./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/600.perlbench_s-210B.champsimtrace.xz" > results/perlbench_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/calculix_2670B.trace.xz" > results/calculix_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core0.trace.xz" > results/cassandra0_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core1.trace.xz" > results/cassandra1_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core2.trace.xz" > results/cassandra2_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core3.trace.xz" > results/cassandra3_perceptron.txt
./bin/champsim_perceptron --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/libquantum_10M.trace.gz" > results/libquantum_perceptron.txt

./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/600.perlbench_s-210B.champsimtrace.xz" > results/perlbench_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/calculix_2670B.trace.xz" > results/calculix_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core0.trace.xz" > results/cassandra0_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core1.trace.xz" > results/cassandra1_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core2.trace.xz" > results/cassandra2_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core3.trace.xz" > results/cassandra3_ashant.txt
./bin/champsim_ashant --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/libquantum_10M.trace.gz" > results/libquantum_ashant.txt

./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/600.perlbench_s-210B.champsimtrace.xz" > results/perlbench_sandeep.txt

./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/calculix_2670B.trace.xz" > results/calculix_sandeep.txt
./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core0.trace.xz" > results/cassandra0_sandeep.txt
./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core1.trace.xz" > results/cassandra1_sandeep.txt
./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core2.trace.xz" > results/cassandra2_sandeep.txt
./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/cassandra_phase0_core3.trace.xz" > results/cassandra3_sandeep.txt
./bin/champsim_sandeep --warmup_instructions 50000 --simulation_instructions 1000000 "traces/ChampSim Traces/libquantum_10M.trace.gz" > results/libquantum_sandeep.txt

#ifndef HYBRID_PREDICTOR_H #define HYBRID_PREDICTOR_H #include <vector> #include <cmath> #include <cstdint> #include <algorithm> #define HISTORY_LENGTH 32 #define WEIGHT_WIDTH 8 #define MAX_WEIGHT ((1 << (WEIGHT_WIDTH - 1)) - 1) #define MIN_WEIGHT (-(1 << (WEIGHT_WIDTH - 1))) #define THRESHOLD_MIN 5 #define THRESHOLD_MAX 40 #define CONFIDENCE_LIMIT 3 class HYBRID_PREDICTOR { public: HYBRID_PREDICTOR(); bool predict(uint64_t ip); void update(uint64_t ip, bool taken); private: int8_t perceptron[1 << 10][HISTORY_LENGTH + 1]; // Table of perceptrons int threshold; int confidence_counter; std::vector<int8_t> history; int compute_output(uint64_t ip); void decay_weights(uint64_t ip); uint64_t fold_history(uint64_t ip); }; #endif

ashant@Ashant:~/ChampSim/branch/ashant$ cat ashant.h
// branch/ashant/ashant.h
#pragma once

#include <vector>
#include <cstdint>
#include <functional>

constexpr size_t GHIST_LENGTH = 32;
constexpr size_t PERCEPTRON_TABLE_SIZE = 1024;
constexpr int WEIGHT_MAX = 127;
constexpr int WEIGHT_MIN = -128;

```
// Perceptron model with bias and weights
struct Perceptron {
    int bias;
    std::vector<int> weights;

    Perceptron() : bias(0), weights(GHIST_LENGTH, 0) {}
};

class BIAS_BOOSTED_PERCEPTRON {
public:
    BIAS_BOOSTED_PERCEPTRON();
    bool predict(uint64_t ip);
    void update(uint64_t ip, bool taken, bool predicted);

private:
    size_t index(uint64_t ip) const;
    int compute_output(const Perceptron& perceptron) const;

    std::vector<int> history;
    std::vector<Perceptron> table;
    std::hash<uint64_t> hasher;
};

ashant@Ashant:~/ChampSim/branch/ashant$ cat ashant.cc
// branch/ashant/ashant.cc
#include "ashant.h"

BIAS_BOOSTED_PERCEPTRON::BIAS_BOOSTED_PERCEPTRON()
    : history(GHIST_LENGTH, 0), table(PERCEPTRON_TABLE_SIZE) {}

size_t BIAS_BOOSTED_PERCEPTRON::index(uint64_t ip) const {
    return hasher(ip) % PERCEPTRON_TABLE_SIZE;
}

int BIAS_BOOSTED_PERCEPTRON::compute_output(const Perceptron& perceptron) const {
    int output = perceptron.bias;
    for (size_t i = 0; i < GHIST_LENGTH; ++i) {
        output += history[i] ? perceptron.weights[i] : -perceptron.weights[i];
    }
    return output;
}

bool BIAS_BOOSTED_PERCEPTRON::predict(uint64_t ip) {
```

```
    Perceptron& perceptron = table[index(ip)];
    int output = compute_output(perceptron);
    return output >= 0;
}

void BIAS_BOOSTED_PERCEPTRON::update(uint64_t ip, bool taken, bool predicted) {
    Perceptron& perceptron = table[index(ip)];
    int output = compute_output(perceptron);

    // Confidence threshold
    int theta = static_cast<int>(1.93 * GHIST_LENGTH + 14);

    // Update only if mispredicted or low confidence
    if ((predicted != taken) || (std::abs(output) <= theta)) {
        int direction = taken ? 1 : -1;

        // Update bias
        perceptron.bias += direction;
        if (perceptron.bias > WEIGHT_MAX) perceptron.bias = WEIGHT_MAX;
        if (perceptron.bias < WEIGHT_MIN) perceptron.bias = WEIGHT_MIN;

        // Update weights
        for (size_t i = 0; i < GHIST_LENGTH; ++i) {
            int input = history[i] ? 1 : -1;
            perceptron.weights[i] += direction * input;
            if (perceptron.weights[i] > WEIGHT_MAX) perceptron.weights[i] = WEIGHT_MAX;
            if (perceptron.weights[i] < WEIGHT_MIN) perceptron.weights[i] = WEIGHT_MIN;
        }
    }

    // Update history
    history.pop_back();
    history.insert(history.begin(), taken ? 1 : 0);
}

ashant@Ashant:~/ChampSim/branch/ashant$



ashant@Ashant:~/ChampSim/branch/ashant$ cat ashant.h // branch/ashant/ashant.h #pragma
once

#include #include #include
```

constexpr size_t GHIST_LENGTH = 32; constexpr size_t PERCEPTRON_TABLE_SIZE = 1024; constexpr int WEIGHT_MAX = 127; constexpr int WEIGHT_MIN = -128;

// Perceptron model with bias and weights struct Perceptron { int bias; std::vector weights;

```
Unset
Perceptron() : bias(0), weights(GHIST_LENGTH, 0) {}
```

};

class BIAS_BOOSTED_PERCEPTRON { public: BIAS_BOOSTED_PERCEPTRON(); bool predict(uint64_t ip); void update(uint64_t ip, bool taken, bool predicted);

private: size_t index(uint64_t ip) const; int compute_output(const Perceptron& perceptron) const;

```
Unset
std::vector<int> history;
std::vector<Perceptron> table;
std::hash<uint64_t> hasher;
```

};

ashant@Ashant:~/ChampSim/branch/ashant$ cat ashant.cc // branch/ashant/ashant.cc #include "ashant.h"

BIAS_BOOSTED_PERCEPTRON::BIAS_BOOSTED_PERCEPTRON() : history(GHIST_LENGTH, 0), table(PERCEPTRON_TABLE_SIZE) {}

size_t BIAS_BOOSTED_PERCEPTRON::index(uint64_t ip) const { return hasher(ip) % PERCEPTRON_TABLE_SIZE; }

int BIAS_BOOSTED_PERCEPTRON::compute_output(const Perceptron& perceptron) const { int output = perceptron.bias; for (size_t i = 0; i < GHIST_LENGTH; ++i) { output += history[i] ? perceptron.weights[i] : -perceptron.weights[i]; } return output; }

bool BIAS_BOOSTED_PERCEPTRON::predict(uint64_t ip) { Perceptron& perceptron = table[index(ip)]; int output = compute_output(perceptron); return output >= 0; }

void BIAS_BOOSTED_PERCEPTRON::update(uint64_t ip, bool taken, bool predicted) { Perceptron& perceptron = table[index(ip)]; int output = compute_output(perceptron);

Unset

```
// Confidence threshold
int theta = static_cast<int>(1.93 * GHIST_LENGTH + 14);

// Update only if mispredicted or low confidence
if ((predicted != taken) || (std::abs(output) <= theta)) {
    int direction = taken ? 1 : -1;

    // Update bias
    perceptron.bias += direction;
    if (perceptron.bias > WEIGHT_MAX) perceptron.bias =
WEIGHT_MAX;
    if (perceptron.bias < WEIGHT_MIN) perceptron.bias =
WEIGHT_MIN;

    // Update weights
    for (size_t i = 0; i < GHIST_LENGTH; ++i) {
        int input = history[i] ? 1 : -1;
        perceptron.weights[i] += direction * input;
        if (perceptron.weights[i] > WEIGHT_MAX)
perceptron.weights[i] = WEIGHT_MAX;
        if (perceptron.weights[i] < WEIGHT_MIN)
perceptron.weights[i] = WEIGHT_MIN;
    }
}

// Update history
history.pop_back();
history.insert(history.begin(), taken ? 1 : 0);
```

}

ashant@Ashant:~/ChampSim/branch/ashant$ update both the files

bin/champsim --warmup_instructions 200000000 --simulation_instructions 500000000 "traces/ChampSim Traces/600.perlbench_s-210B.champsimtrace.xz"

```
bin/champsim --warmup_instructions 200000000 --simulation_instructions 500000000
"traces/ChampSim Traces/600.perlbench_s-210B.champsimtrace.xz" > output.txt
```