

## Introduction

- Power is a critical issue in multicore architectures.
- However, aggressively reducing power consumption could deteriorate an application's performance to an unacceptable level.
- Forecaster** uses a hardware neural network that periodically collects hardware events to predict how much of different hardware structures should be used and reconfigures them accordingly.
- A prediction is accurate if the efficiency is improved, Otherwise, it is considered inaccurate, and the neural network is retrained.

## Motivation

- Hardware resources are not always fully utilized (Fig. 1).
- Programs tend to change their behavior during the execution (Fig. 2).

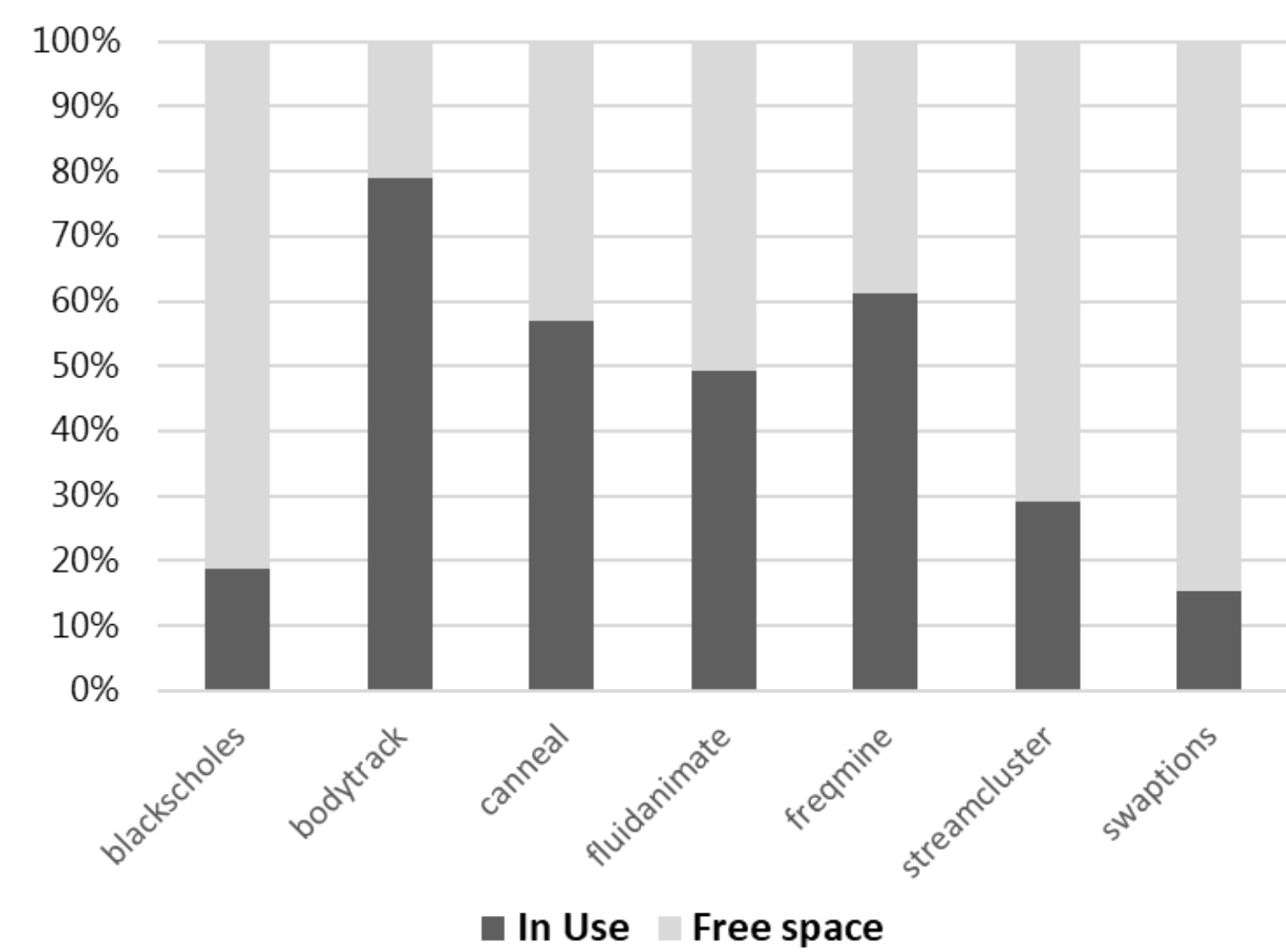


Figure 1: Average shared cache usage among 7 PARSEC Benchmarks

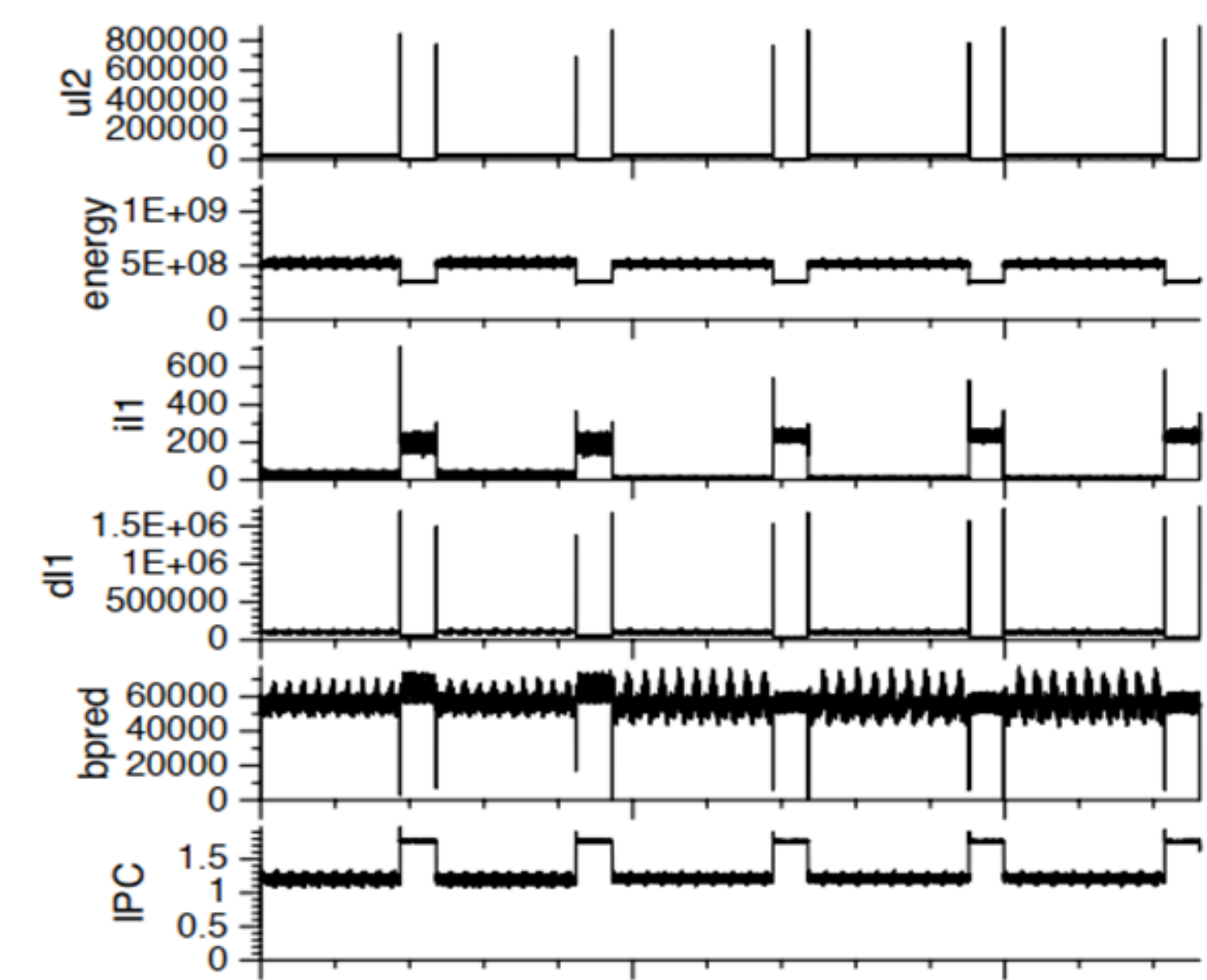


Figure 2: GZIP resource usage during execution [1]

### Main Idea:

- If we can learn the program pattern and dynamically turn off some unnecessary components, then we will get **better energy efficiency**.

## How It Works

### Goal:

- Improve the performance per Watt:  $IPC^3/W$

### Reconfigurable Components:

Components	Value Range	Unit
L2 (Private) Cache	30, 40, 50, 60, 70, 80, 90, 100	Percent
L3 (Shared) Cache	20, 30, 40, 50, 60, 70, 80, 90, 100	Percent

### Forecaster Workflow

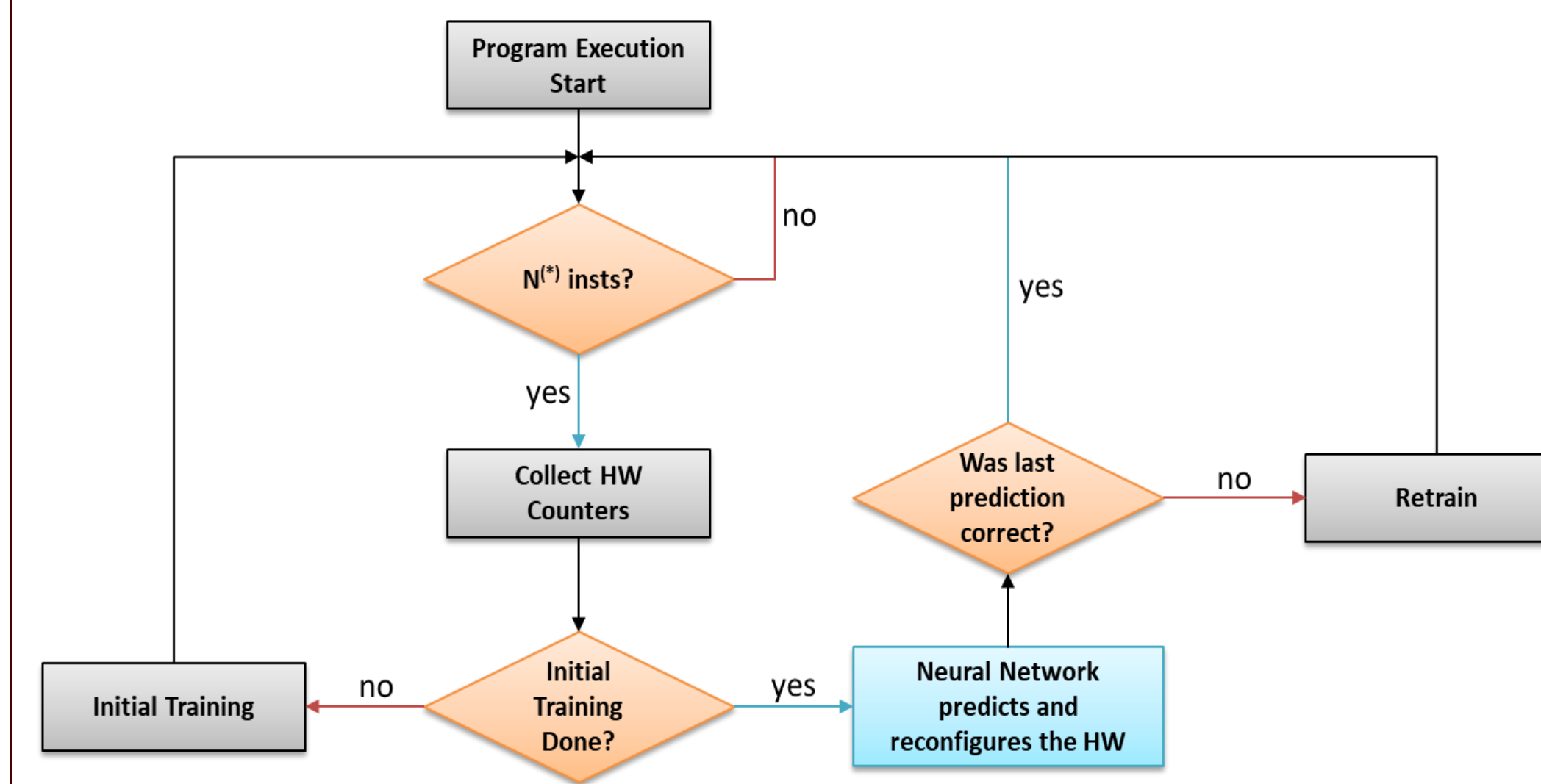


Figure 3: The workflow

(\*) N = interval size (0.2M)

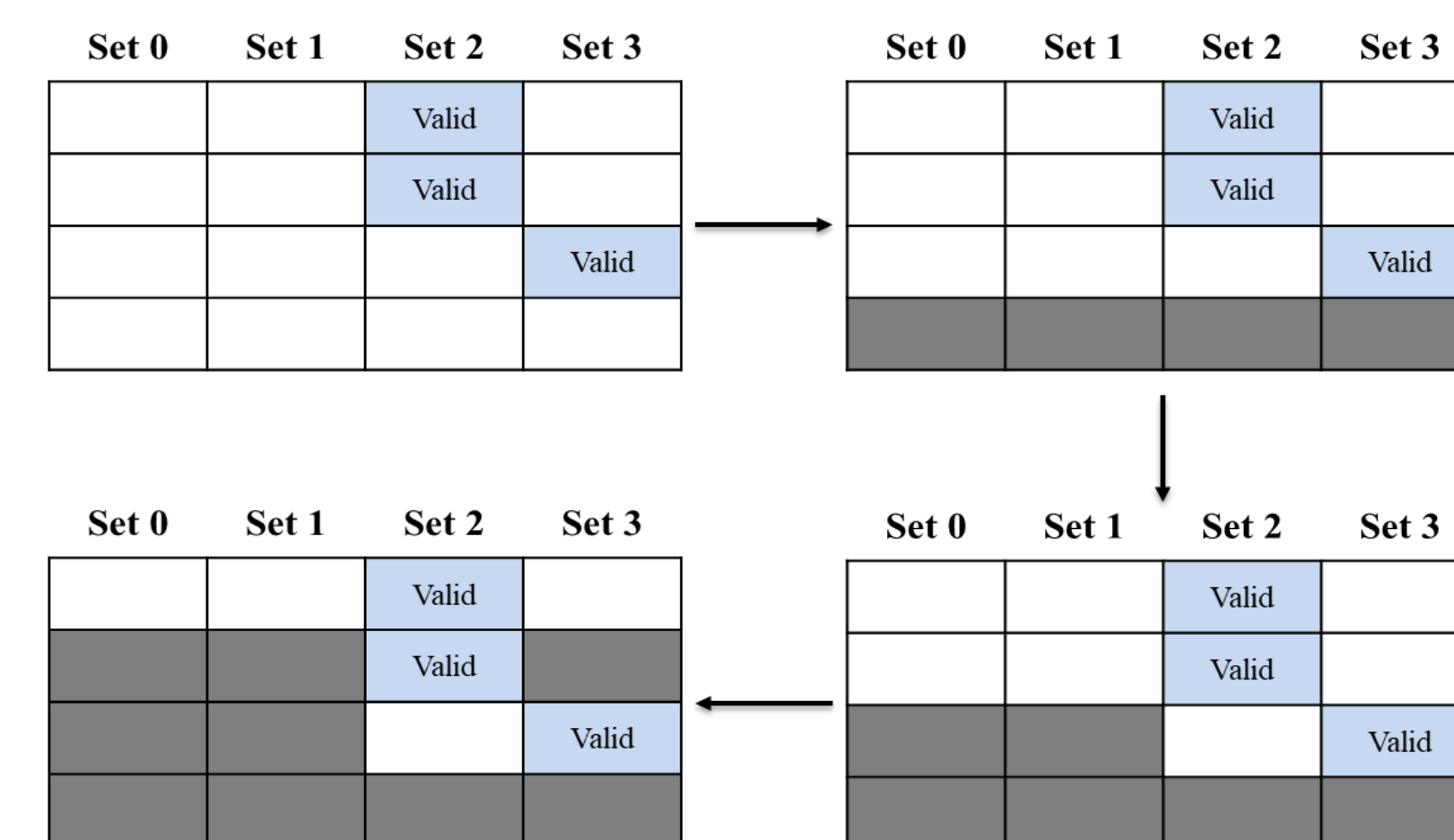
### How big should the interval size be?

- Too big: hardware tuning opportunities missed.
- Too small: too many reconfigurations, more overhead

### Initial Training

- The first  $i$  ( $i=80$ ) intervals of the execution. No prediction.
- The purpose is to collect unbiased preliminary training data.
- Prevent the untrained network to make uneducated guesses and damage the system performance.
- The hardware is reconfigured following the exact rules that we use to train the neural network.

### How to turn off the cache?



- Always leave at least one block per set (to avoid any complication with the cache controller).
- Disabled blocks are skipped in the cache replacement list.

## The Neural Network Predictor

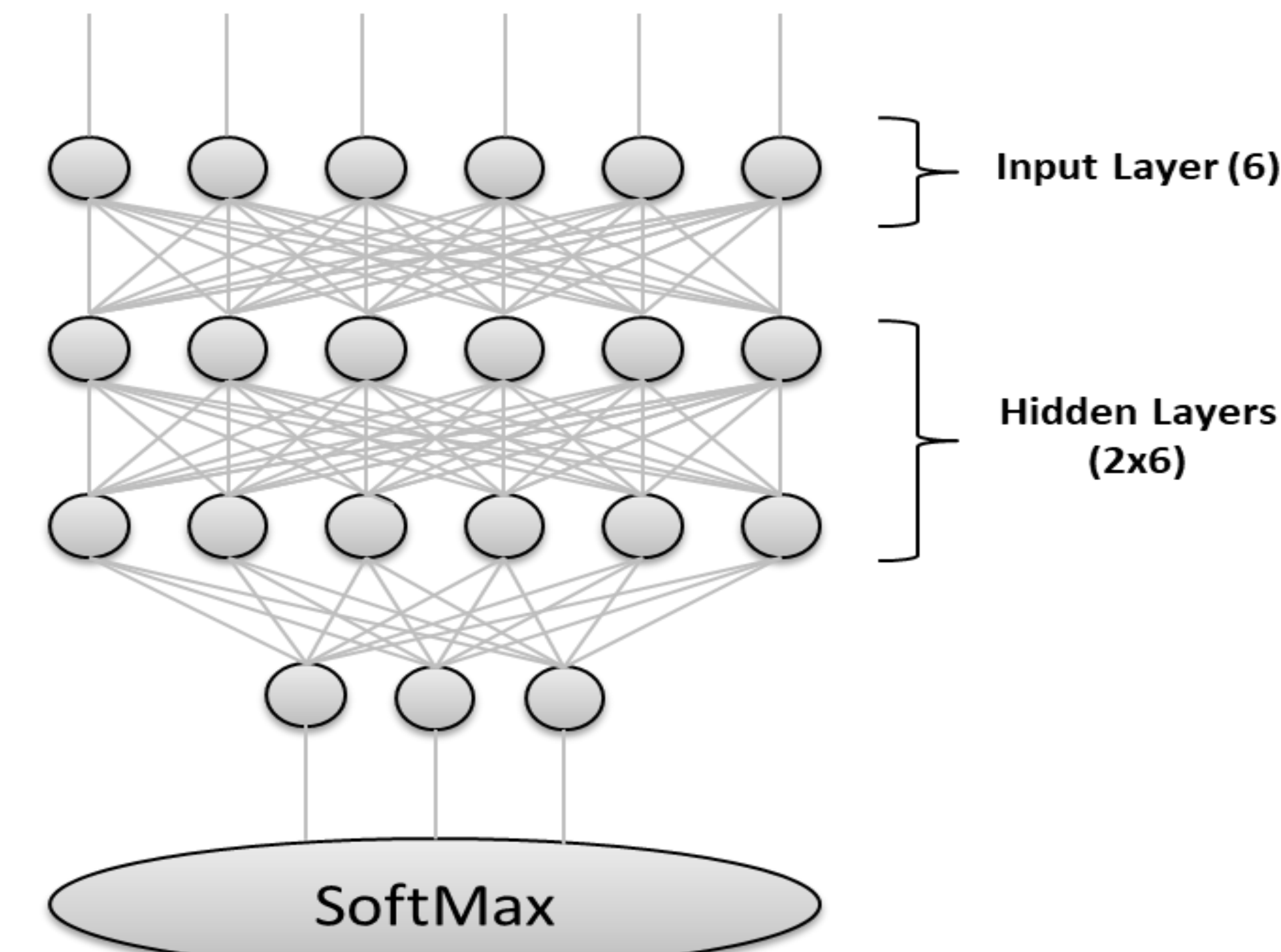


Figure 4: The Model

- Fully connected neural network.
- One model for each component.

### Input:

- Normalized # of floating-point instructions
- Normalized # of integer instructions
- Normalized # of memory instructions
- Normalized # of control instructions
- Normalized # of logic instructions
- Amount of free space of the cache (%)

### Output:

- Increase (+10%)
- Decrease (-10%)
- Maintain (=)

## Experimental Results

### Experimental Setup:

Parameter	4-core 2MB L3	i7-4930MX [2]	i9-9980HK [3]
CPU	4-core @ 3.0Ghz	4-core @ 3.0Ghz	8-core @ 2.4Ghz
Private L1 Cache (I/D)	32KB, 64B line, 8-way	32KB, 64B line, 8-way	32KB, 64B line, 8-way
Private L2 Cache	256KB, 64B line, 8-way	256KB, 64B line, 8-way	512KB, 64B line, 4-way
Shared L3 Cache	2MB, 64B line, 16-way	8MB, 64B line, 16-way	16MB, 64B line, 16-way

[2] Based on the cache configuration of the Intel Core® i7-4930MX

[3] Based on the cache configuration of the Intel Core® i9-9980HK

### Average Power Savings:

- Combination of static and dynamic power.
- i7-4930MX: **4.16%**
- i9-9980HK: **5.49%**

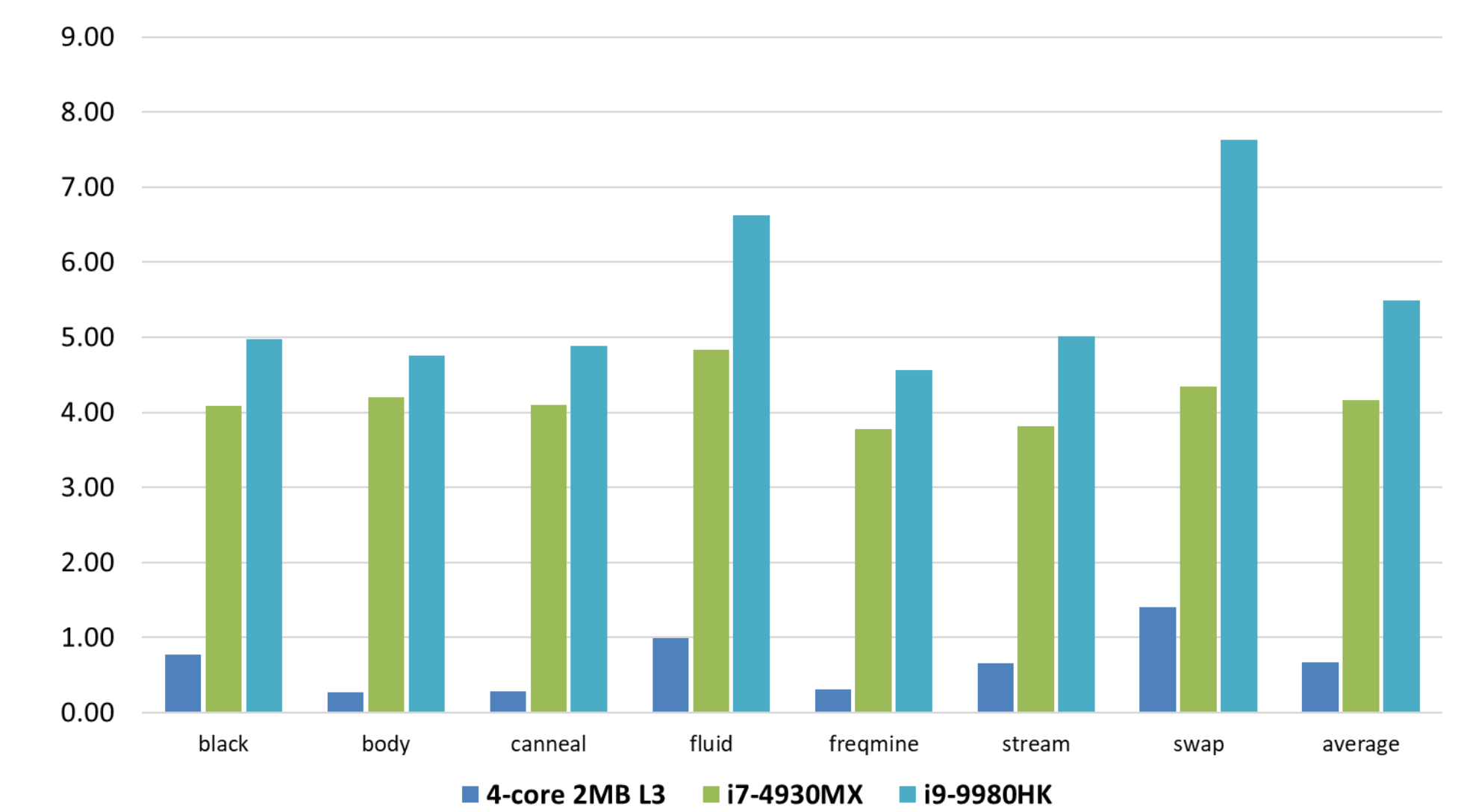


Figure 5: Power savings across 7 PARSEC Benchmarks (%)

### Average Efficiency Gains:

- i7-4930MX: **2.92%**
- i9-9980HK: **4.82%**

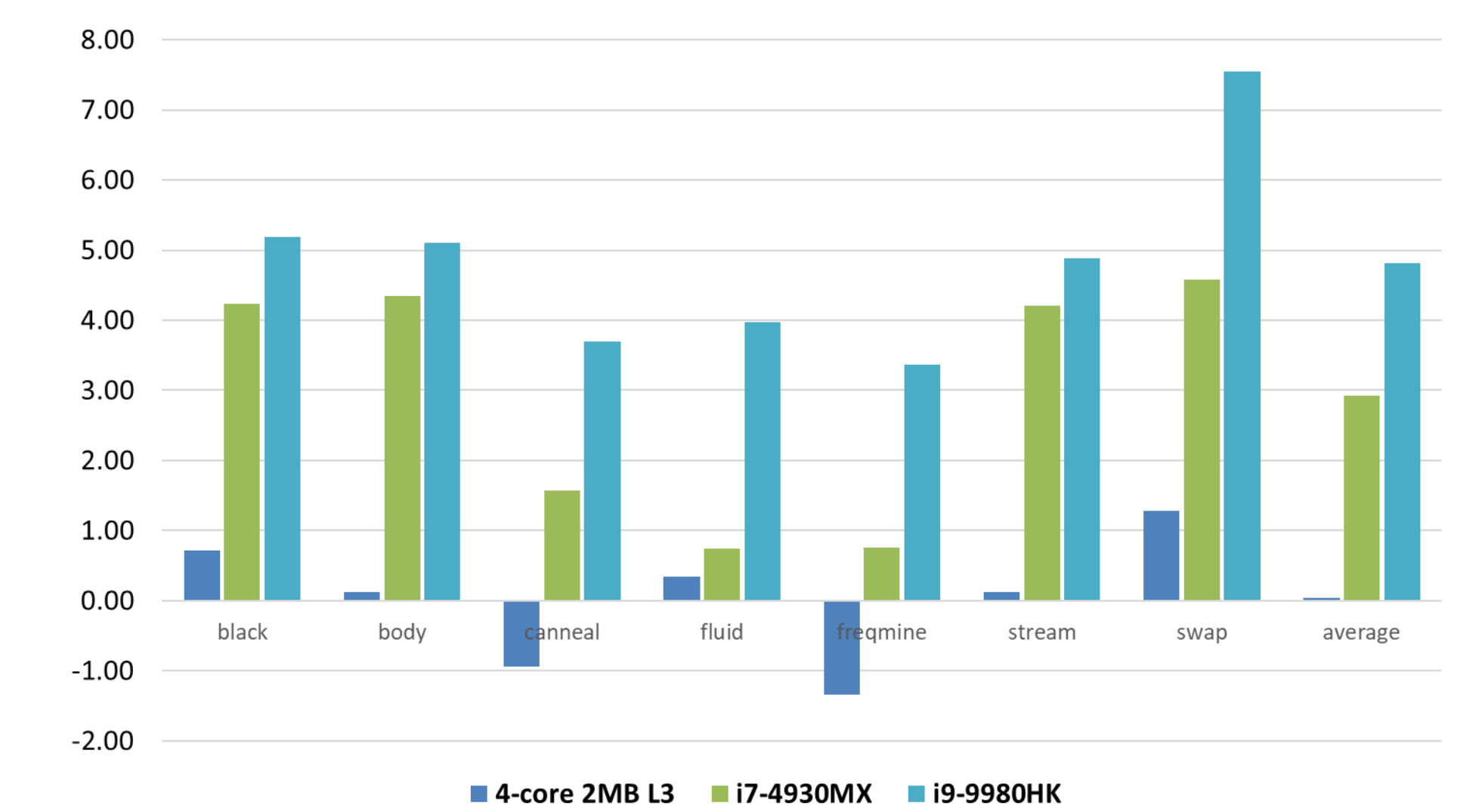


Figure 6: Efficiency gains across 7 PARSEC Benchmarks (%)

### Prediction Accuracy:

- L2 model = 97.31%
- L3 model = 95.40%

### System IPC Overheads:

- i7-4930MX: **-0.46%**
- i9-9980HK: **-0.32%**

## Future Work

- Adding more components:
  - Data Prefetcher
  - Branch Predictor
- Determining the hardware implementation of Forecaster.
- Computing realistic power consumption of Forecaster based on the hardware implementation.
- Implementing continuous learning with Deep Reinforcement Learning.
- Investigating the efficiency of Forecaster in multitasking environment.