1590	0.00	100.00
L1	0.01	0.0390	17.06	82.94
	1.00	0.4770	0.00	100.00
	100.00	0.5311	0.00	100.00

Training time and performance for `LogisticRegression` with changing C and Penalty.

Penalty	C	**Train time (s)**	**Misclass. rate (%)**	**Accuracy (%)**
L2	0.01	0.0487	0.56	99.44
	1.00	0.4005	0.00	100.00
	100.00	0.4009	0.00	100.00
L1	0.01	0.2087	2.44	97.56
	1.00	1.5922	0.00	100.00
	100.00	0.6188	0.00	100.00

Average training time and test performance of LinearSVC (squared_hinge, L2 vs. L1 penalty) at $C = 0.01, 1, 100$.



We trained logistic-regression models on the ML-PUF feature map, varying both the regularization strength C and the penalty type (L1 vs. L2). we observed:

Under-fitting at very low C = 0.01 :

With L2 regularization at C=0.01, the model trains in an average of 0.0295 s and achieves a misclassification rate of 8.06 % (accuracy 91.94 %).

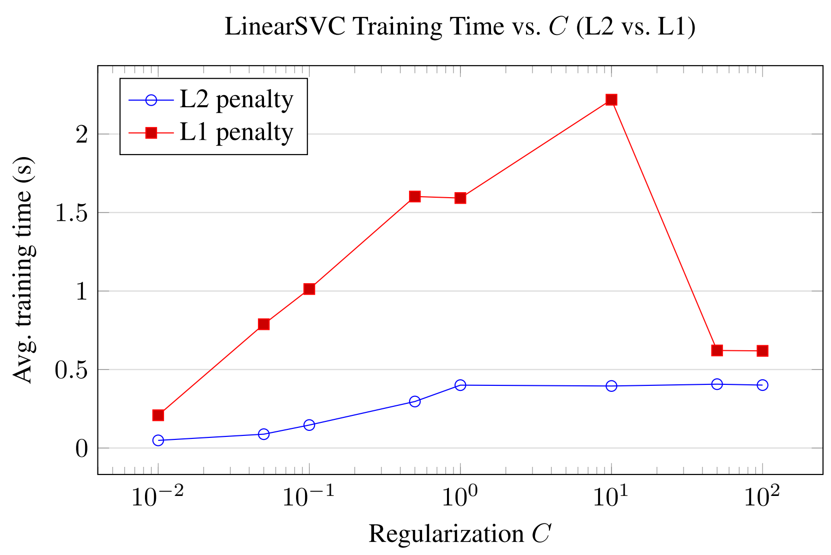
With L1 regularization at C=0.01, the model trains in just 0.0390 s but misclassifies 17.06 % of the test points (accuracy 82.94 %), indicating that L1 drives many weights to zero and under-fits more severely—only about 83 % of samples are correctly labeled. This confirms that under such strong regularization neither method achieves perfect separation, and that L2 under-fits less than L1.

The Perfect-fit with C = 1 : Relaxing regularization to C=1 yields perfect separation for both methods: misclassification drops to 0 % (100 % accuracy). However, L2 achieves this in 0.0872 s, while L1 takes 0.4770 s

Weak-penalty at higher C = 100 : With almost no regularization, both L2 and L1 maintain 100 % accuracy. Training times grow to 0.1590 s for L2 and 0.5311 s for L1, confirming that increasing C beyond 1 offers no accuracy benefit and only adds computational overhead.

LinearSVC

Effect of Penalty Type and Regularization Strength



L2 gives near-perfect separation even at $C = 0.01$ and remains 2 to 3 times faster than L1 at higher C. Use **L2,** $C \approx 1$ for speed and accuracy; use **L1** only if you need a sparse model and can afford slower training.

Strong regularization ($C = 0.01$)**:** L2 trains in 0.049s with 99.44% accuracy; L1 trains in 0.209s with 97.56% accuracy.

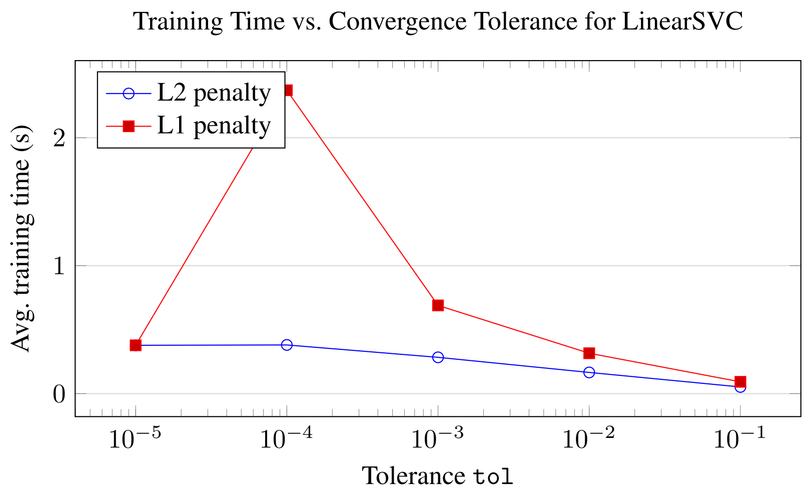
Moderate regularization ($C = 1$)**:** Both penalties reach 100% accuracy. L2 takes 0.400s vs. L1's 1.592s.

Weak regularization ($C = 100$)**:** Accuracy stays at 100%, training times are 0.401s (L2) and 0.619s (L1).

Effect of Convergence Tolerance on Training Time

Penalty	*tol*	**Train time (s)**	**Misclass. rate (%)**	**Accuracy (%)**
5*L2	10^{-1}	0.05215	0.00	100.00
	10^{-2}	0.16559	0.00	100.00
	10^{-3}	0.28363	0.00	100.00
	10^{-4}	0.38035	0.00	100.00
	10^{-5}	0.37755	0.00	100.00
5*L1	10^{-1}	0.09226	0.00	100.00
	10^{-2}	0.31557	0.00	100.00
	10^{-3}	0.68939	0.00	100.00
	10^{-4}	2.37069	0.00	100.00
	10^{-5}	0.37755	0.00	100.00

Average training time and test performance of `LinearSVC` at various convergence tolerances (`tol`), comparing L2 vs. L1 penalties.



Effect of convergence tolerance (`tol`) on training time for LinearSVC with L2 vs. L1 penalties.