Assignment – 3

{Computer Architecture}

{EG-211}

Assignment-3

➔ Submitted by: Vidhish Trivedi (IMT2021055), Sankalp Kothari (IMT2021028) and Siddharth Kothari (IMT2021019)
➔ Submitted to: Prof. Nanditha Rao
➔ Course: Computer Architecture (EG211)

INSTRUCTIONS TO RUN THE CODE:

- Install python libraries (pandas), and install jupyter extension for VScode.
- Run the code file ("IMT2021055_IMT2021028_IMT2021019_Code.py").
- Enter choice 1 for Always Taken,
  choice 2 for Always Not Taken,
  choice 3 for Dynamic predictor.
- If choice 3 is selected, the user is asked to input the number of index bits used to construct a prediction table (in range (2, 20) inclusive).
- Note: Pandas and jupyter extension for VScode are optional and are used only to plot graphs.
- To view graphs in VScode, open the code file, right click and select: "Run Current File in Interactive Window", then proceed as described above.
- IMT2021055_IMT2021028_IMT2021019_a.csv contains misprediction rates corresponding to static prediction (q2.a) and IMT2021055_IMT2021028_IMT2021019_b.csv contains misprediction rates corresponding to dynamic predictions (q2.b).
- Please consider the screenshots attached below.

```
import pandas

# Function to convert a decimal to 32-bit binary string.
def make_binary_32(n):
    n = int(n)
    b = "{0:b}".format(n)
    if len(b) < 32:
        b = "0"*(32 - len(b)) + b
    return(b)

# Read trace file.
with open('./file0.trace', 'r') as f:
    trace_input = f.read().split('\n')
```

```
# trace_address and trace_actual will be used to simulate branch instructions.
# trace_address contains the address of branch instructions in decimal.
# trace_actual contains "T" or "N", indicating if a branch was taken or not.
trace_address = []  # len = 856017
trace_actual = []  # len = 856017

for _ in trace_input:
    l = _.split(' ')
    a = l[0]
    if("T" in _):
        b = "T"
    else:
        b = "N"
    trace_address.append(a)
    trace_actual.append(b)
```

Explanation:

- Pandas library is imported for the graphing logic.
- The binary function converts a given decimal number into a 32 bit binary string.
- We then read the trace file into a list of lists., named trace_input. The lists contain pairs of addresses and whether the branches were taken or not at 0 and 1 index respectively.
- We then iterate over the trace_input and store the zeroth index of all the lists (the addresses) in trace_address list, the taken or not taken status in the trace_actual list.

So now, the trace_address contains the addresses and the trace_actual contains the outcomes of the branches.

ASSIGNMENT-3-Q2-A:

STATIC BRANCH PREDICTOR

HOW IT WAS IMPLEMENTED (ALGORITHM):

Always Taken Predictor:

➢We iterate through the trace file and calculate the total number of taken branches, subtracting this from the total number of branches we get the total number of mispredictions.

Always Not Taken Predictor:

➢ We iterate through the trace file and calculate the total number of taken branches, which gives us the number of mispredictions.

CODE:

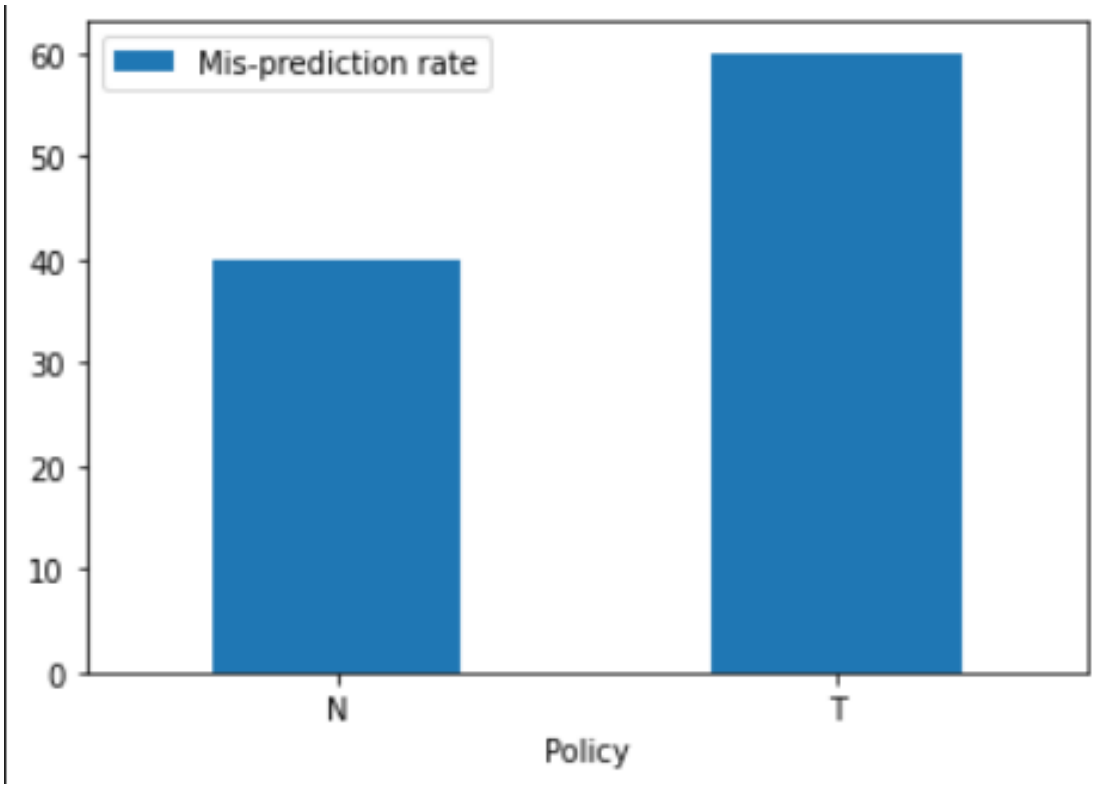
```
# Function to return count of branches which are taken.
# Count of branches not taken = total branches - count of branches taken.
def AlwaysT(tr_ac):
    c_t = 0
    for i in tr_ac:
        if(i == "T"):
            c_t += 1
    return(c_t)
################################################################################
################################################################################
chosen_policy = 0

while(chosen_policy != 1 and chosen_policy != 2 and chosen_policy != 3):
    print()
    print("Choose a prediction policy:\n\t1.) Always Taken Static Predictor\n\t2.) Always Not
Taken Static Predictor\n\t3.) Dynamic Prediction\n")
    chosen_policy = int(input("Enter choice number: "))
print()

if(chosen_policy == 1):  # For QUESTION-2(a).
    print("Using an always taken prediction policy:")
    print("Number of branches taken (correct predictions):", AlwaysT(trace_actual))
    print("Misprediction Rate (in percent):", (100)*((len(trace_actual) -
AlwaysT(trace_actual))/(len(trace_actual))))

elif(chosen_policy == 2):  # For QUESTION-2(a).
    # Count of branches not taken = total branches - count of branches taken.
    print("Using an always not taken prediction policy:")
    print("Number of branches not taken (correct predictions):", len(trace_actual) -
AlwaysT(trace_actual))
    print("Misprediction Rate (in percent):", (100)*((AlwaysT(trace_actual))/(len(trace_actual))))
```

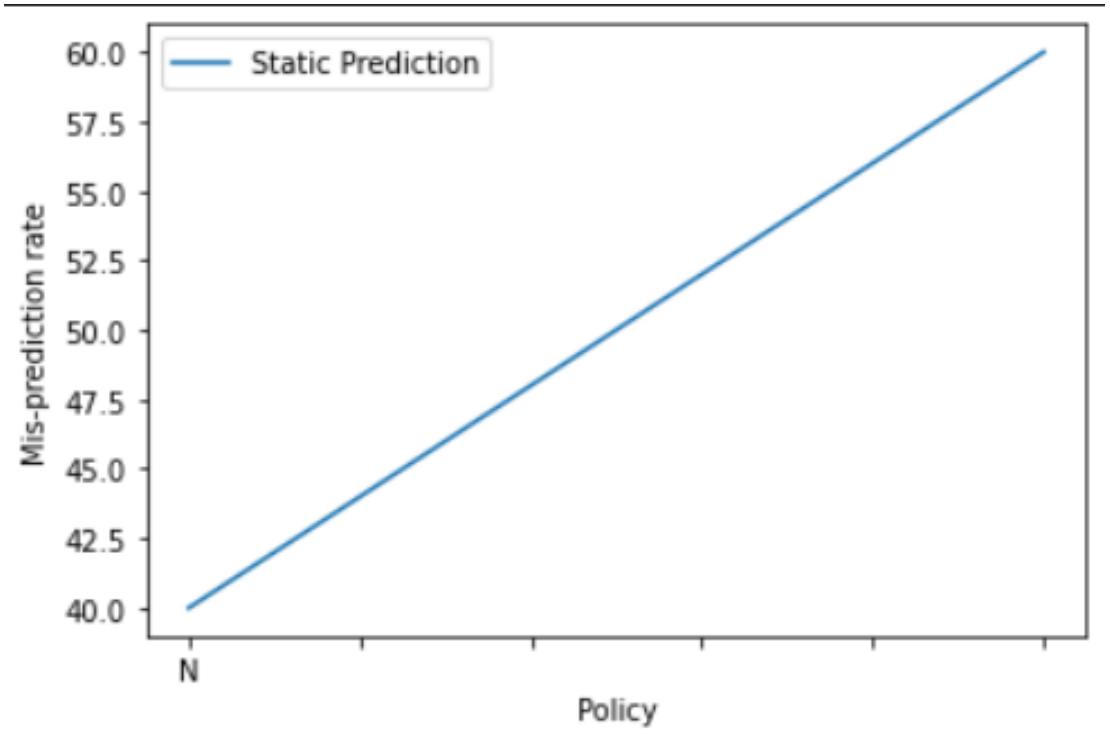
SCREENSHOTS OF WORKING:

```
Choose a prediction policy:
        1.) Always Taken Static Predictor
        2.) Always Not Taken Static Predictor
        3.) Dynamic Prediction

Enter choice number: 1

Using an always taken prediction policy:
Number of branches taken (correct predictions): 342306
Misprediction Rate (in percent): 60.011775467076

  Policy  Mis-prediction rate
0      N            39.988225
1      T            60.011775
```

```
Choose a prediction policy:
        1.) Always Taken Static Predictor
        2.) Always Not Taken Static Predictor
        3.) Dynamic Prediction

Enter choice number: 2

Using an always not taken prediction policy:
Number of branches not taken (correct predictions): 513711
Misprediction Rate (in percent): 39.988224532923994

  Policy  Mis-prediction rate
0      N            39.988225
1      T            60.011775
```

GRAPHS:





EXCEL TABLE CONTAINING VALUES:

	A	B	C
1	Policy	Mis-prediction rate	
2	N	39.98822453	
3	T	60.01177547	
4			

WHICH IS MORE ACCURATE?:

➔ For the given trace file of branch instructions, the “Always Not Taken” predictor is more accurate (39.988 %) than the “Always Taken” (60.011 %). This is subject to change, depending upon the trace file used.

ASSIGNMENT-3-Q2-B:

DYNAMIC BRANCH PREDICTOR

HOW IT WAS IMPLEMENTED (ALGORITHM):

For Dynamic Predictor:

- ➢ After choosing choice 3, the user is asked to enter the number of index bits to be used to construct the prediction table.
- ➢ We then create a prediction table (implemented as a dictionary), where the keys are index bits and corresponding values are the states which are used for predictions (initialized to 0, possible states are 0, 1, 2 and 3).
- ➢ We initialize a variable “correct_predictions” (set to 0) to keep track of the number of branches predicted correctly.
- ➢ Now, we iterate through the given trace file, convert each address to a 32-bit binary number and use string slicing to obtain the index bits.

- ➢ We then use the value associated with this key in the prediction table as the state to make a prediction, stored in a “curr_state” variable.
- ➢ There are 2 cases where predictions will be correct:
  - ○ When curr_state = 0 or 1, and the branch is taken (Prediction: taken when curr_state < 2).
  - ○ When curr_state = 2 or 3, and the branch is not taken. (Prediction: not taken when curr_state >= 2).
  - ○ In both these cases, we increment correct_predictions by 1.
- ➢ Also, we increment curr_state if curr_state != 3 and branch is not taken, and decrement curr_state if curr_state != 0 and branch is taken.
- ➢ We then store this updated value of curr_state as the value associated with the appropriate index bits (key) in the prediction table.
- ➢ Now, we calculate the mispredictions rate as:
  (Total branches - Correct Predictions)/Total)*100

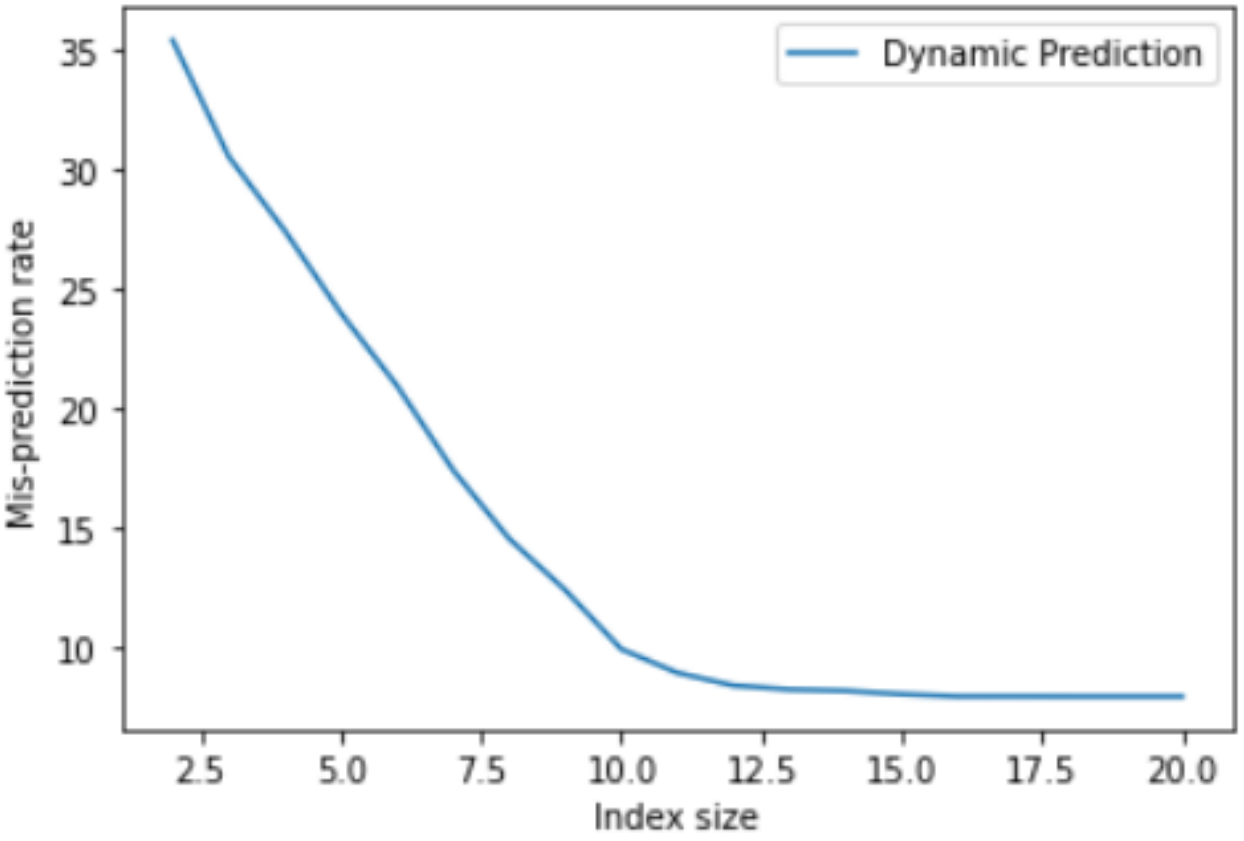
```
elif(chosen_policy == 3):  # For QUESTION-2(b).
    print("Using a dynamic prediction policy:")
    predictor_size = 21
    while(predictor_size < 2 or predictor_size > 20):
        predictor_size = int(input("Enter predictor index size (2 to 20): "))  # States would be
numbered (0, 3) --> (strongly taken, strongly not taken).
        # Convert instruction address in trace_dict from decimal to binary.
        # Size of predictor table is given by 2**(predictor_size) (2 raised to the power of
(predictor_size)).
    ins_binary = {}
    for i in range(2**predictor_size):
        ins_binary[((make_binary_32(i))[-1:-(predictor_size + 1):-1])[-1::-1]] = 0  # Initialised
according to question to strongly taken state.

    # ins_binary acts as prediction table with unique addresses (LSB index bits).

    # 0 --> 1 ==> taken states, 2 --> 3 ==> not taken states.

    # 0 ==> Strongly taken state.
    # 1 ==> Weakly taken state.
    # 2 ==> Weakly not taken state.
    # 3 ==> Strongly not taken state.

    correct_predictions = 0
    mid = 2  # states are: 0, 1, 2, 3.

    for i in range(len(trace_address)):
```

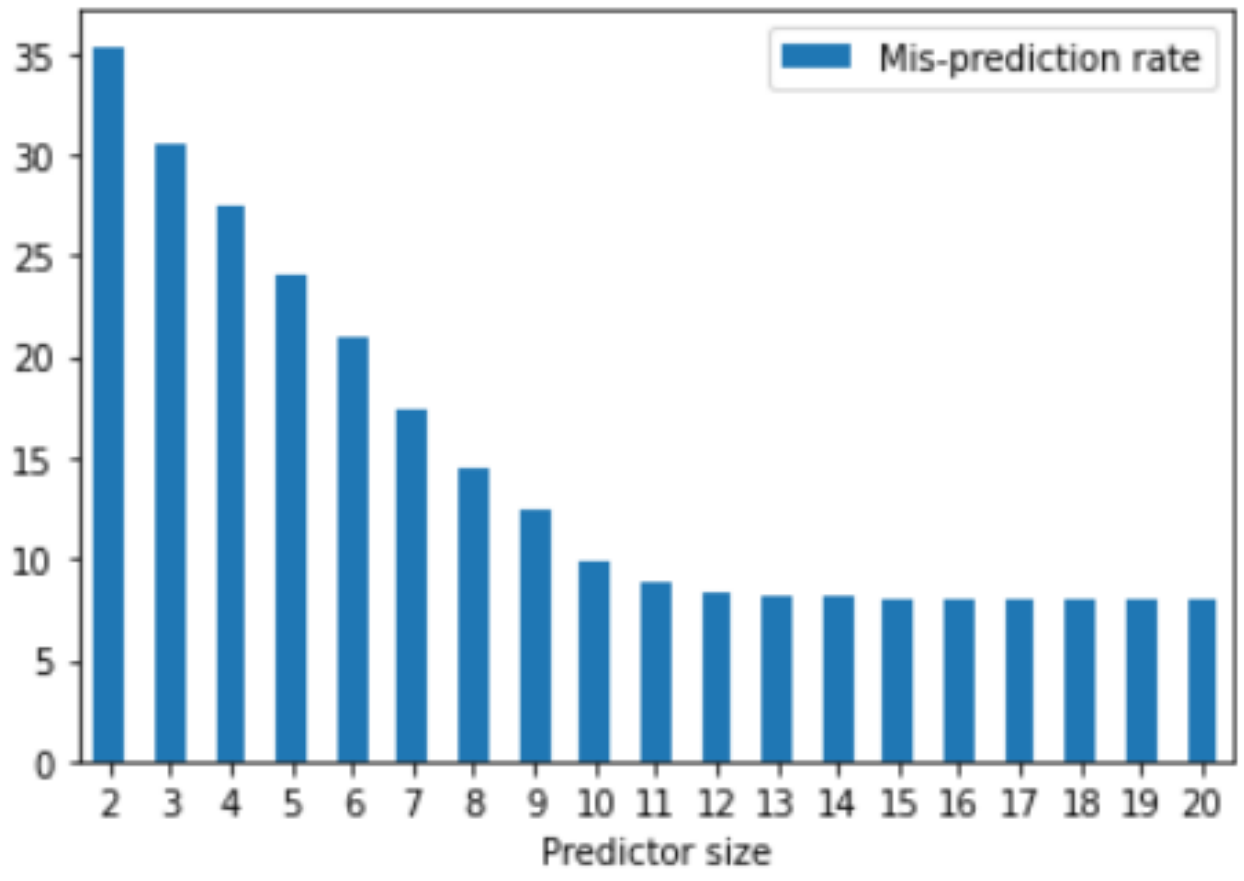
```
        curr_state = ins_binary[((make_binary_32(trace_address[i]))[-1:-(predictor_size + 1):-1])[-1::-1]]

        if(trace_actual[i] == "T"):
            if(0 <= curr_state and curr_state < mid):
                correct_predictions += 1
            if(curr_state != 0):
                curr_state -= 1

        else:  # trace_actual[i] == "N"
            if(mid <= curr_state and curr_state <= 3):
                correct_predictions += 1
            if(curr_state != 3):
                curr_state += 1

        ins_binary[((make_binary_32(trace_address[i]))[-1:-(predictor_size + 1):-1])[-1::-1]] = curr_state

    print("Number of correct predictions:", correct_predictions)
    print("Misprediction Rate (in percent):", (100)*((len(trace_actual) - correct_predictions)/(len(trace_actual))))
```

SCREENSHOTS OF WORKING:

```
Choose a prediction policy:
        1.) Always Taken Static Predictor
        2.) Always Not Taken Static Predictor
        3.) Dynamic Prediction

Enter choice number: 3

Using a dynamic prediction policy:
Enter predictor index size (2 to 20): 3
Number of correct predictions: 594996
Misprediction Rate (in percent): 30.492501901247287

    Predictor size  Mis-prediction rate
0                2            35.351401
1                3            30.492502
2                4            27.394549
3                5            23.954431
4                6            20.923183
5                7            17.397318
6                8            14.528683
7                9            12.383282
8               10             9.913588
9               11             8.922136
10              12             8.392240
11              13             8.226355
12              14             8.169931
13              15             8.030565
14              16             7.941197
15              17             7.942132
16              18             7.938160
17              19             7.938160
18              20             7.938160
```

GRAPHS:


Dynamic Prediction
Mis-prediction rate
35
30
25
20
15
10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Index size


Mis-prediction rate
35
30
25
20
15
10
5
0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Predictor size

	A	B
1	Predictor size	Mis-prediction rate
2	2	35.35140073
3	3	30.4925019
4	4	27.39454941
5	5	23.95443081
6	6	20.9231826
7	7	17.39731804
8	8	14.52868343
9	9	12.38328211
10	10	9.913588165
11	11	8.922135892
12	12	8.392239874
13	13	8.226355318
14	14	8.169931205
15	15	8.030564814
16	16	7.941197429
17	17	7.94213199
18	18	7.938160107
19	19	7.938160107
20	20	7.938160107

BEST MIS-PREDICTION RATE:

➔ The best misprediction rate is 7.93816 %, which is observed for predictors with index sizes 18, 19 and 20 bits.

MISPREDICTION RATE - COMPARISON:

➔ Misprediction rate with always taken predictor: 60.011775 %.
➔ Misprediction rate with always not taken predictor: 39.988225 %.
➔ Minimum = 39.988225 %.
➔ As seen from the data, the closest to half is 20.923183 %, which is observed when the index size is 6 bits.
➔ Number of counters = 2^(number of bits used as index) = 2^(6) = 64.
➔ Size of predictor (in bits): (N + 2)*(2^N)
= (6 + 2)*(2^6)
= (8)*(64) = 512 bits.

MAXED OUT PERFORMANCE OF PREDICTOR:

➔ After the index size becomes 18 bits, misprediction rate does not change. It remains constant at 7.93816 %.

➔ Beyond that, there is no difference in the misprediction rates, and increasing the size the predictor won't make a difference.

GRAPHING LOGIC:

```
# GRAPHING LOGIC.
if(chosen_policy == 3):
    parr = pandas.read_csv("./b.csv")
    ax = parr.plot(x = "Predictor size", y = "Mis-prediction rate", label = "Dynamic Prediction",
ylabel = "Mis-prediction rate", xlabel = "Index size")
elif(chosen_policy == 1 or chosen_policy == 2):
    parr = pandas.read_csv("./a.csv")
    parr["Mis-prediction rate"] = [(100)*((AlwaysT(trace_actual))/(len(trace_actual))),
(100)*((len(trace_actual) - AlwaysT(trace_actual))/(len(trace_actual)))]
    ax = parr.plot(x = "Policy", y = "Mis-prediction rate", label = "Static Prediction", ylabel =
"Mis-prediction rate", xlabel = "Policy")

print()
print(parr)

ax

############################  BAR GRAPH  ################################
if(chosen_policy == 1 or chosen_policy == 2):
    df = pandas.DataFrame({"Policy": parr["Policy"], "Mis-prediction rate": parr["Mis-prediction
rate"]})
    ax = df.plot.bar(x = "Policy", y = "Mis-prediction rate", rot = 0)
elif(chosen_policy == 3):
    df = pandas.DataFrame({"Predictor size": parr["Predictor size"], "Mis-prediction rate":
parr["Mis-prediction rate"]})
    ax = df.plot.bar(x = "Predictor size", y = "Mis-prediction rate", rot = 0)

ax
print()
```

THANK YOU!