Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks

Abstract & Motivation

- **Challenge in Branch Prediction:**
  - Conventional predictors achieve >99% accuracy on most branches.
  - A handful of **hard-to-predict branches** (H2Ps) cause significant IPC losses (up to 37.4% on scaled pipelines and 14.0% on Intel SkyLake).

Objectives of this paper:

(1) Map CNNs to the **global history data** used by existing branch predictors

(2) CNNs enabling us to **amortize offline training** and deploy **ML pattern matching** to improve IPC

(3) Adapt **2-bit CNN inference** to the constraints of current branch prediction units

(4) Establish that **CNN helper predictors** are reusable across application executions on diff inputs

Impact of Variable Iteration Control Structures

Global History Variations:

- Branch predictors use sequences of **IPs and branch directions.**
- Loops with data-dependent iteration counts cause **positional variations** in the global history.

Challenges for Traditional Predictors:

- Methods like **TAGE-SC-L** and **perceptron predictors** depend on fixed positional correlations.

```c
int f(int k, int *uvec, int *vvec) {
  int val1 = 0;
  int val2 = 0;

  if (uvec[k] % 3 > 0) /*Data-Dependent Branch*/
    val1 += 1;

  for(int j = 0; j < (vvec[k]); j++)
    if (vvec[j] % 2 > 0) val2 += vvec[j];

  if (val1 > 0)          /*H2P-1*/
    return val2;
  return 0;
}
```

Example1 - A simple C function illustrate show common program structures cause systematic branch mispredictions.

- **Data-Dependent Branches**
- **Variable Loop Bound**
- **Impact on CPU Performance**

- Predictors perform best with **consistent patterns.**

FP-CNN Architecture(Full-precision)	TP-CNN Architecture (Ternary)
Input Encoding • Convert global branch history (⟨IP, direction⟩) into a **1-hot matrix** using a hash function. $((IP \ll 1) + Dir)\&(2^p - 1)$	**Low-Precision Training (Offline)** • Train with weight clipping, normalization, and quantization so that all weights/activations are are **2-bit (ternary) precision.**
Layer 1 – Convolution • **Two full-precision filters** (one for “not-taken” and one for “taken”) perform 1-wide convolutions over the 1-hot matrix, computing **inner-product scores** ($y = \Sigma w_i x_i + b$). Apply **tanh()**	**Precomputed Lookup Table** • Build a table mapping each ⟨IP, direction⟩ index to a 2-bit response via normalization and quantization.
Layer 2 – Linear Prediction • A fully-connected layer aggregates convolution outputs with positional weights, yielding a **single score**. Apply **sigmoid()**	**Early Convolution via Lookup** • As branches are fetched, retrieve their 2-bit responses from the table and store them in a **FIFO buffer** (performing a **1-wide convolution**).
Final Decision • Predict “**taken”** if the final score is **> 0.5**; otherwise, predict “not-taken.”	**Ternary Inner Product Computation** • For a hard-to-predict branch, compute a ternary inner product (using **popcount and subtraction)** between the FIFO buffer and the 2-bit Layer 2 weights.
Offline Training • Train offline with full-precision weights on runtime-collected data.	**Thresholding for Final Prediction** • Use inverse normalization to set a threshold; if the computed **score exceeds this threshold, predict “taken**,” otherwise “not-taken.”

CNN Global History Model – FP CNN

1.Input Encoding

Source Code\Machine Code	IP LSBs	Not Taken Index	Taken Index
5: `if (uvec[k] % 3 > 0)`			
`400585 test %eax,%eax` `400587 jle,400590`	... 0000111	14	15
8: `for(...;j < (vvec[k]);...)`			
`400627 cmp -0x4(%rbp), %eax` `40062a jg 400599`	... 0101010	84	85
9: `if (vvec[j] % 2 > 0)`			
`4005cb cmp -0x4(%rbp),%eax` `4005cd jle 4005e8`	... 1001101	154	155
11: `if (val1 > 0)`			
`400630 cmpl $0x0,-0xc(%rbp)` `400634 jle40063b`	... 0110100	104	105

Example Global History & Matrix Representation

History Position

...	196	197	198	199	200

IP, Direction

...	40062a T	4005cd NT	40062a T	4005cd NT	40062a NT

1-Hot Index

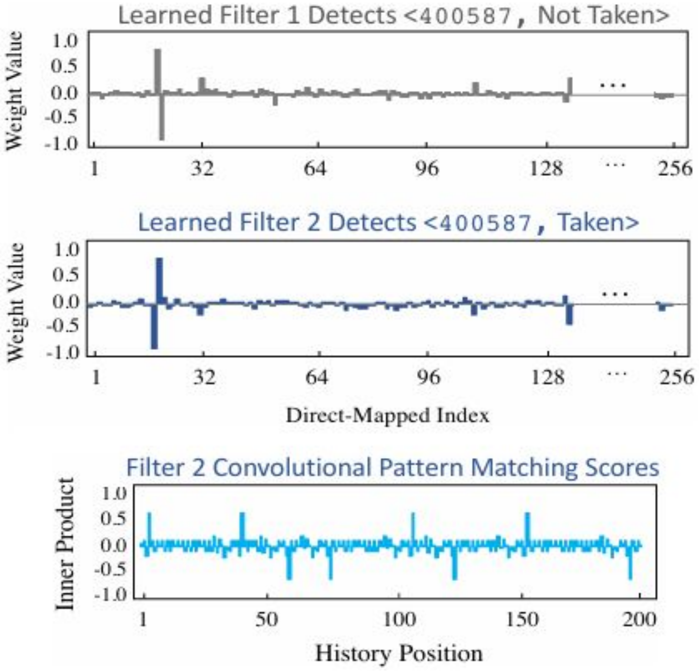
...	85	154	85	154	84

$$\mathbf{X} = \begin{array}{c} \\ \vdots \\ 84 \\ 85 \\ \vdots \\ 154 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccccc} \ldots & 196 & 197 & 198 & 199 & 200 \end{array} \\ \begin{pmatrix} & & & & \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ & & & & \\ 0 & 1 & 0 & 1 & 0 \\ & & & & \end{pmatrix} \end{array}$$

Fig. 2: A CNN is fed H2P-1's global history as a matrix of 1-hot columns with 1's in indices $((IP \ll 1) + Dir)\&(2^p - 1)$.

2. Inner-product scores

$$y = \sum_i w_i x_i + b.$$



TP CNN -On-BPU Inference with 2-Bit CNNs

1. Lookup Table Construction (Ternary CNN)

For m filters of length 2^p, denote the filter weights by W=[w1,w2,…,wm].

Each weight wij corresponds to index i ∈ [1, 2^p] and filter j ∈ [1, m] We have learned normalization parameters μ1j,σ1j,γ1j,β1j from the network's batch normalization layer, which transforms a raw v

$$\hat{y_j} = (y_j - \mu_{1j})(\gamma_{1j}/\sigma_{1j}) + \beta_{1j}$$

We define three quantization bins: [−1, −q],[−q, +q],[q, 1]

where q=0.8 by default (but may be learned). To populate the **2-bit** lookup table TT of size (2^p ×m), we assign:

$$\mathcal{T}[i,j] = \begin{cases} 01, & \text{if } w_{ij} < \frac{-\beta_{1j}}{\gamma_{1j}}\sigma_{1j} + \mu_{1j} - q \\ 11, & \text{if } w_{ij} > \frac{-\beta_{1j}}{\gamma_{1j}}\sigma_{1j} + \mu_{1j} + q \\ 00, & \text{otherwise.} \end{cases}$$

- This precomputation lets us replace on-chip multiplications with a **single lookup** for each ⟨IP, direction⟩ tuple, greatly reducing hardware complexity.

2.Ternary Inner Product

$$P = \text{popcount}(\neg(L1_S \wedge L2_S) \& (L1_V \& L2_V)) - \text{popcount}((L1_S \wedge L2_S) \& (L1_V \& L2_V))$$

where L1S and L1V are the sign and value bits of the FIFO buffer, respectively, and L2S and L2V contain those for the Layer 2 filter.

3.Threshold Final Prediction

$$\text{Pred} = \begin{cases} 1, & \text{if } P > t, \text{where } t = \frac{-\sigma_2}{\gamma_2}\beta_2 - \mu_2 \\ 0, & \text{otherwise} \end{cases}$$

1 - Taken
0 - Not taken

Results

FP CNN

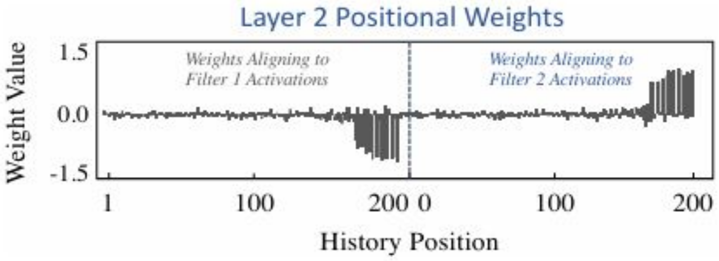


Fig. 4: Layer 2 filter weights represent how much each history position contributes to the final prediction.

TP CNN

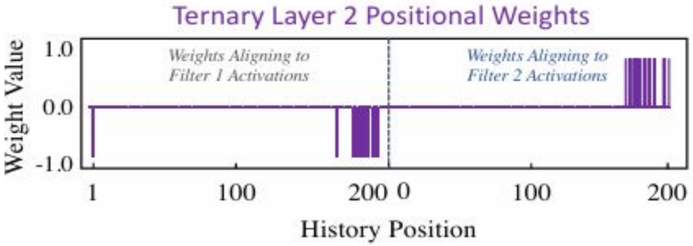


Fig. 5: 2-bit CNN helpers lose fidelity encoding the magnitude of each position's contribution to predictions, but accurately detect ⟨IP, direction ⟩ tuples despite positional variations.

SPECint2017 Benchmark	# Training Folds	# H2Ps (All Phases)	FP-CNN with TAGE 8KB Baseline		TP-CNN with TAGE 8KB Baseline		FP-CNN, Gains Beyond TAGE 64KB	
			% Winners	Mispred. Red. per H2P	% Winners	Mispred. Red. per H2P	% Winners	Mispred. Red. per H2P
600.perlbench_s	4	16	51%	63.2%	18%	26.6%	4%	8.2%
605.mcf_s	8	20	55%	44.8%	28%	27.9%	35%	19.3%
620.omnetpp_s	5	28	71%	33.6%	30%	16.3%	24%	11.2%
623.xalancbmk_s	4	8	39%	27.4%	0%	0.0%	23%	12.8%
625.x264_s	14	7	44%	16.8%	35%	12.0%	33%	12.2%
631.deepsjeng_s	12	49	56%	31.2%	24%	10.0%	12%	15.3%
641.leela_s	10	68	68%	40.7%	44%	15.3%	41%	19.7%
645.exchange2_s	5	19	9%	46.5%	4%	6.0%	0%	0.0%
657.xz_s	5	50	28%	25.2%	29%	15.4%	15%	12.3%
MEAN	7.3	29	47%	36.6%	24%	14.4%	21%	12.3%

TABLE I: CNN Helpers reusably improve accuracy for a large portion of H2Ps. Gains for 21% of H2Ps are beyond the capabilities of TAGE-SC-L when scaled by 8x.

Improved Idea

Instead of just (IP << 1 + dir) & pow(2,p)-1, expand to: (p = 8 bits)

- PC bits (7 bits)
- Opcode bits (e.g., conditional/jump type) (4 - 8 bits)
- Direction (1 bit)

Also the register values can be incorporated.

New_P = 12-16 bits

New_index = ((IP << 4 + Opcode) << 1 + direction) & pow(2,p)-1

Expected Results

- Increase in Winners%.
- Lower MPR on complex dependencies.

Implementation steps

Step 1: Extracted the **opcode** (instruction type) along with ip, direction

Step 2: Encoded into a history matrix using previously defined new_index

Step 3: Built a **FPCNN** with two layers

Step 4: Trained the model using 27 million instructions for 1 epoch (25 hours)

(SPEC 2017 perlbench(**600.perlbench s**)) trace file used

Step 5: Quantized the model weights and updated in **TPCNN**

Step 6: Predicted the direction using TPCNN

Results - Link

Metric	Value
Total Branches	27,772,333
Mispredictions	13,425,785
Misprediction Rate	48.35%
MPKI	483.5

Future Work- We have to train the model for 40 epochs so models trains and predicts correctly.