University of Central Florida

Department of Computer Science

CDA 5106: Spring 2025

Machine Problem 2: Branch Prediction

# Deep Learning and Knowledge Distillation-Based Branch Prediction

by

**Group 4**

Honor Pledge:

*"I have neither given nor received unauthorized aid on this test or assignment."*

|   |   |
|---|---|
| Student's electronic signature: | Nicholas Gray |
| Student's electronic signature: | Sara Belal |
| Student's electronic signature: | Esperandieu Elbon |
| Student's electronic signature: | Ricardo Barrios |

***Abstract*—Accurate branch prediction is pivotal for today's high performance CPUs, yet conventional table-based schemes such as GShare or TAGE possess complex designs difficult to iterate upon. Recent perceptron and deep-learning predictors, such as BranchNet, raise accuracy but at the expense of transparency and silicon cost, making them hard to study and deploy widely. We present a lightweight knowledge distillation framework that trains compact student networks to emulate a reference predictor's decisions without access to its internal weights. Using GShare as the teacher, three models were evaluated with and without distillation on 18 collected traces: they exceeded the teacher in 14 traces and distillation improved performance in some of the other cases, demonstrating that the addition of distillation can impart richer understanding than supervised learning alone. Our pipeline opens a path for reverse engineering proprietary predictors, stress-testing their vulnerabilities, and iteratively fine-tuning distilled models to advance branch prediction accuracy at minimal computational cost.**

## I. Introduction

Over the past two decades, the demand for computational speed has outpaced what micro-architectural advances alone can deliver, making predictive computing essential. An example of this is branch prediction, which attempts to predict the outcome of a branch before the result is known. By forecasting the outcome of a branch before its evaluation, processors gain substantial performance and energy benefits, albeit at the risk of heavy penalties when predictions fail.

Classic counter and table-based schemes such as Smith, GShare [8], and TAGE [10] already exceed 90–95% accuracy on many workloads. However, they plateau on harder patterns, prompting growing interest in machine-learning approaches. Starting with perceptron predictors [3], researchers have explored both online [11] and offline [7] learning strategies. Deep networks improve accuracy but face a size–latency trade-off: large models excel but are impractical in hardware, while small ones are more likely to underfit. In other domains, knowledge distillation [2] mitigates this gap by training compact models to imitate larger teachers.

This report frames branch prediction as a binary-classification problem, using GShare as a reference. We evaluate a feed-forward autoencoder and two convolutional neural networks (CNNs), and propose a custom distillation loss that guides the networks toward GShare-like behavior. Initial experiments saw our deep models surpass GShare on most benchmarks, with distillation boosting the rest. A second round improved raw accuracy further, though distillation occasionally underperformed compared to only supervised training.

The remainder of the report is organized as follows: Section 2 surveys traditional and ML-based predictors and knowledge distillation; Section 3 details model architecture and training; Section 4 describes implementation; Section 5 presents results and analysis; Section 6 concludes and outlines future work.

## II. Background

### A. Traditional Predictors

The earliest branch prediction algorithms, such as the Smith predictor, used counter based systems to predict branch outcomes. These algorithms were simple and poor (40% misprediction rate), and did not fully capture the branch prediction environment. This led to the development of table based predictors, such as the Bimodal predictor, which kept a table of counters that were accessed based on an index generated using the branch's program counter. This greatly improved performance, but lacked an understanding of previous prediction history. GShare [8] was designed to accommodate this, including a global history register to modify the accessing index, leading to a small performance improvement.

Traditional predictors greatly improved in performance with the release of TAGE [10], which extends a base predictor with a set of prediction tables using different branch history lengths. This predictor led to enormous performance improvements, and later iterations [9] are considered state-of-the-art and have been implemented in current microprocessors. However, TAGE is a complex architecture that is difficult to iterate upon, leading researchers to pursue other possible architectures for improving branch prediction.

### B. Machine Learning Predictors

An early use case of machine learning in branch prediction is the Perceptron predictor [3]. Utilizing a table of weights alongside a global history register to make a prediction from a weighted sum, this architecture was able to outperform GShare over 20 years ago, before the introduction of TAGE. Perceptrons are linear models, however, and have difficulty capturing many of the complex intricacies of branch prediction.

Therefore, researchers have turned to deep learning for branch prediction. Deep neural networks [5] capture intricate patterns but require large datasets for training. Online learning adapts weights after every branch but limits model size, so newer works adopt offline training on massive traces and freeze the model at run time. BranchNet [11] employed convolutions to rival TAGE, but its complexity limits usability. Studies of smaller networks [7] sacrifice some accuracy but improve hardware and iteration practicalities. All such models learn solely from outcomes, however, and it is an open question if performance can be improved further by internalizing the heuristics of traditional predictors.

### C. Knowledge Distillation

Knowledge distillation [2] is a technique that trains a compact student network to reproduce the behavior of a larger or otherwise inaccessible teacher model. Instead of relying only on ground truth labels, the student also learns from the teacher's soft targets – the predicted probability distribution the teacher assigns to each input. These soft targets convey richer information than ground truth labels alone: they encode the teacher's assessment of class similarities and decision boundaries, allowing the student to inherit much of the teacher's

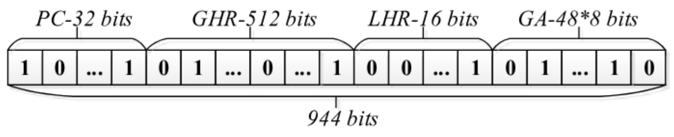

Fig. 1. Input vector format for deep learning models, adapted from [7].

representational power even when the teacher's architecture is hidden or unknown.

## III. Design

### A. Model Architecture

Three neural network architectures were implemented based off the models used in [7]: an autoencoder based feed forward network, a small convolutional neural network (CNN) based on LeNet [6], and a larger CNN based on AlexNet [4]. Autoencoders are a type of neural network architecture that compress the network into a smaller representation in the middle, before expanding back to the original input size. Convolutional neural networks use kernels to process a small section of the input, processing over the entire input in a sliding window fashion. All models' output layers were a single neuron wrapped in a sigmoid function for binary classification.

Each deep learning model had the same 944 history bit input, adapted from [7]. The model's input is a fixed 944 bit vector, organized as a 32-bit program counter, 16-bit local history register, 512-bit global history register, and the lowest 8 bits of the last 48 branch addresses. Figure 1 shows the organization of the bits for the models' inputs.

GShare was chosen as the teacher model due to existing implementation from the machine problem section of the final project. The GShare predictor keeps a table of $2^m$ three-bit saturating counters, initialized to 4, and an $n$-bit global history register, initialized to 0. For a branch at program counter $PC$, form the index by XOR-ing the lower $n$ bits of $PC$ with the global history and concatenating the result with the remaining $m - n$ bits of $PC$. The selected counter predicts taken if its value is $\geq 4$, otherwise not taken. After the actual outcome, the counter is incremented or decremented (clipped to the range $[0, 7]$), and the global history is updated by right-shifting and inserting the real outcome into the most significant bit.

### B. Training

Figure 2 shows the supervised learning and knowledge distillation pipeline used for training the deep learning models to perform branch prediction. For each model architecture, two models were trained – one trained only against the branch outcome in a supervised fashion, and the other trained on both the branch outcome and GShare's prediction. The latter is the standard procedure for knowledge distillation, as it has been shown keeping the supervised ground truth alongside the teacher model target allows the student model to better match the performance of the teacher model.

All models were trained using the binary cross entropy (BCE) loss for both supervised learning and knowledge distillation training. Binary cross entropy captures the probability

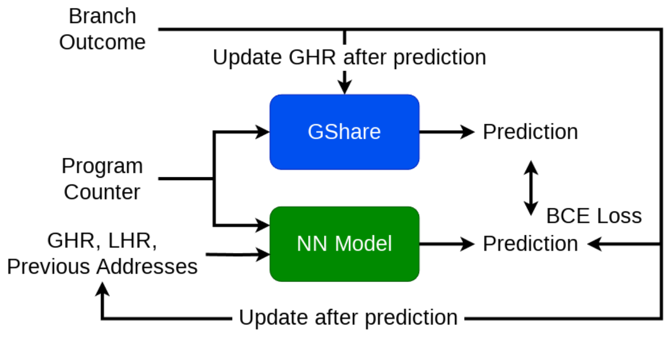

Fig. 2. Supervised learning and knowledge distillation training pipeline.

difference between a predicted binary probability $\hat{y}_i$ and a ground truth binary probability $y_i$, and gives harsher penalties the greater the binary probabilities differ. The equation for binary cross entropy can be found below:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} \Big[ y_i \log(\hat{y}_i) + (1 - y_i) \log\big(1 - \hat{y}_i\big) \Big]$$

Binary cross entropy is a popular loss function for supervised learning for binary classification, but knowledge distillation generally uses the Kullback-Leibler divergence to better capture the distribution differences between a student and teacher model. However, given the binary nature of branch prediction and GShare's direct taken/not-taken output, knowledge distillation is better represented as a binary classification problem.

### C. Dataset

14 traces were collected for this project – 3 given from the original machine problem (GCC, JPEG, Perl), and 11 found from a GitHub repository of unknown program origin [1]. All trace files were processed to be in the format "[hexadecimal program counter] t/n", where t/n indicates if the branch was taken or not. All traces contain at least 1.5 million branch instructions.

Two training sessions were designed for this project, with the results of the first experiment guiding the second. The first experiment was trained on a smaller training set made from unknown traces 1, 10, and 12. These traces each contained around 1.7 to 2 million branches each. After the results of the first experiment, a second training session was conducted with a larger training set made from unknown traces 2, 3, 6, and 10. This training set contained 3 traces with over 3 million branches each, representing a significantly larger training set. This session was performed to understand how training set size could affect model and distillation performance.

## IV. Implementation

GShare's implementation was carried over from the machine problem, and configuration parameters ($m = 12, n = 2$) were set based on a grid search over the average misprediction rate across the GCC, JPEG, and Perl traces. All deep learning

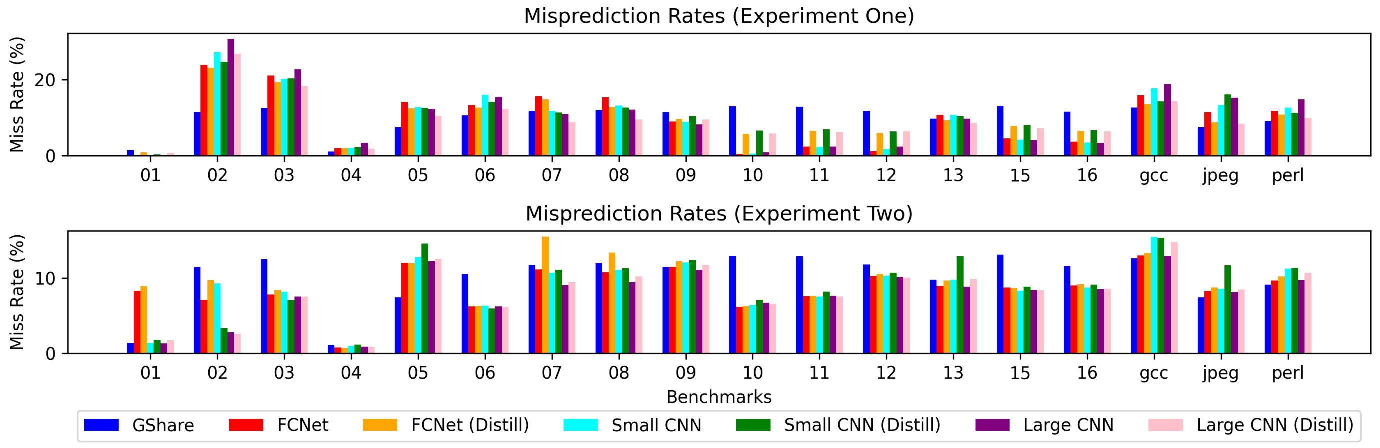

Fig. 3. Misprediction Rates of deep learning models (supervised and distilled) versus GShare Predictor. Experiment One models trained on traces 1, 10, and 12. Experiment Two models trained on traces 2, 3, 6, and 10.

models were implemented manually in Python using the PyTorch library. The traces were preprocessed into vector format before training to minimize training time overhead. The GShare prediction results were also pre-generated in order to minimize repeated processing.

All training was done on an RTX 3090 GPU and AMD Ryzen 9 5900X CPU, and training took less than 1 hour for all models collectively for each experiment. Each model was trained for five epochs with the Adam optimizer and an initial learning rate of $1 \times 10^{-4}$ with a $10\%$ step-off after the 4th epoch in the first experiment and after the 3rd epoch in the second experiment.

## V. Evaluation

All models were evaluated via misprediction rate across 14 collected traces – 3 traces given from the machine problem (GCC, JPEG, Perl), and 11 traces collected unknown from unknown origin. Each trace was fed to the models sequentially, and the misprediction rate is the proportion of branches the model predicted incorrectly by the total number of branches in each trace.

## VI. Results

Figure 3 shows the benchmark results for each experiment. comparing each model's supervised learning only vs. additional distillation training results, and comparing the models to the teacher GShare. A notable trend overall is model performance is correlated with model size, matching with the common understanding that larger model sizes can perform better at complex tasks.

Experiment One had at least one machine learning model achieved superior performance in 10 out of 18 trace benchmarks. For those traces, distillation led to worse performance compared to supervised learning, but still had superior performance compared to GShare. For the other traces where the models performed worse than GShare, distillation training improved performance. Both results indicate that knowledge distillation was able to guide the models to mimic GShare's behavior to some extent. Among GCC, JPEG, and Perl traces, the machine learning models were unable to beat GShare, but distillation brought the models to similar performance.

Experiment Two had at least one model have superior performance in 14 out of 18 benchmarks. Interestingly, the larger training set size caused knowledge distillation to either not affect training or lead to worse models, in some cases training the model to perform worse than GShare. This can be seen in GCC, JPEG, and Perl, where distilled models had inferior performance compared to their supervised counterparts, despite the supervised models performing worse than GShare themselves. However, it is notable that the larger training set size led to more consistent benchmarks across architectures, and performance better matched GShare on GCC, JPEG, and Perl.

## VII. Summary

In this report, we explore the capabilities of using deep learning models for branch prediction, and the viability of using knowledge distillation to improve capabilities by distilling capabilities of the high quality branch predictor GShare into deep neural networks. Our results showed that deep learning is able to have superior performance to GShare on 14 of 18 benchmarks, and knowledge distillation is able to improve model capabilities in some scenarios. Future improvements could focus on using superior branch predictors, such as TAGE or BranchNet, as teachers, testing model size limits, and investigating which input bits are most useful according to the models.

These results could allow for many future possible works, including reverse engineering of proprietary predictors, stress-testing vulnerabilities, and fine-tuning of distilled models to improve their branch prediction accuracy further.

## References

[1] Vittal Battula. Branch prediction programming. https://github.com/saivittalb/branch-prediction-programming, 2021.

[2] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[3] Daniel A Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206. IEEE, 2001.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] Yonghua Mao, Huiyang Zhou, Xiaolin Gui, and Junjie Shen. Exploring convolution neural network for branch prediction. *IEEE Access*, 8:152008–152016, 2020.

[8] Scott McFarling. Combining branch predictors. Technical report, Technical Report TN-36, Digital Western Research Laboratory, 1993.

[9] André Seznec. Tage-sc-l branch predictors. In *JILP-Championship Branch Prediction*, 2014.

[10] André Seznec and Pierre Michaud. A case for (partially) tagged geometric history length branch prediction. *The Journal of Instruction-Level Parallelism*, 8:23, 2006.

[11] Siavash Zangeneh, Stephen Pruett, Sangkug Lym, and Yale N Patt. Branchnet: A convolutional neural network to predict hard-to-predict branches. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 118–130. IEEE, 2020.