CrystalBall: Statically Analyzing Runtime Behavior via Deep Sequence Learning [1]

Group 26: Heyi Xu, Fangyi Dai, Ye Li

[1] CrystalBall: Statically analyzing runtime behavior via deep sequence learning
S. Zekany, D. Rings, N. Harada, M. A. Laurenzano, L. Tang and J. Mars
2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

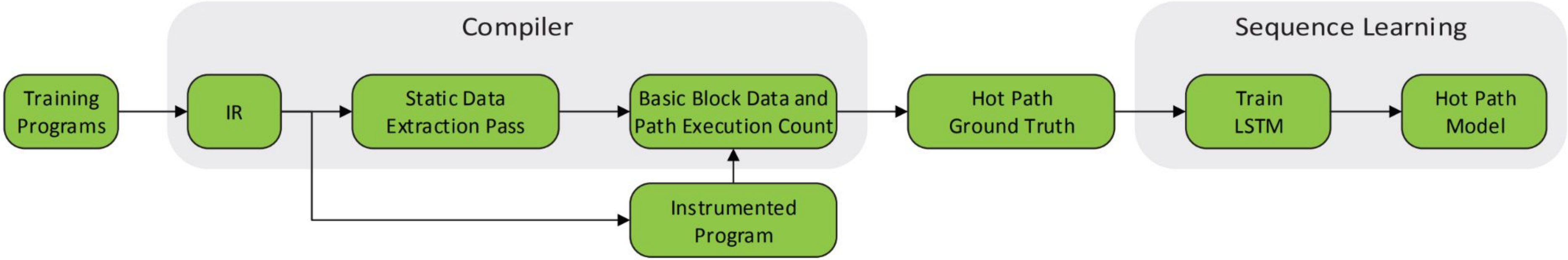
Motivation: Why Predict Hot Paths?

- **Runtime Behavior Drives Software Performance**
  - Many compiler optimizations require knowledge of execution paths
  - Debugging, testing, and security rely on understanding runtime behavior
  - Real programs spend most of their time in a small fraction of paths

Challenges

1. Hotness is a dynamic property, dependent on inputs and runtime context.
2. Static code alone does not show execution frequency.
3. Control-flow graphs can generate millions to billions of paths.
4. Existing static heuristics are language-specific and limited.
5. Modeling dynamic behavior from static IR is difficult.

What Does CrystalBall Do?



1) Converts IR basic blocks into feature vectors
2) Treats each path as a sequence of operations
3) Trains an LSTM to classify paths as hot or cold
4) Enables static hot path prediction without running the program

Key Ideas

1. **Compiler IR**
   a. IR is language-independent
   b. IR encodes both semantic and low-level information
   c. Better suited than high-level source code
2. **Deep Sequence Learning**
   a. Hot/cold patterns emerge from opcode sequences
   b. LSTMs capture long-range dependencies
   c. Removes need for manually engineered features

Path Enumeration (Ball–Larus)

- **CrystalBall** enumerates paths with Ball–Larus numbering
- Every path has a unique, reversible ID
- Efficient enumeration without executing the program

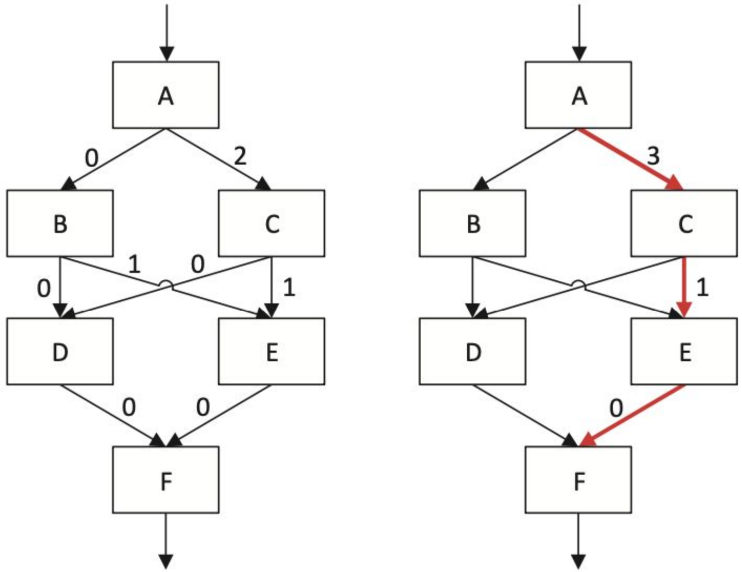


Fig. 1: Example of function path enumeration using Ball-Larus algorithm (left - edge weights between basic blocks, right - example of path reconstruction)

Static Feature Extraction

- Count opcode types per basic block
- Convert each BB → vector
- Path = sequence of vectors

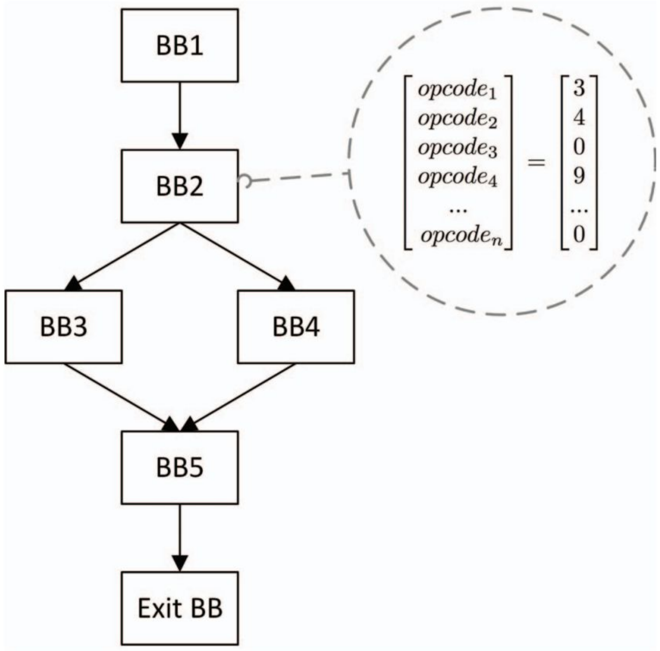
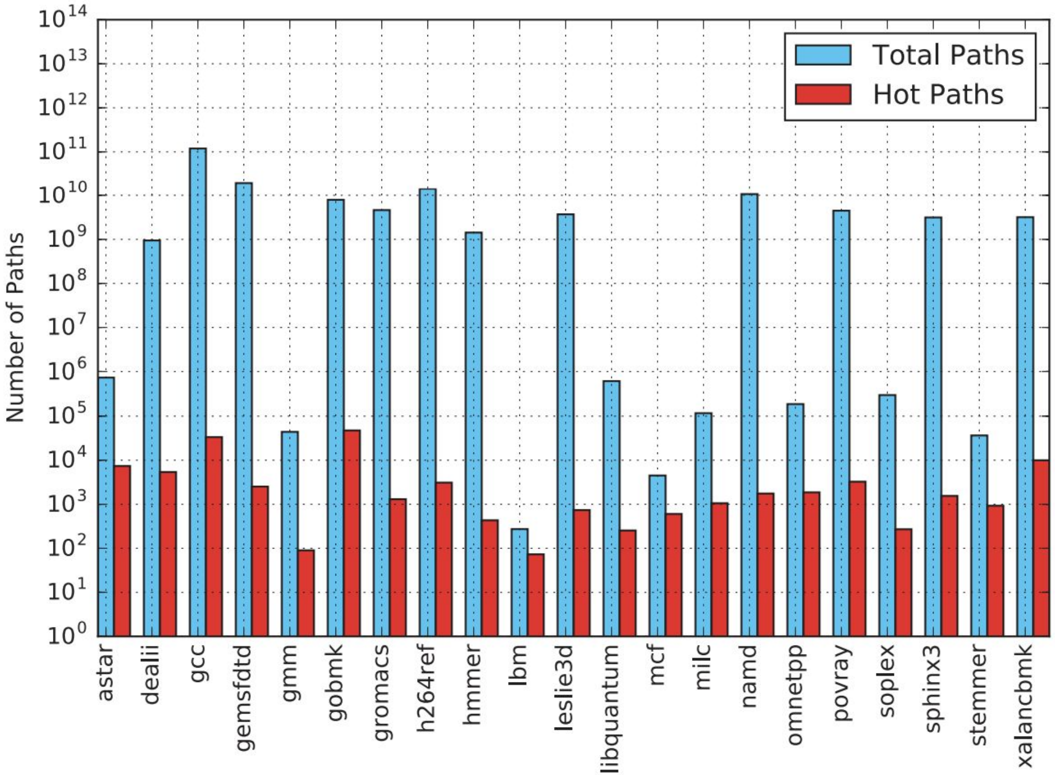


Fig. 5: Basic block feature vector extraction

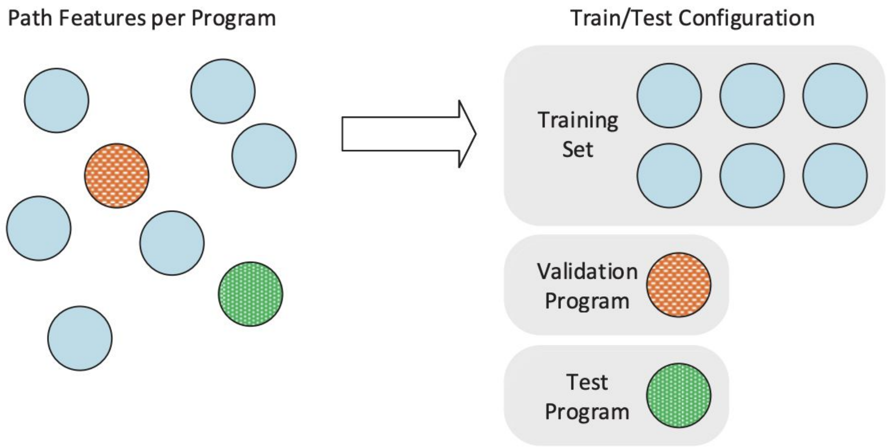
Path Sampling

- Path explosion: billions of possible paths
- CrystalBall samples 2000 cold paths/function
- Always keeps all hot paths



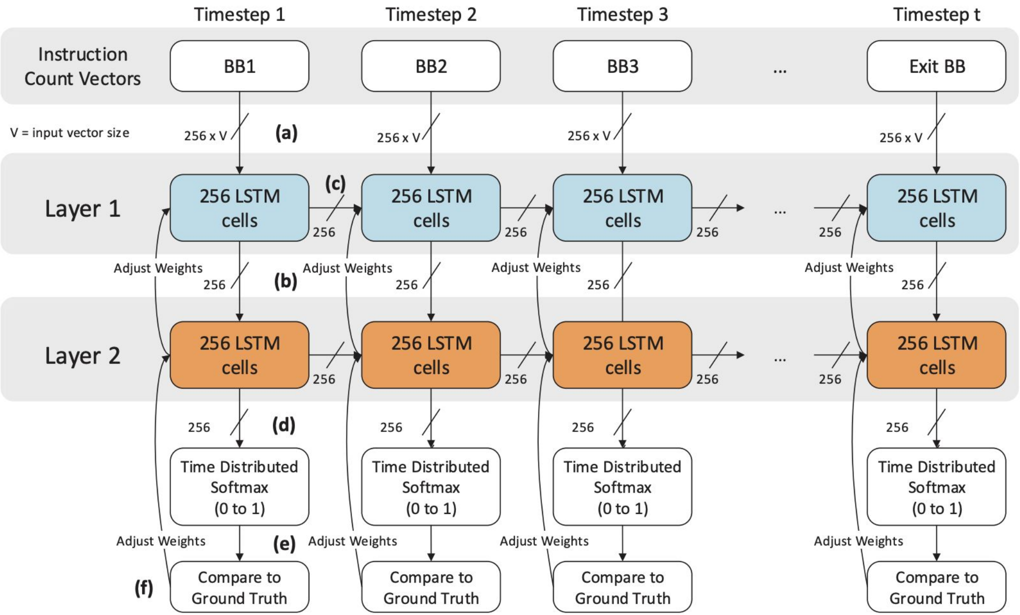
Training Setup

- Leave-one-program-out (LOPO)
- Prevents overfitting and data leakage
- Matches realistic compiler scenarios



Model Architecture

- 2-layer LSTM, 256 units each
  (Long short-term memory a type of recurrent neural networks)
- Sequence-to-label structure
- Time-distributed softmax + BPTT



Benchmark Details

- **SPEC CPU2006** (C, C++, Fortran)
- **Sirius kernels** (C++)
- 21 total programs
- Dynamic profiling with reference inputs

Program	Language	Lines of Code	Suite
astar	C++	5,842	SPEC CPU2006
dealii	C++	81,810	SPEC CPU2006
gcc	C	484,953	SPEC CPU2006
gemsfdtd	Fortran	11,580	SPEC CPU2006
gmm	C++	236	Sirius
gobmk	C	190,118	SPEC CPU2006
gromacs	C	72,220	SPEC CPU2006
h264ref	C	51,578	SPEC CPU2006
hmmer	C	35,992	SPEC CPU2006
lbm	C	1,155	SPEC CPU2006
leslie3d	Fortran	3,807	SPEC CPU2006
libquantum	C	3,454	SPEC CPU2006
mcf	C	2,685	SPEC CPU2006
milc	C	15,042	SPEC CPU2006
namd	C++	2,127	SPEC CPU2006
omnetpp	C++	14,200	SPEC CPU2006
povray	C++	140,892	SPEC CPU2006
soplex	C++	41,463	SPEC CPU2006
sphinx3	C	18,280	SPEC CPU2006
stemmer	C++	865	Sirius
xalancbmk	C++	296,028	SPEC CPU2006

Hot Path Distribution

- **Few** paths dominate **majority** of runtime
- Supports feasibility of static prediction

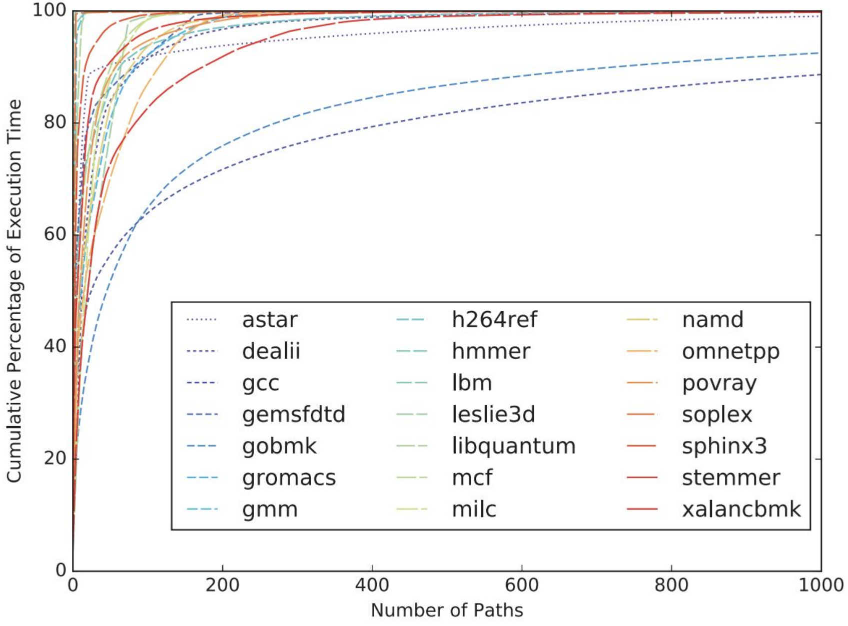
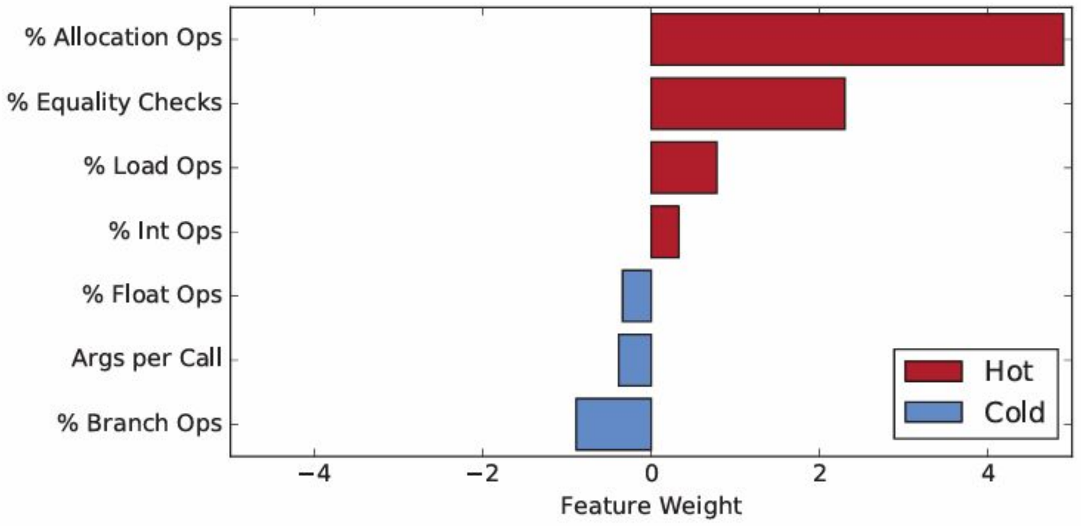


Fig. 9: Paths responsible for cumulative runtime

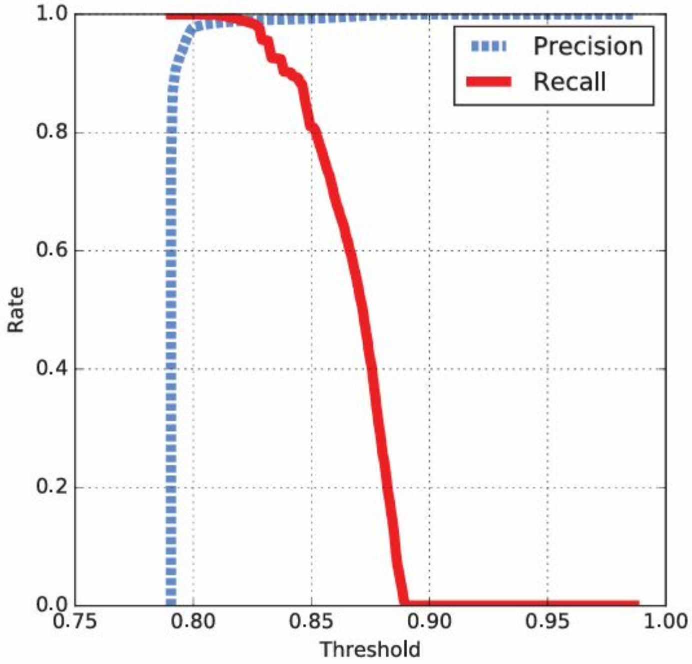
Feature Analysis

- Static IR features that correlate with hot/cold
- Allocations often appear before heavy computation
- Branch-heavy paths explode combinatorially, so only a few are hot
- Equality checks often appear in error-handling or rare paths



Evaluation Metric

- Accuracy fails under heavy class imbalance
- F1 score depends on threshold and is misleading
- AUROC = threshold-independent, robust metric
- Generated by plotting the true positive rate versus the false positive rate across the entire range of possible thresholds



Results

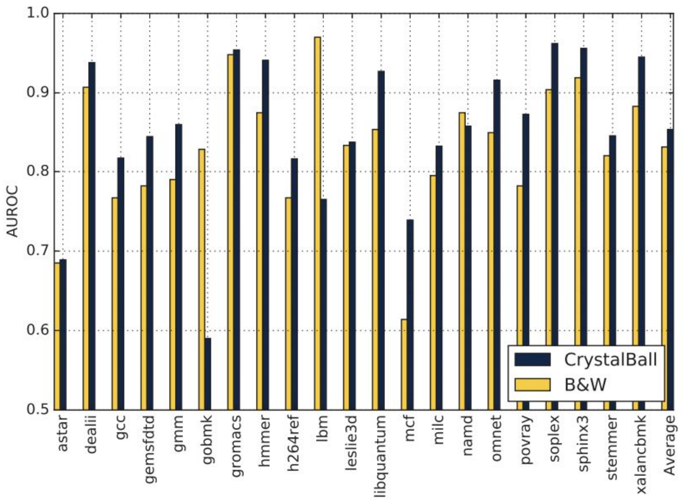


Fig. 12: AUROC by program

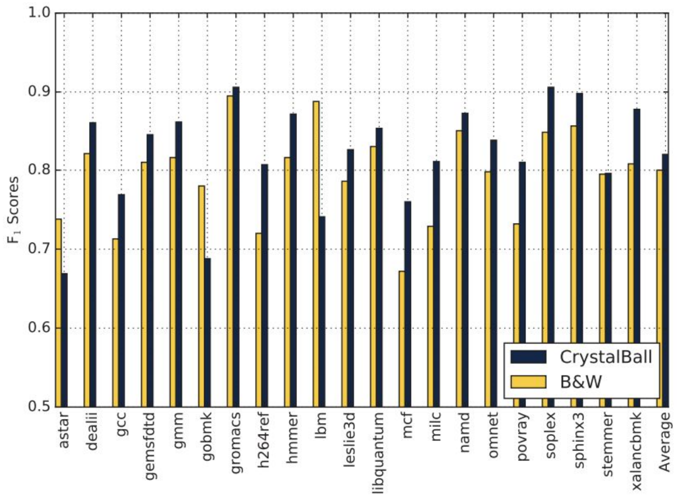


Fig. 13: F_1 scores by program

- **CrystalBall: 0.85 AUROC**
- B&W (baseline): 0.83 AUROC
- Language-independent (it works on LLVM IR)
- does not rely on hand-designed features

Strength

1) LSTM models the order of basic blocks, while feature engineering not
2) Language independence- no need to retrain for each language
3) No feature engineering
4) More training data and more memory gains better performance

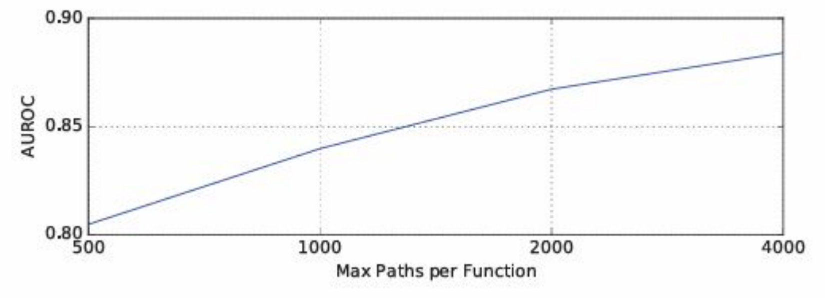


Fig. 14: Performance by max paths for *povray*

Weakness

1) Requires path downsampling, introduces bias
2) Cannot scale to very large CFGs
3) Needs dynamic profiling labels
4) Coverage depends on input representativeness
5) Large training cost

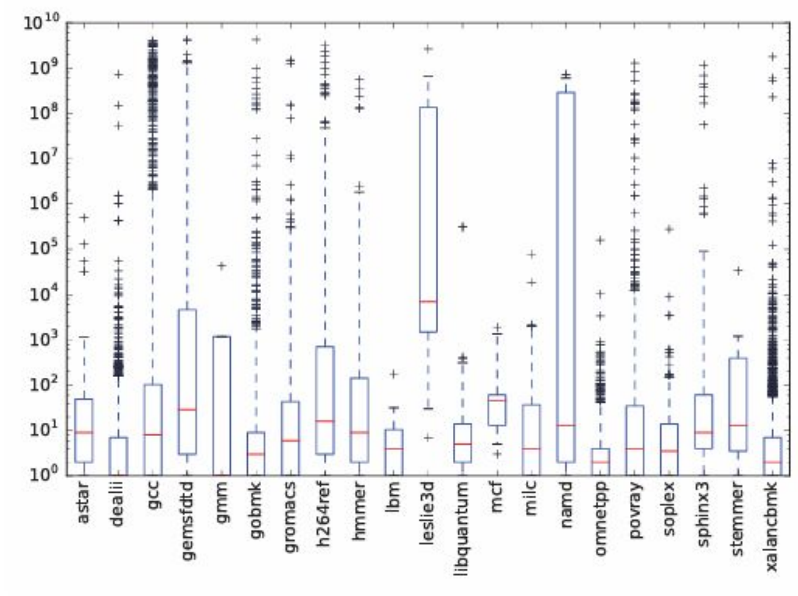


Fig. 10: Path counts per function

Conclusion

1. Deep learning can infer runtime behavior from static IR.
2. ***CrystalBall*** performs better than prior static approaches.
3. Opens the door to compiler optimizations guided by ML.