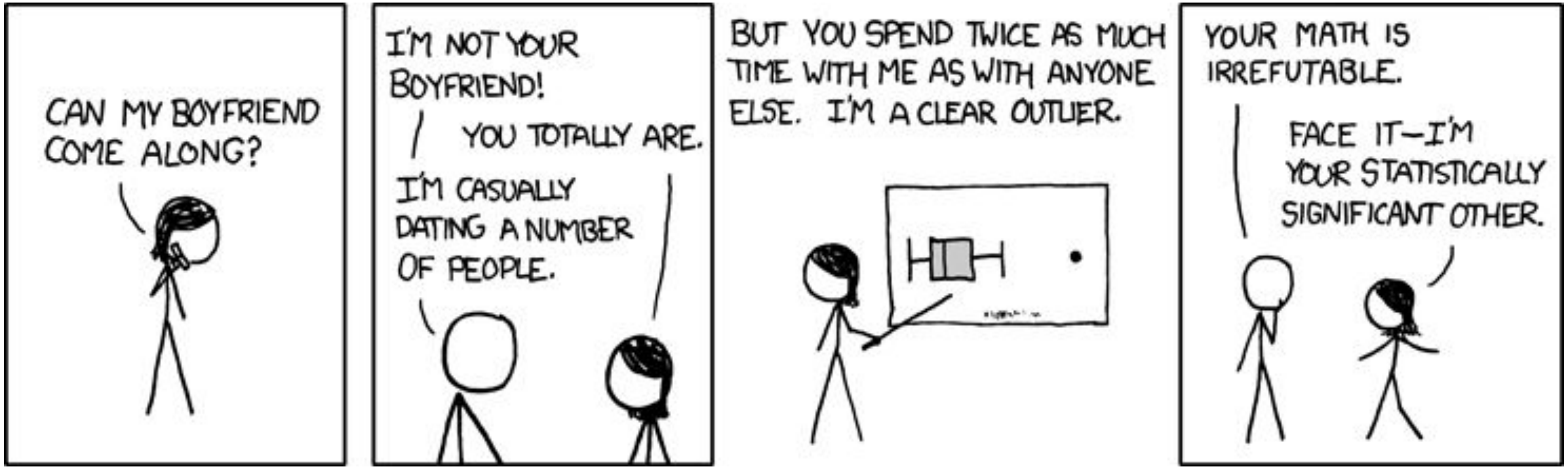
Machine Learning in Compiler Optimization

Authors: Zheng Wang and Michael O'Boyle

Qi Zhang, Shengpu Tang, Yanqi Wang, Yiting Shen

Main Takeaways

- Machine Learning (ML) based compilation is a trustworthy and exciting direction for compiler research.
- Data is fuel to ML based research.



Why do we need ML?

- Compilers have two jobs - translation and optimization
  - Compiler writers develop heuristics with the hope of improving performance
  - ML can serve as a predictor of the optima
- Machine-Learning Compilation
  - Automation!
  - Evidence-based science

OpenCL Kernel code snippet

```
kernel void square(global float* in, global float* out){
    int gid = get_global_id(0);
    out[gid] = in[gid] * in[gid];
}
```

(a) Original OpenCL kernel

```
kernel void square(global float* in, global float* out){
    int gid = get_global_id(0);
    int tid0 = 2*gid + 0;
    int tid1 = 2*gid + 1;
    out[tid0] = in[tid0] * in[tid0];
    out[tid1] = in[tid1] * in[tid1];
}
```

(b) Code transformation with a coarsening factor of 2

Example: Best Thread Coarsening Factor

Thread Coarsening Pros & Cons

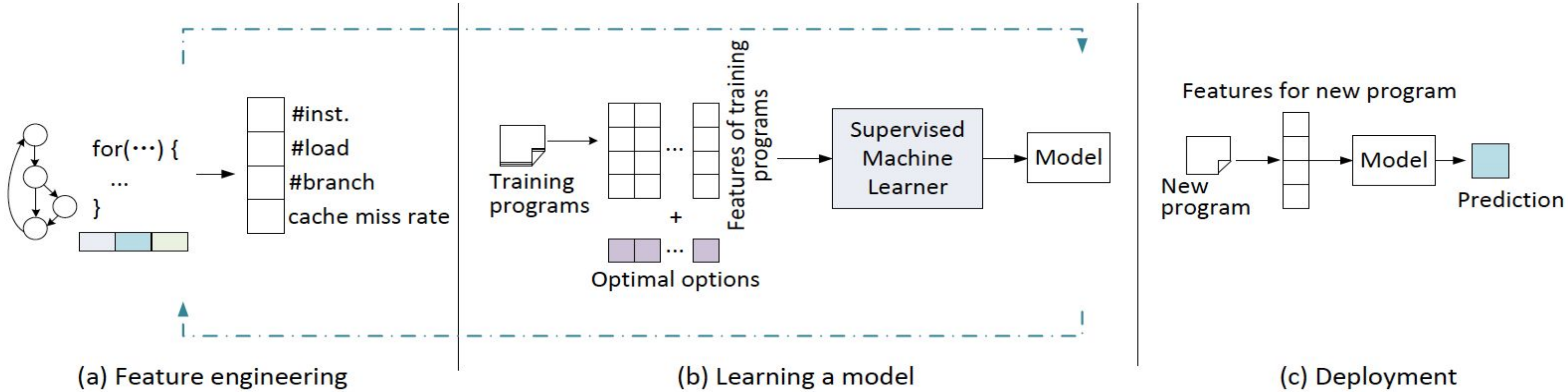
Pros:

- Increase instruction-level parallelism
- Reduce # of memory-access operations
- Eliminate redundant computations

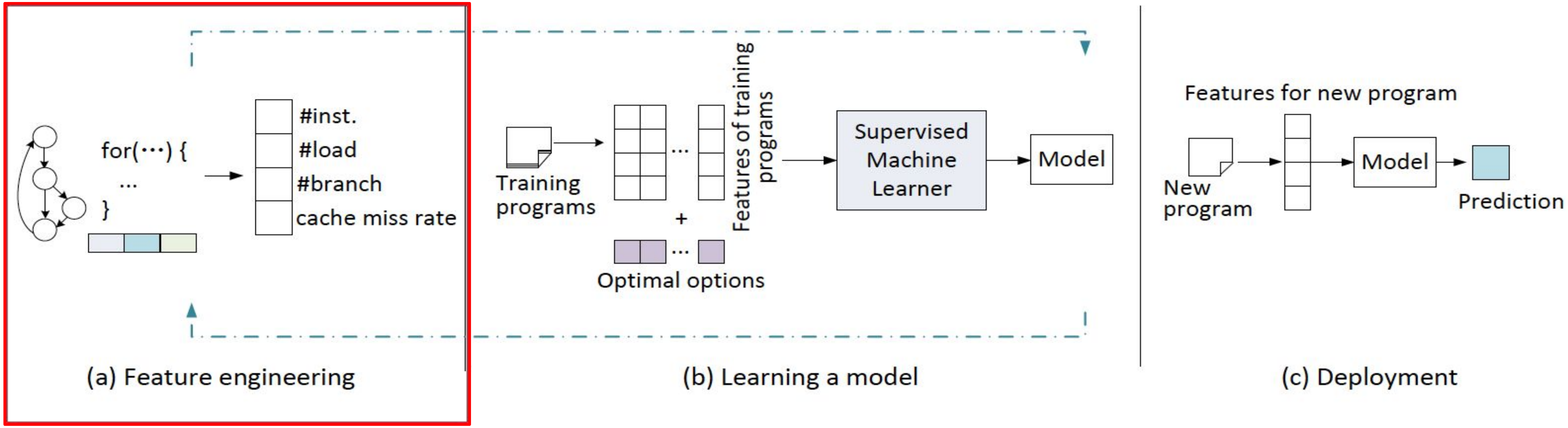
Cons:

- Reduce the total amount of parallelism
- Increase the register pressure

Architecture Overview



Stage 1: Feature Engineering



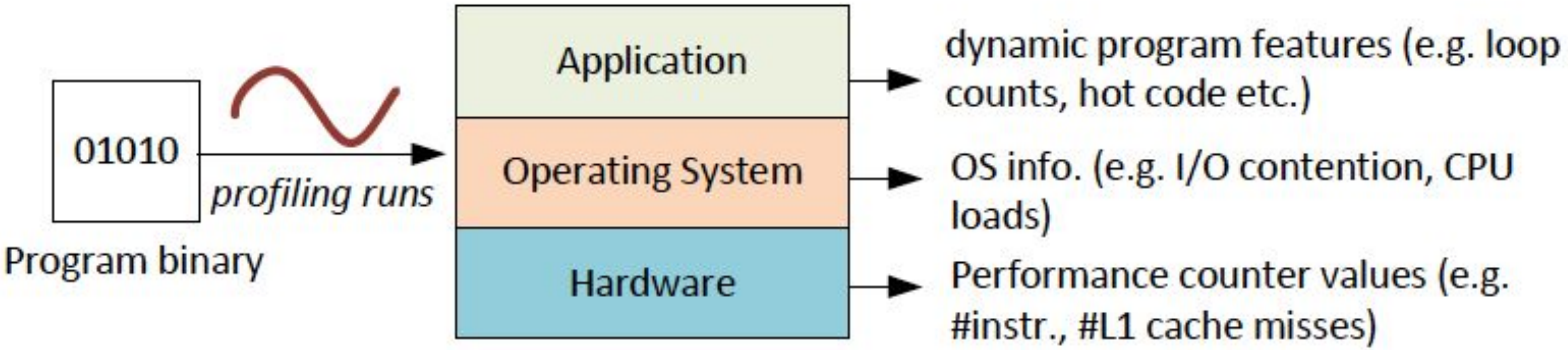
Static Code Features

Extracted from the intermediate representations

Description	Examples
Arithmetic instructions	#floating point instr., #integer instr., #method call instr.
Memory operations	#load instr, #store instr.
Branch instructions	#conditional branch instr, #unconditional branch instr
loop information	#loops, loop depth
parallel information	#work threads, work group size

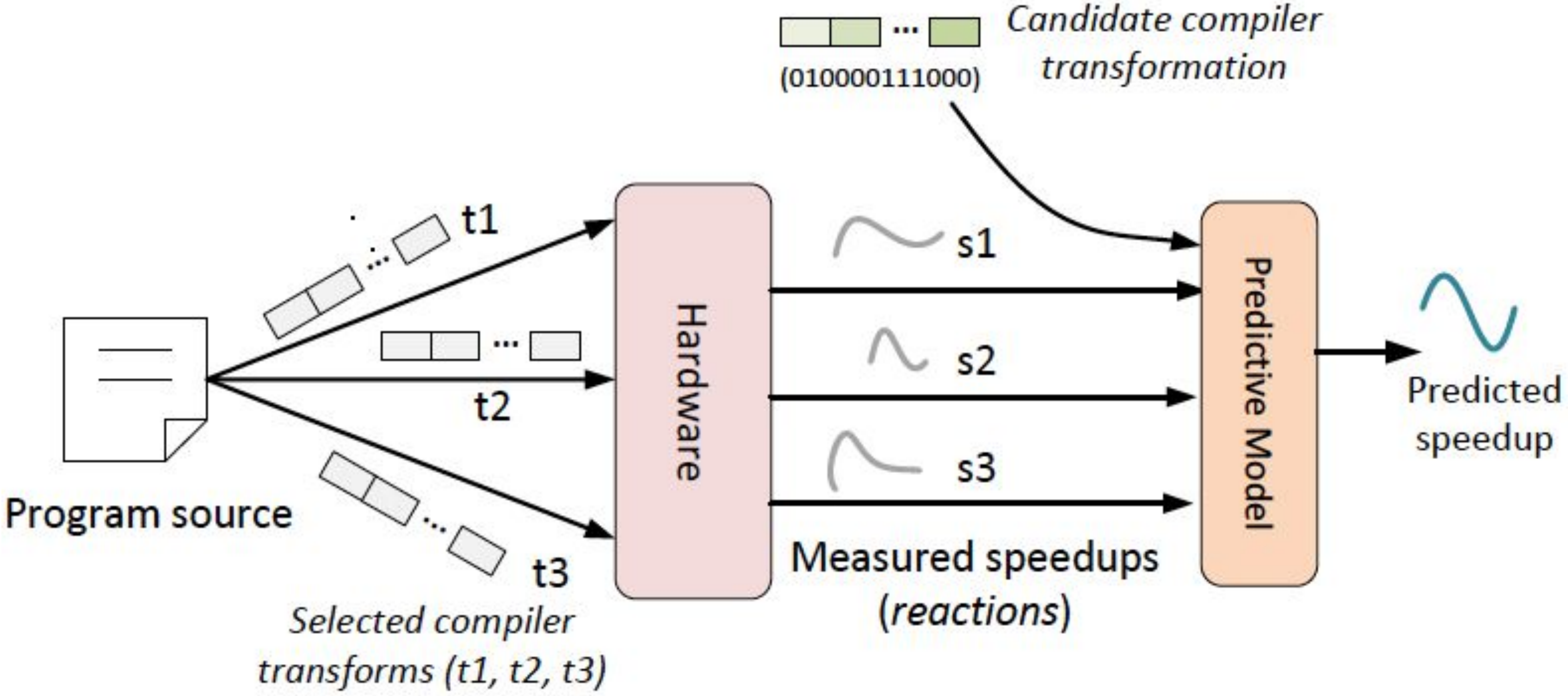
Dynamic Features

Extracted from multiple layers of the runtime environment.



Reaction Based Features

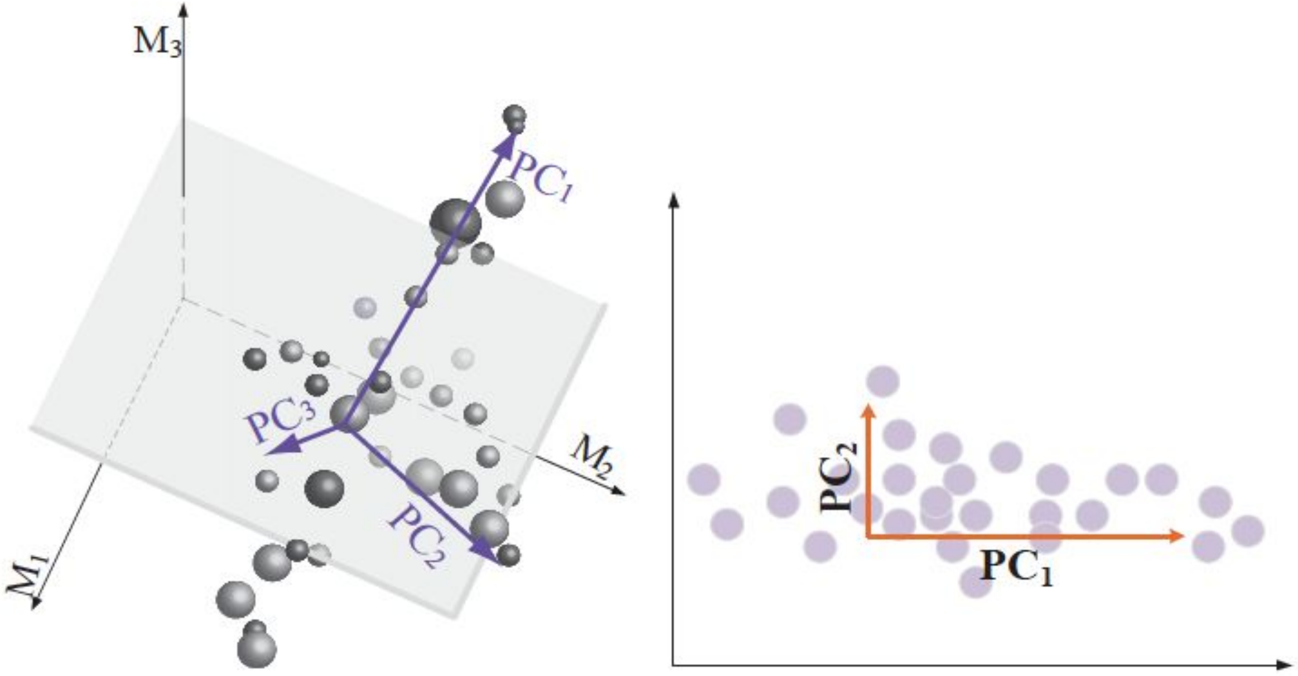
Carefully selected compiler options.



Feature Selection and Dimension Reduction

Problem: Too many features -> need a lot more training examples

Solution: Feature space dimension reduction



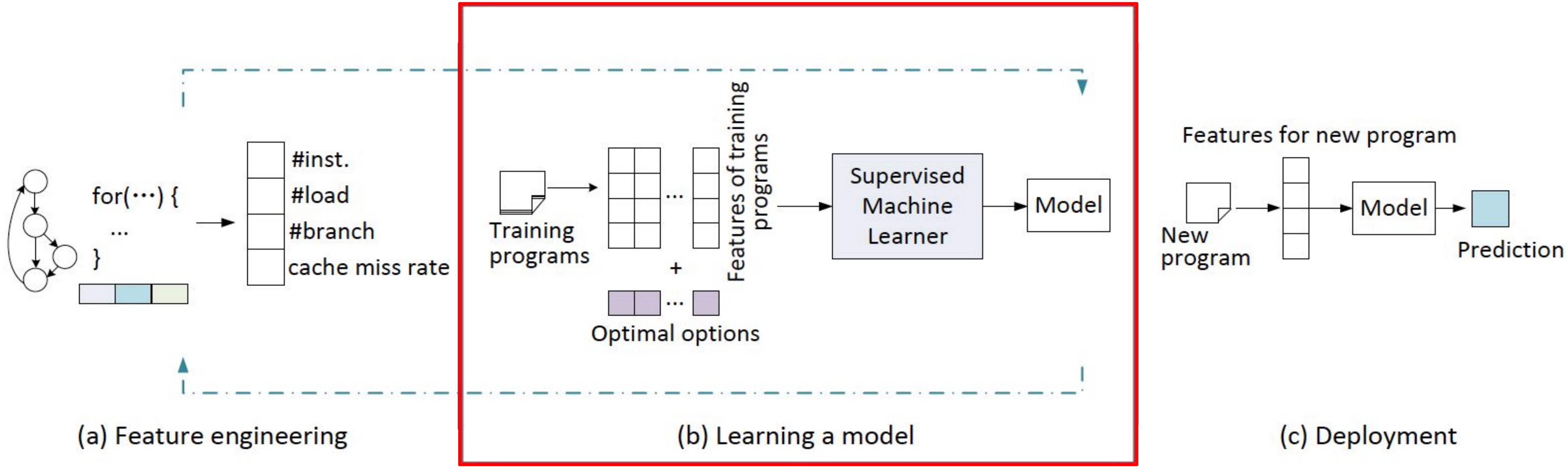
(a) Original feature space (b) Reduced feature space

Example: Best Thread Coarsening Factor

```
kernel void square (global float* in,
                    global float* out) {
    int gid = get_global_id(0);
    out[gid] = in[gid] * in[gid];
    ...
}
```

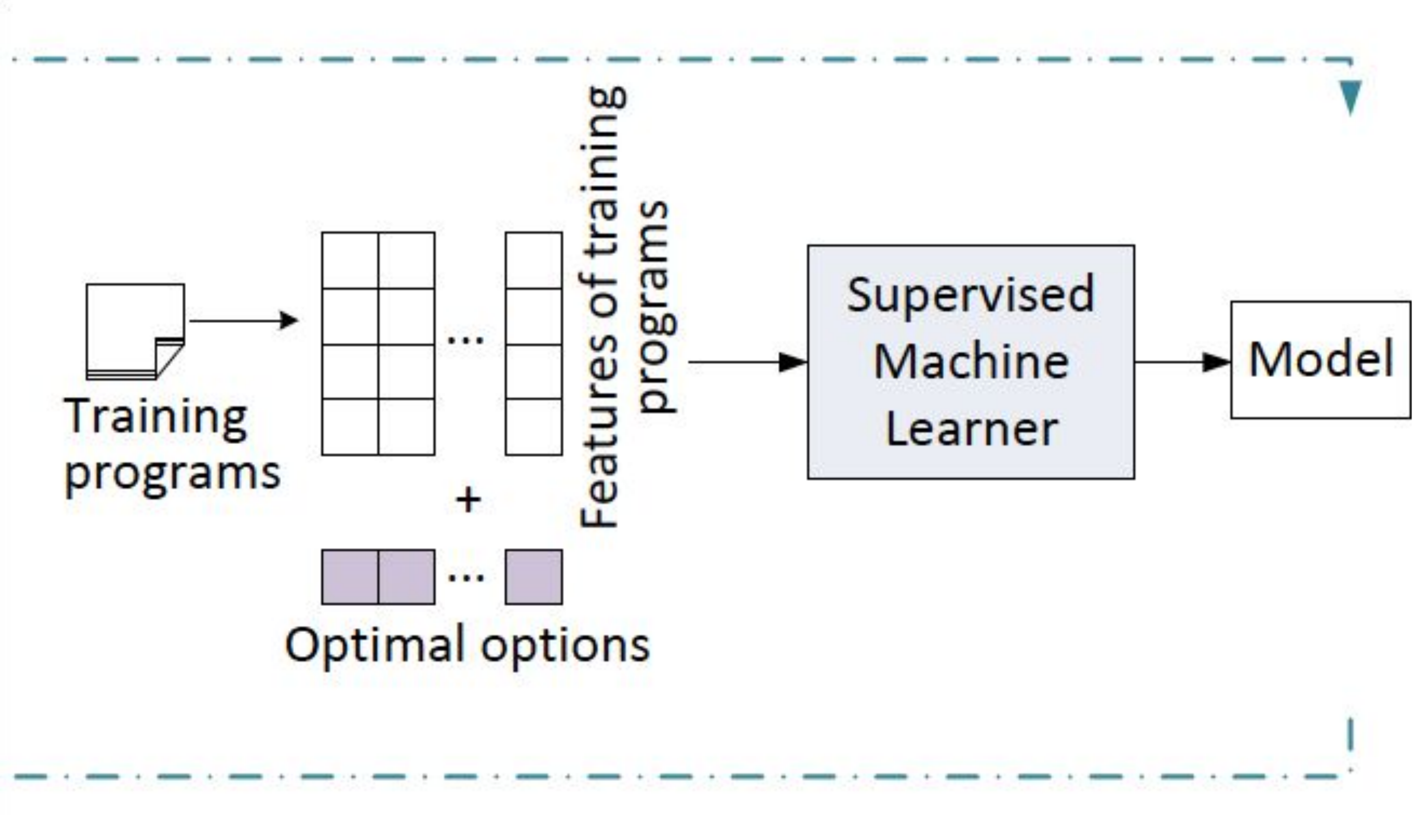
Feature Description	Feature Description
# Basic Blocks	# Branches
# Divergent Instr.	# Instrs. in Divergent Regions
(# instr. in Divergent regions)/(# total instr.)	# Divergent regions
# Instrs	# Floating point instr.
Avg. ILP per basic block	(# integer instr.) / (# floating point instr.)
# integer instr.	# Math built-in func.
Avg. MLP per basic block	# loads
# stores	# loads that are independent of the coarsening direction
# barriers	

Stage 2: Learning a Model



Models

Supervised vs. Unsupervised

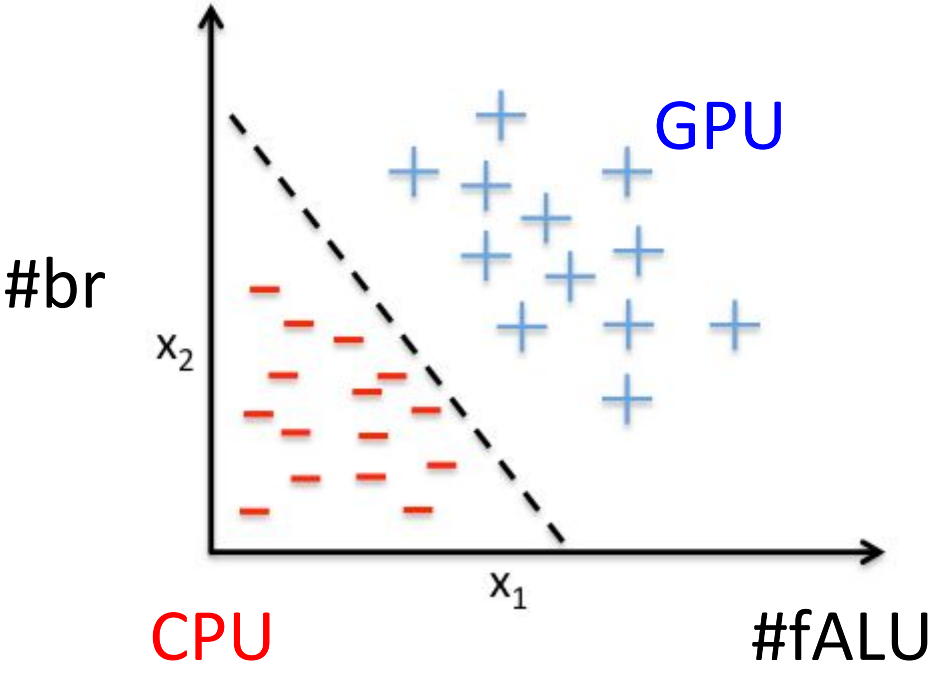


Supervised Learning: Classification

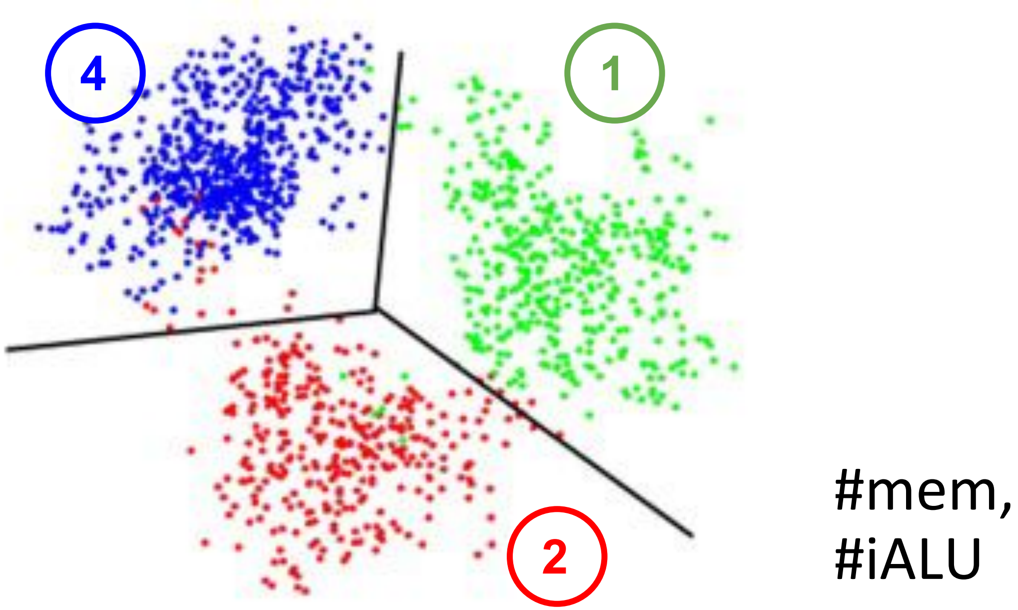
Goal: Separate different types of program by a decision boundary

Given a new unseen program, knows what to do

Best device to run



Best coarsening factor

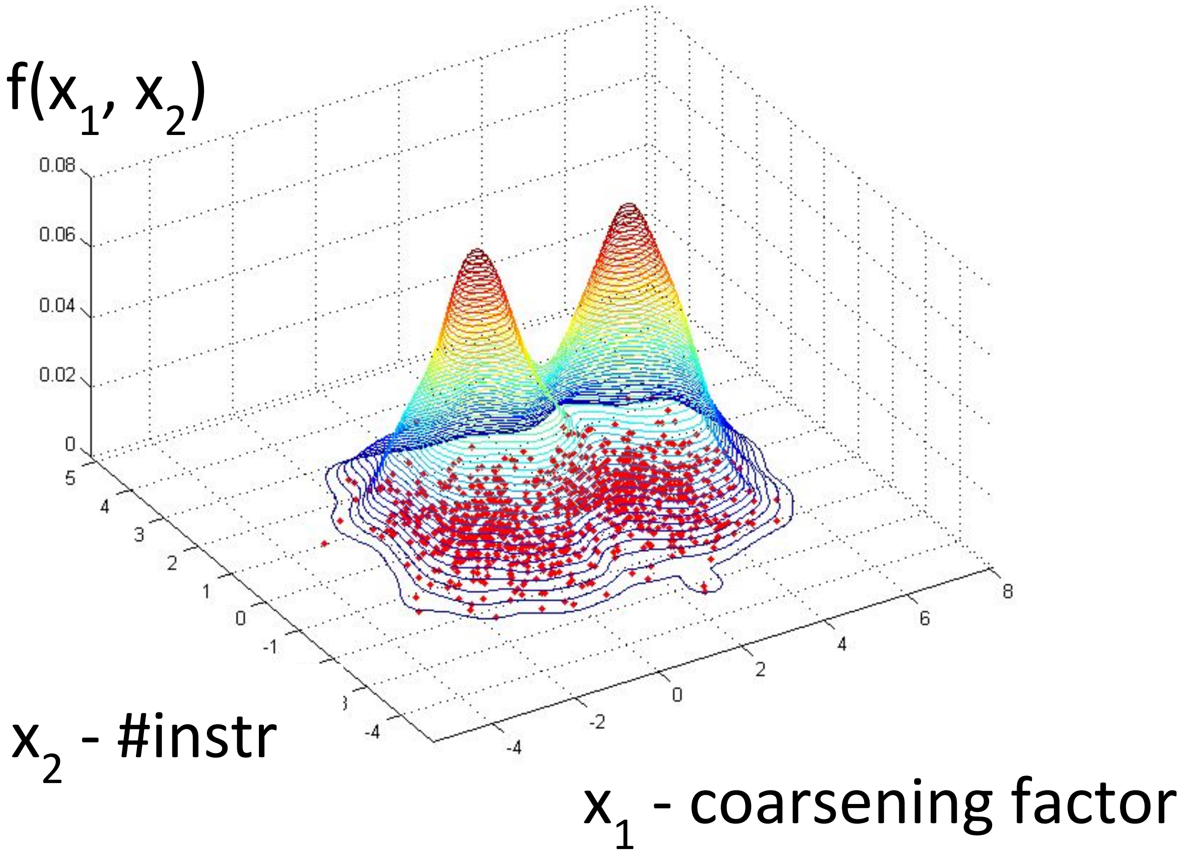


Supervised Learning: Regression

Goal:

Learn a function to predict

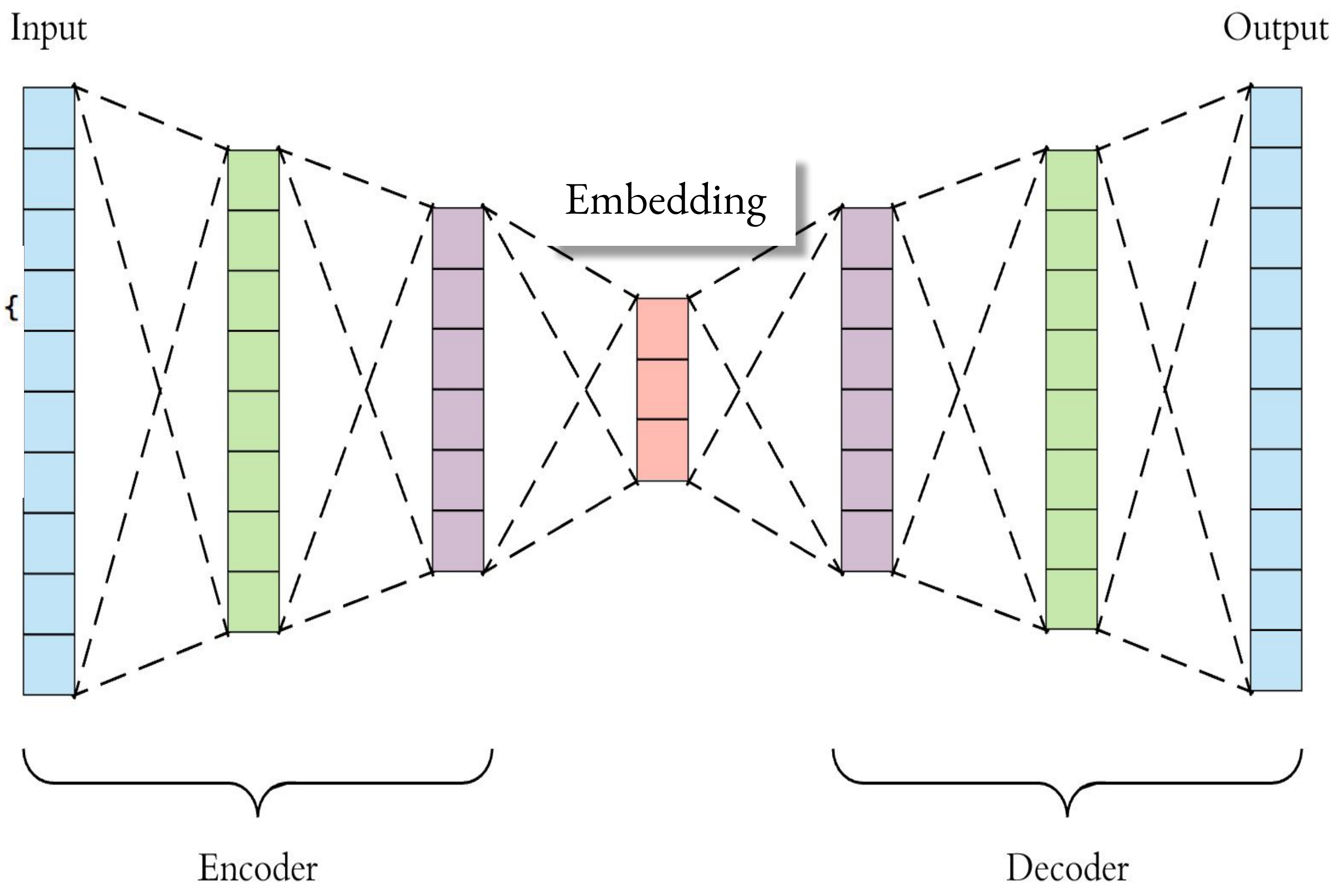
- Power consumption
- Latency
- Exec. time



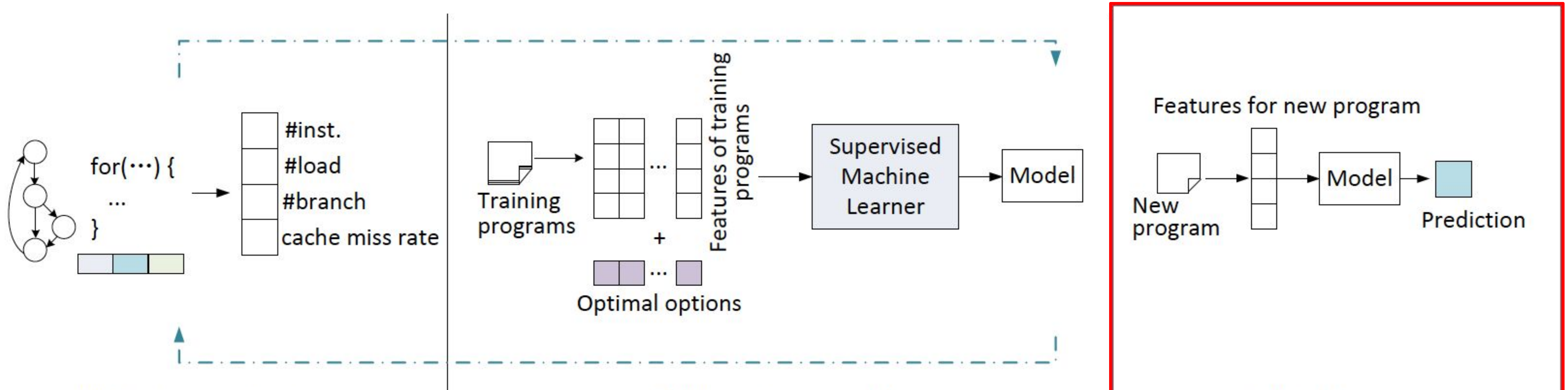
Unsupervised Learning: Auto-encoder

Learn best feature representation

```
kernel void square (global float* in,
                    global float* out) {
    int gid = get_global_id(0);
    out[gid] = in[gid] * in[gid];
    ...
}
```



Stage 3: Deployment

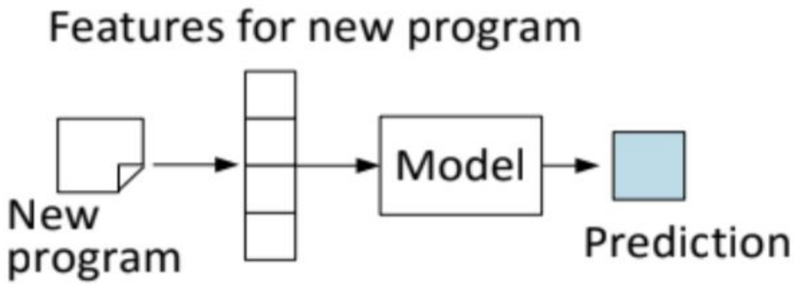


(a) Feature engineering (b) Learning a model (c) Deployment

Deployment

How to utilize the learned model:

1. Extract the features of the input program
2. Feed the extracted feature values to the learned model to make a prediction
3. Apply the predicted results



Application

What hardware problems can machine learning solve?

- ➢ Optimize sequential programs
  - target a fixed set of compiler options
  - represent the optimization problem as a multi-class classification problem – where each compiler option is a class.

- ➢ Optimize parallel programs
  - provide the potential for high performance and energy-efficient computing

Optimize Sequential Programs - Example

Goal: Predict the optimal loop unroll factor

Approach:

- Decision tree based model
    - Multi-class Classification
    - Label: loop unroll factor $l \in [0, 15]$, 16 classes in total

Optimize Parallel Programs - Example

Goal: Predict the scheduling policy

Approach:

- Regression based neural net
- SVM classifier

Challenges & Limitations

- Training cost
- Garbage in, garbage out
- Unable to invent new program transformations
- Unable to prove the validity