Meta-IDS: A Multi-stage Deep Intrusion Detection System with Optimal CPU Usage

Nadia Niknami, Vahid Mahzoon, and Jie Wu

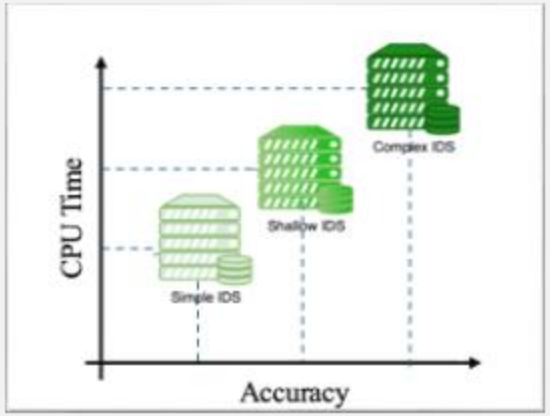
Temple University

Outline

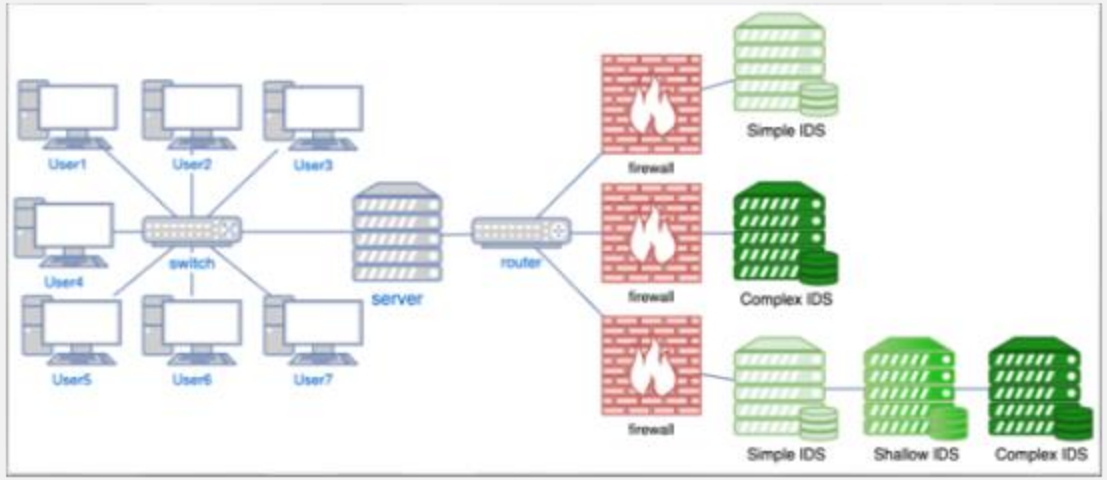
Introduction

- Cyber-Security
  - Protecting network against cyber threats
- Intrusion Detection System (IDS):
  - Monitoring network traffic
  - Identifies suspicious patterns
  - Generating alerts
- Deep learning has shown great success in cybersecurity.
- Challenges:
  - High-complexity deep learning models
  - High computational overhead
  - Dynamic network traffic
  - Resource-constrained devices
  - Need for efficient and accurate NIDS

Motivation



IDS Model	Neural Network	#Features
Simple IDS	No hidden layer	Very few
Shallow IDS	One/two hidden layer	A few
Complex IDS	Multiple hidden layers	All

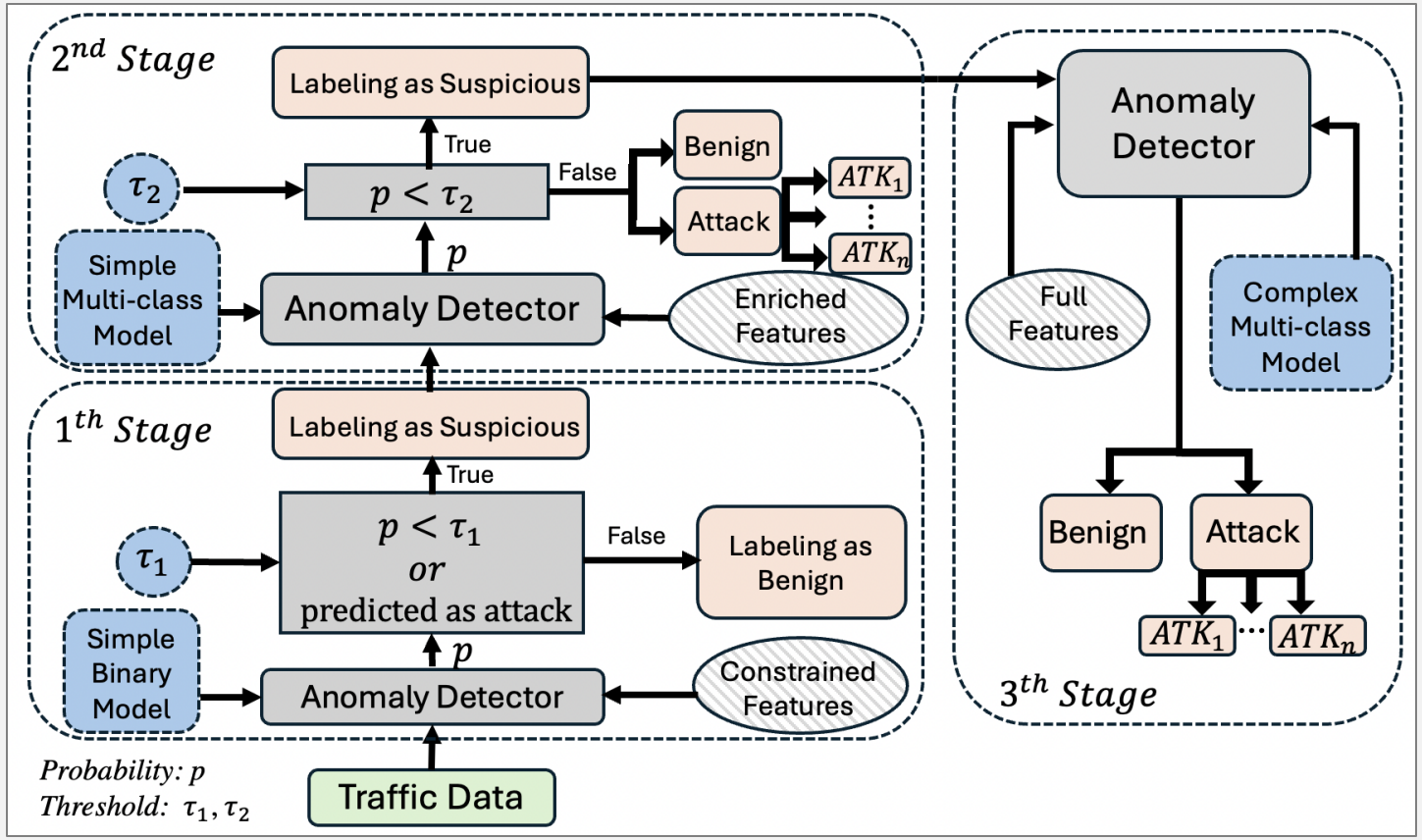


- Depends of scale of traffic and the sensitivity of attack detection different IDS models
  - Simple
  - Shallow
  - Complex

- Trade-off Between CPU Time and Accuracy
  - Simple IDS uses less CPU time but has lower accuracy.
  - Shallow IDS strikes a balance between CPU time and accuracy.
  - Complex IDS achieves the highest accuracy but requires the most CPU time

- Managing computing resources dynamically
- Optimized performance
- Efficient resource utilization enhancing parallel processing capabilities

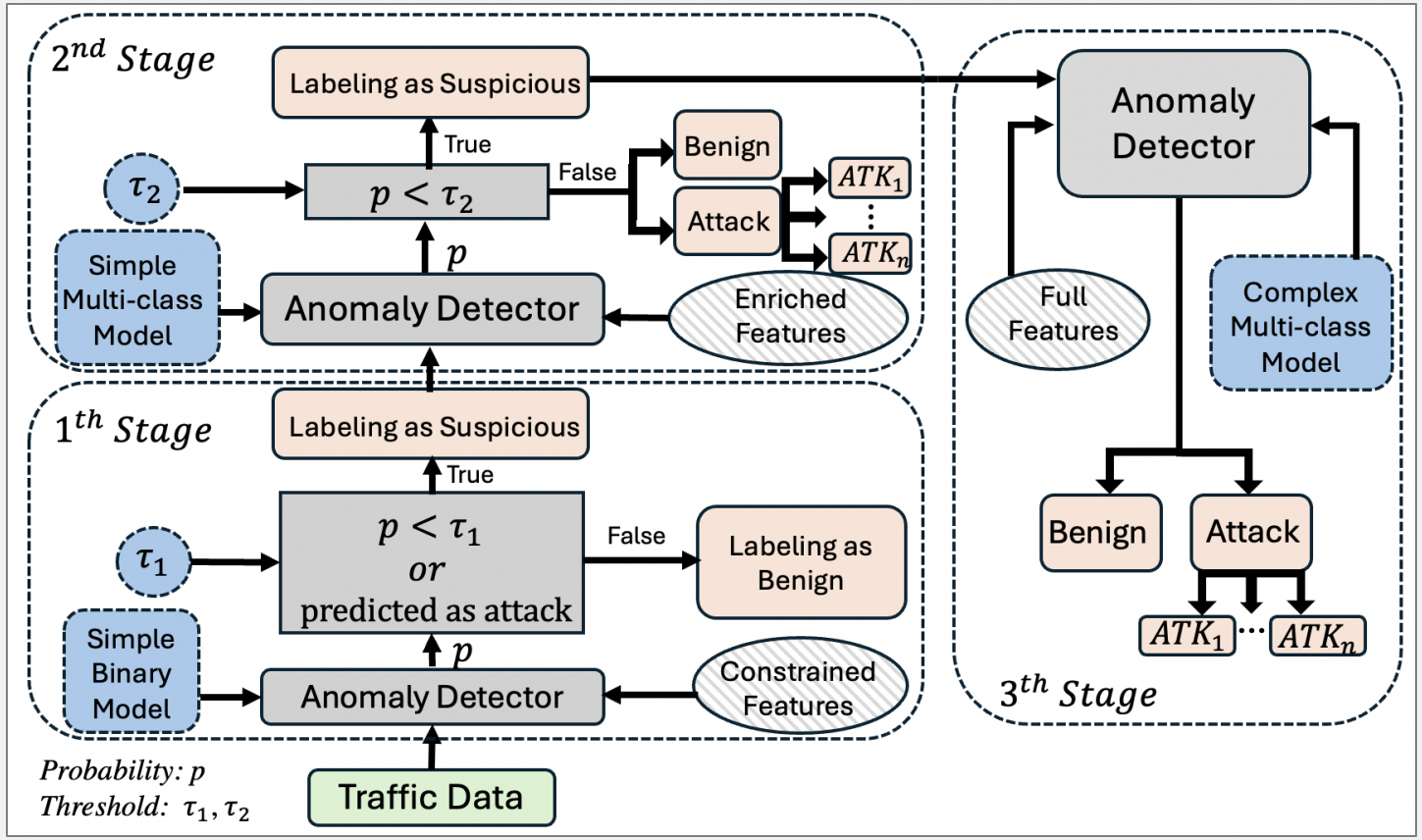
The Proposed Approach: Meta-IDS



- Inspired by **Meta-computing** principles, our framework divides the intrusion detection process into **three stages**, each equipped with unique capabilities and resource allocations.

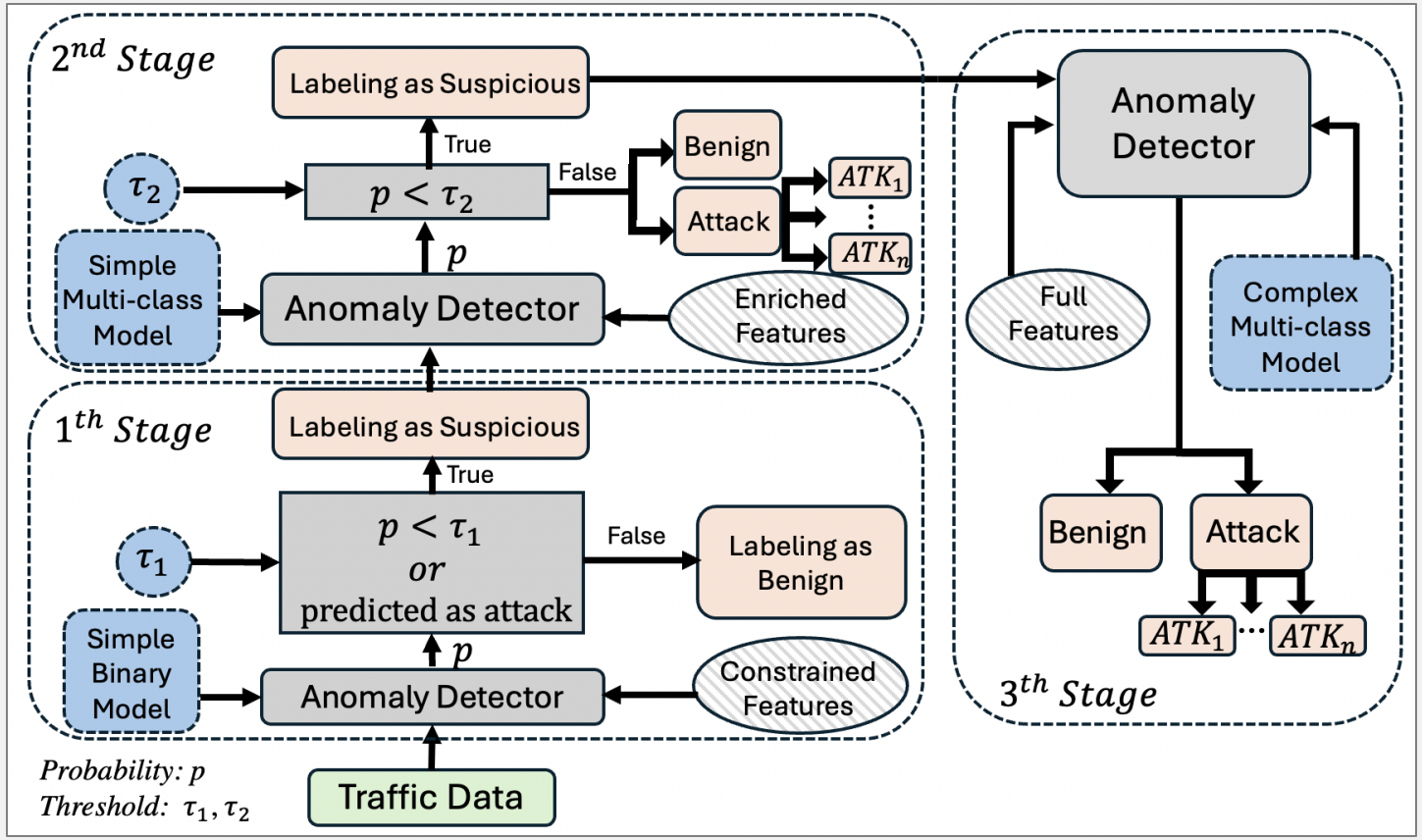
- This architecture:
  - Dynamically allocates analysis tasks
  - Reducing communication
  - Processing overheads
  - Enhancing the system's flexibility
  - Enhancing the system's scalability

The Proposed Approach: Meta-IDS



- **First stage:**
  - A simple neural network
  - Without hidden layers
  - With very few features
  - Binary classification
- **Second stage:**
  - A simple multi-class neural network
  - With more features
  - Classify any known attack or benign classes
- **Third stage:**
  - A complex multi-class neural network
  - Encompassing all available features
  - Comprehensive detection on data samples that were not satisfactorily classified during second stage

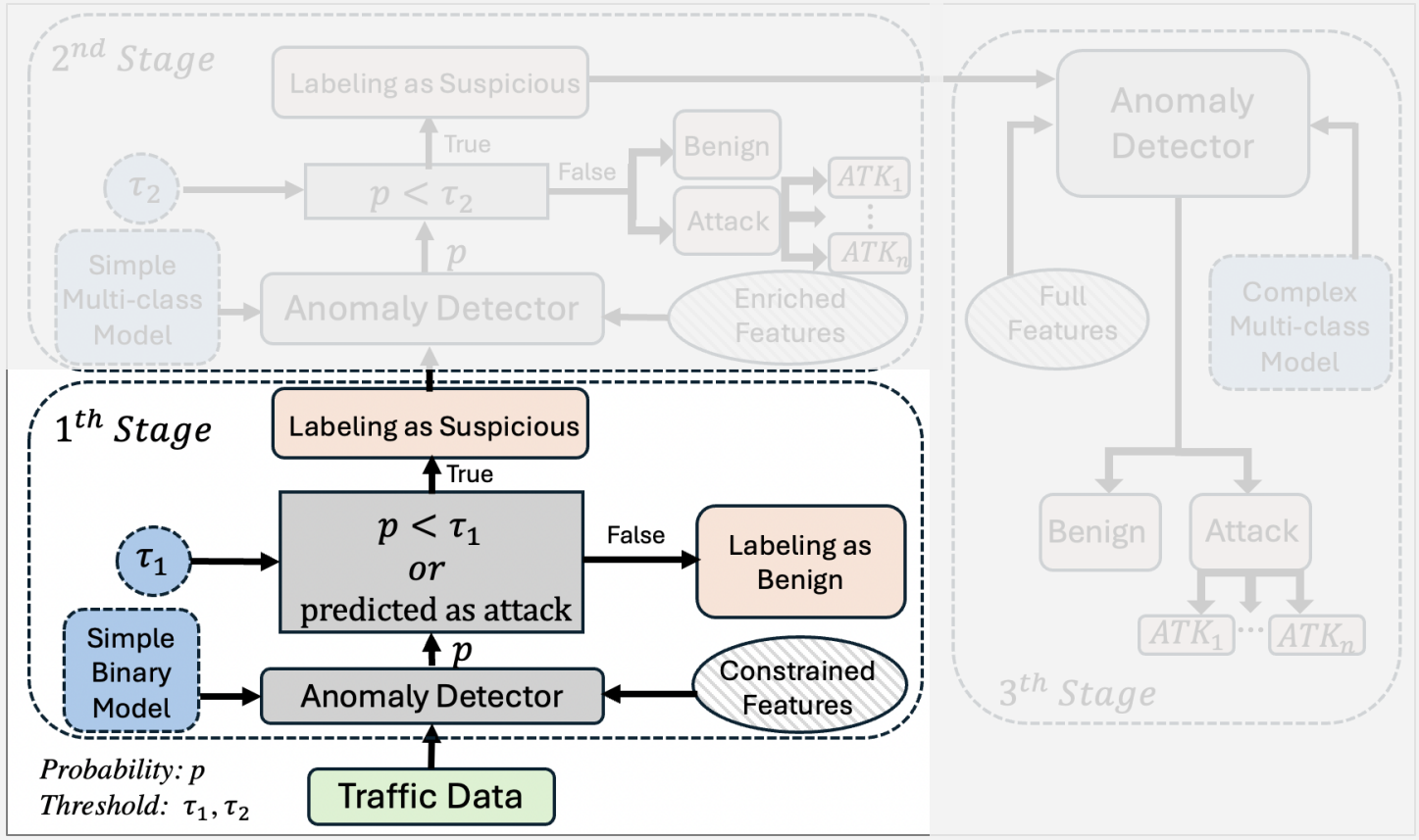
The Proposed Approach: Meta-IDS



Algorithm 1 Meta-IDS Algorithm

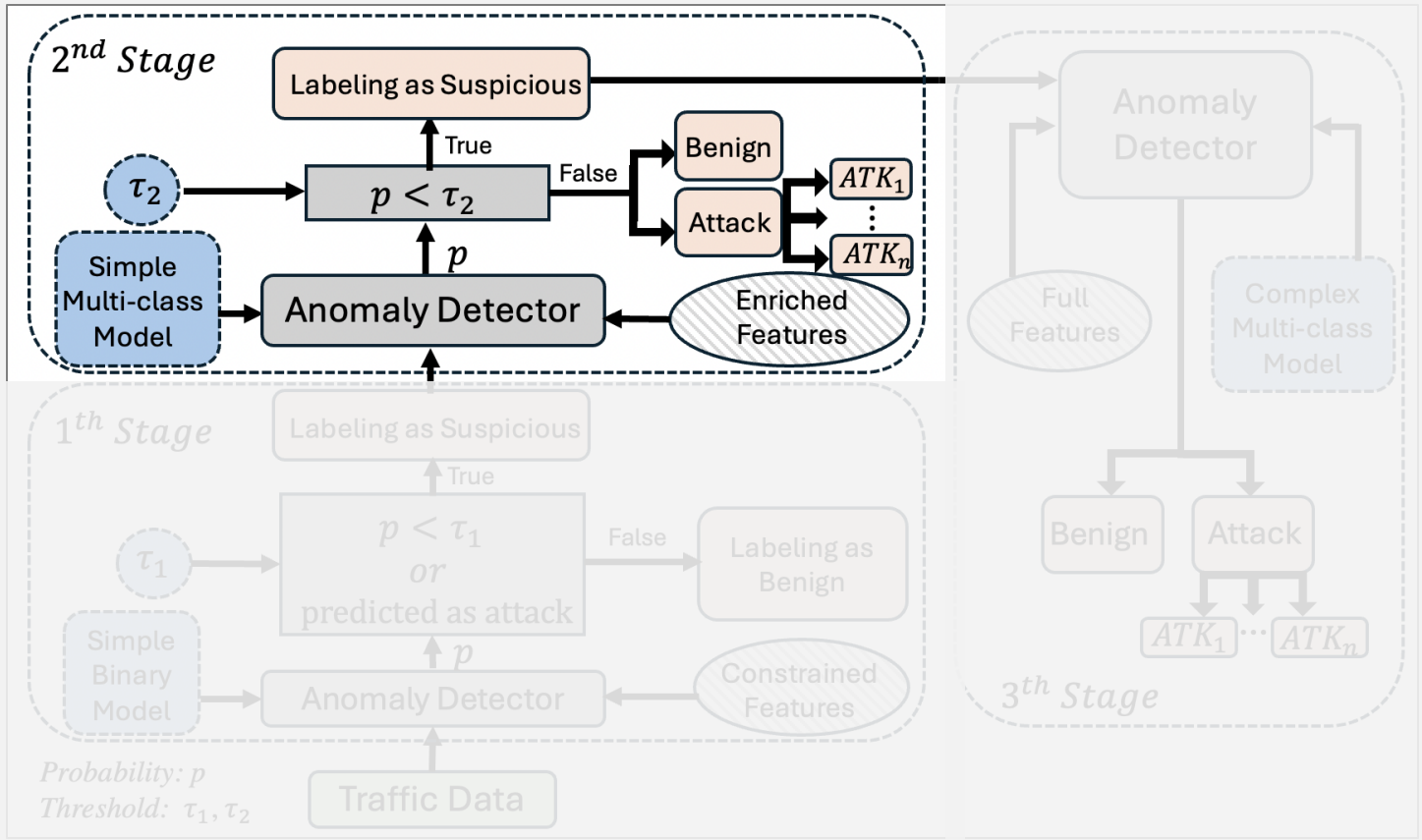
Input $(X, Y) = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, $\mathcal{F}_1$, $\mathcal{F}_2$, τ_1, τ_2
Output $\hat{Y} = \{\hat{y}_1, \ldots, \hat{y}_N\}$: Predicted Labels
for each data point x_i in X **do**
    $p_1, c_1 \leftarrow \text{BINARYMODEL}(x_i, \mathcal{F}_1)$
    if $p_1 < \tau_1$ **or** $c_1 ==$ malicious **then**
        $p_2, c_2 \leftarrow \text{SIMPLEMULTICLASSMODEL}(x_i, \mathcal{F}_2)$
        if $p_2 < \tau_2$ **then**
            $\hat{y}_i \leftarrow \text{COMPLEXMULTICLASSMODEL}(x_i)$
        else
            $\hat{y}_i \leftarrow c_2$
        end if
    else
        $\hat{y}_i \leftarrow c_1$
    end if
end for

Meta-IDS: First Stage



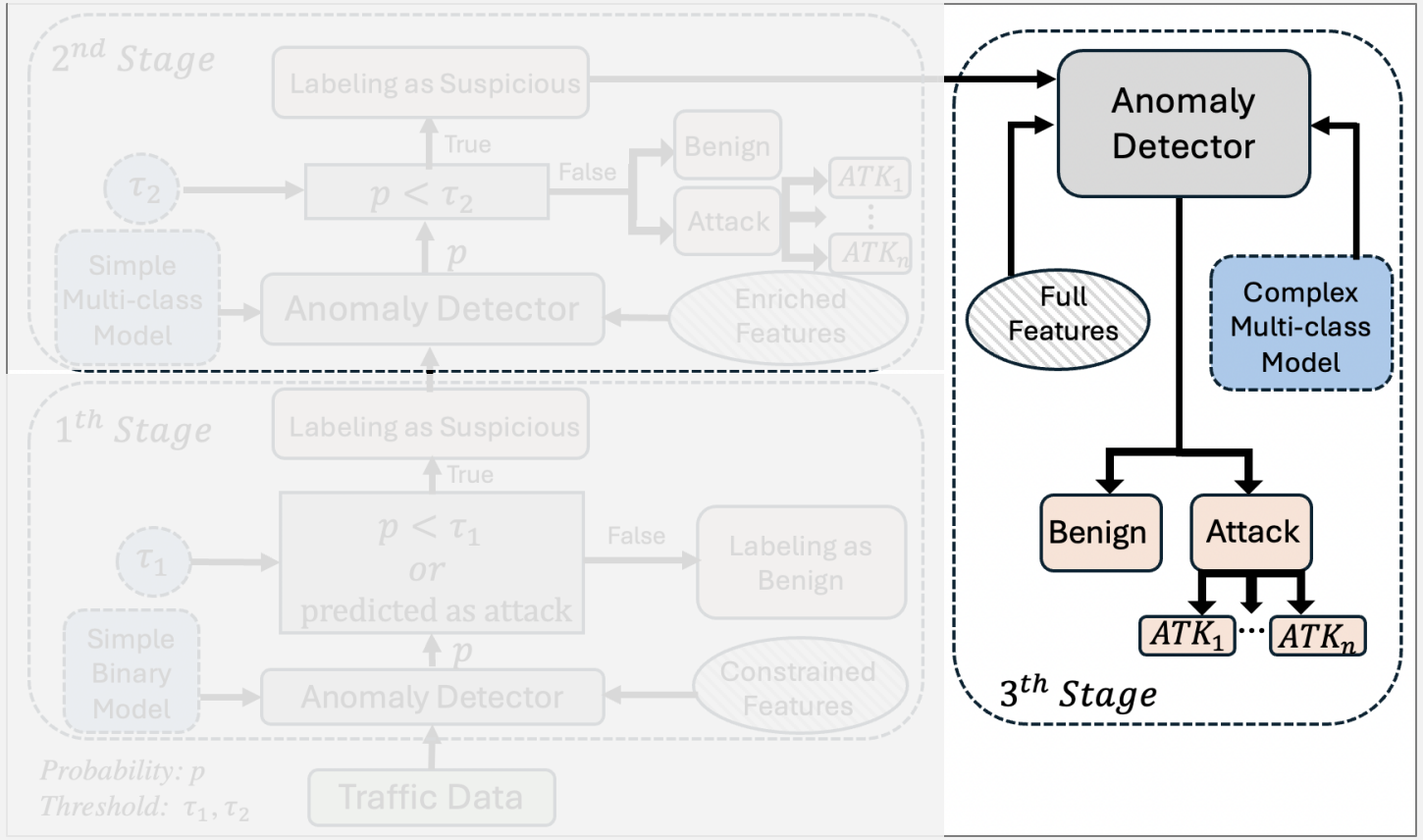
- In the initial stage, a simple neural network for binary classification (with very few features) is utilized to identify straightforward attacks based on predefined thresholds and probabilities.
- If the model confidently predicts a traffic flow as benign, it requires no further processing.
- If a sample's probability of belonging to a class falls below a threshold τ_1 or the model predicts the sample as an attack, it advances to the next stage as suspicious traffic.

Meta-IDS: Second Stage



- During this stage, a simple multi-class classification model is utilized for further evaluation of suspicious traffic in the first stage.
- The classifier generates probabilities for each known attack class as well as the benign class, and the class with the highest probability is assigned as the predicted class.
- Samples with probabilities falling below a predefined threshold τ_2 are classified as unknown and forwarded to the third stage.

Meta-IDS: Third Stage



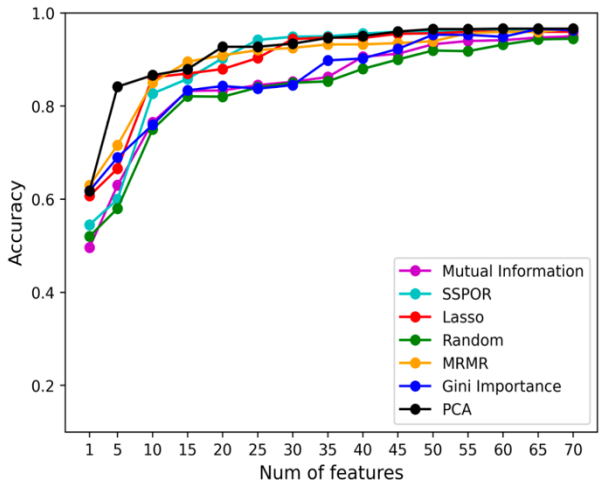
- The last stage involves the utilization of a complex multi-class neural network (with three hidden layers), encompassing all available features.
- The classifier generates probabilities for each known attack class as well as the benign class, and the class with the highest probability is assigned as the predicted class.
- The main aim of this stage is to address false positives, particularly instances where benign network activity is incorrectly identified as malicious.

Feature Selection

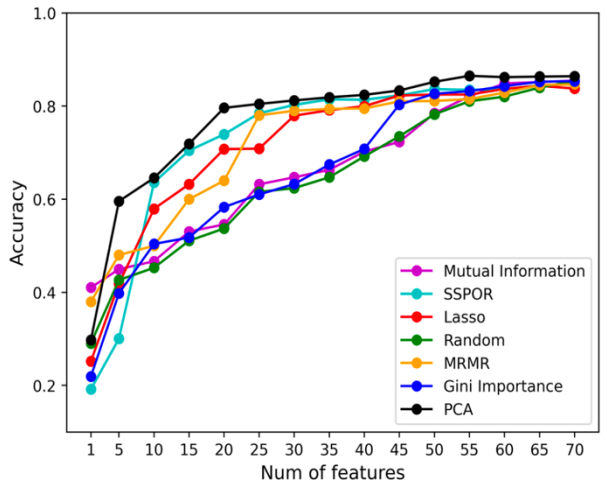
- We used different feature selection methods to select the most informative features from CICFlowMeter features to design a lightweight intrusion detection system.

- Approximately 35 features are adequate to achieving satisfactory accuracy in binary classification, while a minimum of 50 features is necessary for acceptable accuracy in multi-class classification.

- It is evident that PCA as a feature extraction algorithm exhibits the highest accuracy compared to different feature selection methods.



(a) Binary Classification

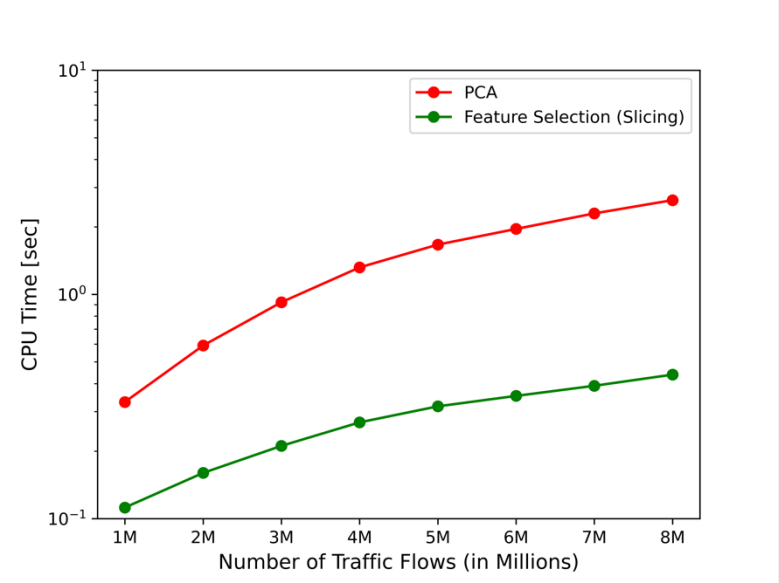


(b) Multi-Class Classification

Feature Selection

- Despite its superior performance, PCA was not considered for our experiment due to its significant time overhead.

- Unlike other methods where the chosen features can be readily applied through straightforward slicing operations, PCA necessitates transforming the original data into a new coordinate system via matrix multiplication.



Evaluation

- Dataset CICIDS2017 : 5 days using 14 machines, variety of attacks

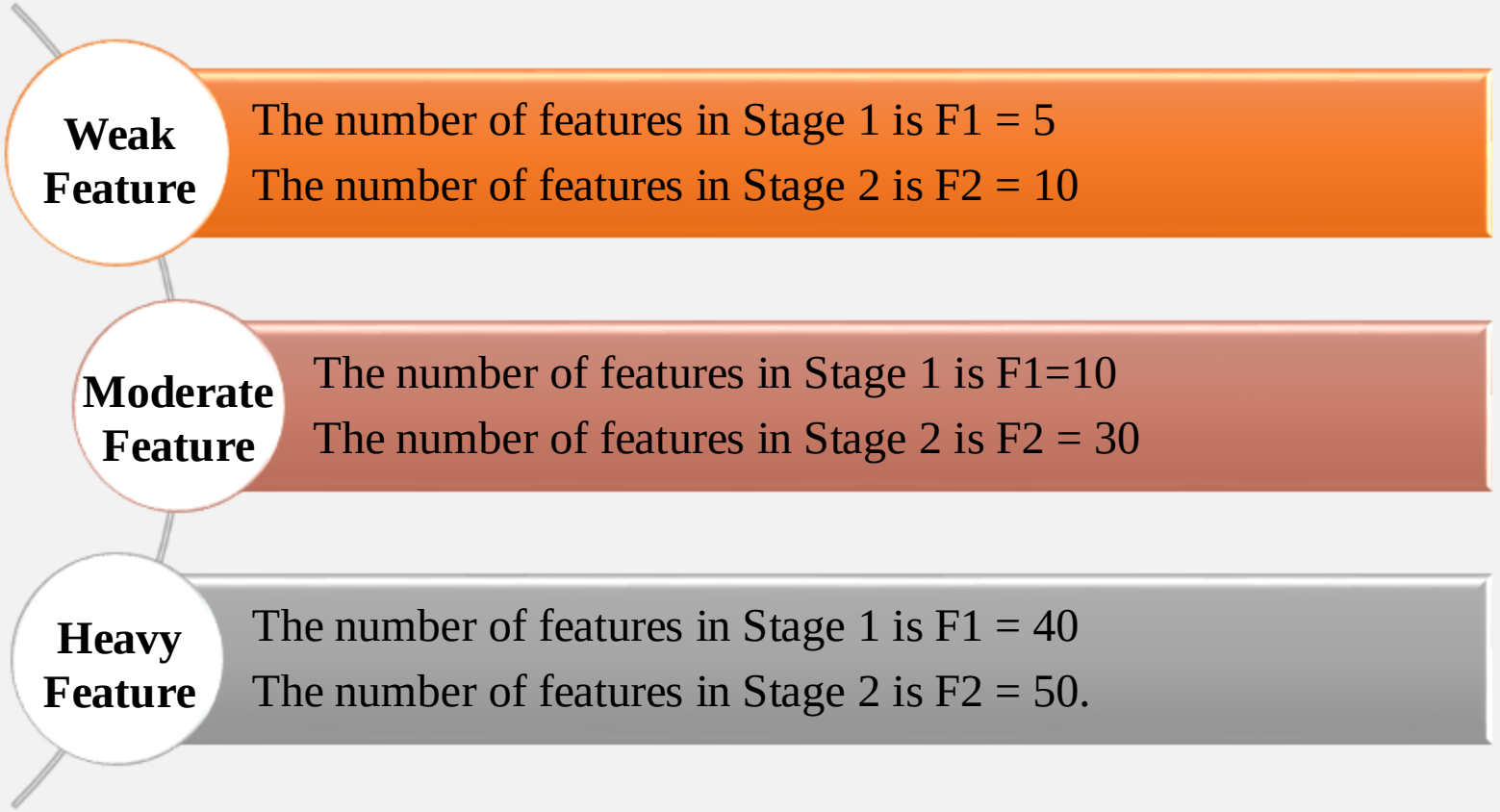
- Preprocessing:
  - Data samples containing NaN values were eliminated.
  - Data samples featuring negative values for attributes that necessitate non-negative values were discarded.
  - One- hot encoding was applied to the Protocol column, resulting in three new features.

- We excluded the dataset pertaining to infiltration attacks from our analysis due to the exceedingly small number of samples available for this type of attack.

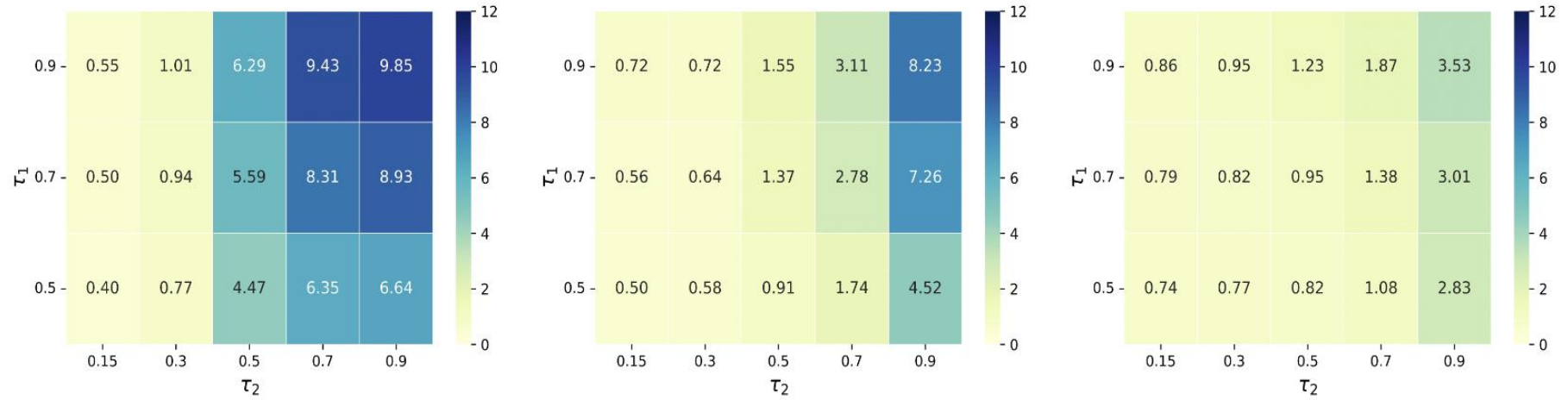
Label	Category in CICIDS2017	Number of Samples
Benign	Benign	1982759
DoS	DoS Hulk	230123
	DoS Slowloris	5796
	DoS Slowhttp	5499
	DoS GoldenEye	10293
	Heartbleed	11
Web Attack	Brute Force - XSS	2159
	Brute Force - Web	1507
	SQL Injection	21
DDoS	DDoS	128025
Brute-Force	FTP-Patator	7935
	SSH-Patator	5897
Bot	Bot	1956
PortScan	PortScan	158804

Experimental Design

Scenarios

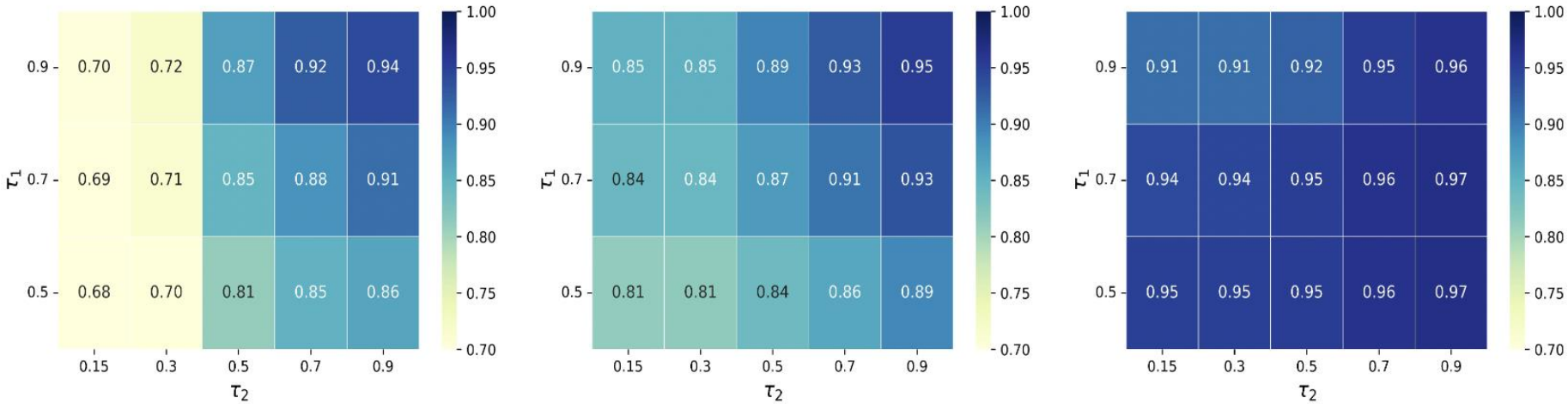


Given these scenarios, we select different values of τ_1 and τ_2 and obtain accuracy and CPU time on 8 million of testing data points for each case.



(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 7: CPU time (sec) for different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) on varying thresholds τ_1 and τ_2.



(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 8: Accuracy for different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) on varying thresholds τ_1 and τ_2.

Utility Function

- Determining the optimal values for thresholds τ_1 and τ_2, considering both CPU time and accuracy:

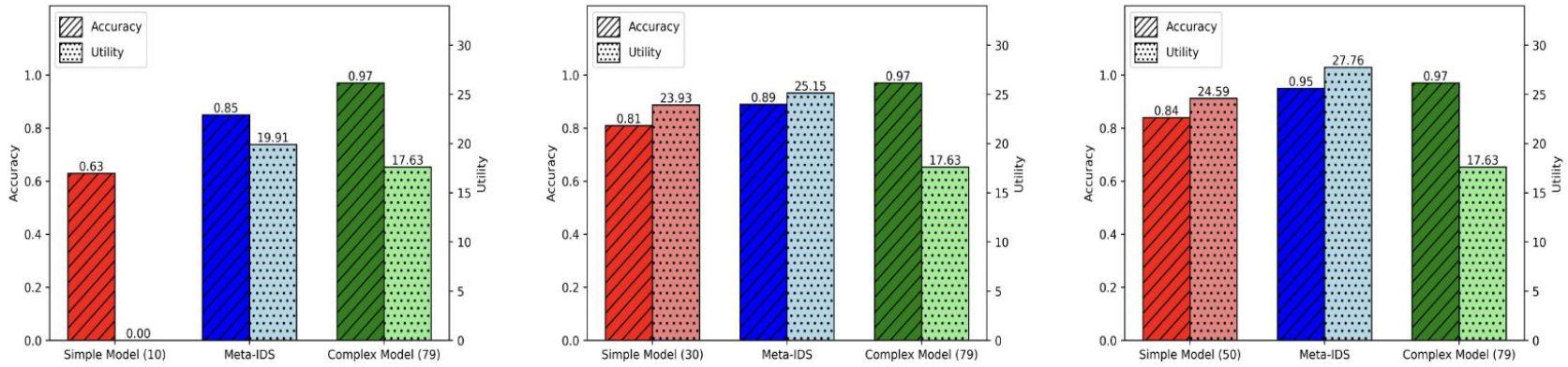
$$Utility(\tau_1, \tau_2) = \begin{cases} Score(\tau_1, \tau_2), & \text{if } Accuracy(\tau_1, \tau_2) > 0.8 \\ 0 & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} Score(\tau_1, \tau_2) =& (W_{ACC} \times Accuracy(\tau_1, \tau_2)) \\ & -(W_T \times Time(\tau_1, \tau_2)). \end{aligned} \quad (1)$$

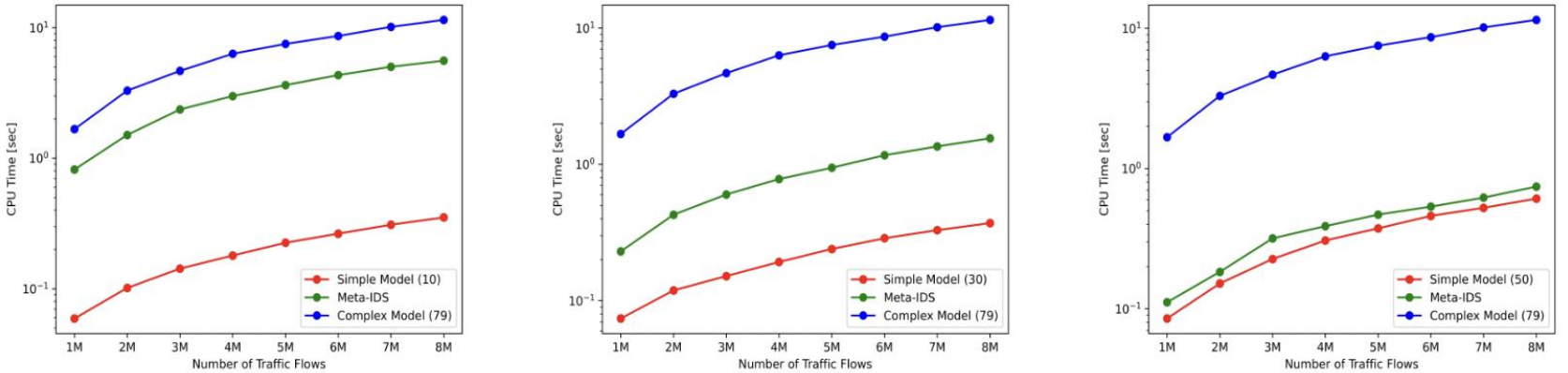
- A higher utility score indicates a more desirable balance between accuracy and CPU time.

- Example:
  - If we can enhance accuracy by 0.01, there would be an additional 0.3 seconds of CPU time.
  - Under such circumstances, we may set weights as follows: $W_{ACC} = 30$ and $W_T = 1$.



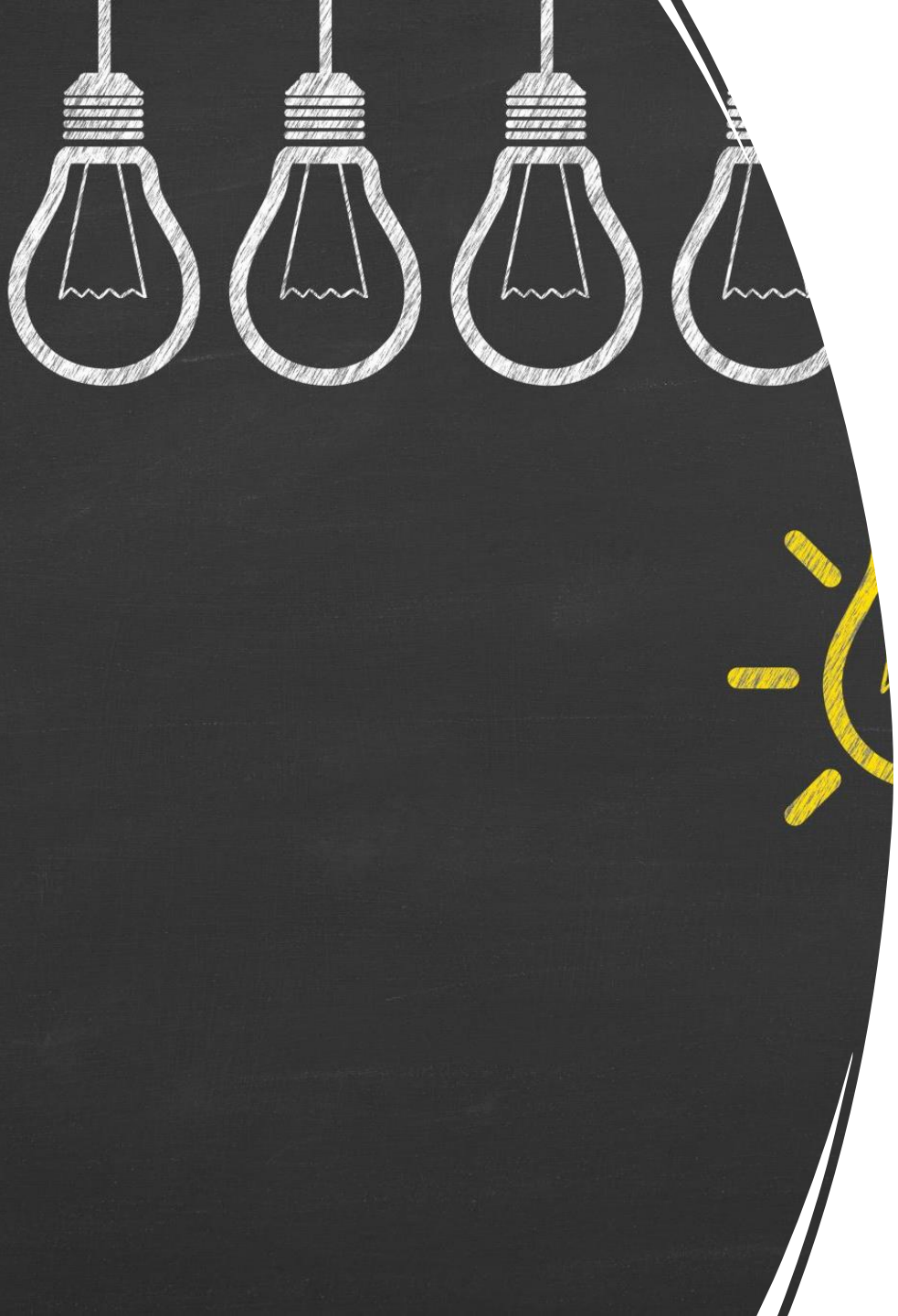
(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 9: Accuracy and Utility for the best thresholds in different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) in Figs. 7 and 8. Part (a) shows accuracy for *Weak-Feature Scenario* when $\tau_1 = 0.7$ and $\tau_2 = 0.5$. Part (b) shows accuracy for *Moderate-Feature Scenario* when $\tau_1 = 0.9$ and $\tau_2 = 0.5$. Part (c) shows accuracy for *Heavy-Feature Scenario* when $\tau_1 = 0.5$ and $\tau_2 = 0.15$. We considered $W_{ACC} = 30$.



(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 10: CPU time for the best thresholds in different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) in Figs. 7 and 8. Part (a) shows CPU time for *Weak-Feature Scenario* when $\tau_1 = 0.7$ and $\tau_2 = 0.5$. Part (b) shows CPU time for *Moderate-Feature Scenario* when $\tau_1 = 0.9$ and $\tau_2 = 0.5$. Part (c) shows CPU time for *Weak-Feature Scenario* when $\tau_1 = 0.5$ and $\tau_2 = 0.15$. We considered $W_{ACC} = 30$.



Conclusion

- Our hierarchical framework for intrusion detection, inspired by meta-computing principles effectively balances accuracy and computational efficiency.
- Through a series of experiments, we demonstrate the superior performance of our approach concerning CPU utilization and accuracy metrics in the binary and multi-class classification of traffic flows.
- Key to the success of our methodology is the careful selection of thresholds (τ_1 and τ_2) governing the transition between stages. Our analysis highlights the critical role of these thresholds in optimizing resource utilization while maintaining high levels of accuracy.