Using Machine Learning to Predict Branch Probabilities

Zachary Eichenberger, Shin Lee, Harvin Murmick, Shivan Prasad

Learning Branch Probabilities in Compiler from Datacenter Workloads

Easwaran Raman
Google
eraman@google.com

Xinliang David Li
Google
davidxl@google.com

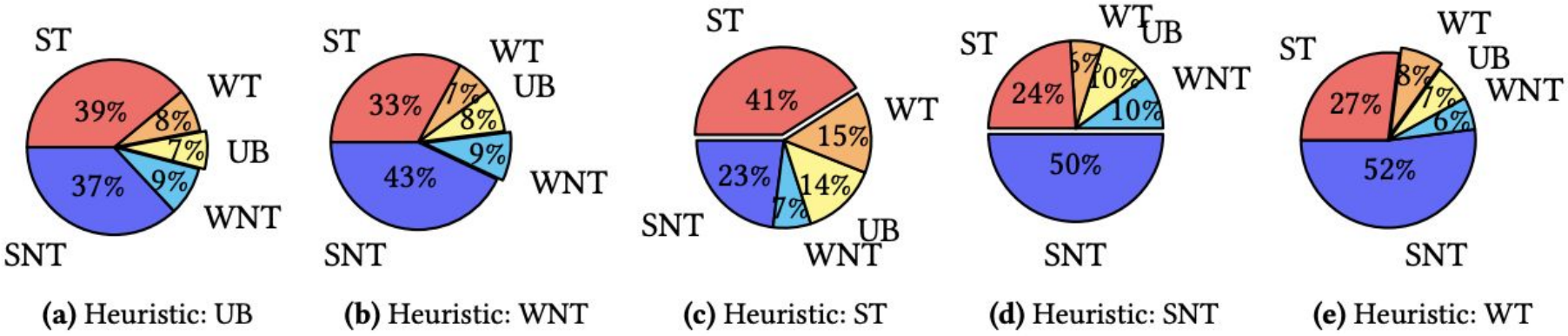
Branch Prediction Heuristics

- LLVM Compiler: Relies on branch probability analysis
  - E.g. to optimize code layout
  - Can be obtained via profiling – accurate, but can be difficult
- What happens when profiling data is not available?

Observation	Heuristic prediction
Explicit programmer labelling	That label
Edge is loopback	Strongly Taken
Equality comparisons (floats & ptrs)	Weakly not taken
Otherwise	Unbiased

Heuristic Limitations

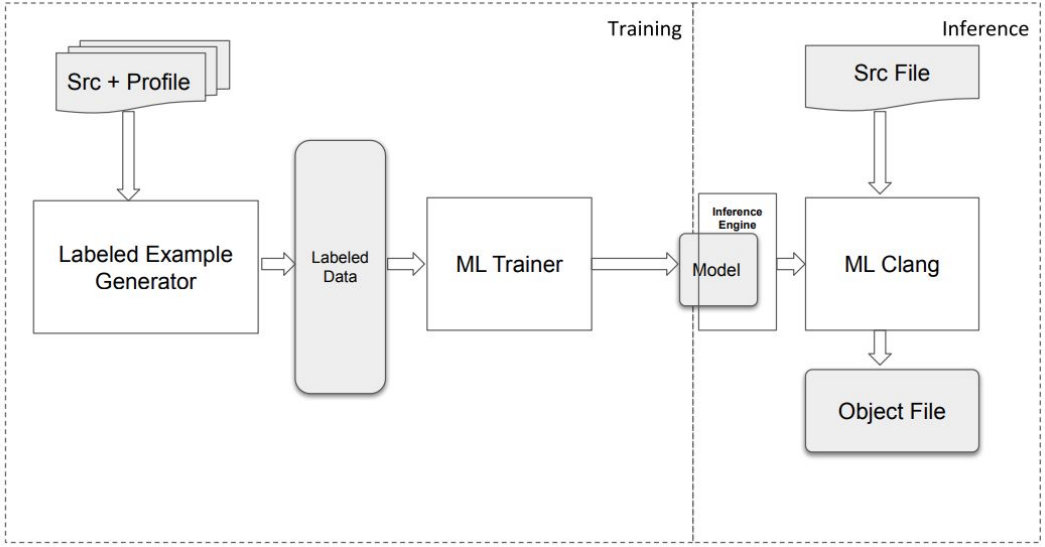
- Simple → Conservative Predictions
- Reality
  - Heuristic accuracy is poor
  - Most branches are strongly taken/not taken



(a) Heuristic: UB (b) Heuristic: WNT (c) Heuristic: ST (d) Heuristic: SNT (e) Heuristic: WT

Solution: ML Approach

1. Select all features from compiler that could help determine the bias of a branch
2. Pass to ML model to predict branch probabilities using this larger feature set
3. Use predicted branch probabilities to optimize code layout



Data Flow Features

- Capture Opcode & types of inputs to branch (const, var, etc.)

```
%10 = %9 + 2
%11 = %8 * 3
%12 = icmp eq, %11, %10
br %12, label %t, label %f
```

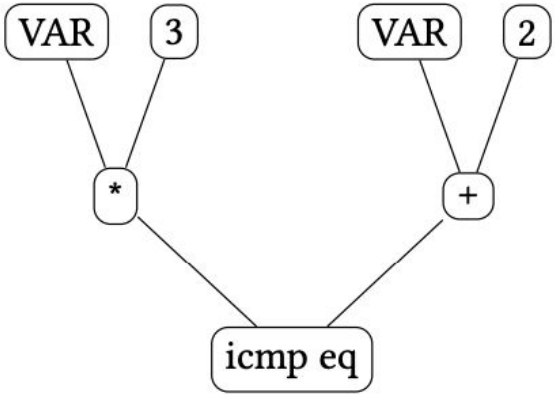


Figure 4. Expression tree

Control Flow Features

- Encode basic CFG shape
- Look at blocks that are control-dependent on branch edge
- Encode most frequently executed function and its attributes

Frequent Function attributes

- Inline, noinline, _always_inline_, cold, etc.
- Function name (embedded)

Loop Features

Numerical Features

- Loop depth, number of BBs, number of Exit Blocks, etc.

Categorical Features

- Is exit edge, is backedge, is destination within loop, etc.

Miscellaneous Features

Function Features

- Information about the enclosing function
- Number of instructions, Basic Blocks, edges in CFG

File Name Features

- Branches in repetitive uses of same file (such as header files) may have similar behavior
- Extract file name from branch destination in debug pass, add it to feature set

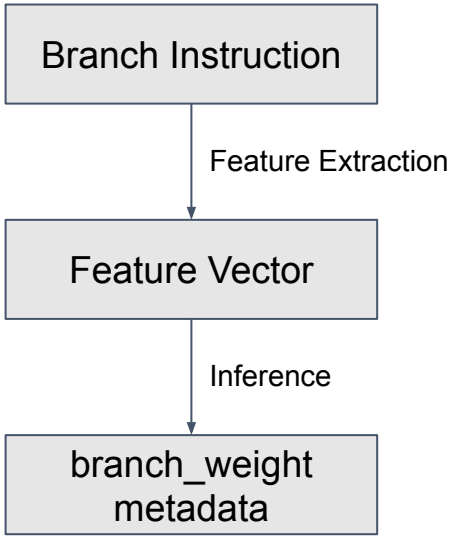
Model Architecture + Training

- Labels obtained by profiling training data and extracting calculated branch probabilities
- Training Data: > 7M unique static branch instructions

Parameter	Value
Num. Layers	5
Hidden activation	ReLU
Final activation function	Sigmoid
Optimizer	Adagrad
Batch size	200
Epochs	100

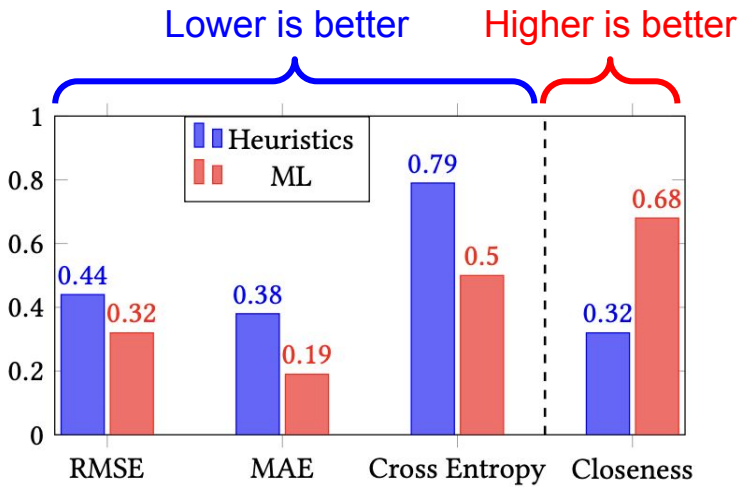
Model Inference

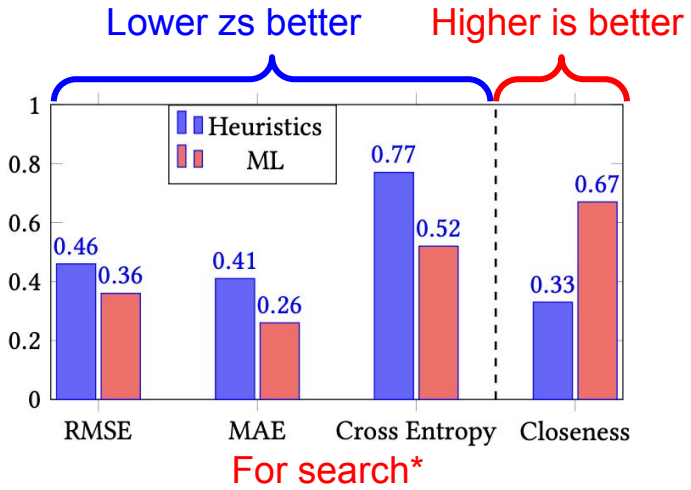
- Inference used within new LLVM pass
  - Load model at start of pass
  - During pass, gather features for each branch
  - Feed formatted input to obtain prediction label
  - Finally use prediction to annotate branch instruction LLVM metadata



Evaluation

- Tested on 10% of examples at random
  - ML model outperforms heuristics ~67% of time
- Fairly robust to distribution shifts (when tested on search)





Evaluation: Error % vs Heuristics

- Heuristic line “jumps” due to fixed % predictions.
  - ML benefits from continuous preds.
  - Unclear how beneficial this actually is in practice

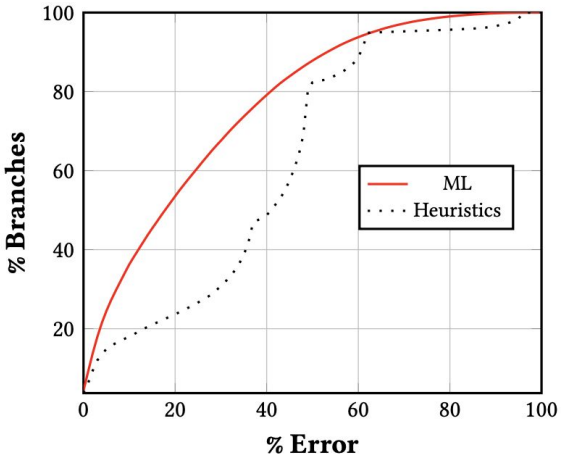


Figure 7. Prediction Error

Evaluation: Real World Applications

- Tested on search, other apps.
  - ~ 1% geometric mean speedup .

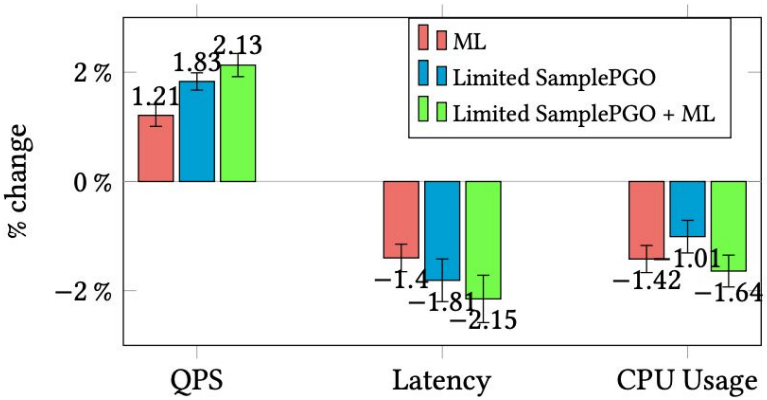


Figure 11. Performance of Search application

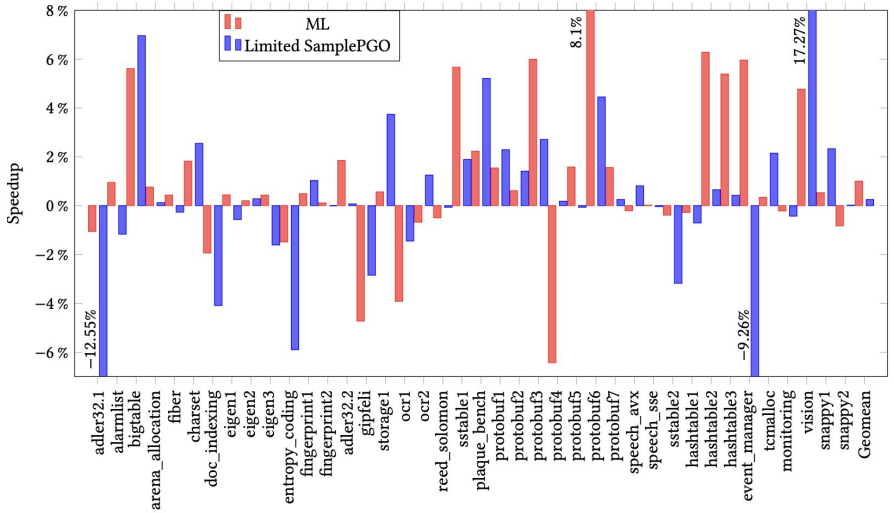


Figure 10. Benchmark Suite Results

Conclusion

What's cool:

- Significant accuracy boost over heuristics
- Relatively simple methodology, but enables use of more features to make predictions
- Avoids hand-engineering (mostly)

Issues:

- Would have liked to see which features were most weighted / an ablation
- Relatively minor improvement in terms of speed
- Adds overhead to compiling (but not much)
- Unsure of generalization abilities

Questions!!