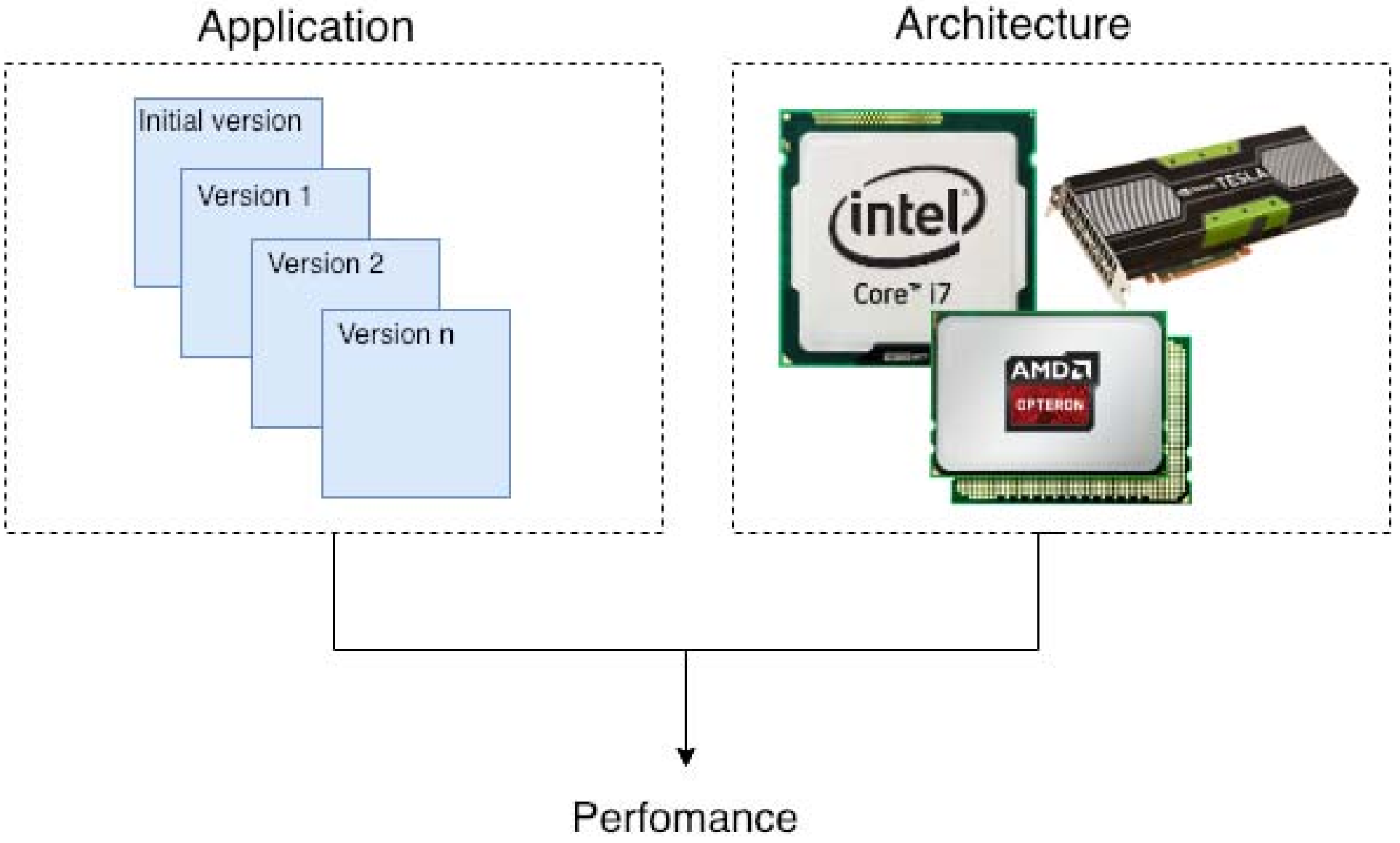
Automatic Source-to-Source Optimizations using Machine Learning

Maksim Berezov, Centre de recherche en informatique

Advisor:
Corinne Ancourt, MINES ParisTech CRI, PSL Research University

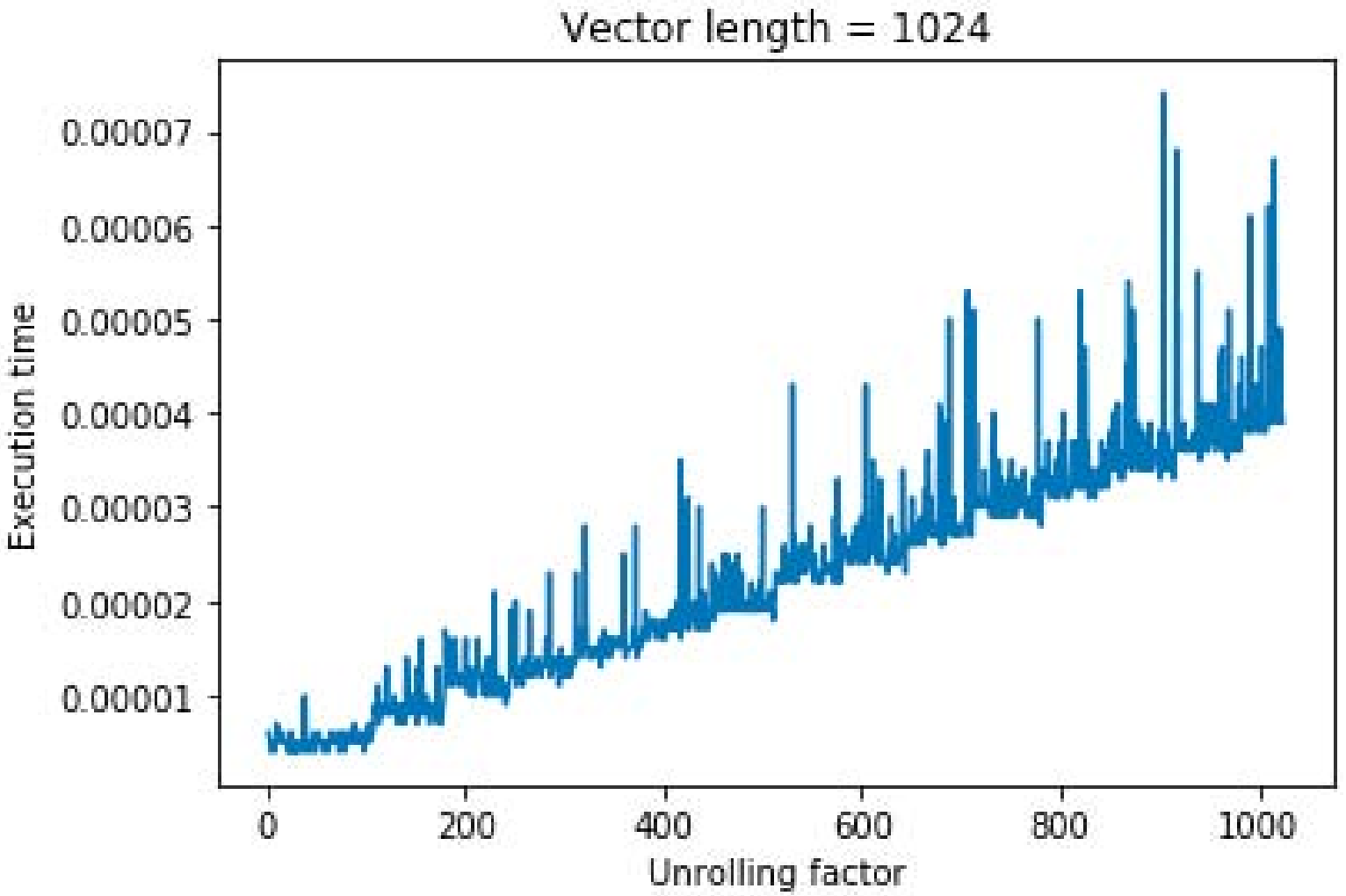
February 30, 2019


Application
Initial version
Version 1
Version 2
Version n
Architecture
intel
Core i7
TESLA
AMD
OPTERON
Perfomance

Context

- All possible transformations on a source code that produce an equivalent code form an optimization set.
- To perform a beneficent optimization we need to know suitable values of the parameters for each transformation (for example, for <loop unrolling> transformation, we need to know <unrolling factor> parameter)

Execution time for vector multiplication with different unrolling factors



Source-to-Source transformations

The benefits of source-to-source transformations:

- Program remains understandable
- We can easily track the result
- Transformations step-by-step

The most interesting for us are operations related to loops.
Considering the transformation set is <unrolling, tiling, interchange>.

https://pips4u.org/

Optimization complexity

Search space is very large:

- m is #transformations
- s is sequence size
- Number of transformation sequences: m^s
- If we take order of transformations into account: $s! * m^s$

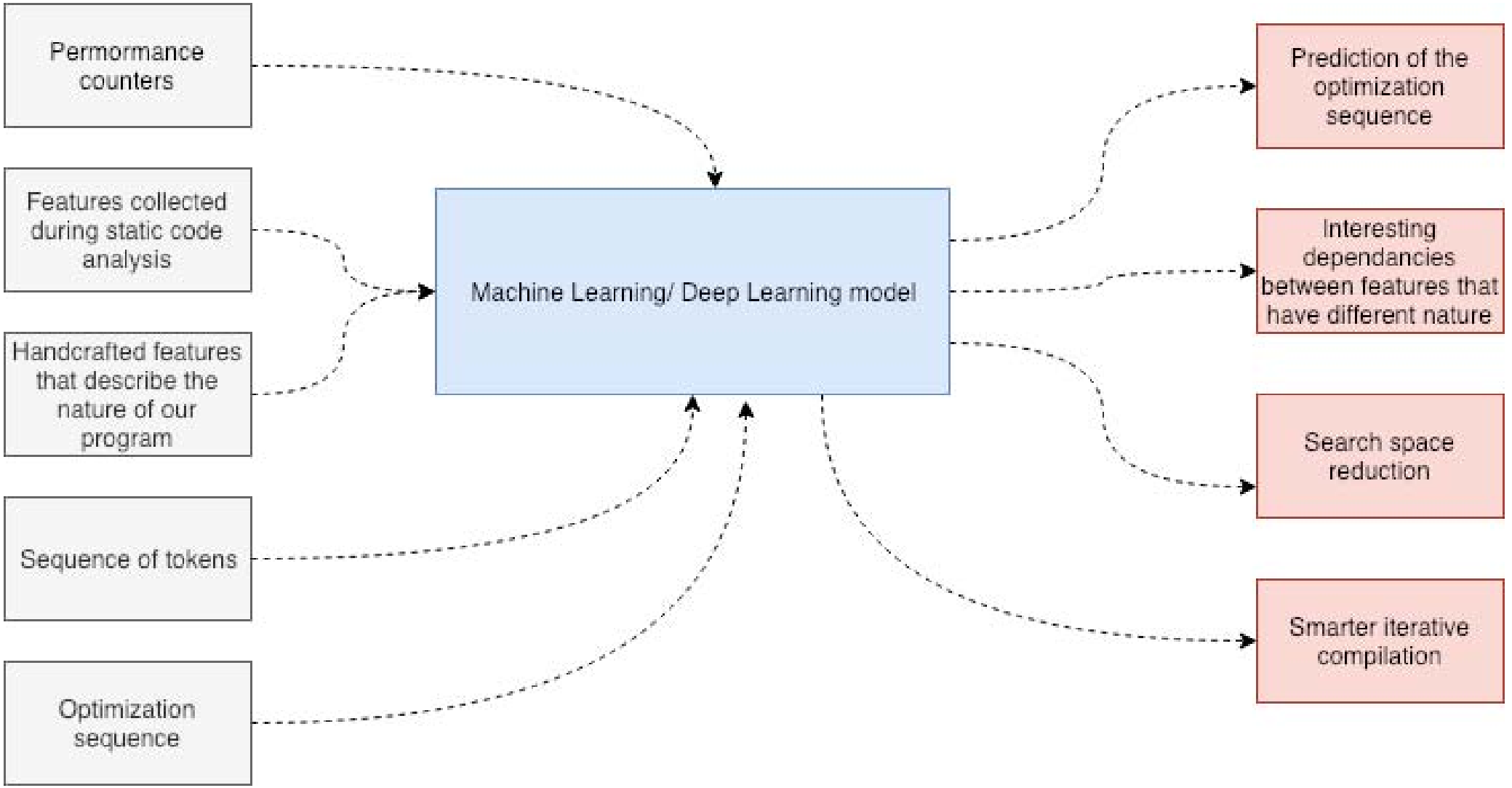
Naive Matmul Complexity

- Let’s consider a naive matrix multiplication algorithm and only 2 transformations (unrolling, tiling, interchange) for square 1024x1024 matrices
- Number of transformation sequences: ~ 6.59*10^12 (if all transformations are legal, but they might be not)
- If we take order of transformations into account: 5! * 6.59*10^12

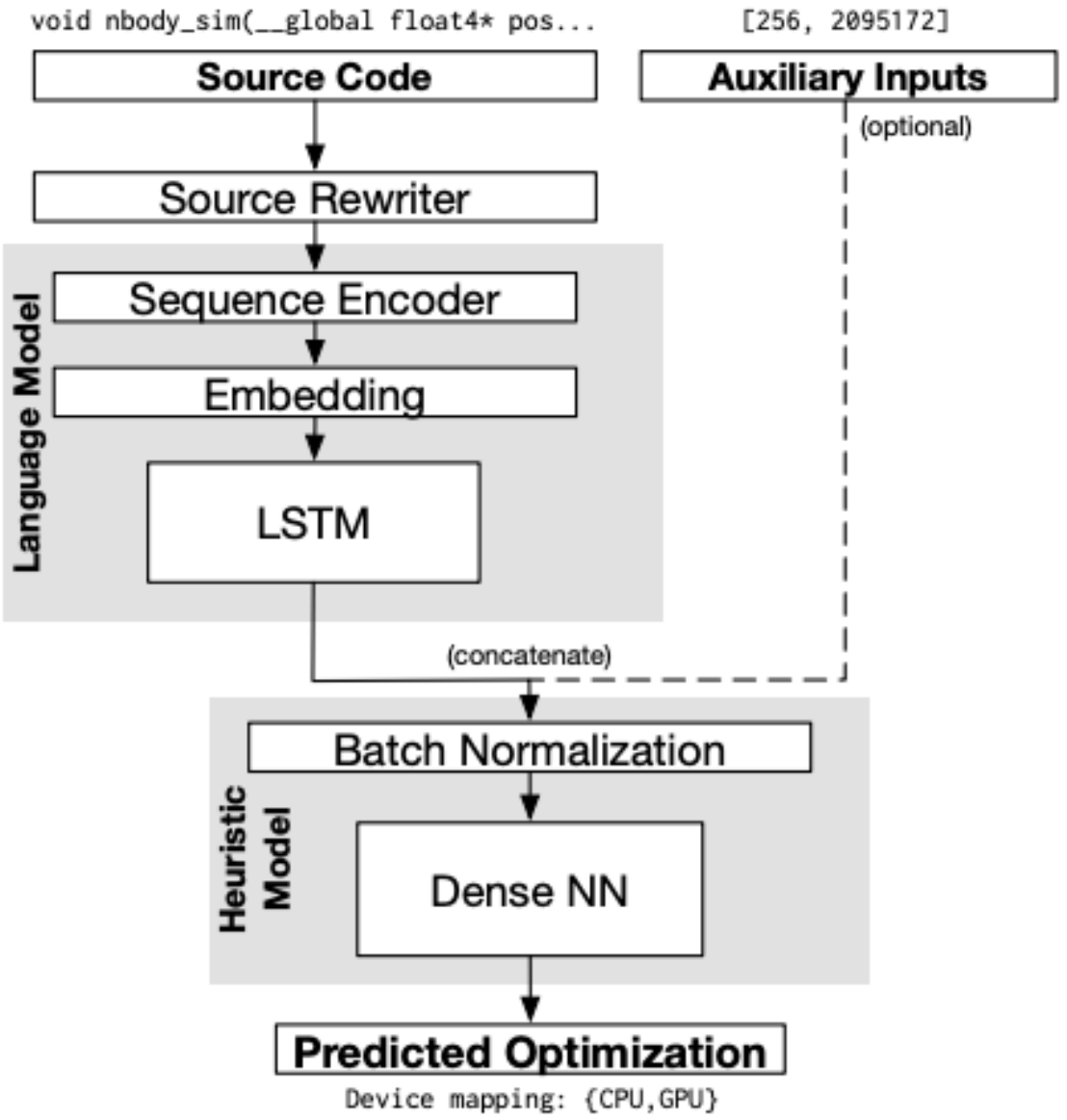
Why Machine Learning?

- A lot of data from different sources (raw code, intermediate representation, performance counters, etc)
- Many heuristics work successfully in this domain which is a prerequisite for smarter generalization
- Large optimization space

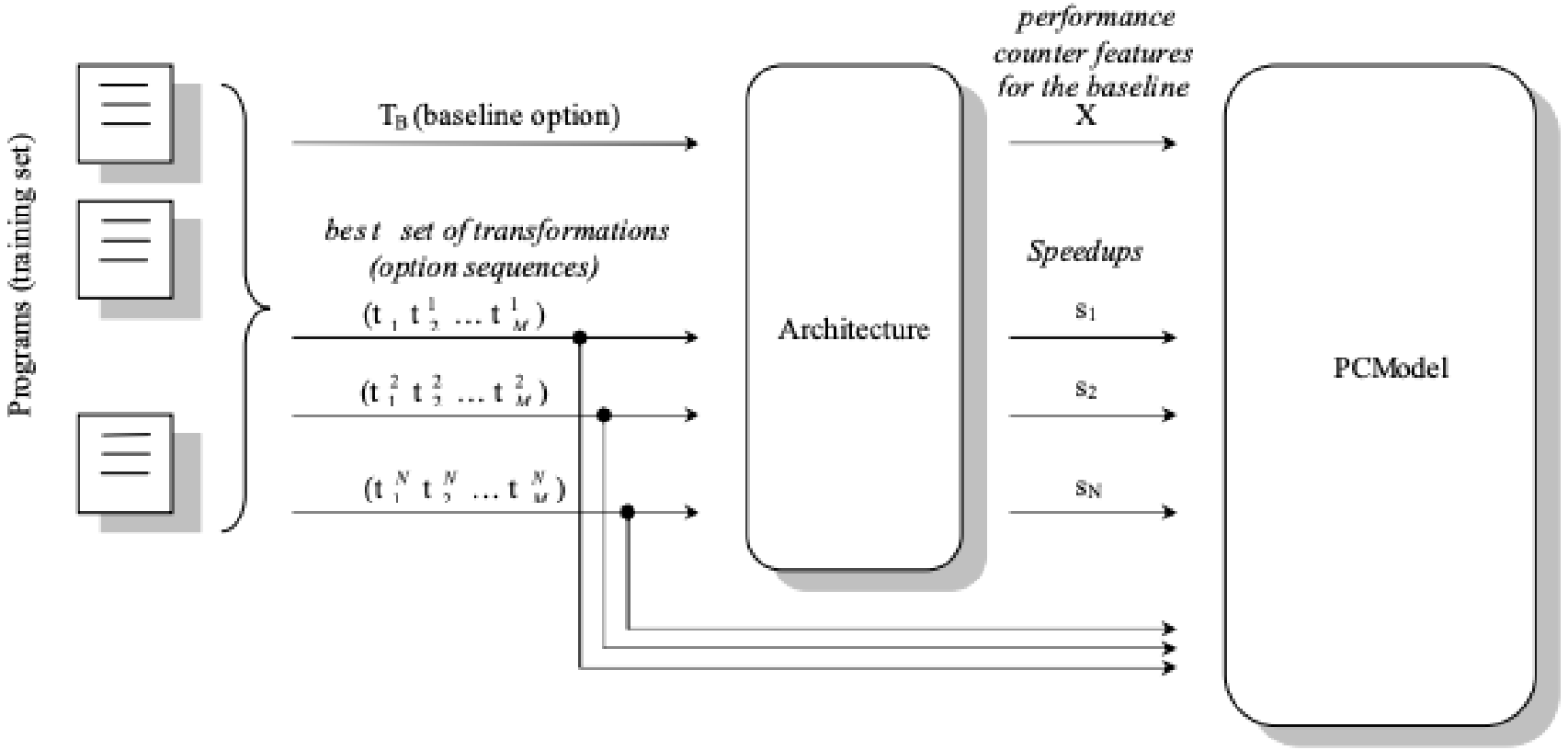
What inputs can we use and what can we get?

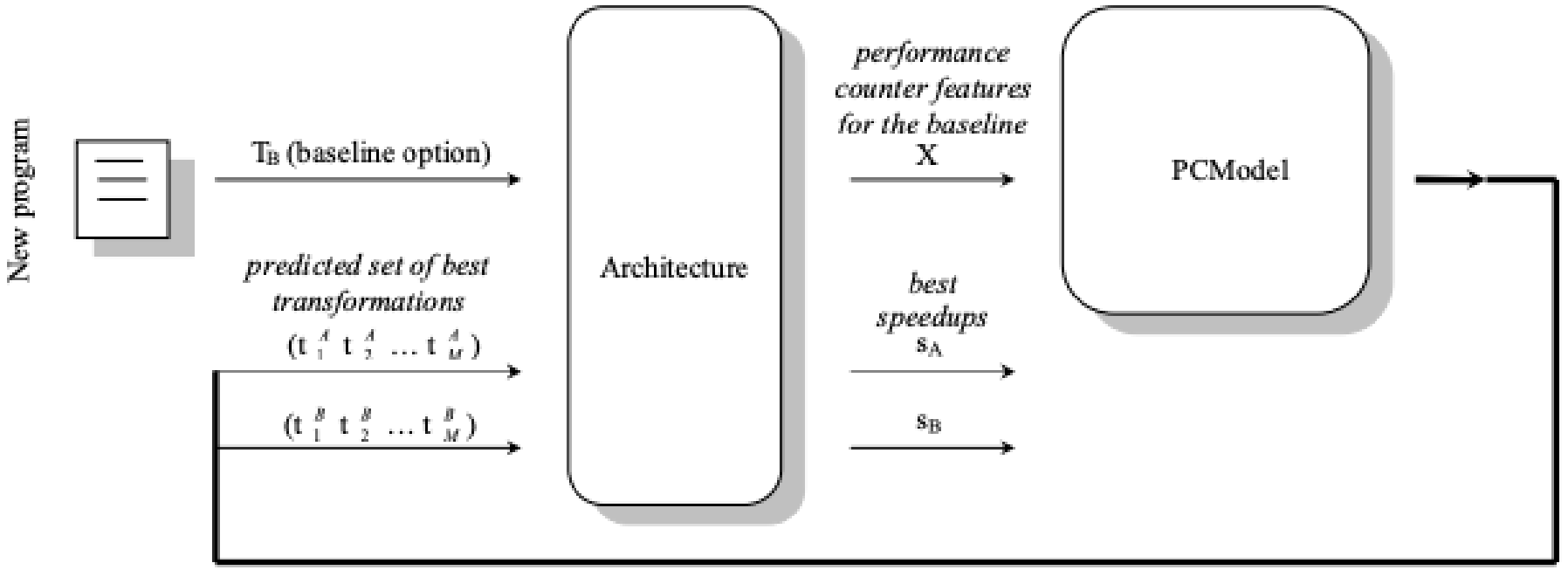


Cummins C. et al. End-to-end deep learning of optimization heuristics



Cavazos J. et al. Rapidly selecting good compiler optimizations using performance counters




New program
T_B (baseline option)
predicted set of best transformations
(t^A_1 t^A_2 ... t^A_M)
(t^B_1 t^B_2 ... t^B_M)
Architecture
performance counter features for the baseline
X
best speedups
s_A
s_B
PCModel

Frameworks/ Tools

- PIPS: source-to-source compilation framework for analyzing and transforming C and Fortran programs https://pips4u.org/
- Locus: language for program optimization *
- PAPI: Performance Application Programming Interface http://icl.utk.edu/papi/
- Scikit-learn: Machine Learning library for Python https://scikit-learn.org/stable/
- PyTorch: Deep Learning framework for Python https://pytorch.org/

* Thiago S. F. X. Teixeira, Corinne Ancourt, David Padua, and William Gropp. 2018. Locus: A System and a Language for Program Optimization

Thank you for your attention!