Using Support Vector Machines to Learn How to Compile a Method

Ricardo Nabinger Sanchez, J. Nelson Amaral, Duane Szafron
University of Alberta, Edmonton, AB, Canada
Marius Pirvu, Mark Stoodley
IBM Toronto Software Laboratory, Markham, ON, Canada

OOPSLA 2006

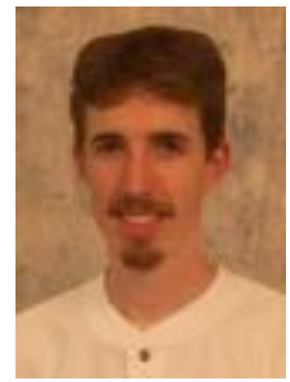
ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications
October 22-26, Portland, Oregon, USA

John Cavazos and Michal O'Boyle, "Method-specific dynamic compilation using logistic regression." OOPSLA 2006.

March 2008, Xangrilá, RS. Ricardo Sanchez

October 2008, IBM Toronto Software Laboratory, Markham, ON, Kevin Stoodley, Mark Stoodley, and Marius Pirvu:
Can we use machine learning to improve compilation decisions in Testarossa?

November 2008: University of Alberta, Edmonton, AB, Canada
Duane Szafron, Michael Bowling, Ricardo Sanchez
We should try Support Vector Machines.



IBM Toronto Software Laboratory

May 2009-August 2009: Ricardo spends the Summer working with mainly with Marius Pirvu at the IBM Toronto Software Laboratory

The Research Question

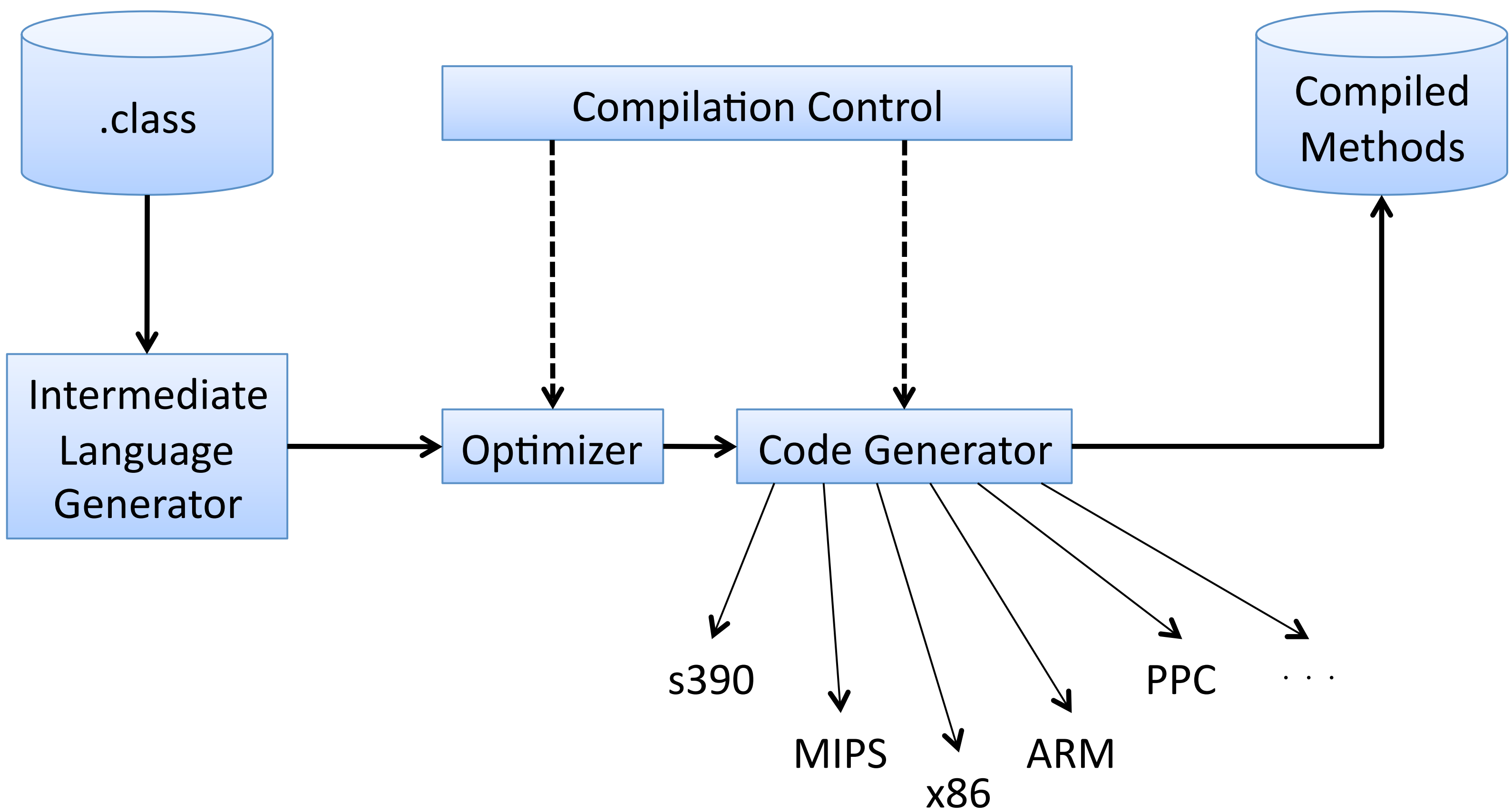
Can Support Vector Machines (SVMs) improve on the selection of code transformations done by human developers?

Characterize methods using *features*.

Learn to associate features with compilation strategies.

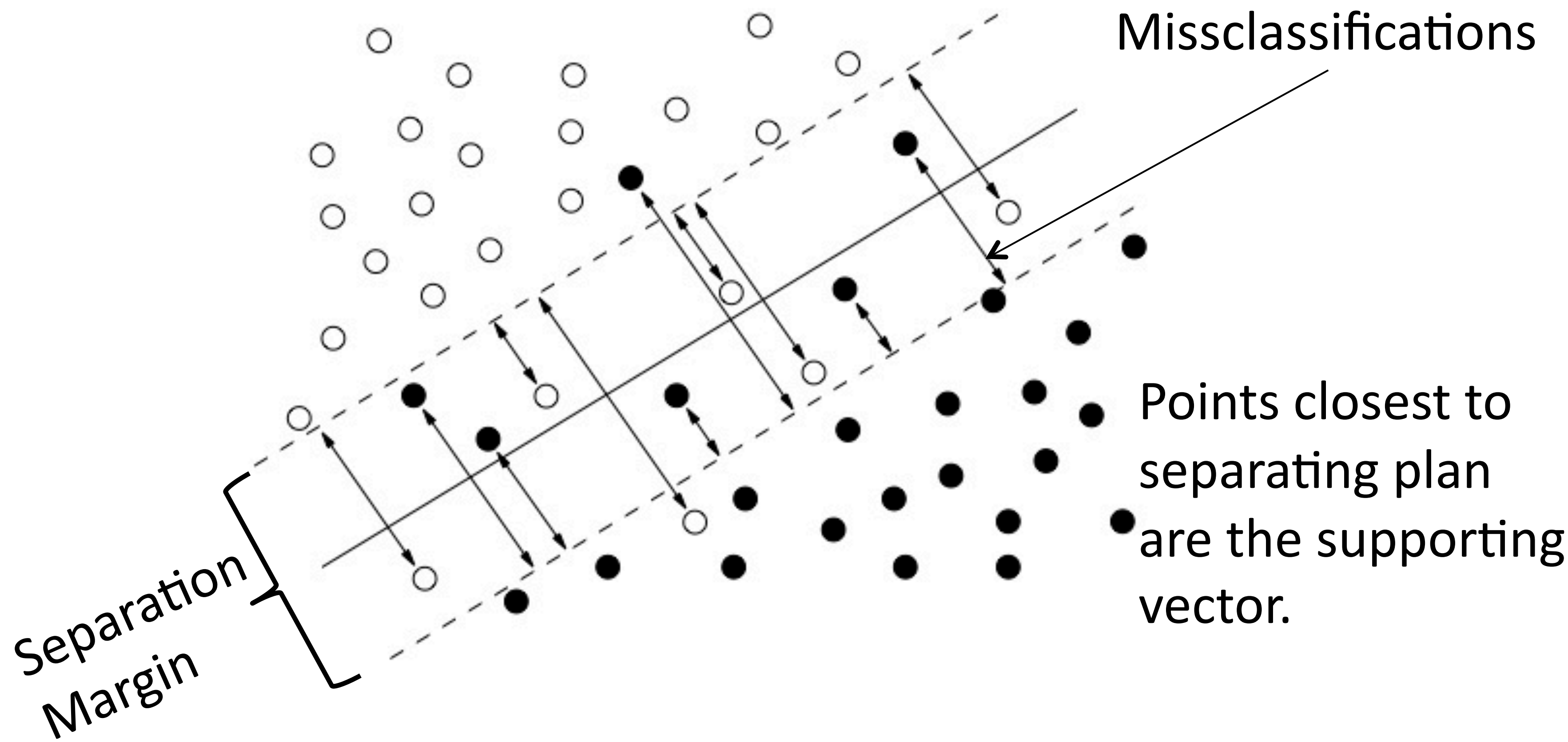
Strategies can be selected on a per-method basis.

Testarossa



.class
Compilation Control
Compiled Methods
Intermediate Language Generator
Optimizer
Code Generator
s390
MIPS
x86
ARM
PPC
. . .

Support Vector Machines

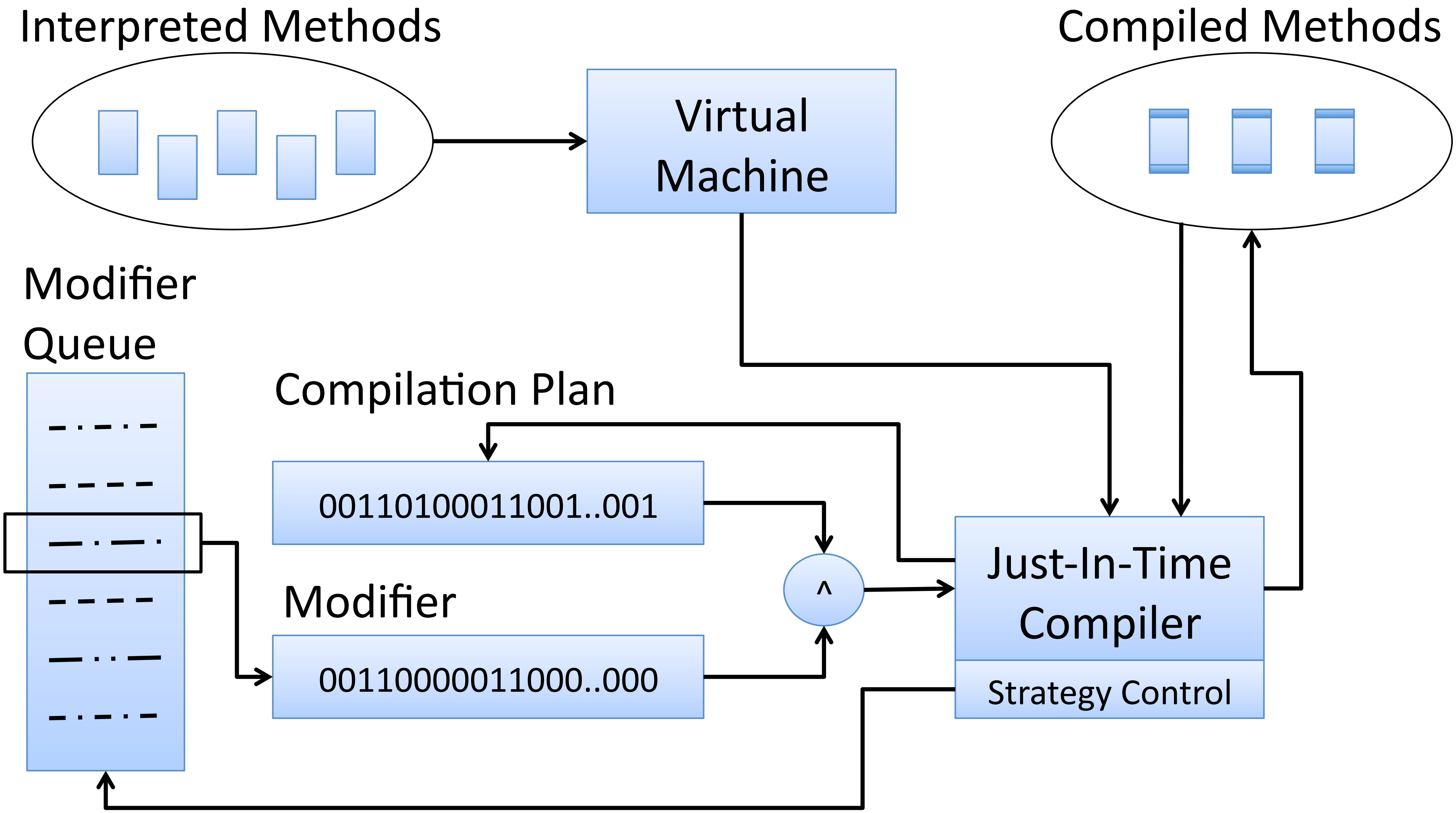


A parameter C in the implementation of the SVM specifies the maximum separation margin.

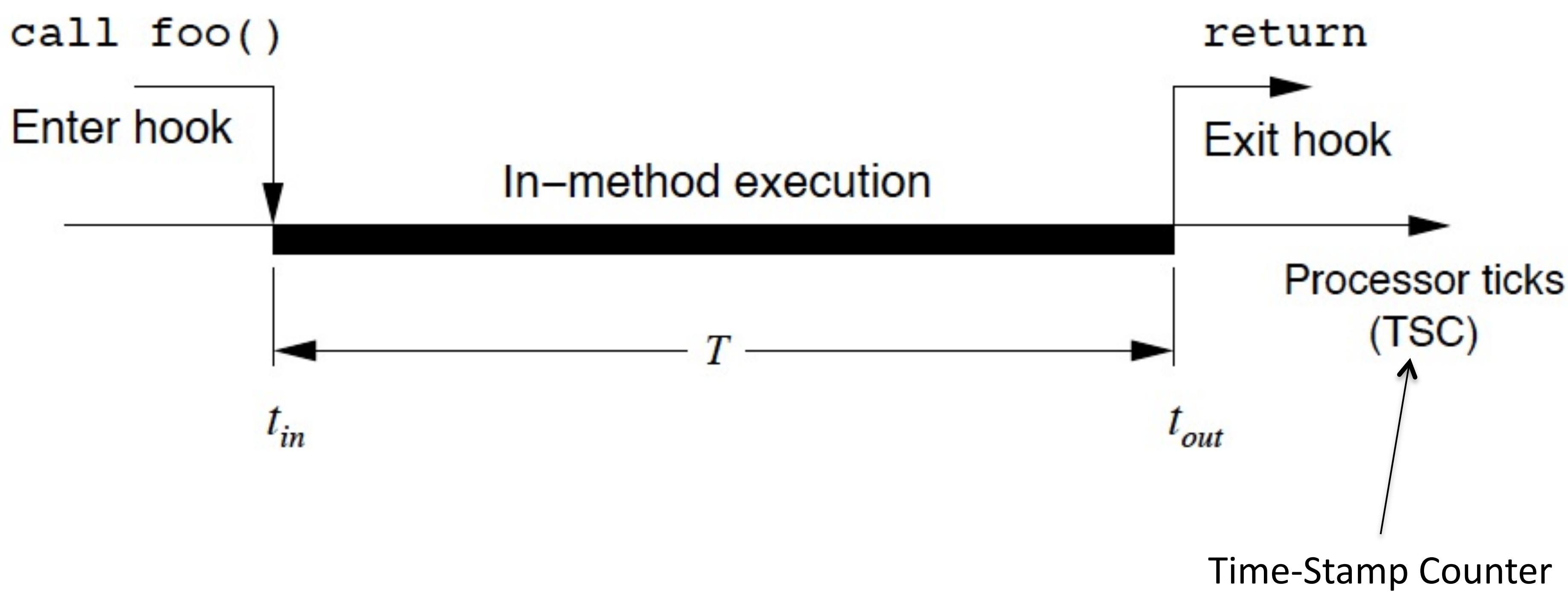
SVMs in Testarossa

- 51 features to describe each method
  - 51-dimension space search
- More than 70 code transformations
  - More than 2^{70} classes
- Why not non-linear kernels
  - Data is already highly dimensional
  - No need to project it to higher dimensions

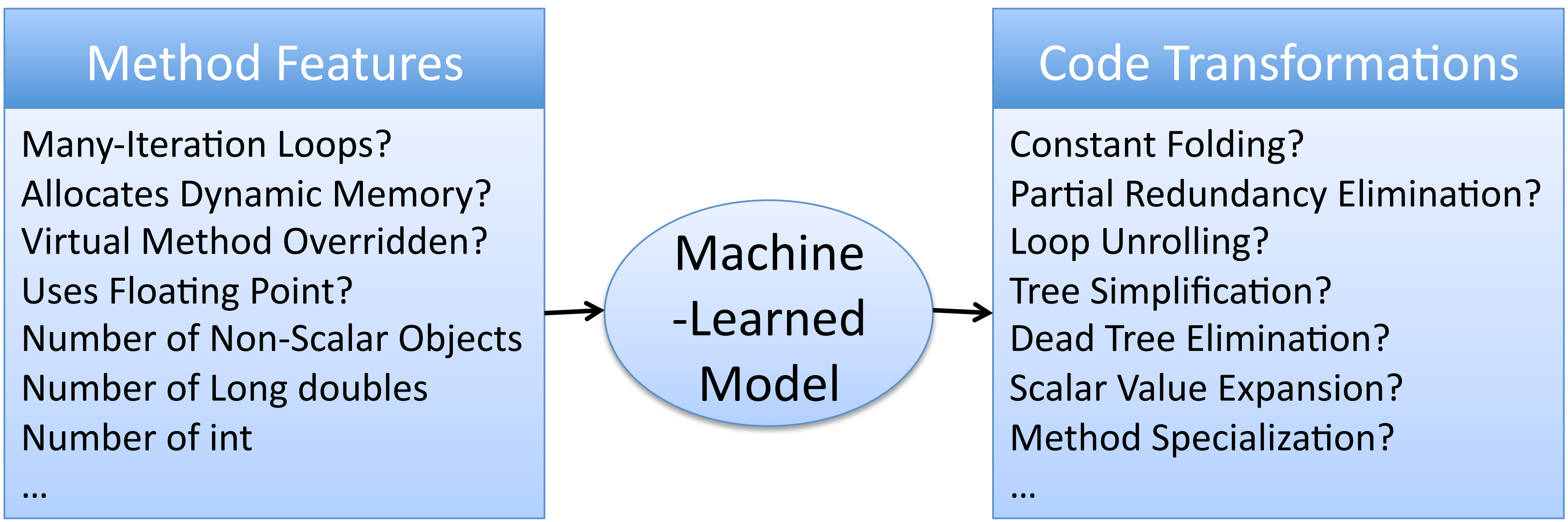
Data Collection



Measuring Time



Goal


Method Features
Many-Iteration Loops?
Allocates Dynamic Memory?
Virtual Method Overridden?
Uses Floating Point?
Number of Non-Scalar Objects
Number of Long doubles
Number of int
…
Machine
-Learned
Model
Code Transformations
Constant Folding?
Partial Redundancy Elimination?
Loop Unrolling?
Tree Simplification?
Dead Tree Elimination?
Scalar Value Expansion?
Method Specialization?
…

Ranking Plans

Let (i,p,h) represent a method **i** compiled with a compilation plan **p** at a level of hotness **h**:

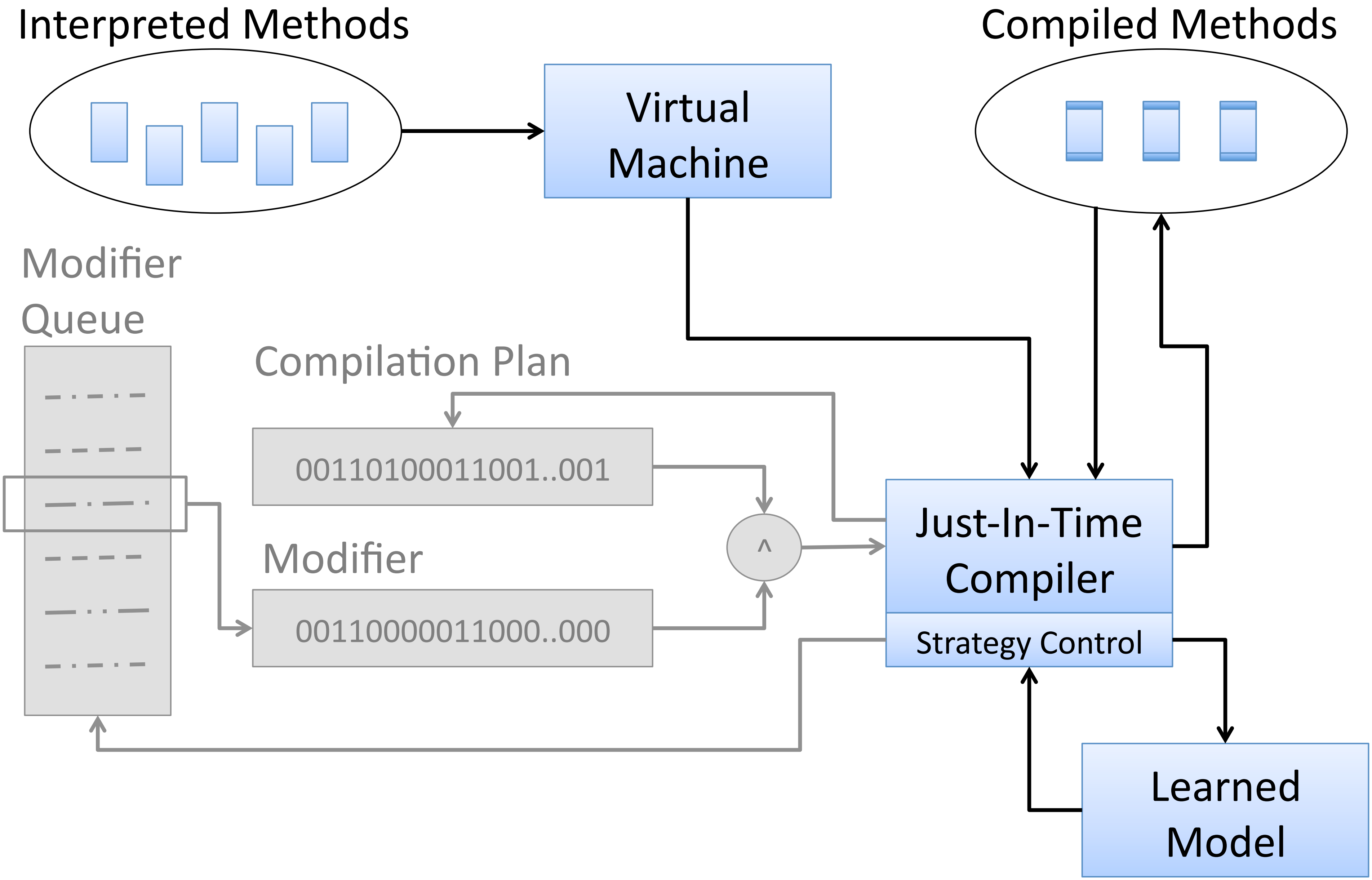
$$\text{Value(i,p,h)} = \frac{\text{Total running time of all invocations of (i,p,h)}}{\text{Number of invocations of (i,p,h)}} + \frac{\text{Compilation time for (i,p,h)}}{\text{Threshold for h}}$$

For each method **i**, select the top **t** plans for training of the SVM.

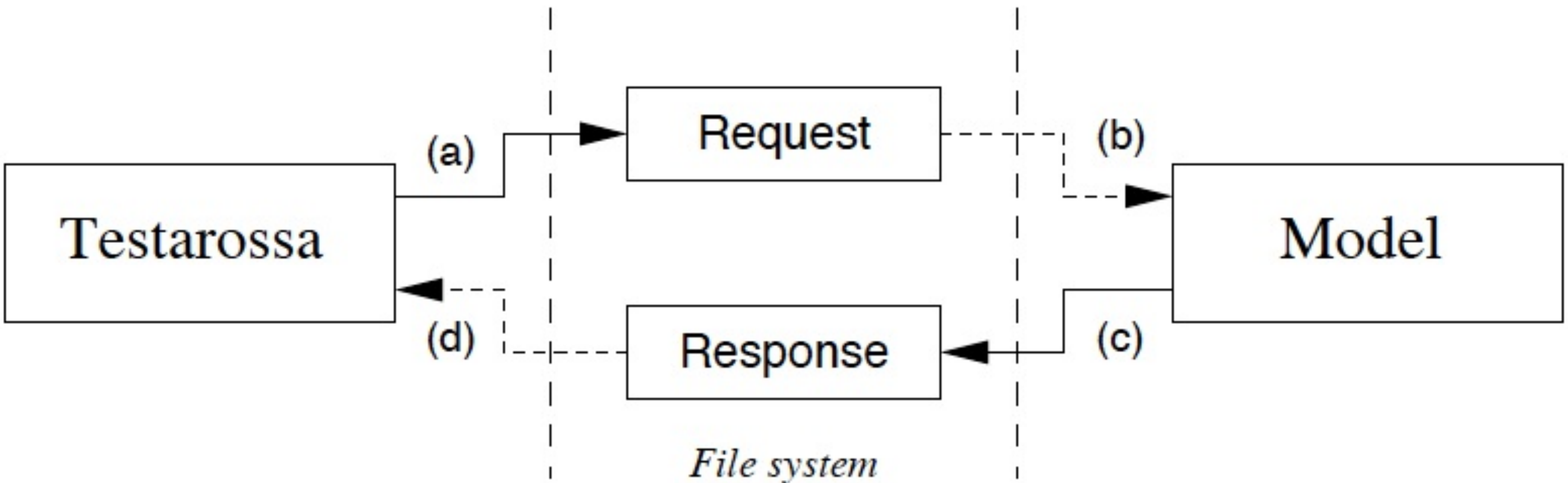
The valued of the lowest plan must be at least **f** % of the best.

In this research **t = 3**, and **f = 95%**.

Using Learned Model



Socket-Based Communication Between Compiler and Model



Used *named pipes* (Unix) to communicate between Compiler and Model

Data Set Sizes

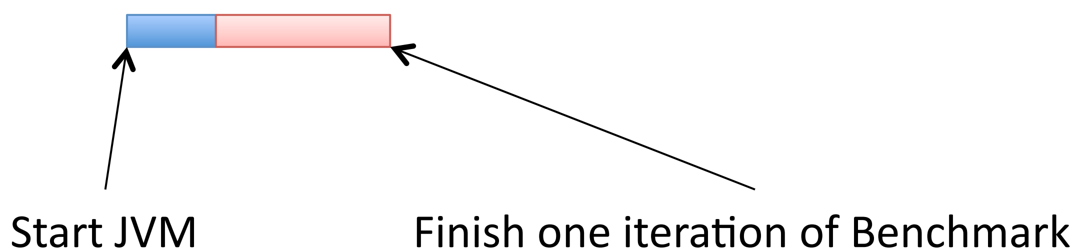
Compilation Level	Merged Data				Ranked Data (training)			
	Data Instances	Unique Classes	Unique Feature Vectors	Vector: Instance Ratio	Instances	Classes	Feature Vectors	Vector: Instance Ratio
Cold	1,551,545	1,421,717	1,175	1:1,320	2,326	949	1,094	1:2.12
Warm	1,577,157	1,455,947	1,153	1:1,368	2,213	1,590	1,108	1:1.99
Hot	2,543,564	2,229,364	1,201	1:2,118	2,073	1,379	1,069	1:1.94

Experimental Platform

- AMD Blade Server
  - 16 nodes
    - 2 Quad-Core Opteron/Node
    - 2 GHz
    - 8 GiB of RAM
    - 20 GiB swap space
    - CentOS GNU/Linux
- Development version of Testarossa

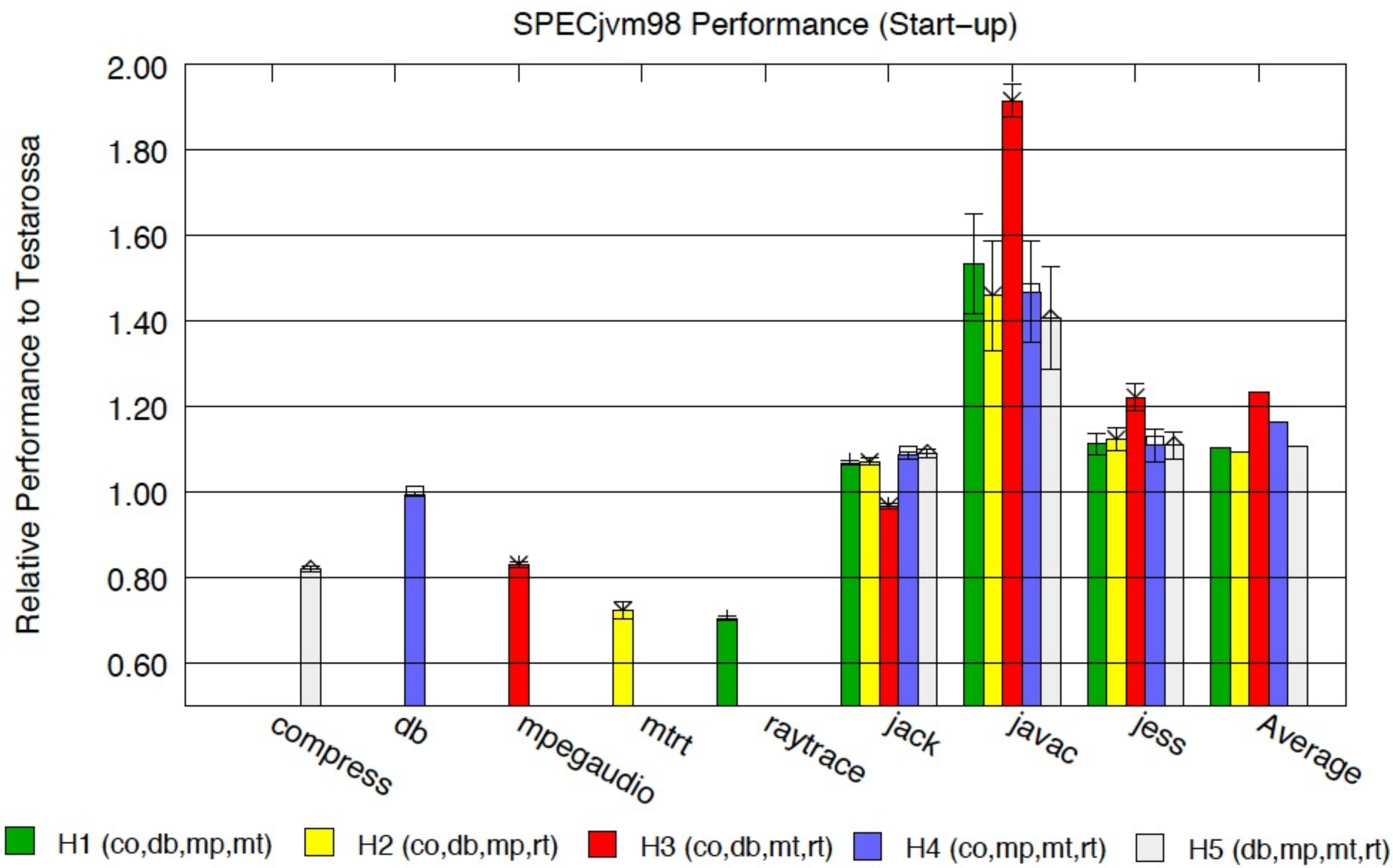
StartUp × Throughput Performance

StartUp Performance:

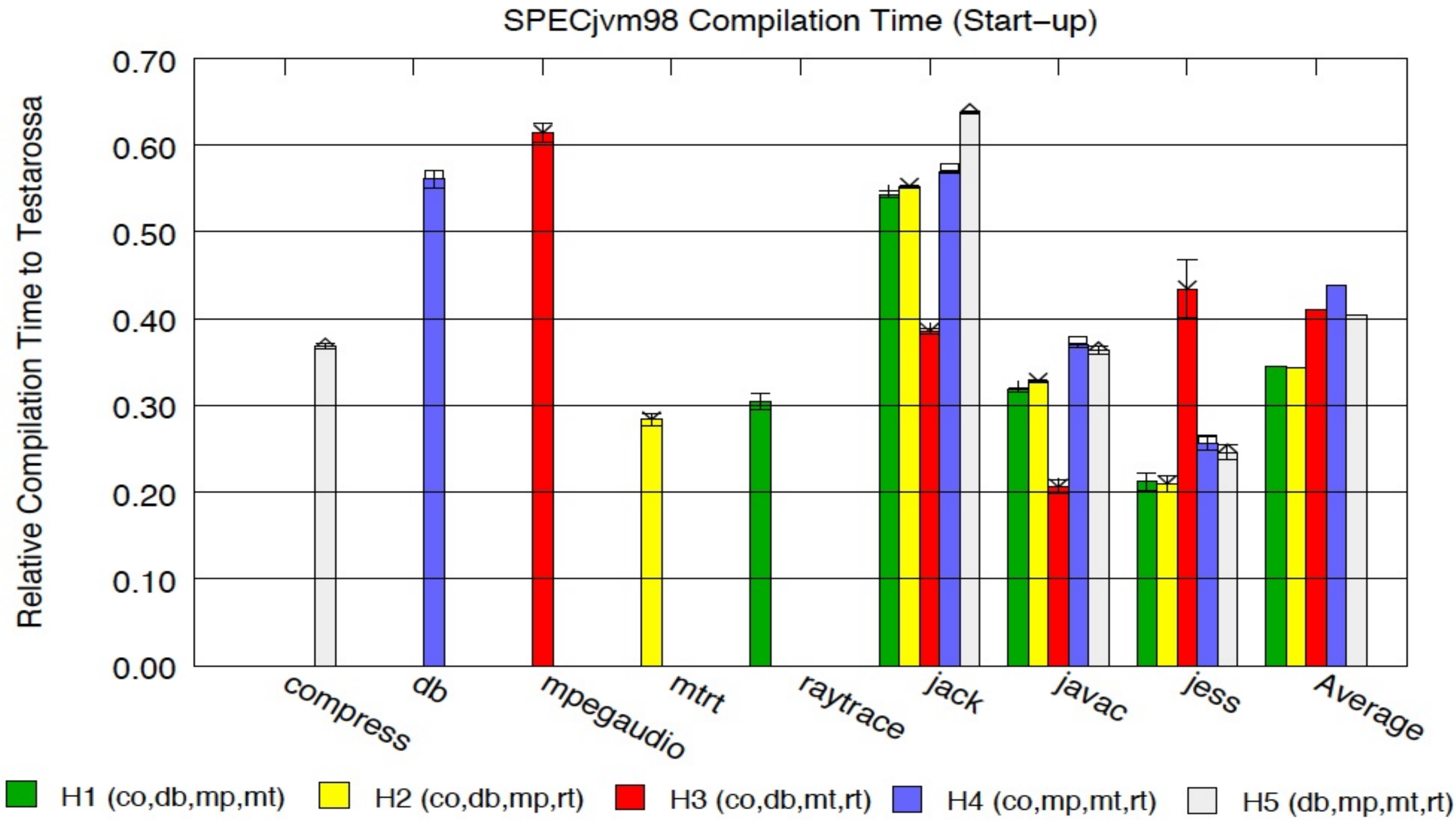


Throughput Performance:

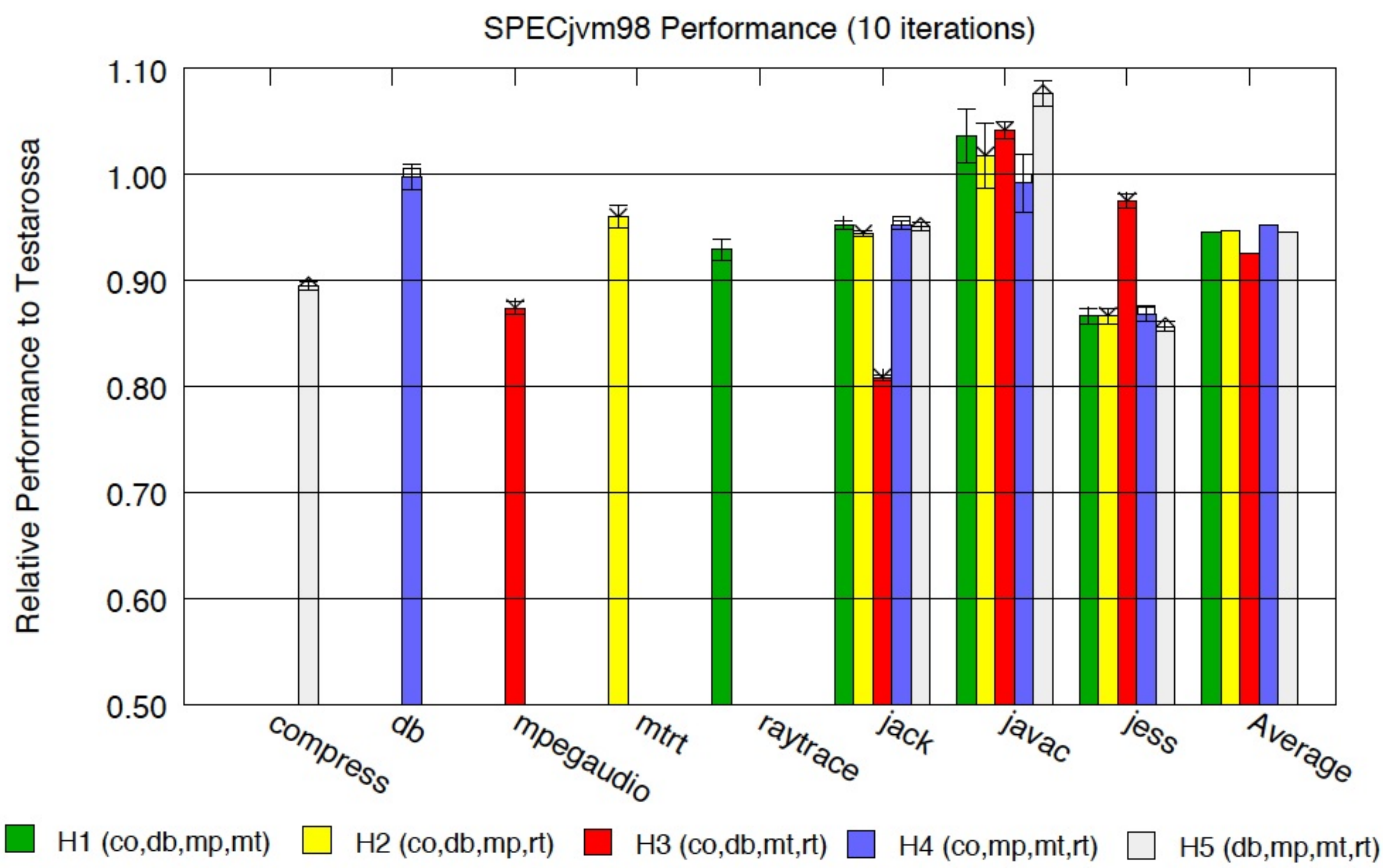
StartUp Performance (SPECjvm98)



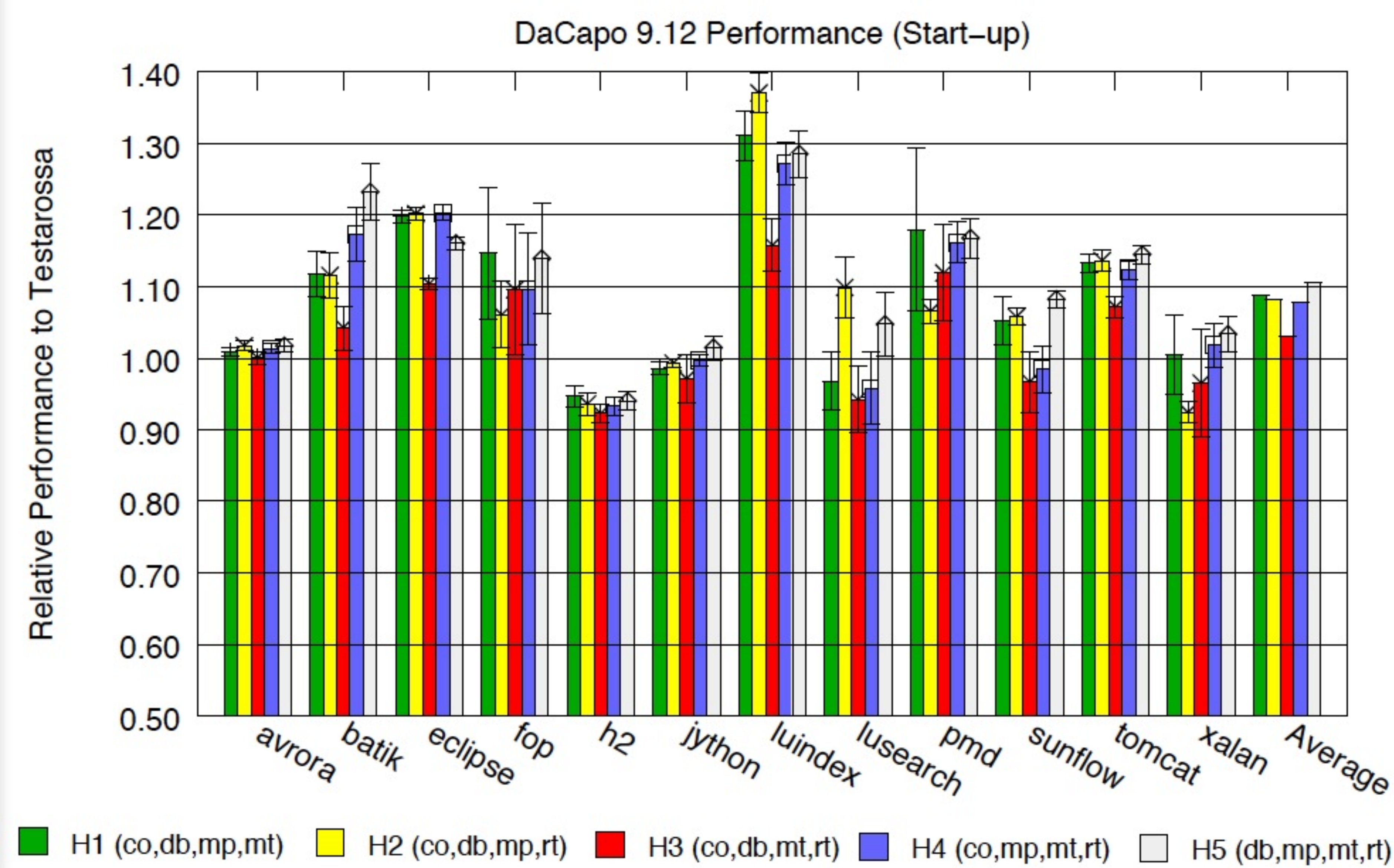
Compilation Time Reduction for StartUp (SPEC jvm98)



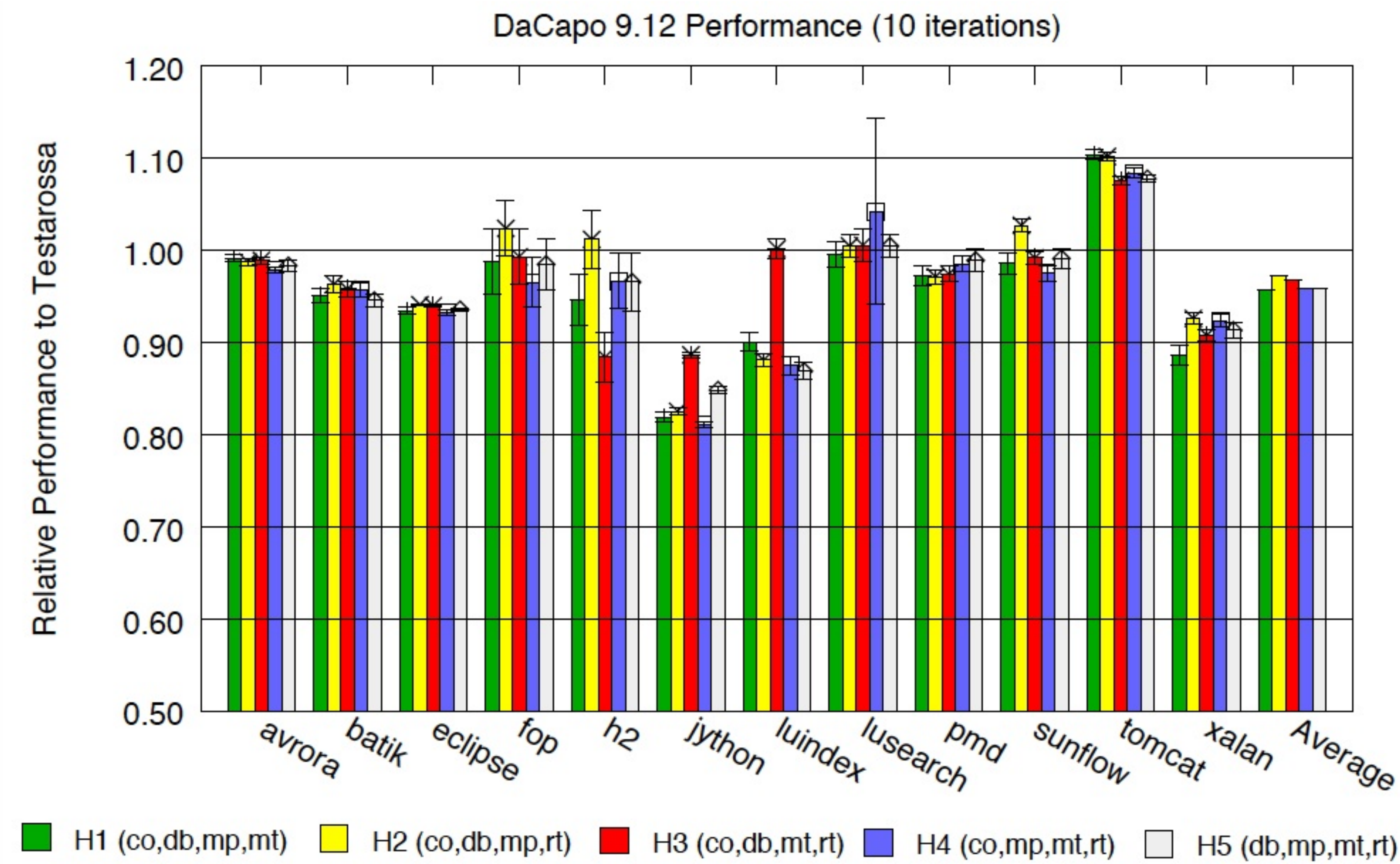
Throughput Performance (SPECjvm98)



StartUp Performance DaCapo



Throughput DaCapo



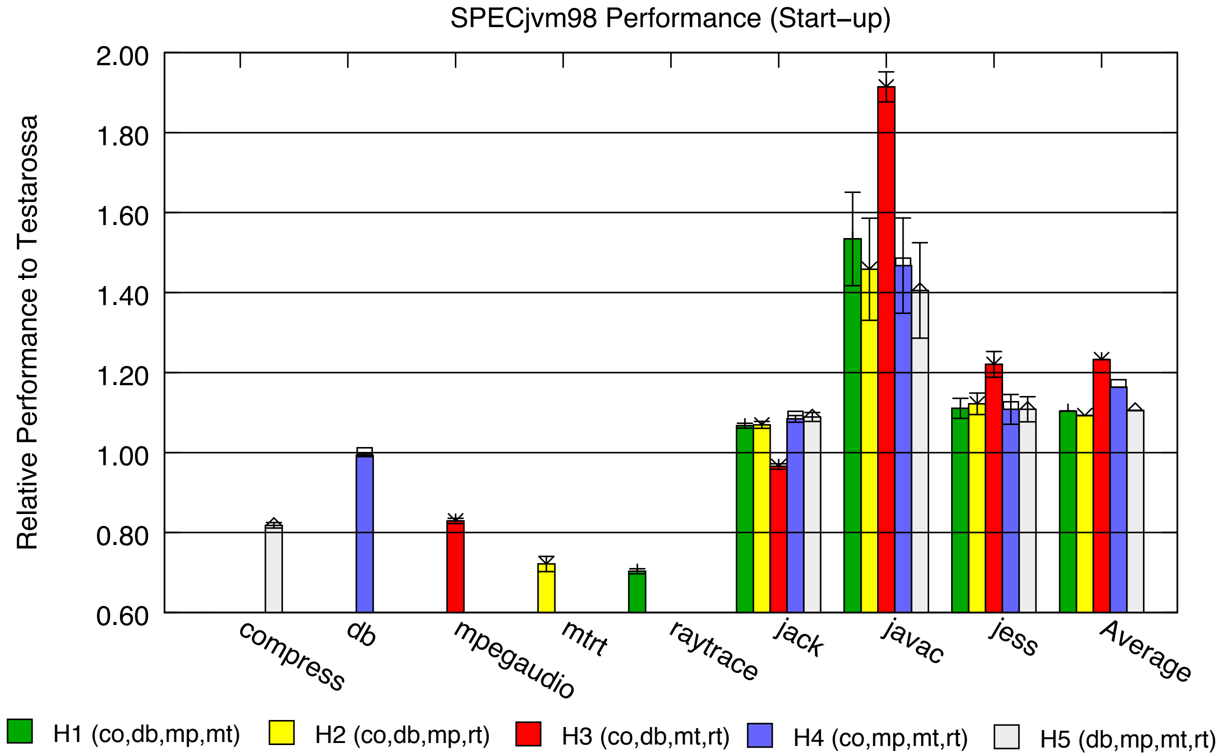
Influence of Inlining

For the previous performance results we collected method features and applied the model before inlining.

Inlining may change method features significantly.

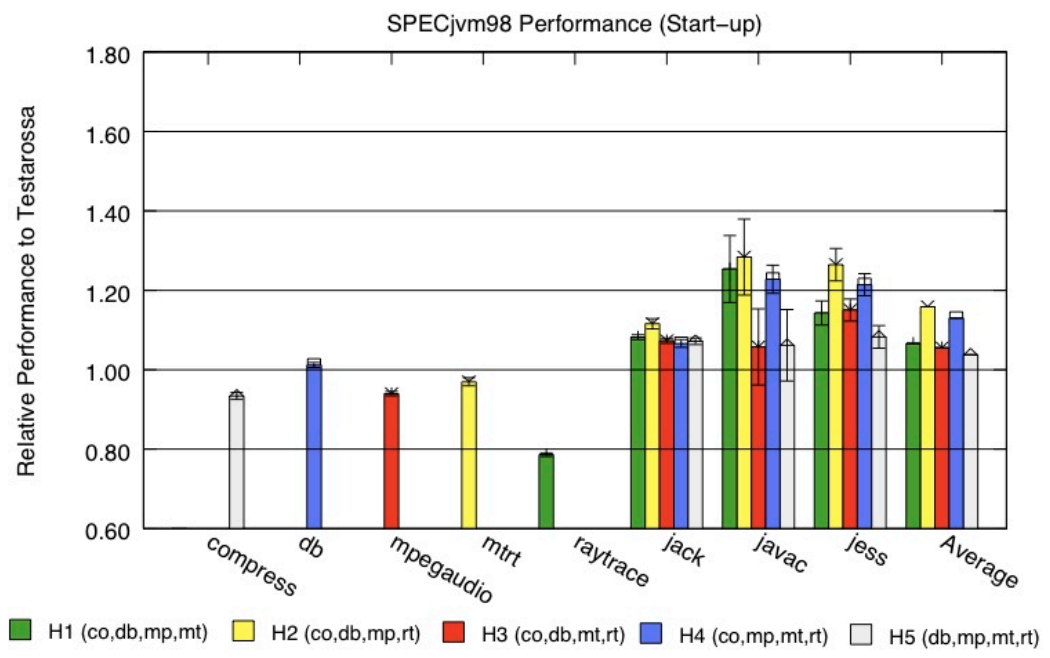
What would the results be if method features were measured after inlining?

StartUp Performance (SPECjvm98)

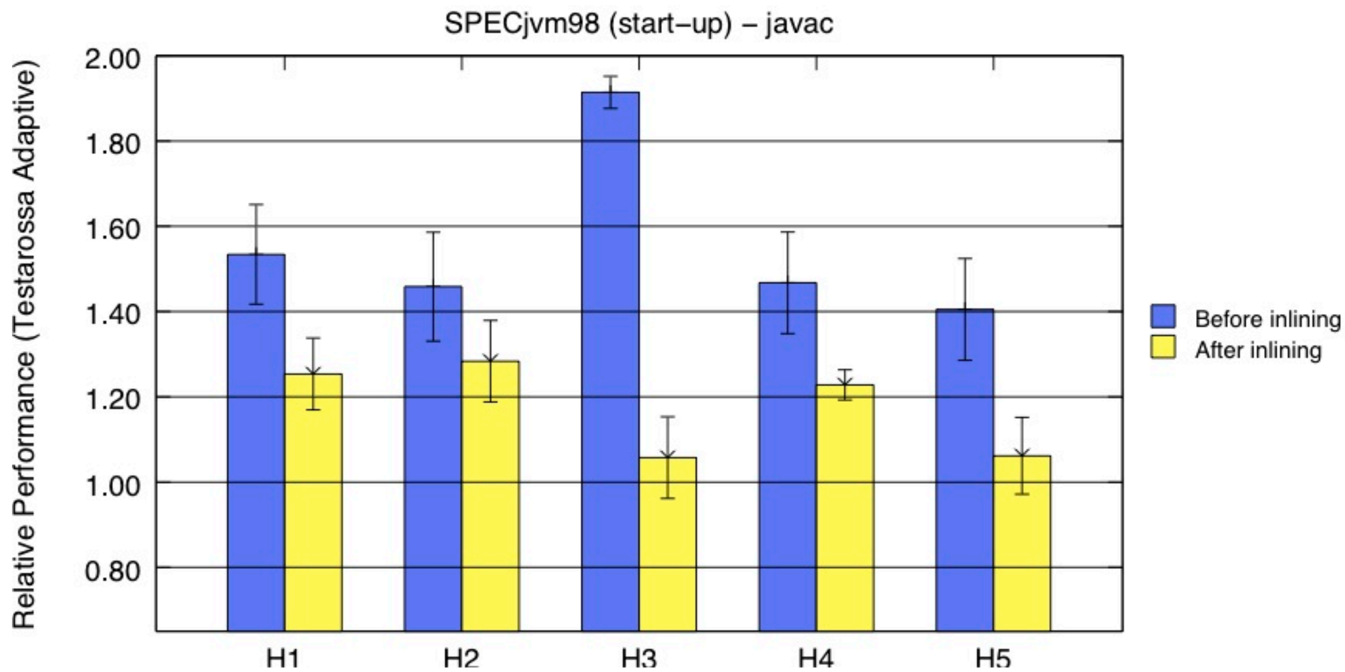


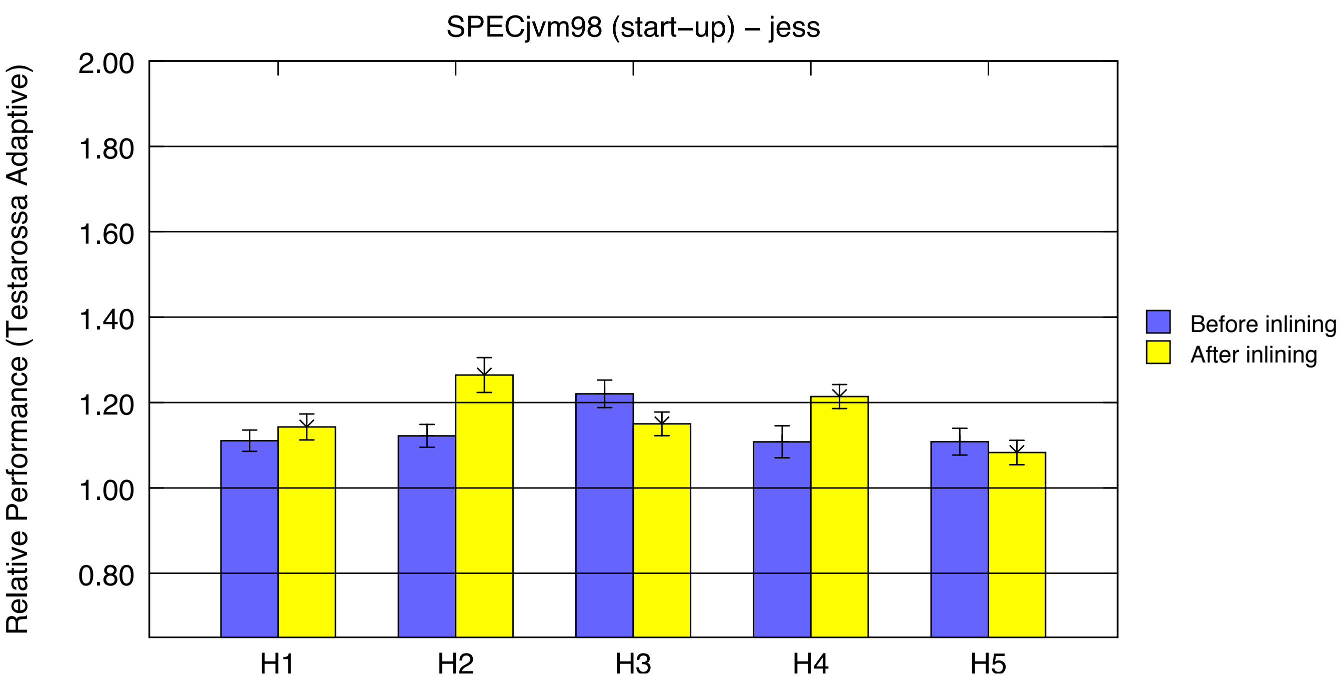
Before Inlining

After Inlining



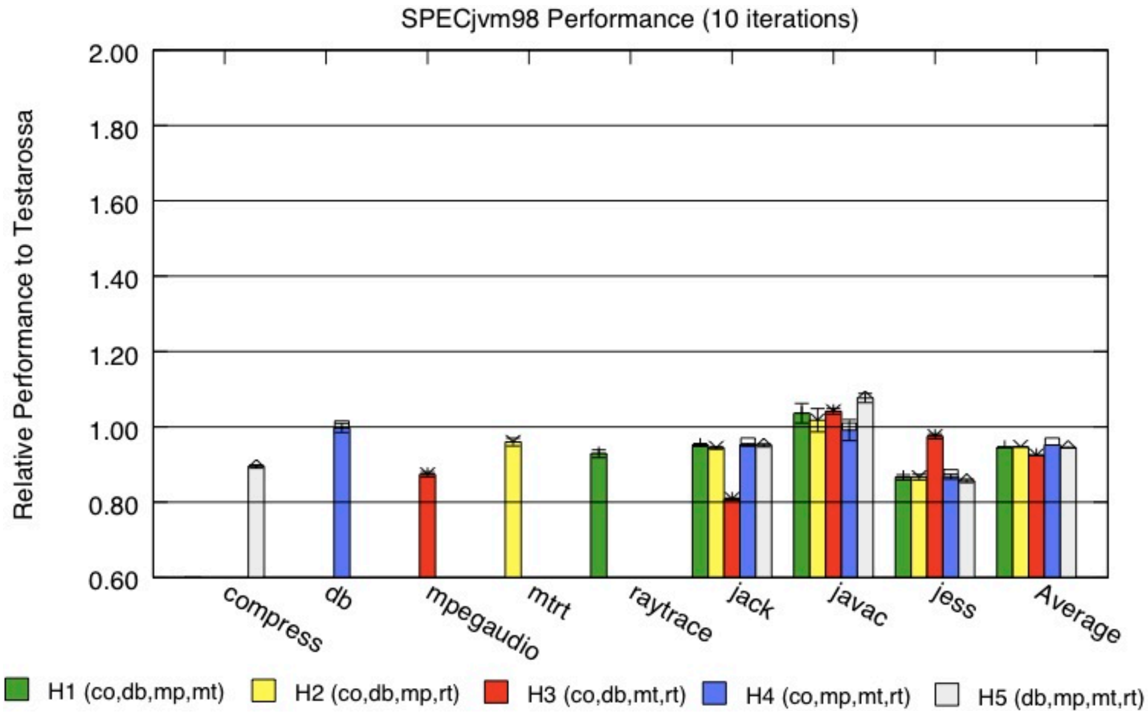
JavaC StartUp Performance





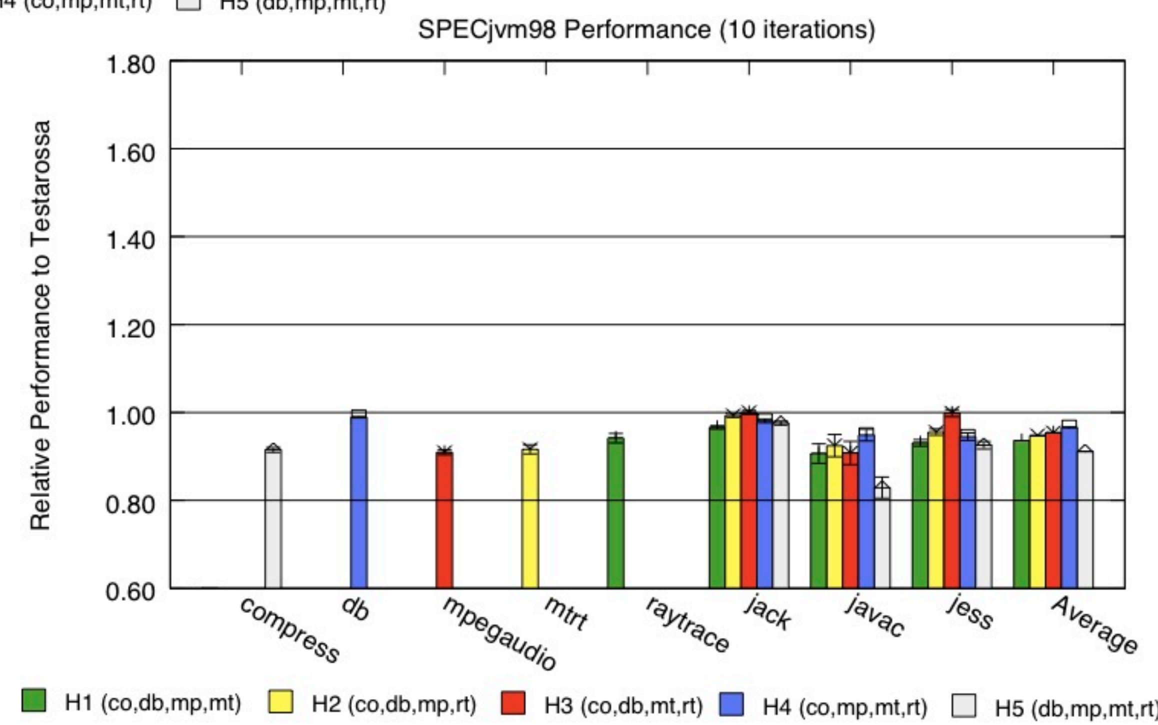
Jess StartUp Performance

Throughput Performance (SPECjvm98)

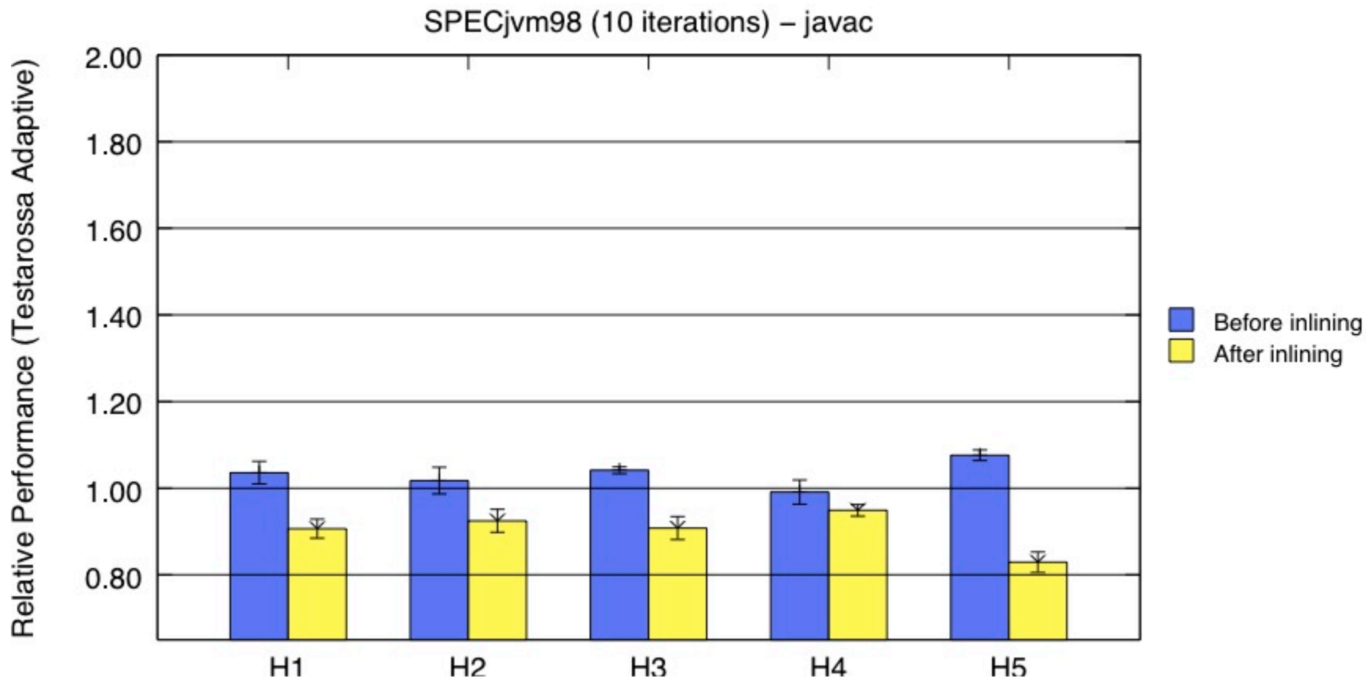


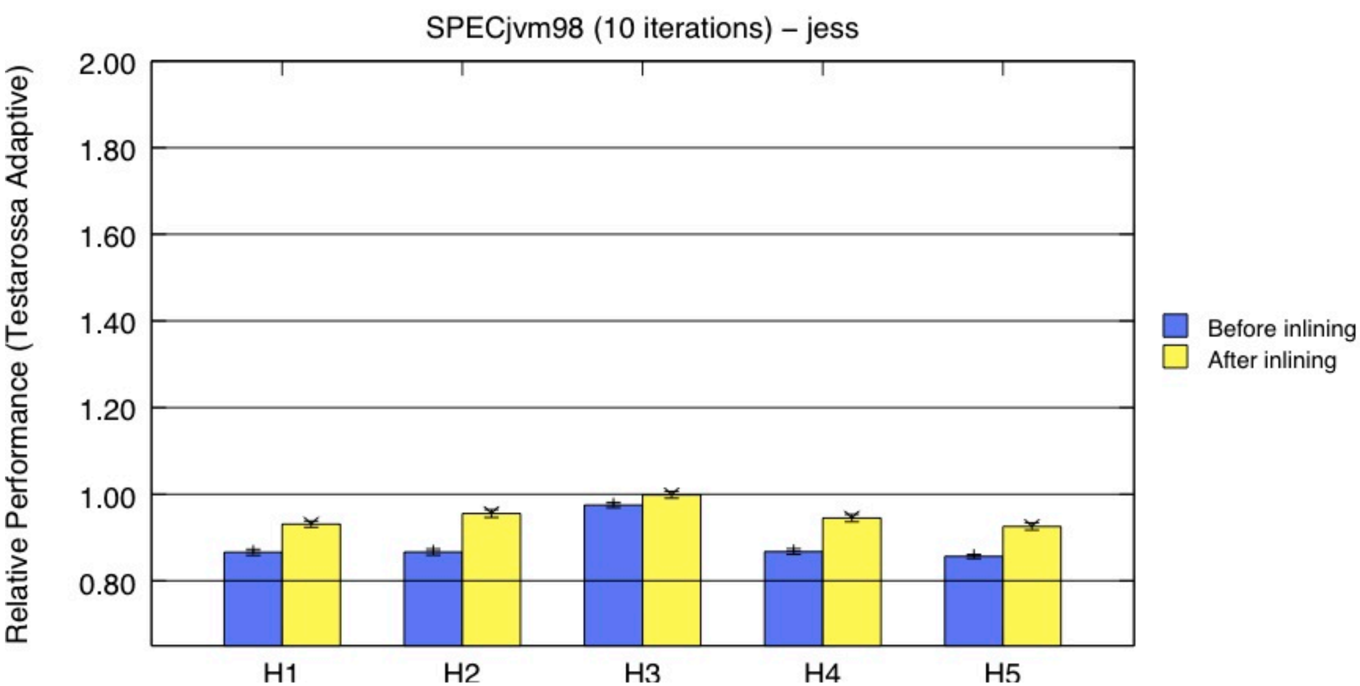
Before inlining

After inlining



javac
Throughput





jess
Throughput

What have we learned?

- **Overall**: SVM-based models outperform Testarossa's heuristics for start-up performance.
  - But it underperforms Testarossa for throughput performance.
- **Surprising**: significant reduction in compilation time.
- **Puzzling**: Collecting method features after inlining did not yield greater performance gains.
- **Pleasantly positive**: model generalized from SPEC benchmaks to DaCapo.