IoT & its Security (ITIT-9307) Project

Title: Machine Learning Methods in Tasks Load Balancing Between IoT Devices and the Cloud

Presented By
Saumya Thakor (2024WNC-010)

Introduction

Introduction

- IoT ecosystems power smart industries (farms, cities, hospitals) with devices ranging from a few to thousands.
- AI enables real-time decision-making, transforming IoT from data collectors to adaptive systems.
- Cloud Computing supports IoT but has high costs, performance issues, and privacy risks.

Motivation

- Problem: Optimizing task distribution between IoT devices and Cloud for efficient resource use.
- Challenge: Estimating device computational capacity and task complexity without a profiler.
- Goal: Using ML to predict task runtime complexity and match tasks to capable devices, reducing Cloud reliance.

Author's Perspective

- **Objective:** Optimize task distribution between IoT devices, server, and Cloud using ML to predict task runtime complexity (Big O).
- **Framework:** Uses LLMs to estimate Big O, matches tasks to devices based on processing capabilities, execution time, and energy.
- **Steps:**
  1. Profile devices to assess capabilities and build historical data.
  2. Predict task Big O using pre-trained LLM.
  3. Estimate task parameters (I/O transfer, execution time, energy) and select suitable devices.
  4. Check device availability via polling; offload tasks, prioritizing devices over server and Cloud.
- **Key Innovation:** Big O-based task matching without profiling, reducing Cloud reliance and enhancing efficiency.

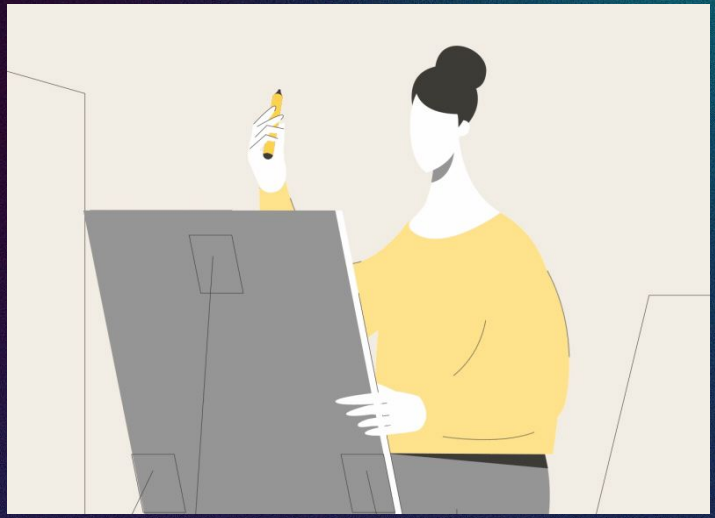
Challenges

- **Author's Solution Issues:**
  1. Unclear profiling for I/O, deadlines, and CPU cycles, leading to unreliable execution time estimates.
  2. Uneven task distribution, with low-complexity tasks skewed due to high-complexity task occupation.
  3. Potential Big O misprediction by LLMs, affecting task-device matching.

- **Replication Challenges:**
  1. Dataset unavailability (device parameters and Big O labels); generated datasets caused bias toward Cloud offloading.
  2. No viable solution found to replicate framework without introducing bias.

Novelty

- **Synthetic Dataset:** Generates realistic dataset (40,000 samples) with device parameters and Big O labels, solving dataset unavailability.
- **Random Forest Model:** Predicts optimal device for tasks, avoiding LLM Big O misprediction and unclear profiling.
- **Balanced Distribution:** Uses device constraints (CPU, memory, energy) and stratified sampling to reduce uneven task allocation and Cloud bias.
- **Interactive Interface:** Gradio UI enables real-time task-device matching with probability visualization.
- **Scalable Framework:** Handles diverse IoT devices and task complexities efficiently.



Data Generation for IoT Device Classification

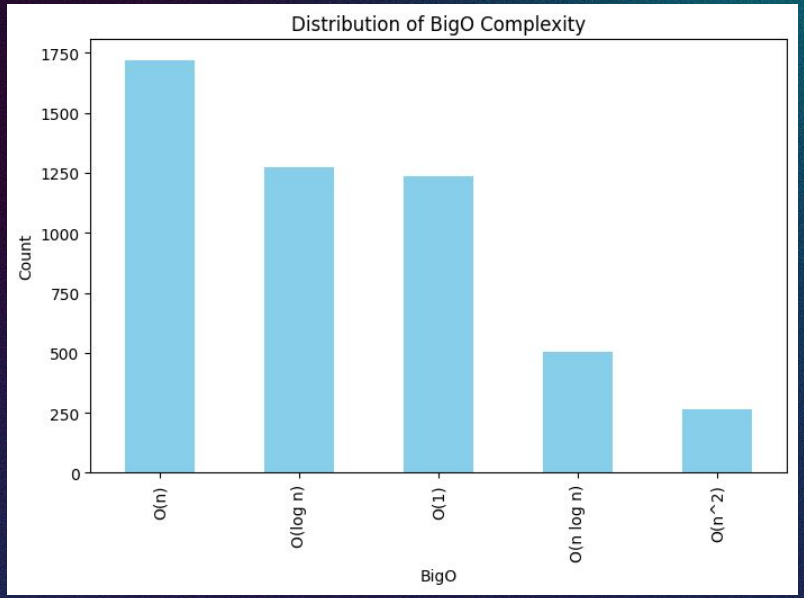
Objective: Create a dataset to classify IoT devices based on performance metrics and computational complexities.

Key Features:

- **Big-O Notations: Simulates complexities like O(1), O(log n), O(n), etc.**
- **Device Profiles: Includes Raspberry Pi, Arduino, ESP32, etc.**
- **Metrics: Input Size, CPU Speed, Memory, Power Cost, Latency, Bandwidth, Energy.**

Data Generation:

- **Device-specific configurations with noise.**
- **10% random label flip for variability.**



Data Preprocessing and Feature Engineering

Steps:

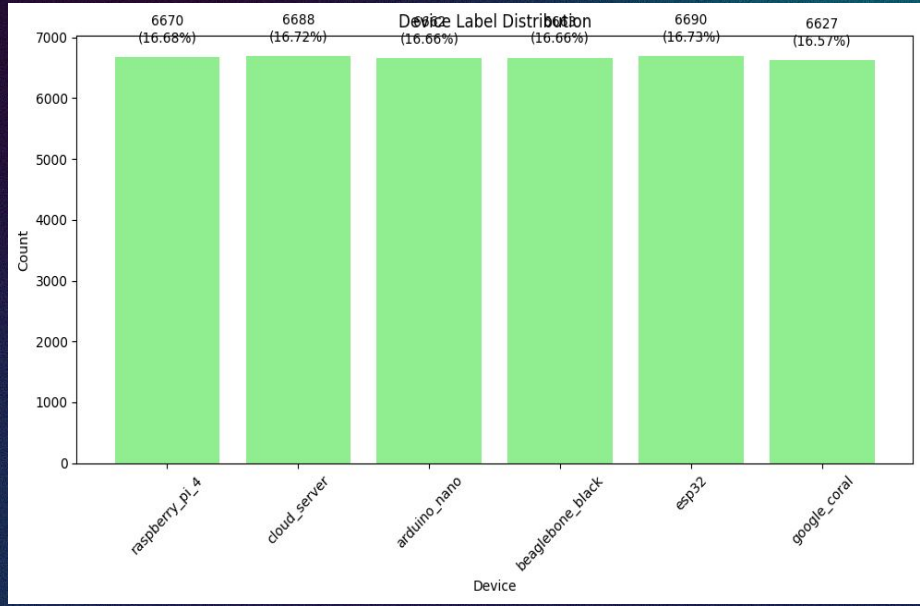
- **Lognormal Distribution: For input sizes across devices.**
- **Random Sampling: For device metrics.**
- **Normalization: Applied StandardScaler to normalize features.**

Label Generation:

- **Random assignment with equal distribution and label noise.**

Resulting Data:

- **Shape: 40,000 samples, 9 features, 1 label.**
- **Features: CPU speed, memory, latency, input size, bandwidth, energy, power cost.**
- **Target: Device classification.**



Feature Alignment and Minimal Noise

Feature Alignment:

- **Aligned key features (CPU speed, memory, input size, bandwidth) to match IoT devices (e.g., Raspberry Pi > Arduino/ESP32).**
- **Ensured realistic device behavior (latency, energy) for accurate classification.**

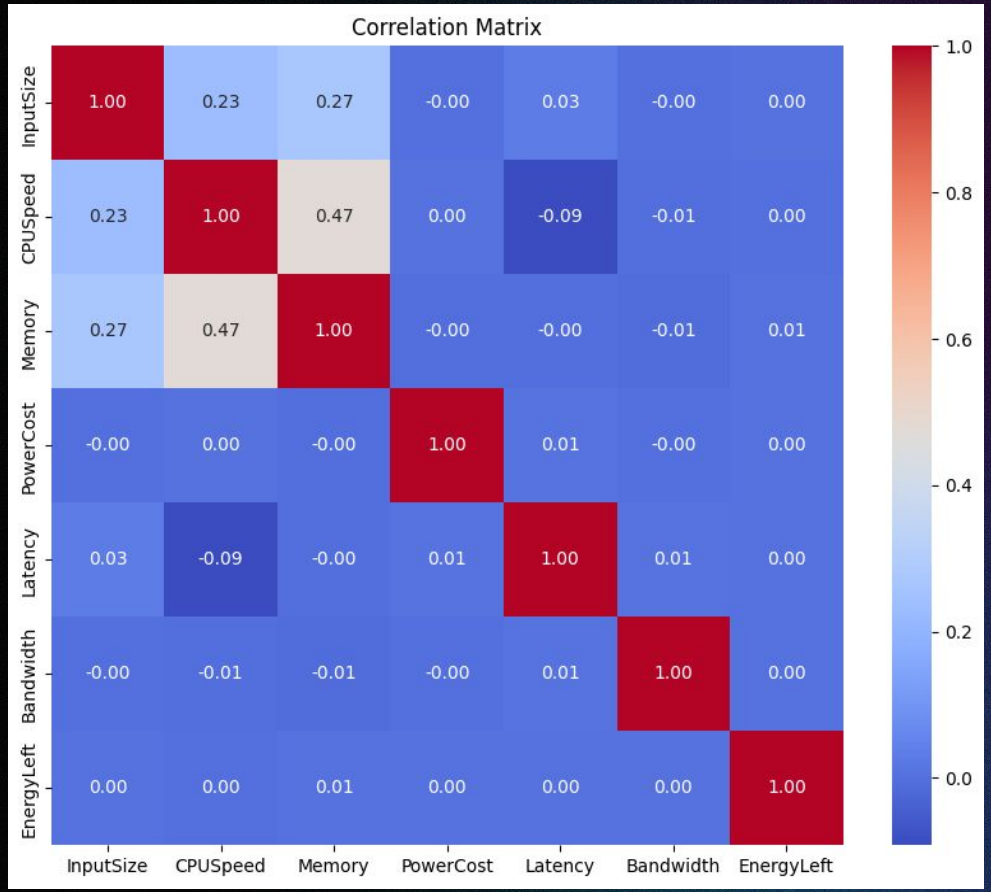
Minimal Noise:

- **Added noise (mean = 1.0, std dev = 0.05) to simulate real-world variations without distorting data.**

Resulting Data:

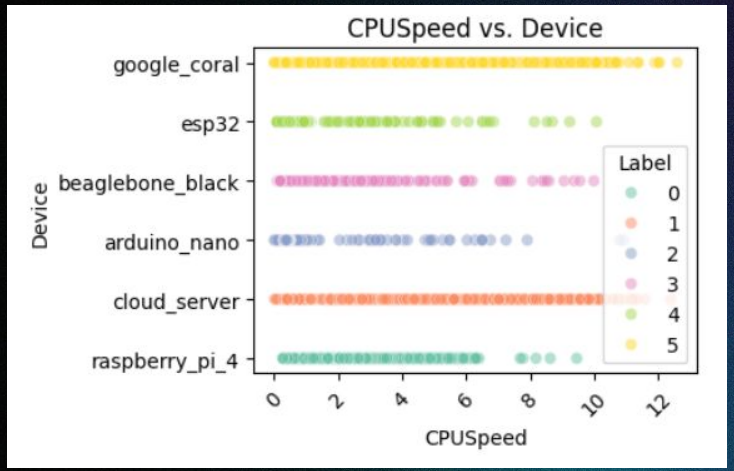
- **Consistent device profiles for effective classification.**
- **Minimal noise maintained data integrity while simulating real-world conditions.**

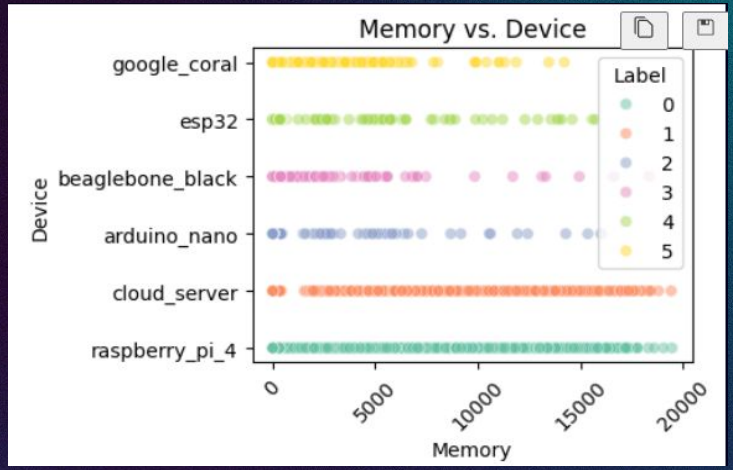
Exploratory Data Analysis



Correlation Matrix

Exploratory Data Analysis





	BigO	InputSize	CPUSpeed	Memory	PowerCost	Latency	Bandwidth	EnergyLeft	Label
0	O(log n)	204	4.495138	8256.383567	0.511239	82.368487	1010.719139	77.103445	5
1	O(n^2)	287	0.512864	2579.444241	0.381517	68.757061	1200.677027	95.737664	3
2	O(n)	328	2.573923	4861.880551	0.678005	39.986274	685.136421	85.816082	5
3	O(n)	317	1.401045	3292.646379	0.761401	16.948996	758.929547	80.891191	5
4	O(1)	3000	7.501359	6002.235194	0.838002	30.813739	948.253061	8.419157	1

Data Preprocessing and Train-Test Split

Big-O Mapping:

- **Converted time complexities to numeric scale:**
  - **O(1) → 1, ..., O(n^2) → 5.**

Feature & Label Separation:

- **Dropped 'Label' column to create feature matrix X and target vector y.**

Scaling:

- **Standardized features using StandardScaler to normalize across varied units (e.g., CPU speed vs. latency).**

Data Split:

- **Performed stratified 70-30 train-test split to maintain label distribution balance.**
- **Ensures fair evaluation and avoids class imbalance during training.**

Model Training

Trained Models with Fine-Tuned Hyperparameters:

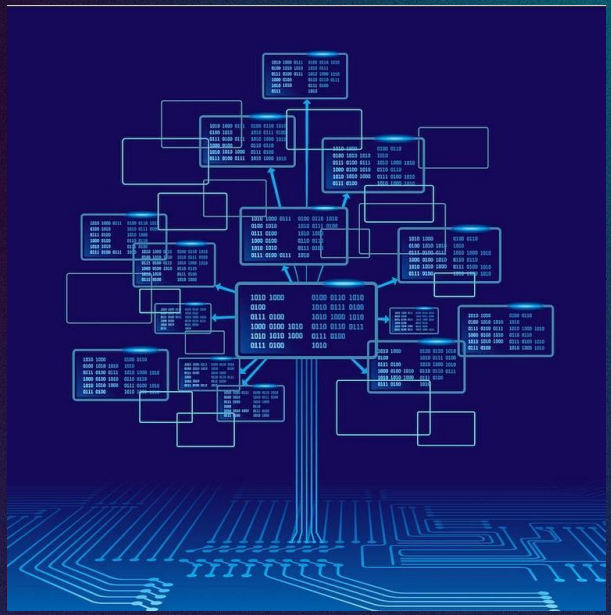
- **Logistic Regression: C=1.0, max_iter=2000**
- **Random_Forest:_estimators=150,max_depth=15, min_samples_split=10**
- **Gradient Boosting: learning_rate=0.15, max_depth=5**
- **SVM (RBF): C=1.5, probability=True**

Evaluated on Test Data:

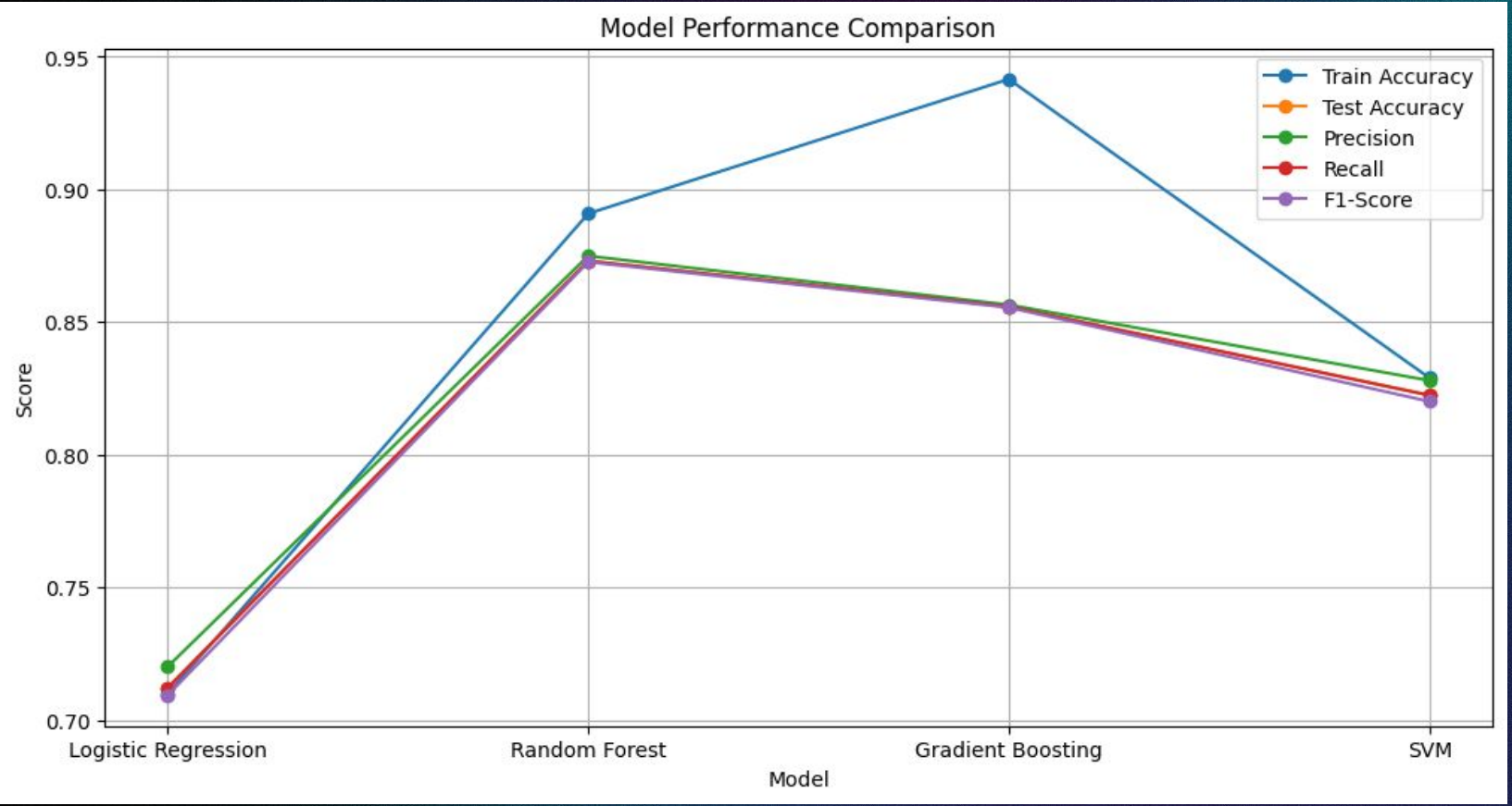
- **Used metrics: Accuracy, Precision, Recall, F1-score (weighted)**
- **Also captured Training Time for each model.**

Cross-Validation (CV):

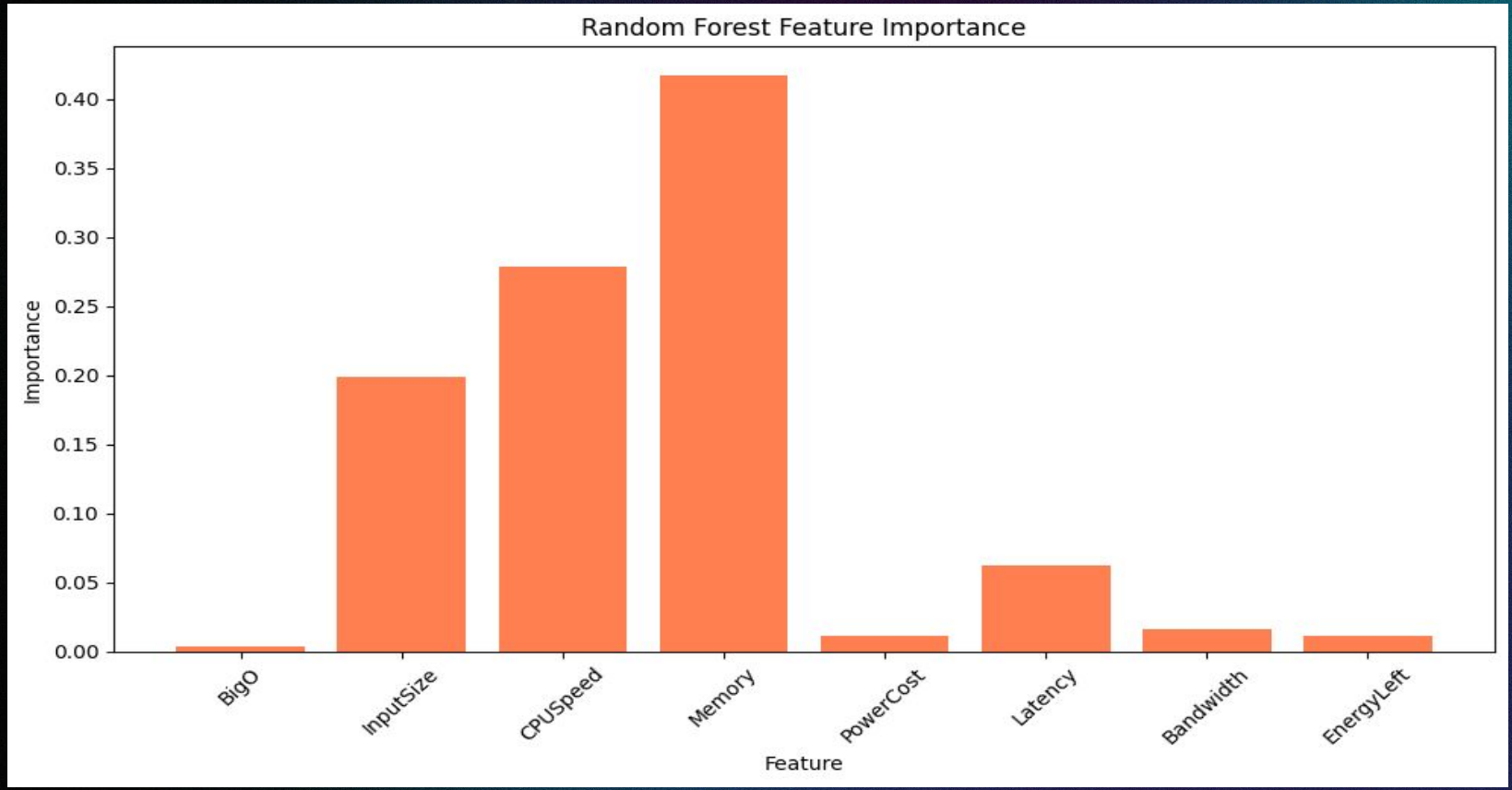
- **5-fold CV used to calculate mean ± std of CV Accuracy for each model, ensuring generalization and robustness.**



Performance Comparison



Visualisation Summary



Device Selection Logic

Goal: Select the most optimal device based on task requirements and device specifications using ML predictions.

1. Constraint Filtering:

Only consider devices that meet all task-defined constraints:

CPUSpeed ≥ MinCpuSpeed

Memory ≥ MinMemory

PowerCost ≤ MaxPowerCost

Latency ≤ MaxLatency

Bandwidth ≥ MinBandwidth

EnergyLeft ≥ MinEnergyLeft

2. Feature Vector Construction:

For each eligible device, build a feature vector:
[BigO, InputSize, CPUSpeed, Memory, PowerCost, Latency, Bandwidth, EnergyLeft]

3. Model Prediction:
Scale features using pre-fitted StandardScaler

Use predict_proba() to estimate each device's suitability

Device with the highest probability score is selected as the optimal choice

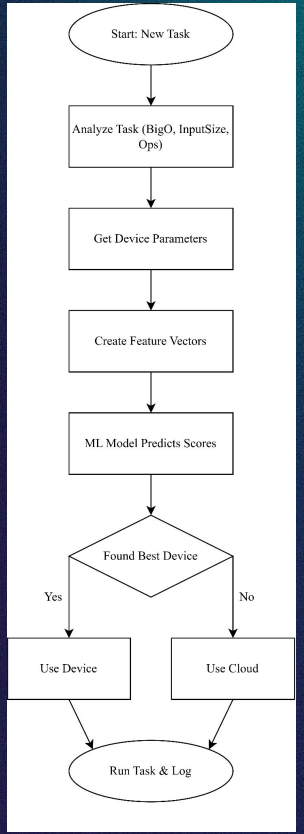
4. Fallback Option:

If no device meets constraints:
→ Return 'cloud_server' with default probability [0.0, 1.0, 0.0, 0.0, 0.0, 0.0]

Implementation flow

The provided flowchart outlines a systematic process for selecting the most suitable IoT device to execute a given task. This workflow takes into account the task requirements, device capabilities, and availability to ensure efficient and effective task completion.

1. Start: This is the initiation point of the workflow. When a new task needs to be executed within the IoT system, this process begins.
2. Analyze task: The first crucial step is to thoroughly analyze the requirements of the new task. This involves identifying key parameters that will influence the choice of the device.
3. Get device parameters: In this step, the system retrieves the relevant parameters and capabilities of all the available IoT devices. The parameters include memory, latency, bandwidth, CPU speed, energy left etc.

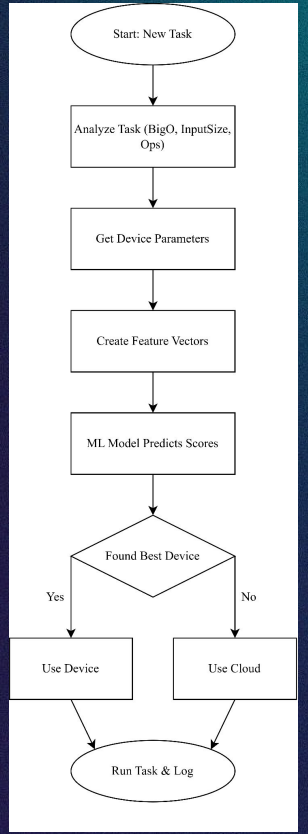


Implementation Flow (Cont.)

4. Create feature vectors: Once the task parameters and device parameters are obtained, the system creates feature vectors for both.

- **Task Feature Vector: This vector numerically represents the analyzed task parameters BigO, InputSize, Ops etc.**
- **Device Feature Vector: This vector numerically represents the capabilities of each IoT device processing power, memory, connectivity, etc.**

5. ML model predicts scores: This step indicates the use of an ML model to predict a suitability score for each available IoT device based on the feature vector. The ML model would have been trained previously on historical data of task executions on different devices, learning the relationships between task requirements and device performance or efficiency.

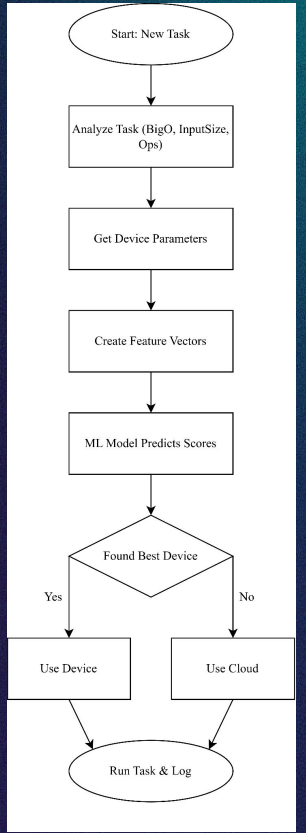


Implementation Flow (Cont.)

6. Found best device: Based on the scores predicted by the ML model, the system selects the device with the highest score as the best device for the current task.

- **Yes (Use device): If the selected best device is available, the workflow proceeds to utilize that device for executing the task.**
- **No (Use cloud): If the selected best device is not available, the workflow takes an alternative path and decides to use cloud resources for executing the task.**

7. Run task & log: Regardless of whether the task is executed on the selected IoT device or in the cloud, this final step involves running the task and logging relevant information.




Thank
You!