Machine Learning Approach for Loop Unrolling Factor Prediction in High Level Synthesis

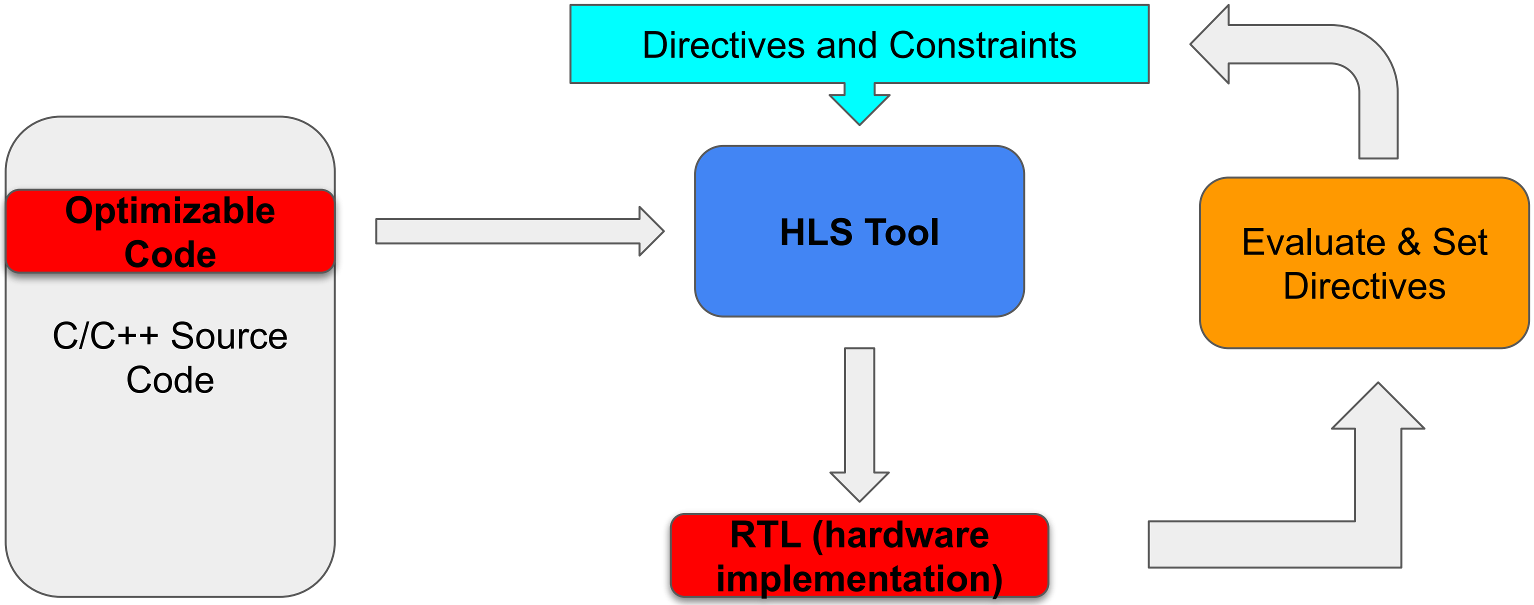
Authors: Georgios Zacharopoulos, Andrea Barbon, Giovanni Ansaloni and Laura Pozzi

Mehar Singh, Anisha Aggarwal, Parth Raut
(Group #10)

High Level Synthesis (HLS)

- HLS frameworks allow **hardware circuits** to be described at **higher abstraction** for customizable hardware
  - Languages like C or C++
- Differs from **Register Transfer Level** (RTL) methods which describe circuits in terms of registers, logical operations, and data movements
  - Languages like Verilog
- HLS allows engineers to deal with hardware design without having to know low-level details
  - Focus on **functionality**
  - Optimizing from code to hardware descriptions is done **automatically**

High Level Synthesis (HLS)



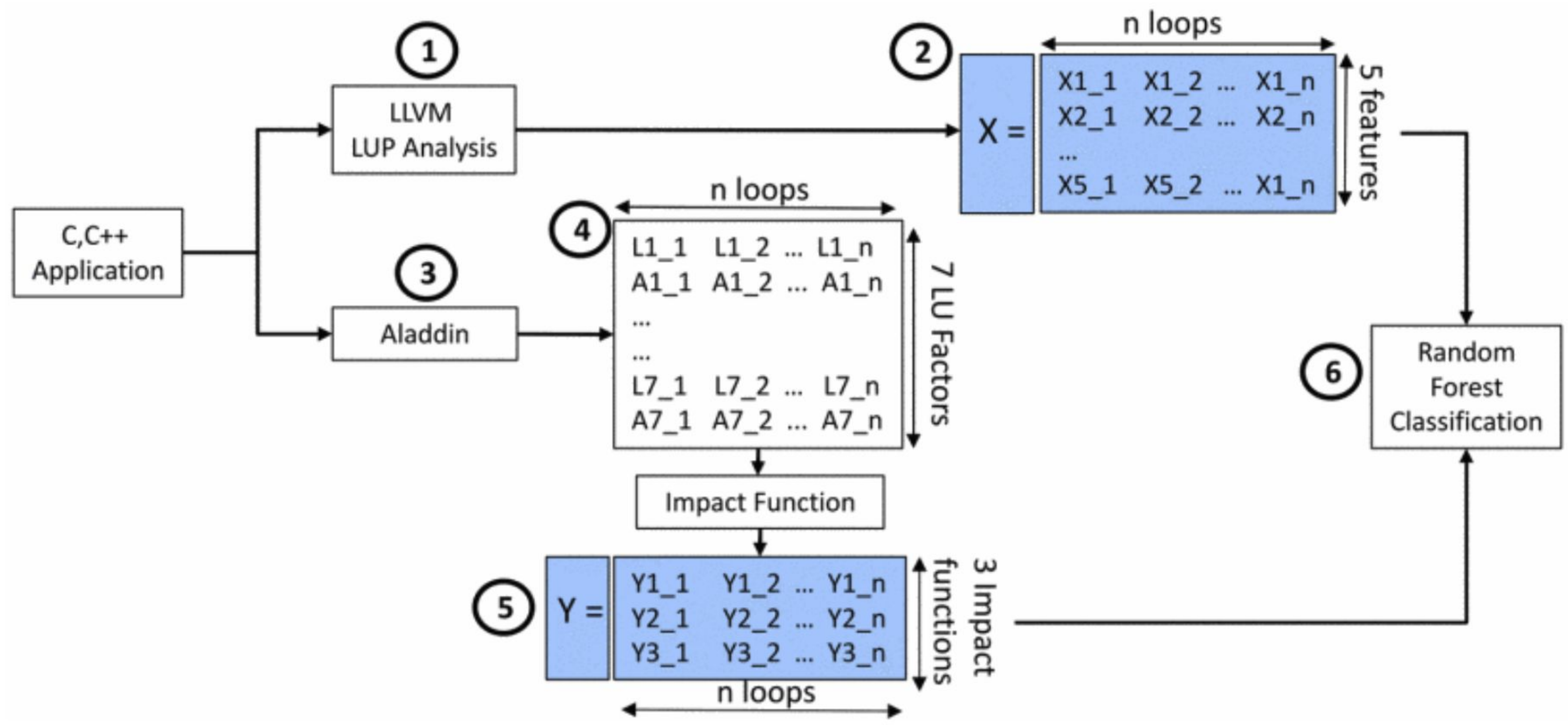
Hardware Loop Unrolling

- HLS is often used to customize hardware to **optimize loops** for **hardware accelerators**.
- Loop unrolling impacts the **performance** of hardware accelerators due to:
  - **High area cost** for duplicated logic
    - Area cost is the size of the chip occupied by the accelerator
  - **Loop-carried dependencies** & **frequent memory access**
    - Causes accelerators to act more sequentially

Paper's Contributions

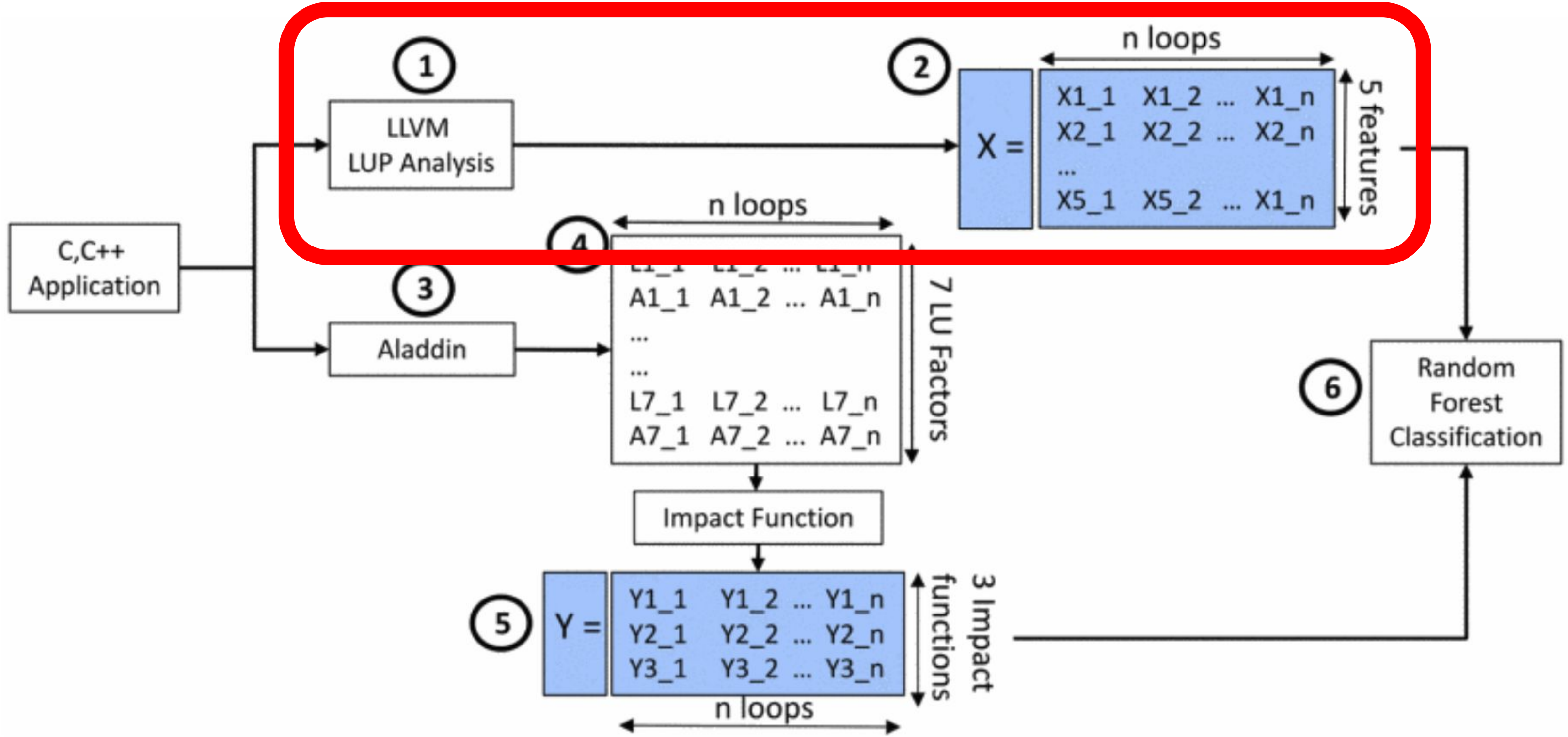
1. Trained a **Random Forest (RF) classifier** to predict unrolling factors for loops in HLS designs
2. Developed an **automated framework** in LLVM to **extract features** (for both training and inference) as input to the classifier

Method



C,C++
Application
1
LLVM
LUP Analysis
2
n loops
X =
X1_1 X1_2 ... X1_n
X2_1 X2_2 ... X2_n
...
X5_1 X5_2 ... X1_n
5 features
3
Aladdin
4
n loops
L1_1 L1_2 ... L1_n
A1_1 A1_2 ... A1_n
...
...
L7_1 L7_2 ... L7_n
A7_1 A7_2 ... A7_n
7 LU Factors
Impact Function
5
Y =
Y1_1 Y1_2 ... Y1_n
Y2_1 Y2_2 ... Y2_n
Y3_1 Y3_2 ... Y3_n
n loops
3 Impact functions
6
Random
Forest
Classification

Method



C,C++
Application
1
LLVM
LUP Analysis
2
X =
n loops
X1_1 X1_2 ... X1_n
X2_1 X2_2 ... X2_n
...
X5_1 X5_2 ... X1_n
5 features
3
Aladdin
4
n loops
A1_1 A1_2 ... A1_n
...
...
L7_1 L7_2 ... L7_n
A7_1 A7_2 ... A7_n
7 LU Factors
Impact Function
5
Y =
Y1_1 Y1_2 ... Y1_n
Y2_1 Y2_2 ... Y2_n
Y3_1 Y3_2 ... Y3_n
n loops
3 Impact functions
6
Random
Forest
Classification

Steps 1 & 2: Feature Selection for RF Classifier

1. Length of critical path
2. Loop trip count
3. The presence of loop carried dependencies
4. The # of load instructions
5. The # of store instructions

different features compared to original MIT paper

FEATURE VECTORS SELECTED BY STEPHENSON ET AL. [9].

Features - X Vector 1	**Features - X Vector 2**
# Operands	*# Floating Point Operations*
Range Size	*Loop Nest Level*
Critical Path	*# Operands*
# Operations	*# Branches*
Loop Trip Count	*# Memory Operations*

Steps 1 & 2: LLVM Feature Extraction Pass

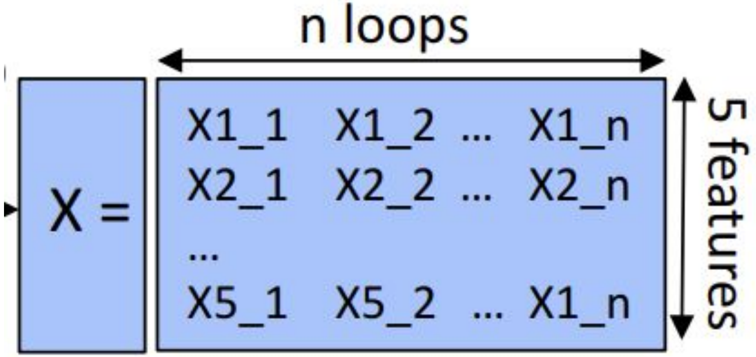
Algorithm 1 LLVM Analysis Pass - Loop Unrolling Prediction Analysis

Input: Application written in C, C++
Output: X (Feature Vector)

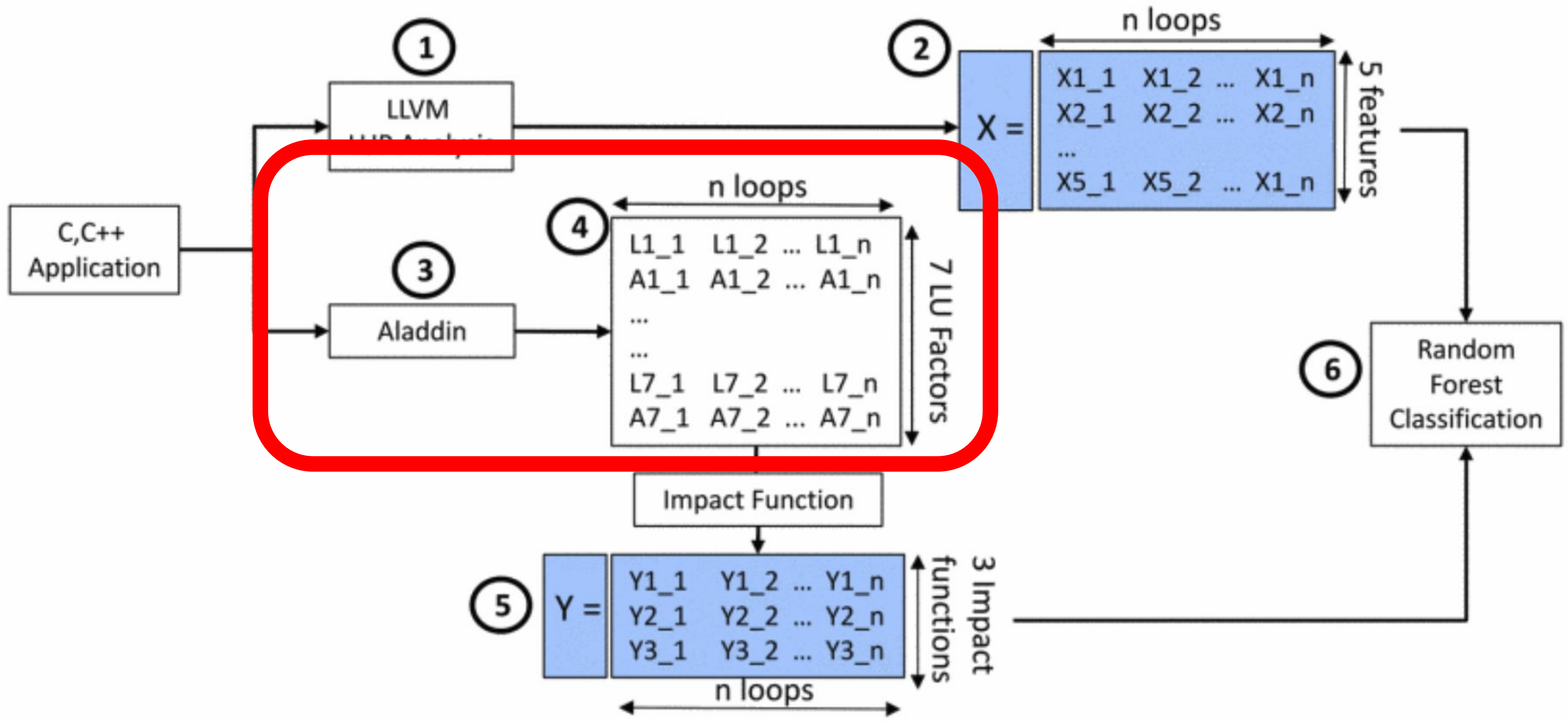
```
function RunOnFunction( )
    for BB in Function do
        if L=getLoopForBB() then
            LoopUnrollingPredictionAnalysis(BB,L)

function LoopUnrollingPredictionAnalysis(Basic Block BB, Loop L)
    LI=getLoopInfoAnalysis()
    SE=getScalarEvolutionAnalysis()
    DA=getDependenceAnalysis()
    /* Gather Features for X Vector */
    x1=getCriticalPath(BB)
    x2=getTripCountForLoop(L)
    x3=getLoopCarriedDependencies(BB)
    x4=getNumberOfLoadInstructions(BB)
    x5=getNumberOfStoreInstructions(BB)
```

extracts all 5 features for each loop



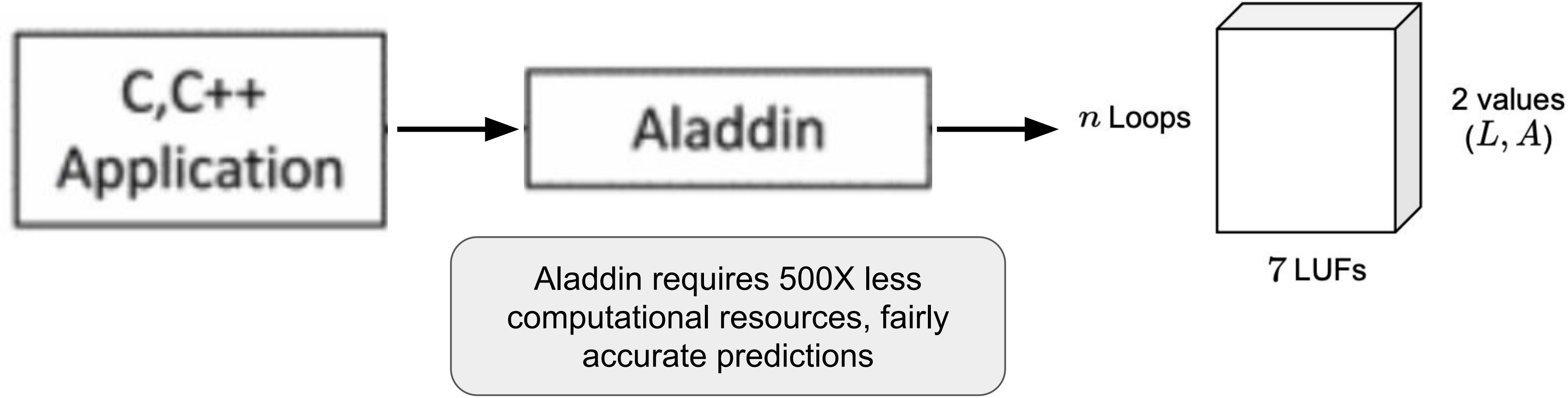
Method



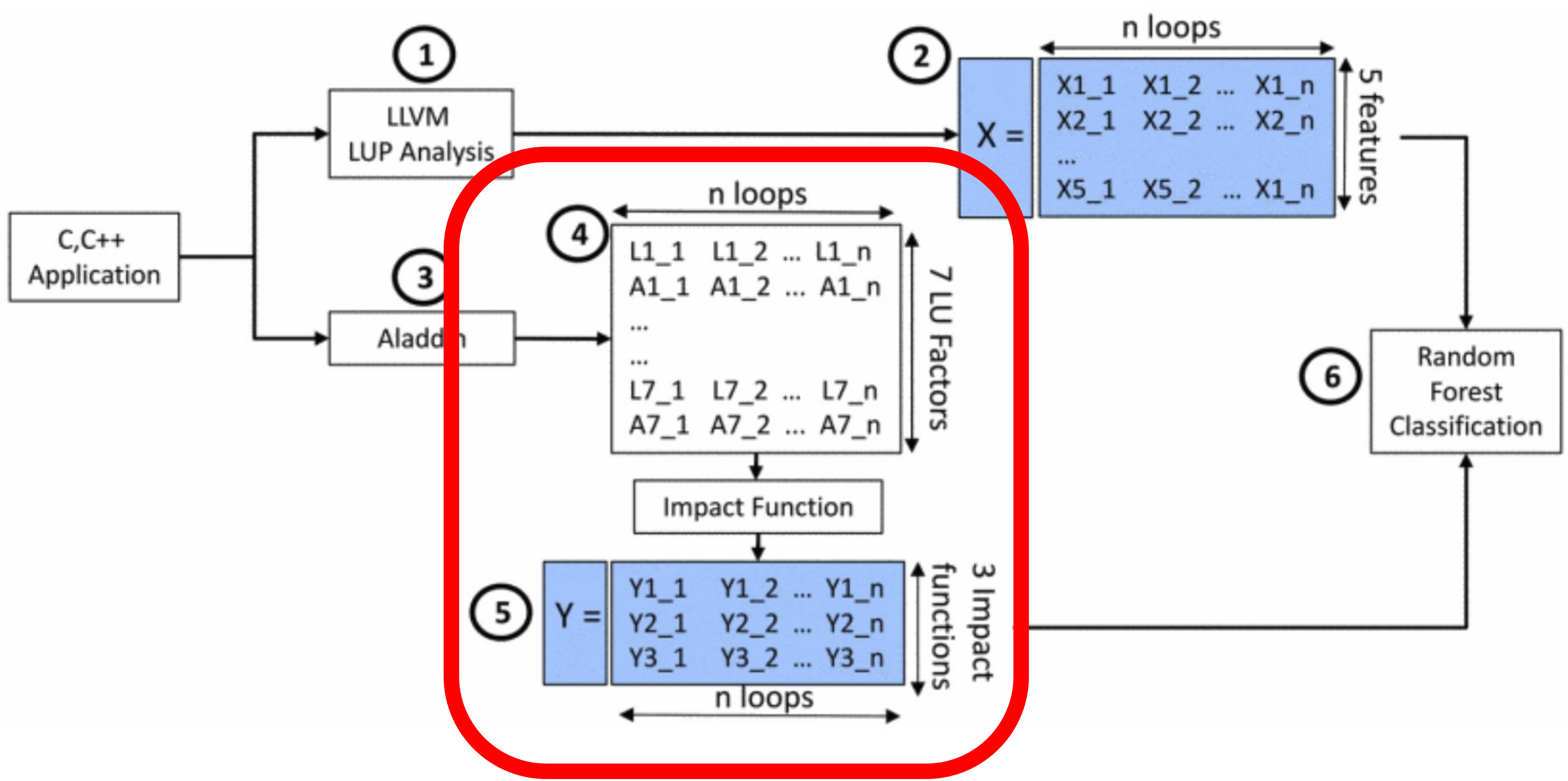
1
LLVM
2
n loops
X =
X1_1 X1_2 ... X1_n
X2_1 X2_2 ... X2_n
...
X5_1 X5_2 ... X1_n
5 features
C,C++
Application
3
Aladdin
4
n loops
L1_1 L1_2 ... L1_n
A1_1 A1_2 ... A1_n
...
...
L7_1 L7_2 ... L7_n
A7_1 A7_2 ... A7_n
7 LU Factors
Impact Function
5
Y =
Y1_1 Y1_2 ... Y1_n
Y2_1 Y2_2 ... Y2_n
Y3_1 Y3_2 ... Y3_n
n loops
3 Impact functions
6
Random
Forest
Classification

Steps 3 & 4: Approximating L & A with Aladdin

- Input: C/C++ programs to simulate
- Output: **latency** (L) and **area** (A) **values** for 7 loop unroll factors (LUFs)
    - LUFs: 1, 2, 4, 8, 16, 32, 64



Method



1
LLVM
LUP Analysis
2
n loops
X =
X1_1 X1_2 ... X1_n
X2_1 X2_2 ... X2_n
...
X5_1 X5_2 ... X1_n
5 features
C,C++
Application
3
Aladdin
4
n loops
L1_1 L1_2 ... L1_n
A1_1 A1_2 ... A1_n
...
...
L7_1 L7_2 ... L7_n
A7_1 A7_2 ... A7_n
7 LU Factors
Impact Function
5
Y =
Y1_1 Y1_2 ... Y1_n
Y2_1 Y2_2 ... Y2_n
Y3_1 Y3_2 ... Y3_n
n loops
3 Impact functions
6
Random
Forest
Classification

Step 4 & 5: Impact Function

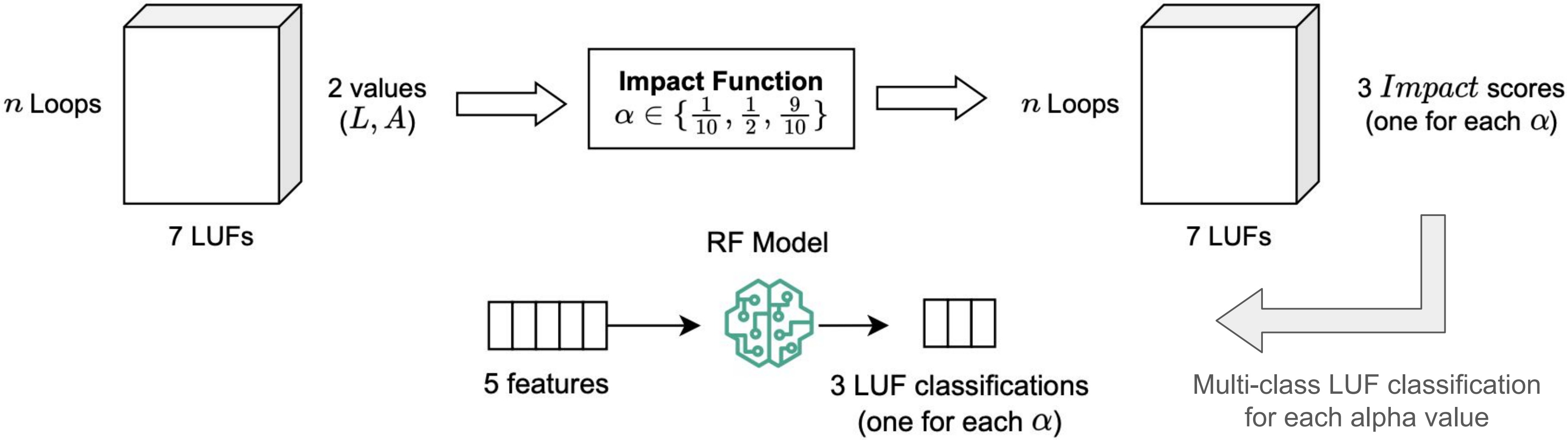
- Tradeoff between performance (latency) & required resources (area)

$$I(L,\ A) = \alpha \cdot \frac{(L_1 - L)}{L_1} + (1 - \alpha) \cdot \frac{(A_1 - A)}{A_1}, 0 \le \alpha \le 1$$

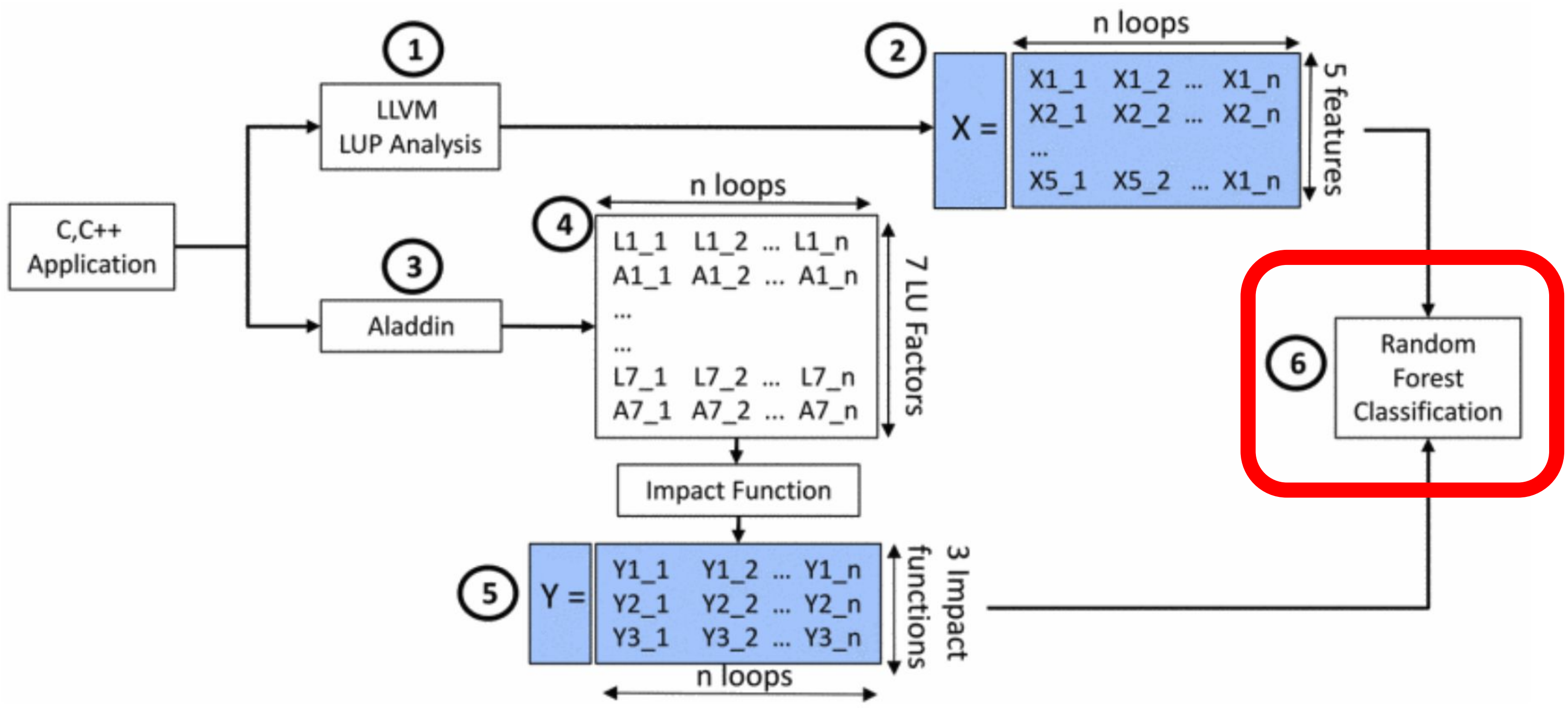
- L & A = latency & area of function synthesized as accelerator for chosen LUF
- L_1 & A_1 = latency & area of function synthesized as accelerator when LUF is 1
- α = relevance of latency & area
- The impact score is used to generate the ground truth for the RF model during training
    - The LUF that produces the highest impact score is considered the ground truth

Step 4 & 5: Impact Function

- Apply the impact function for 3 α values
  - α = 0.5: Optimize L & A equally
  - α = 0.9: Optimize L over A
  - α = 0.1: Optimize A over L

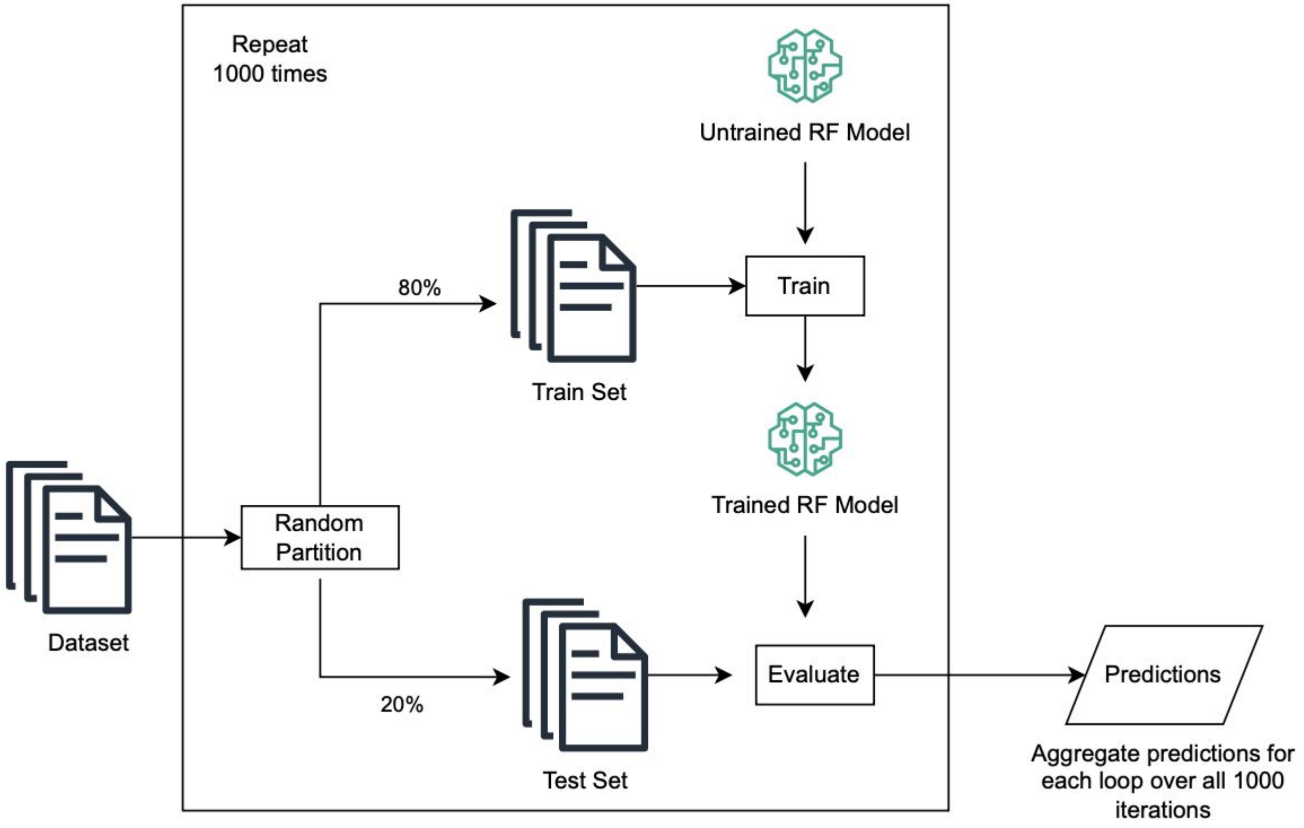


Method



1
LLVM
LUP Analysis
2
n loops
X =
X1_1 X1_2 ... X1_n
X2_1 X2_2 ... X2_n
...
X5_1 X5_2 ... X1_n
5 features
C,C++
Application
3
Aladdin
4
n loops
L1_1 L1_2 ... L1_n
A1_1 A1_2 ... A1_n
...
...
L7_1 L7_2 ... L7_n
A7_1 A7_2 ... A7_n
7 LU Factors
Impact Function
5
Y =
Y1_1 Y1_2 ... Y1_n
Y2_1 Y2_2 ... Y2_n
Y3_1 Y3_2 ... Y3_n
n loops
3 Impact functions
6
Random
Forest
Classification

Step 6: Random Forest (RF) Classification

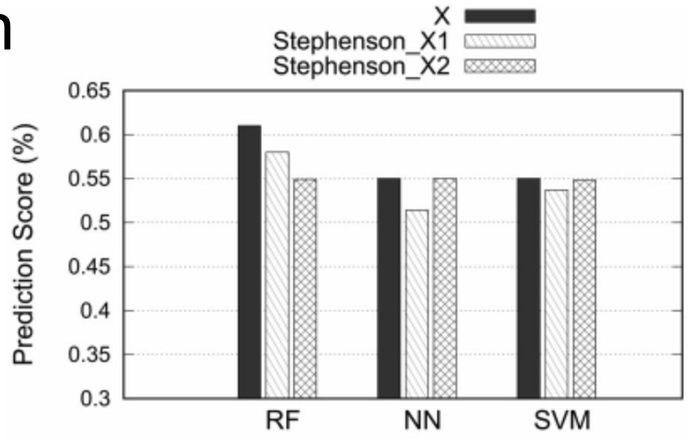


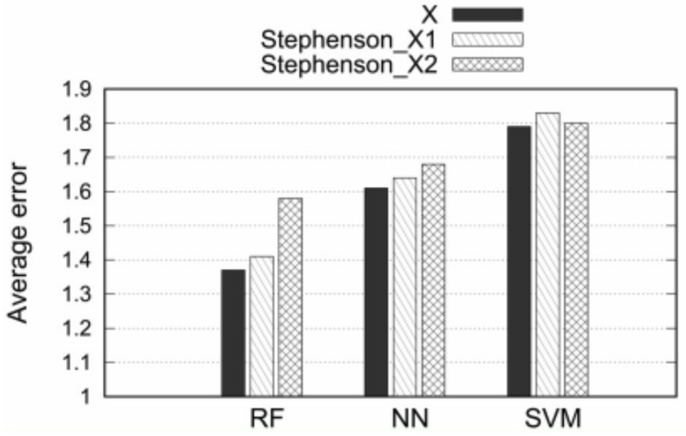
Evaluation Metrics

- *Prediction score:* percentage of optimal LUFs correctly identified on test set
- *Average error:* average distance between correct & predicted LUFs

Results: Model & Feature Selection

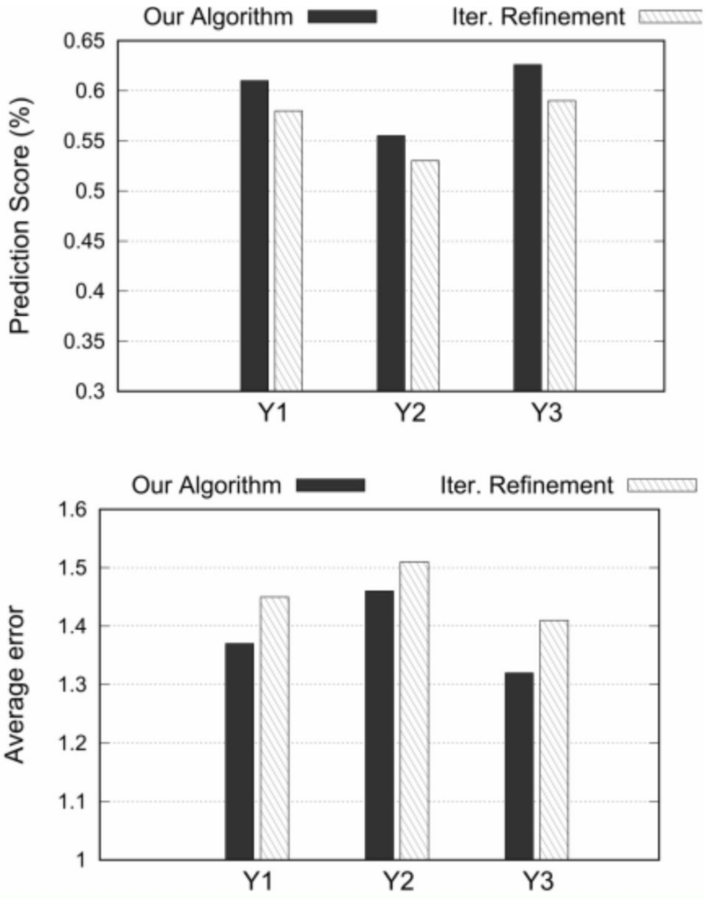
- Compared feature selection against Stephenson et. al. (original MIT paper)
- α = 0.5
- Used 3 different models: RF, Nearest Neighbor (NN), and Support Vector Machine (SVM)
- Paper presents **superior feature selection** and **choice of model**





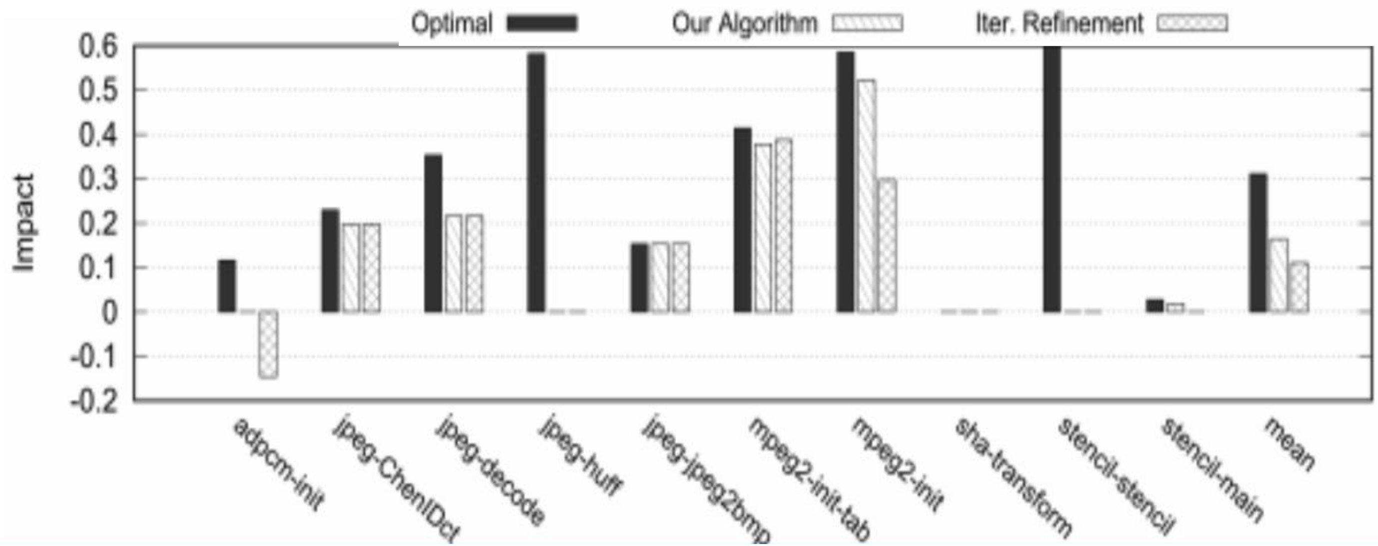
Results: Training Method

- Compared against an Iterative Refinement (IR) approach
    - IR first trains the model on a dataset, and then trains it again on a disjoint dataset
- Y1: α = 0.5
- Y2: α = 0.9
- Y3: α = 0.1
- Paper algorithm **outperforms** Iterative Refinement



Results: Impact Score

- Compared against an Iterative Refinement (IR) approach
- Improvement or comparable impact against IR in all cases
- Poor performance on jpeg-huff and stencil-stencil, tests with complex loop structure



Group Commentary

- Positives:
  - Tested 3 α values to consider the latency/area tradeoff
  - Considered some loop dependencies (MIT considered none)
  - Using Aladdin over full synthesis for computational efficiency
- Limitations:
  - Model may be too simplistic to understand complex looping structures
    - Evidenced by poor performance on jpeg-huff & stencil-stencil
  - Their 80-20% train/test split is random across all 1000 iterations
  - Feature extraction may be too simplistic
    - “Has Loop Carried Dependencies” feature - does not consider dependency distance
  - Regression to predict impact scores may be more suitable than classification
    - The LUF that maximizes predicted impact score should be selected
    - In classification, if ground truth LUF = 6, a predicted LUF of 1 and 5 would be considered equally wrong

Thank you!

Q&A