Towards Autonomous Compiler Design Using Machine Learning

Zheng Wang
School of Computing
University of Leeds

Work in collaboration with

Chris Cummins

Zhanyong Tang

Pavlos Petoumenos

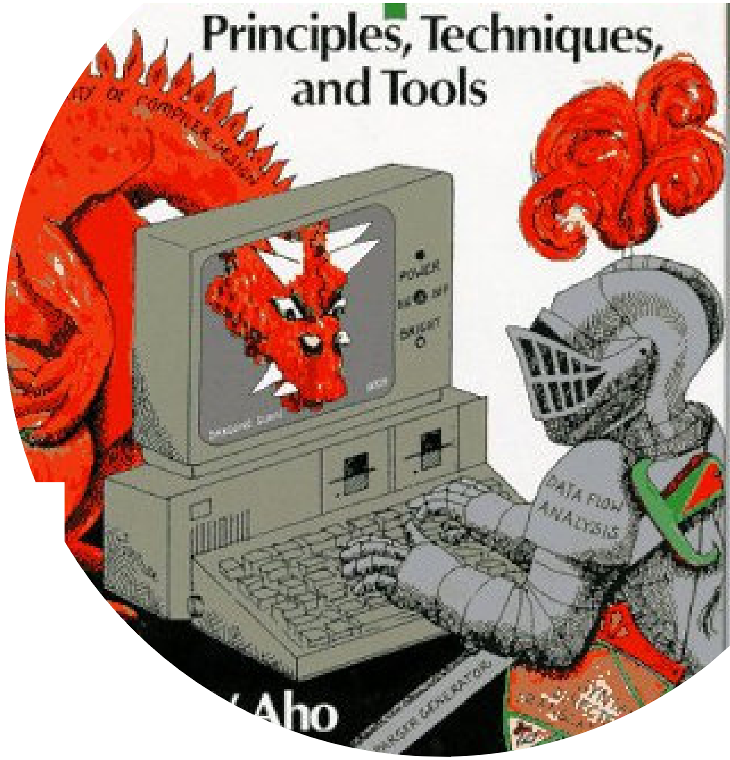
Guixin Ye

Hugh Leather

William Ogilvie

Huanting Wang

Better compiler = happier users



- Faster binary = better user experience 🙂
- Lower hardware requirement or power = £ saved

Writing compiler optimisation heuristics **by hand** is time-consuming and hard

- **Machine learning** promises:
  - **Better compilers**
  - **Less development cost**

Learn to search the optimisation space

ML algorithm

```
a[1] = b[1] + c[1]
a[2] = b[2] + c[2]
a[3] = b[3] + c[3]
a[4] = b[4] + c[4]
a[5] = b[5] * c[5]
```

State

Choose a (valid) transformation

{a[1], a[2]}, {a[32], a[43]}, {a[3], a[4]}

```
a[1:2] = b[1:2] + c[1:2]
a[3]   = b[3] + c[3]
a[4]   = b[4] + c[4]
a[5]   = b[5] * c[5]
```

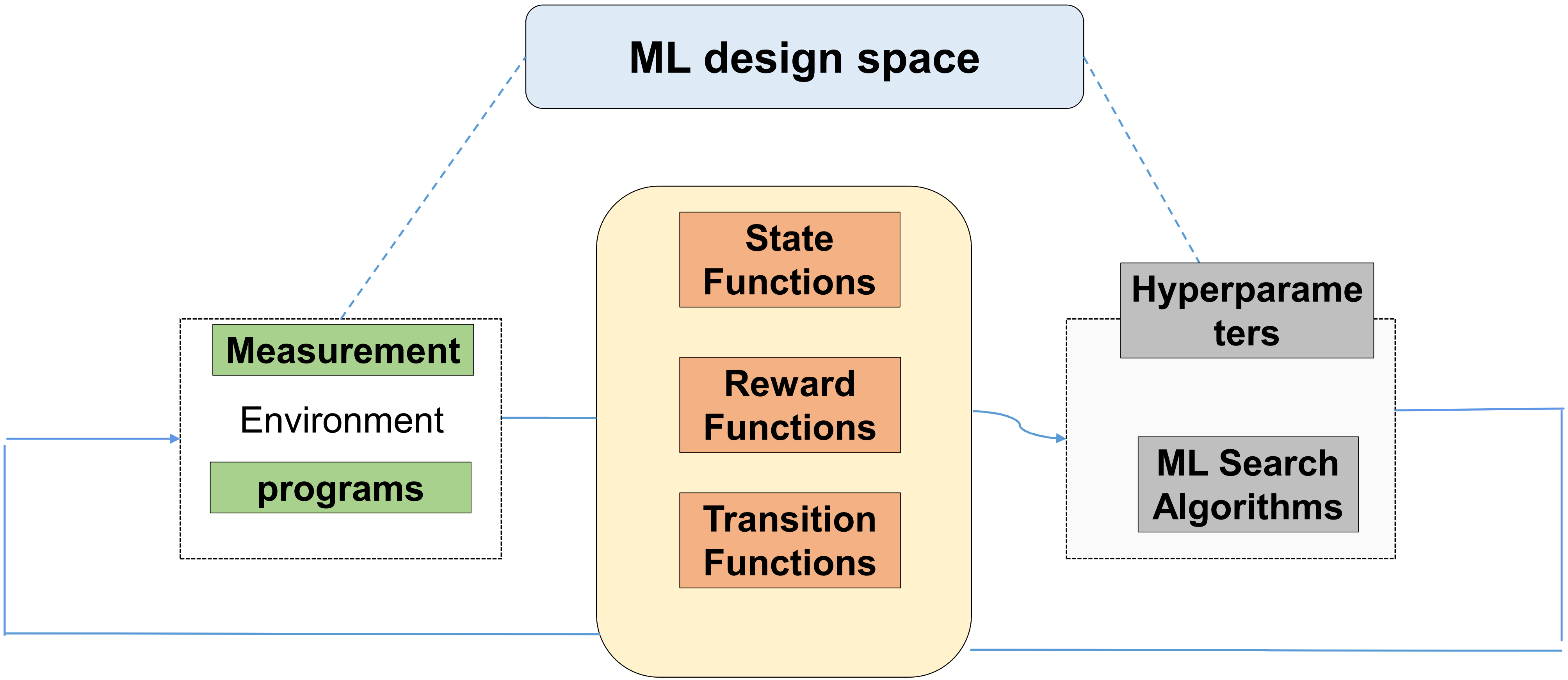
New State

Evaluate the Reward
(Speedup or code size)

Iterate

Slides adapted from Saman Amarasinghe @ MIT

ML components for compiler search



Supersonic: AutoML for compiler optimisation

Lowering the barrier of integrating ML into compilers

API

python™

Supersonic

CompilerGym

RAY

Python library

Developers describes the compiler problem

Automatically selected and Tuned ML components

H. Wang, Z. Tang, C. Zhang, J. Zhao, C. Cummins, H. Leather, Z. Wang,
Automating Reinforcement Learning Architecture Design for Code Optimization, CC 2022

User defines the search space

```
import Supersonic as ss

actions=[“-O3",  ... ]

class task(ss.PolicyInt):
        def __init__(self, benchmarks,  *arg):
            #Initialise an environment

        def run(self):
            #How to execute the compiled code

        def step(self, code, action):
            #take an action to transform/compile the code
```

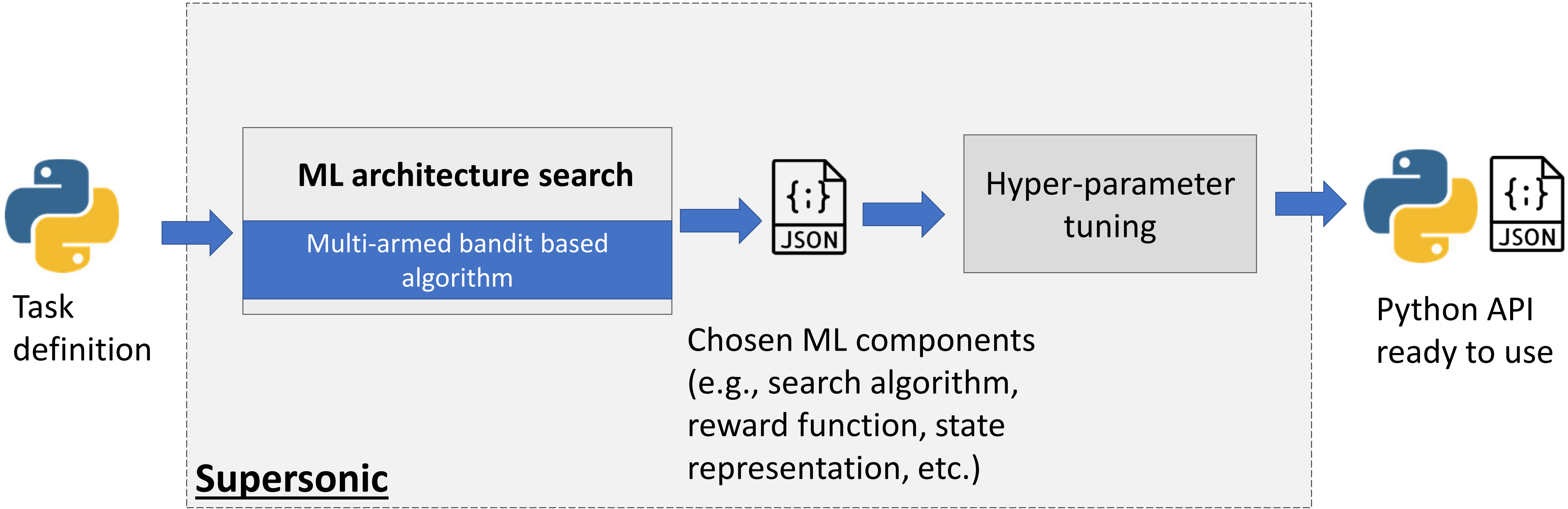
Transformation options

Benchmarks for tuning

Measurement interface

How to apply an action

Automatically find and tune the ML architecture

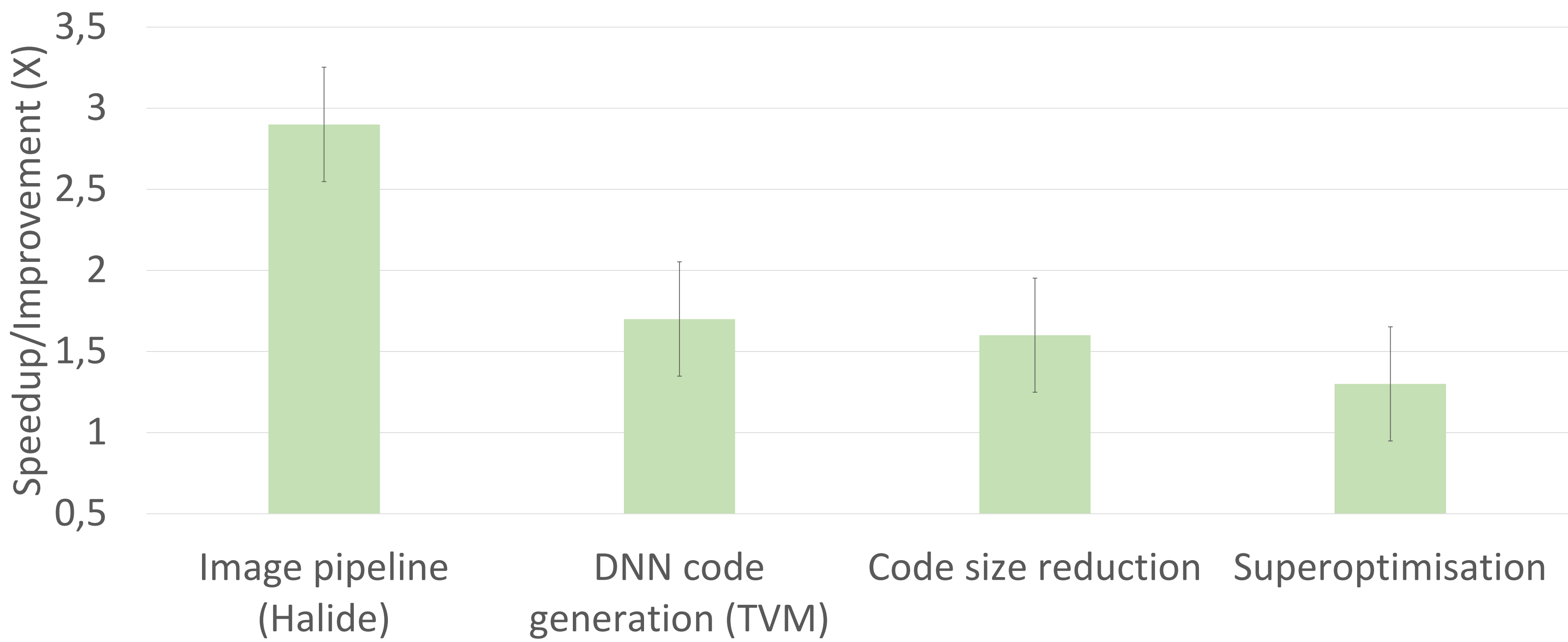


Case studies

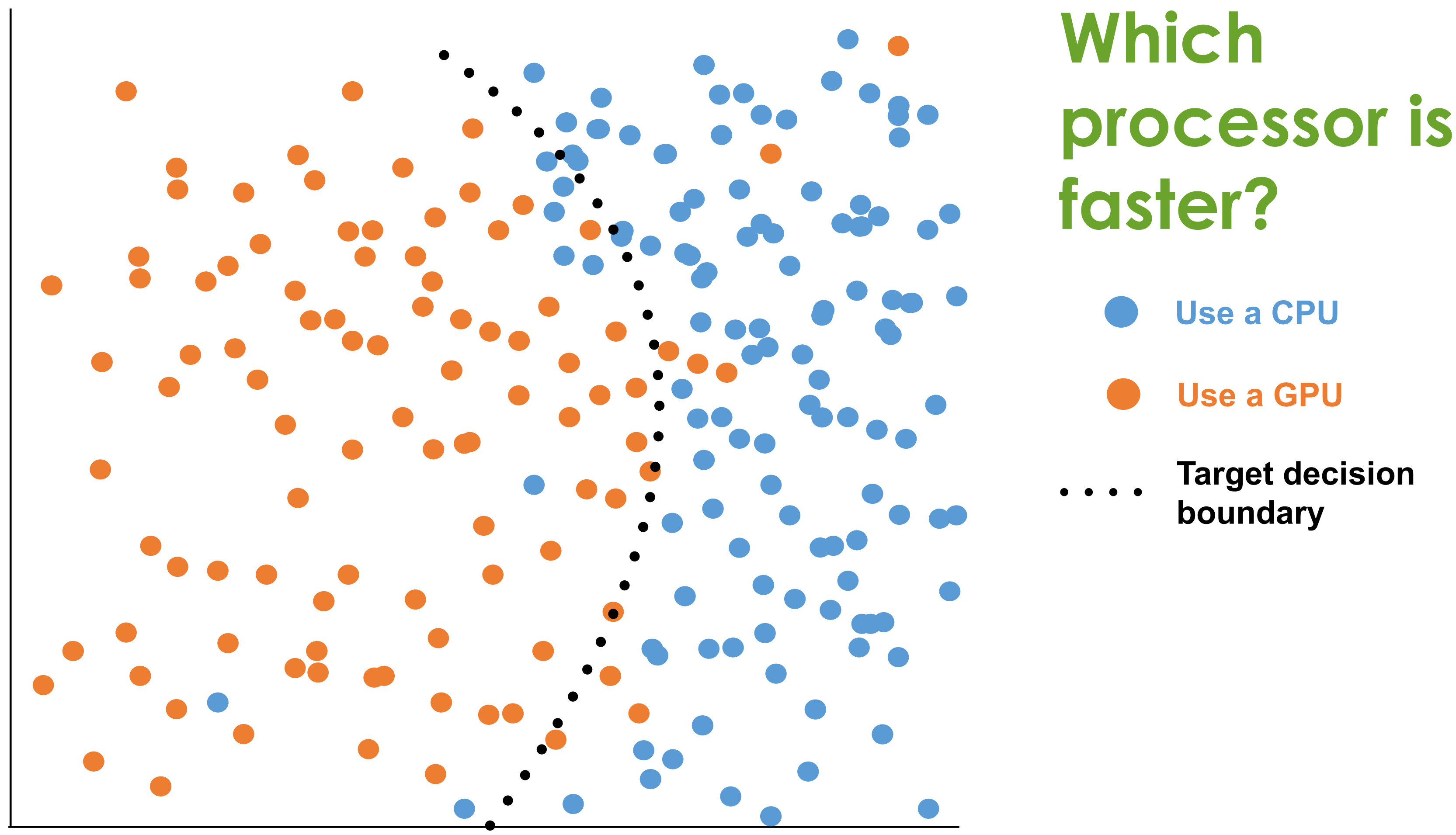
Optimizing Image Pipelines (Halide)	Neural Network Code Generation (TVM)	Code Size Reduction	Superoptimization
➢ 4x prior methods	➢ 7x prior methods	➢ 4 prior methods	➢ 3 prior methods

Compare to hand-tuned ML approaches

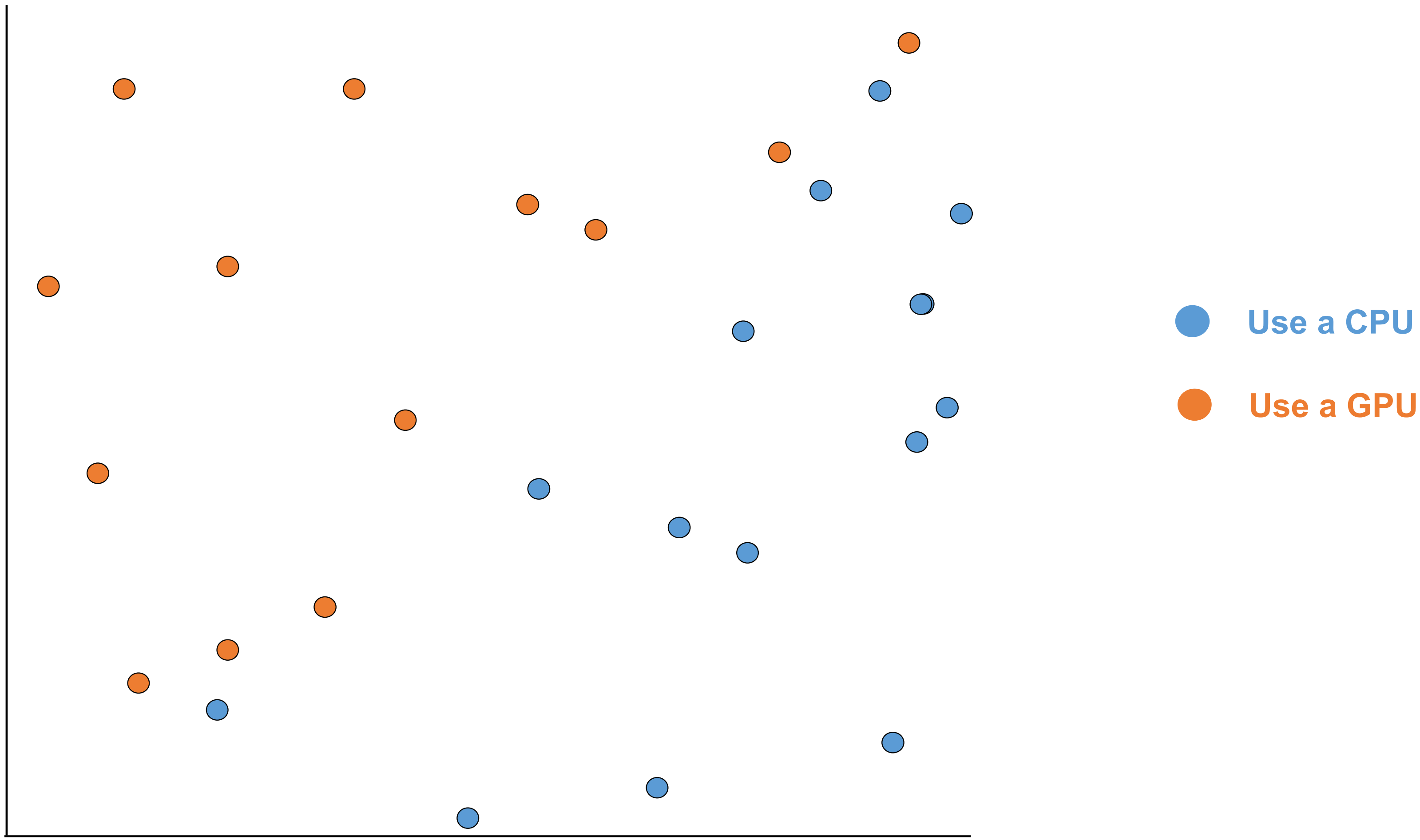
Even better than hand-crafted ML



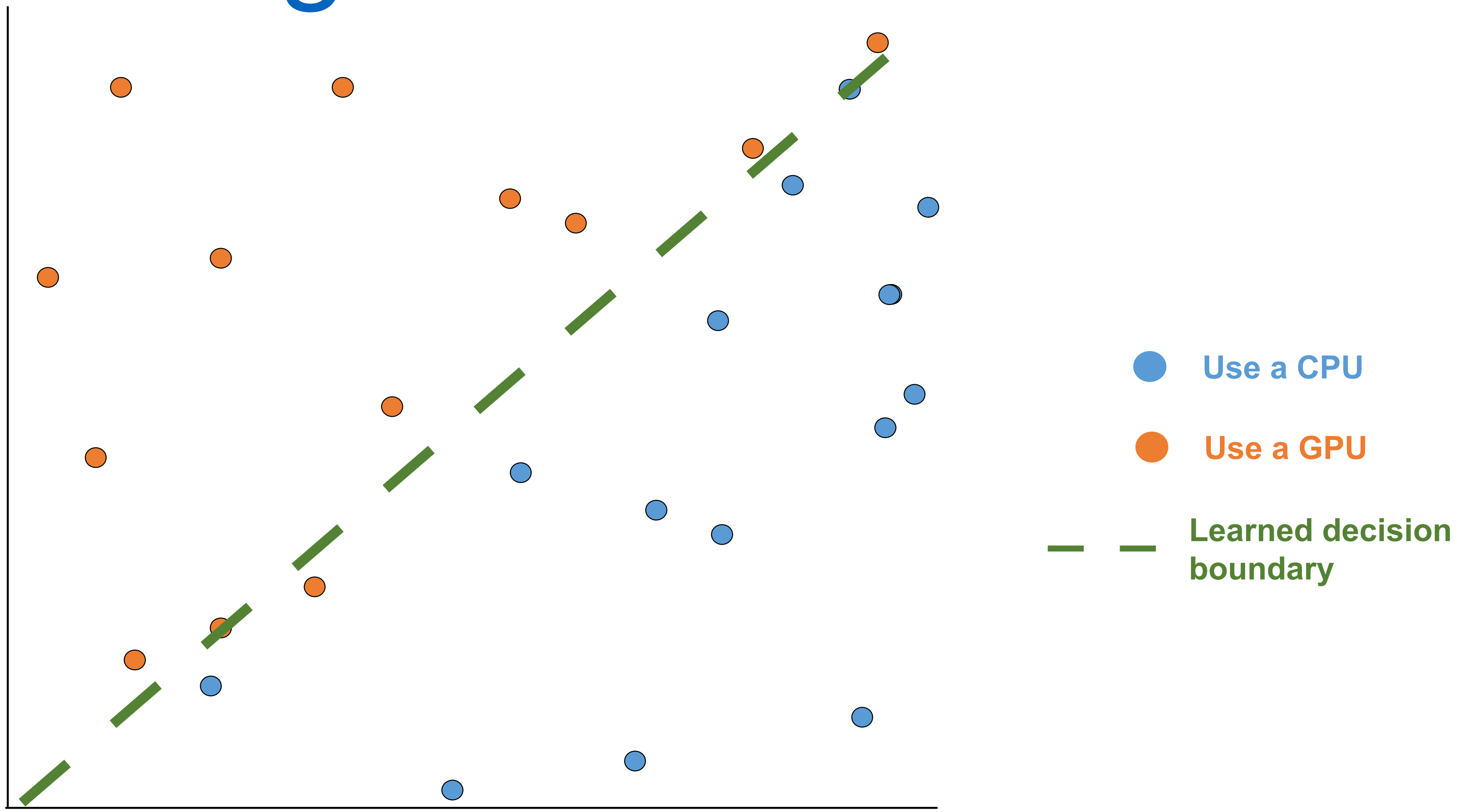
ML for predictive modelling



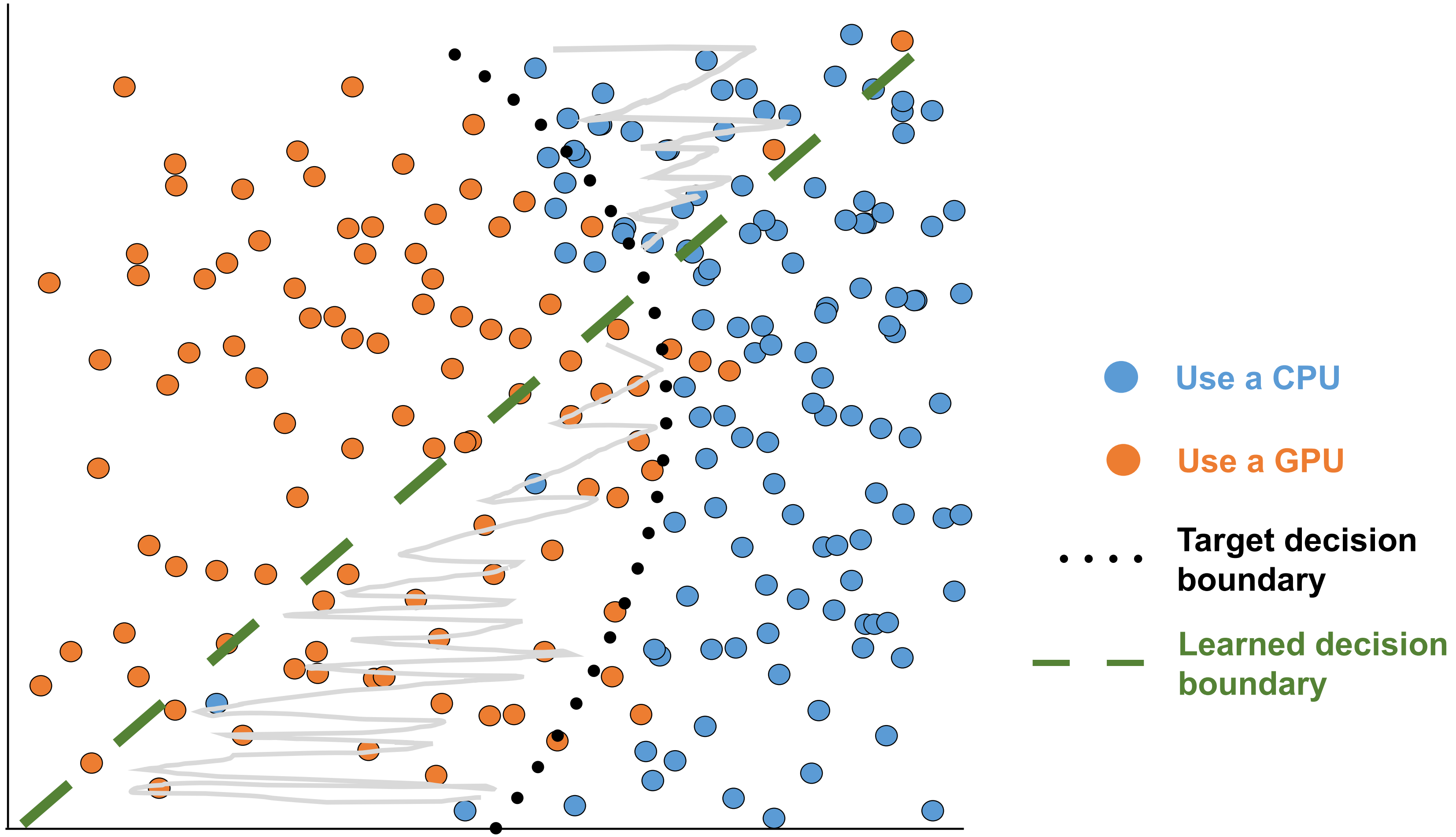
We actually have:



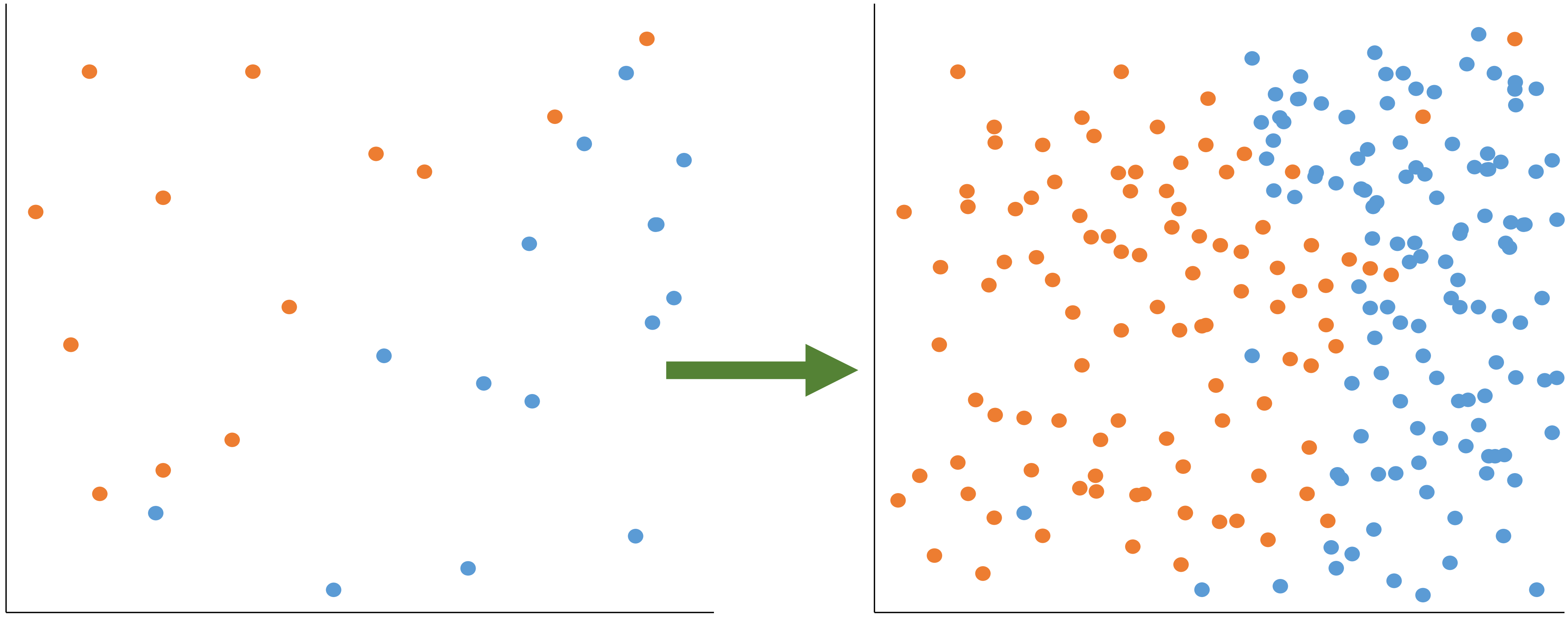
This would give us a model like:



Insufficient training data give an inaccurate model

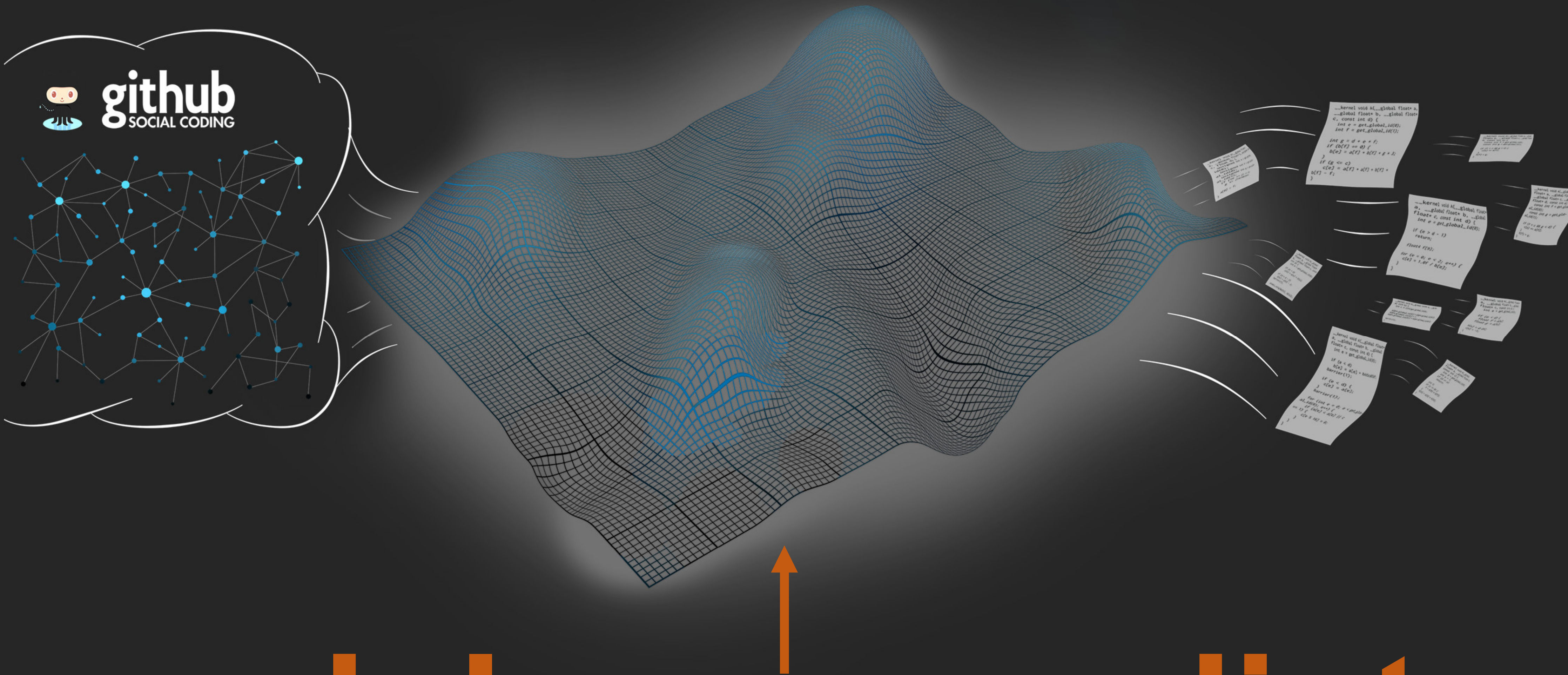


What we need



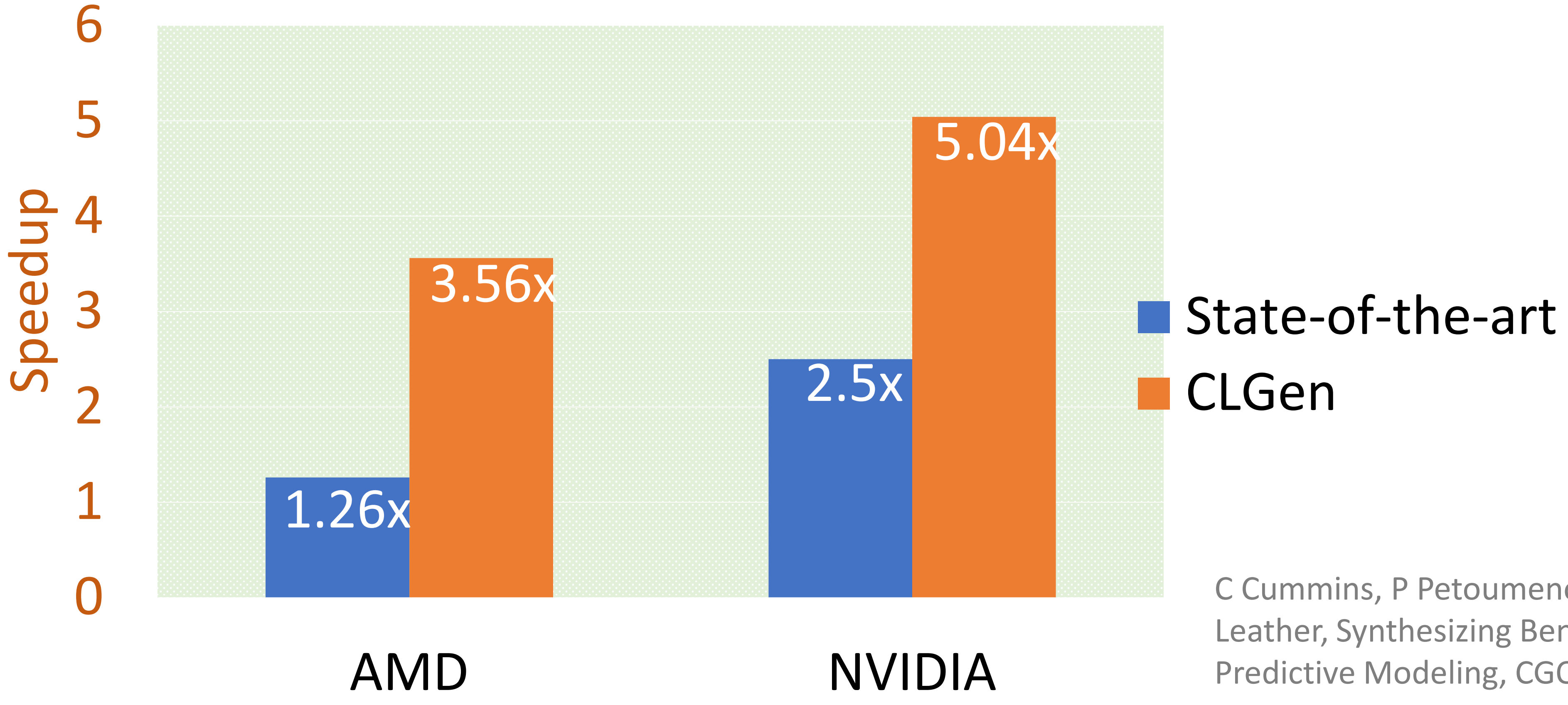
from this

to this



model source distr.

71 benchmarks, 1,000 synthetic benchmarks



C Cummins, P Petoumenos, Z Wang, H Leather, Synthesizing Benchmarks for Predictive Modeling, CGO 2017

Language agnostic

- Generate JavaScript (JS) programs to test JS compilers
- Uncovered 170+ unique bugs from

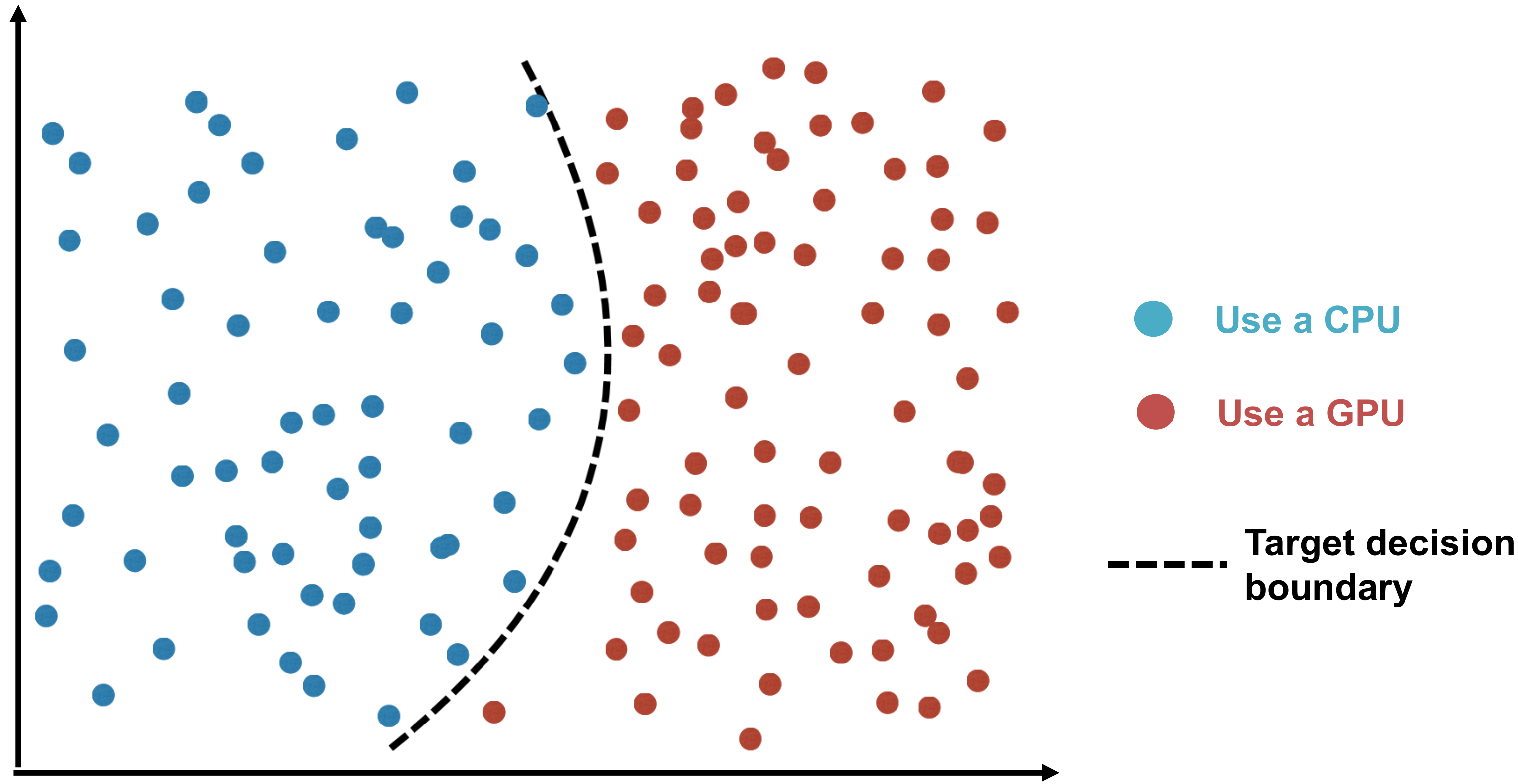
Chrome V8 , Safari JavaScriptCore, MS edge ChakraCore, Firefox SpiderMonkey, etc.

142 bugs have been verified, 120+ have been fixed.

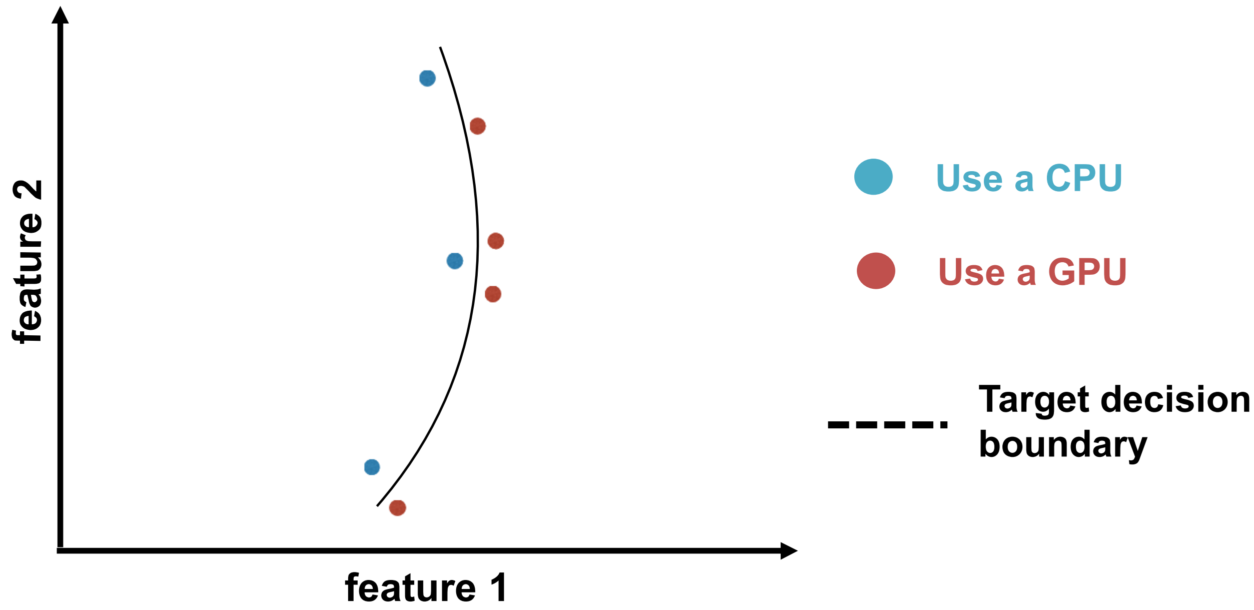
G. Ye, Z. Tang, S. Tan, X. Sun, Z. Wang, *Automated Conformance Testing for JavaScript Engines via Deep Compiler Fuzzing, PLDI 2021*

ML for predictive modelling– Recap

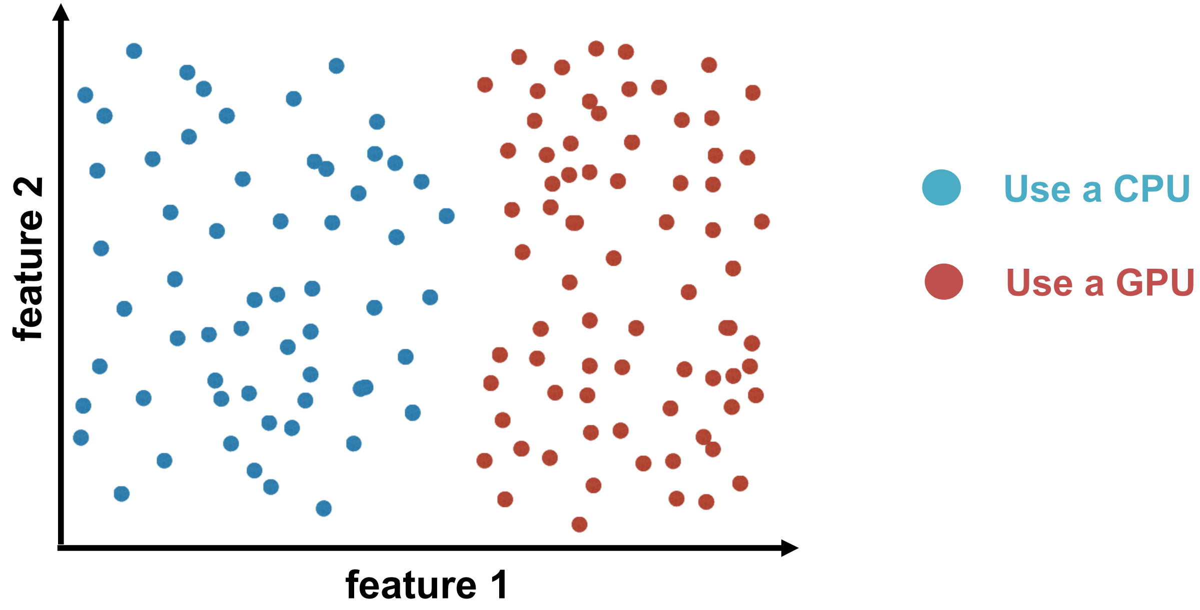
Labelling training programs can be very expensive



Well, we actually only to profile these.



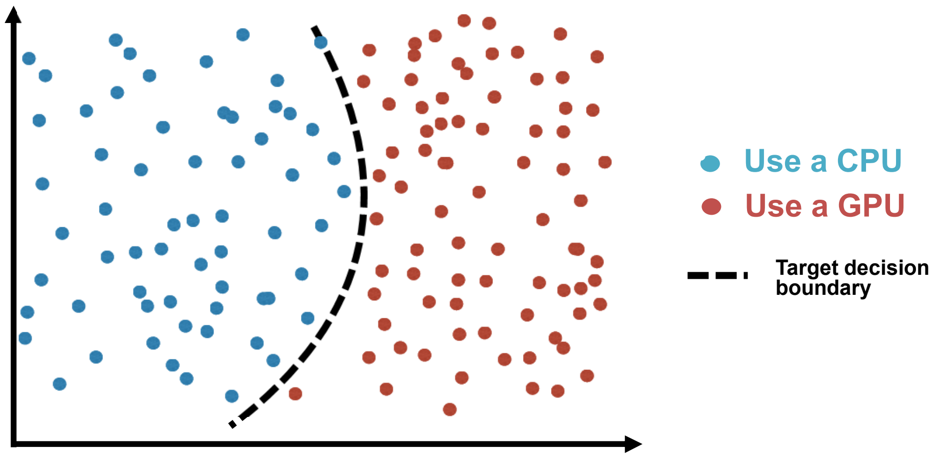
So these were a complete waste of time!



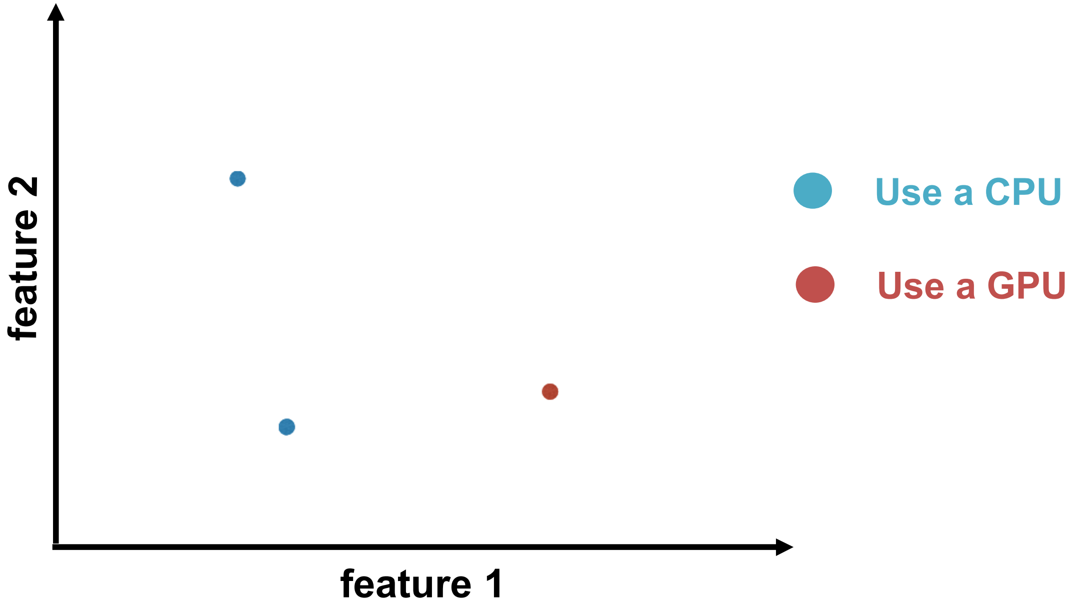
Random profiling inevitably leads to redundancy

What do we do about it?

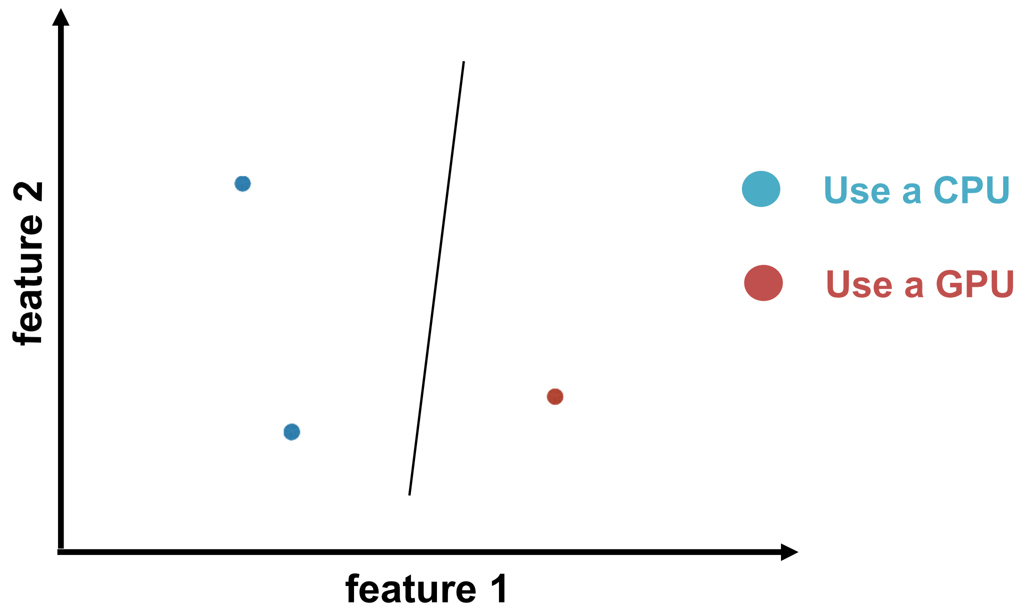
- We cannot know where the informative examples lie
- But, we can let the algorithm make an educated guess – active learning



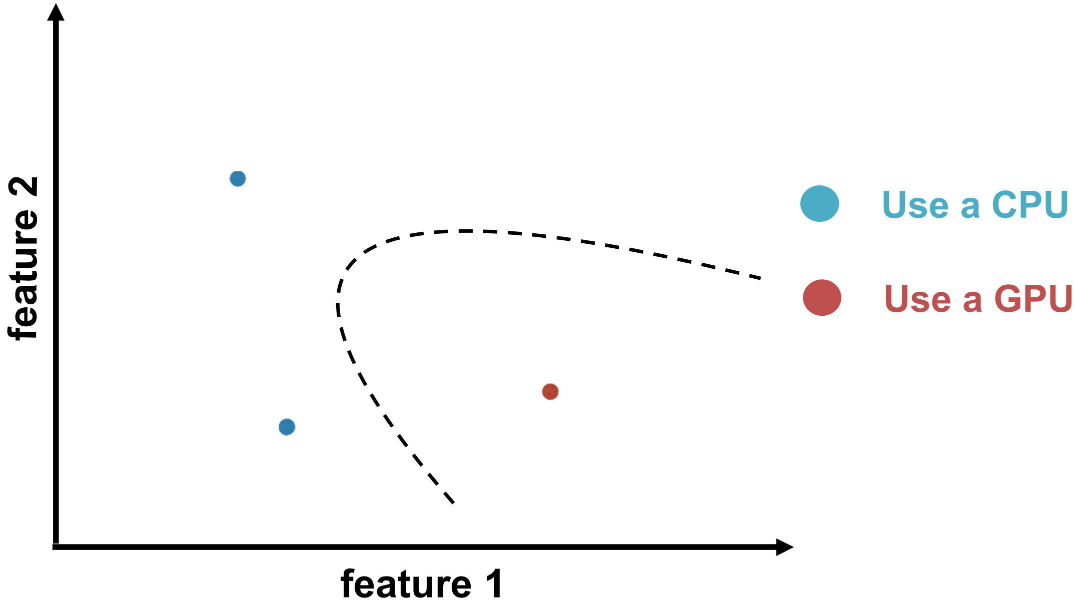
We start with a few random examples



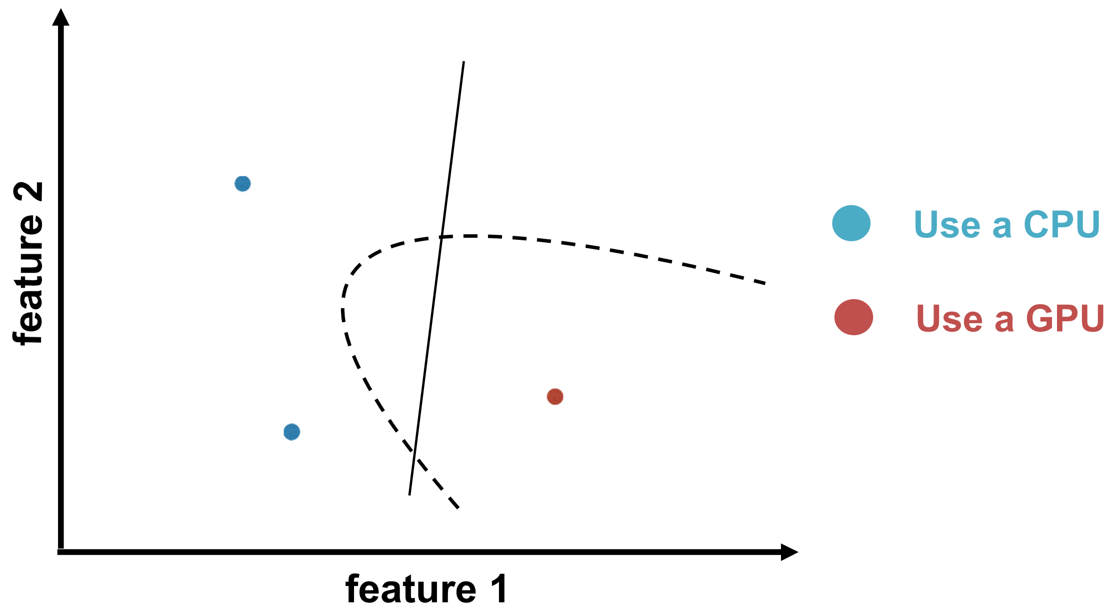
We form multiple intermediate models



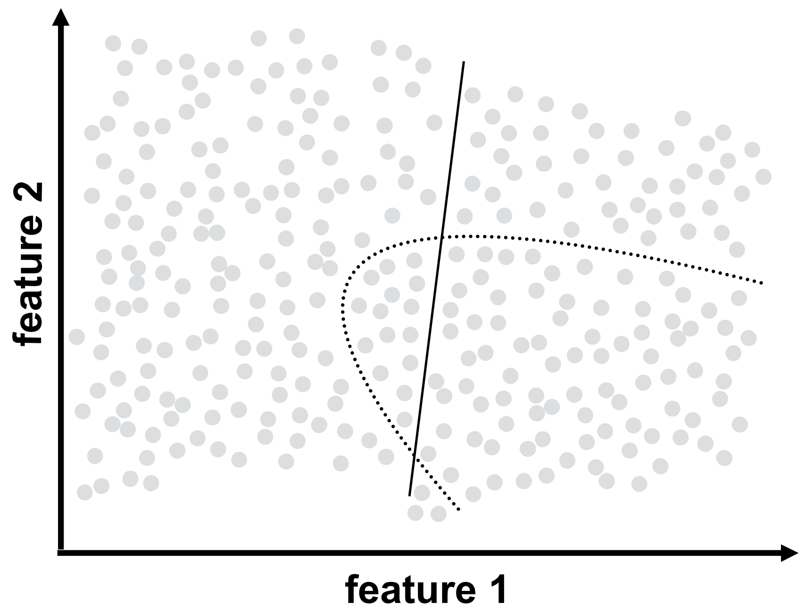
Each with a distinct algorithm



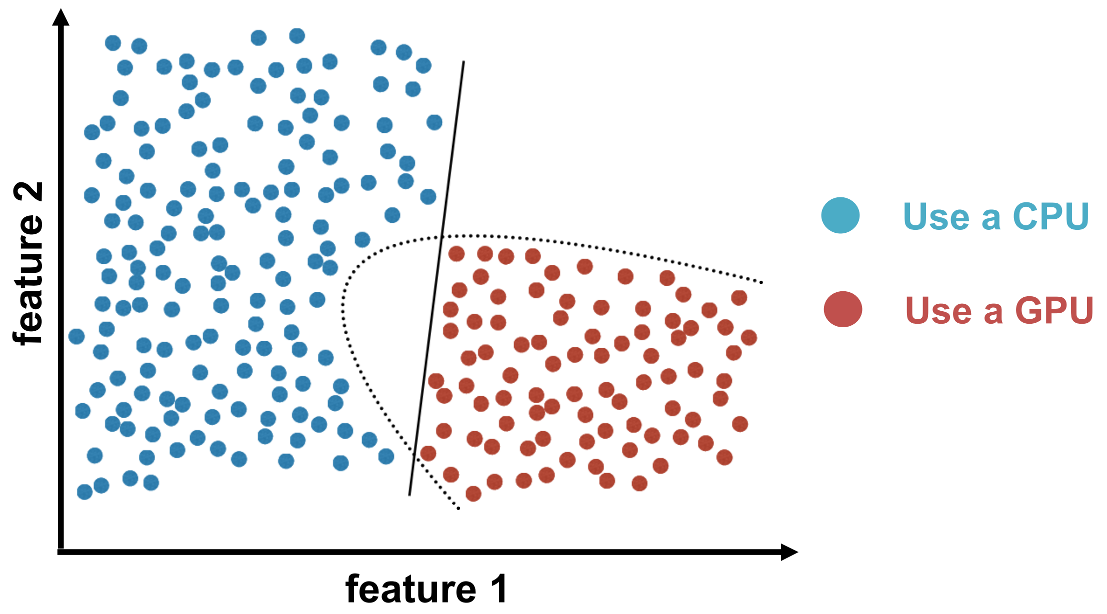
A “committee” of different models



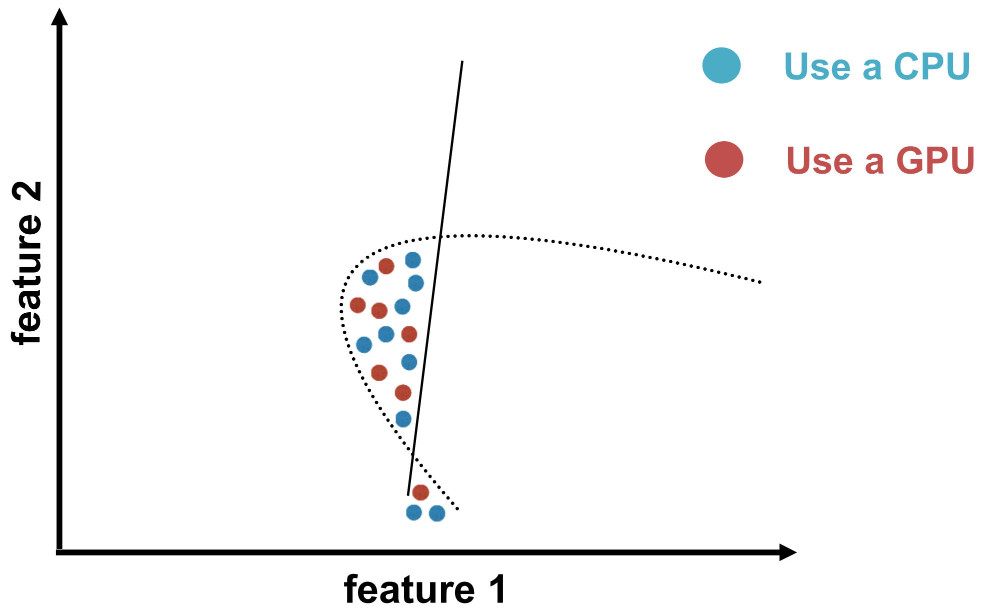
So what example do we learn from next?



Broadly the “Committee” will agree…

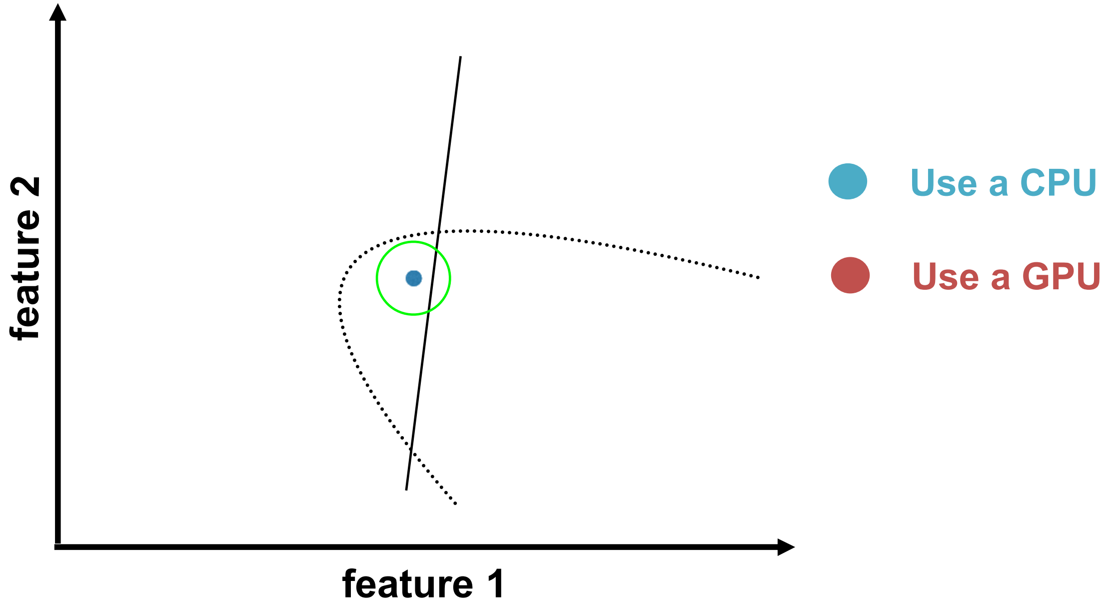


Sometimes, they disagree…

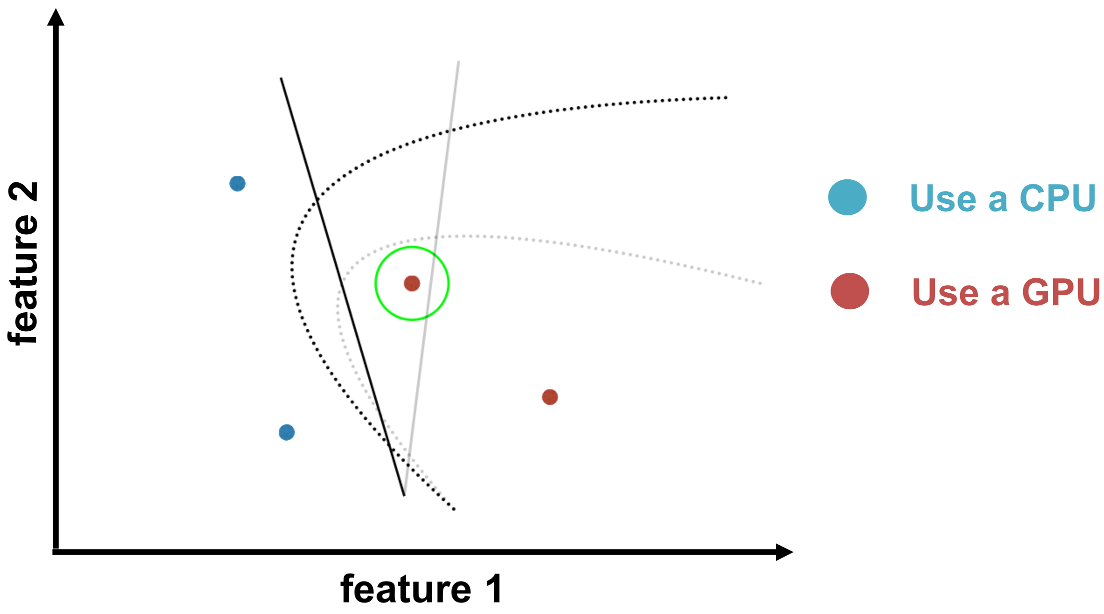


Disagreement regions hold the greatest potential to improve the collective knowledge — learn from there!

We select one of these examples to label properly



Then rebuild the intermediate models



Notice the region of disagreement has shrunk

Eventually the distinct models will converge

4x faster on average

SPAPT Benchmarks

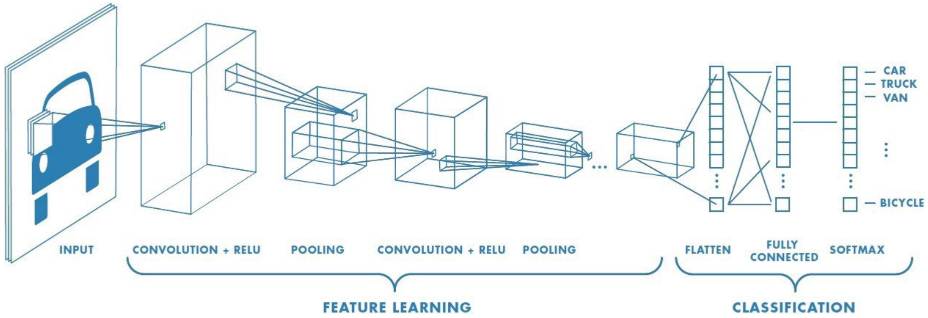
Optimisation space : 5x108 to 1x1027

We reduce the profiling time for labelling by 4x.

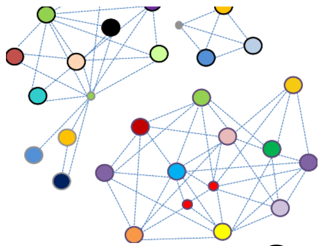
Minimizing the cost of iterative compilation with active learning, CGO 2017

Challenges ahead

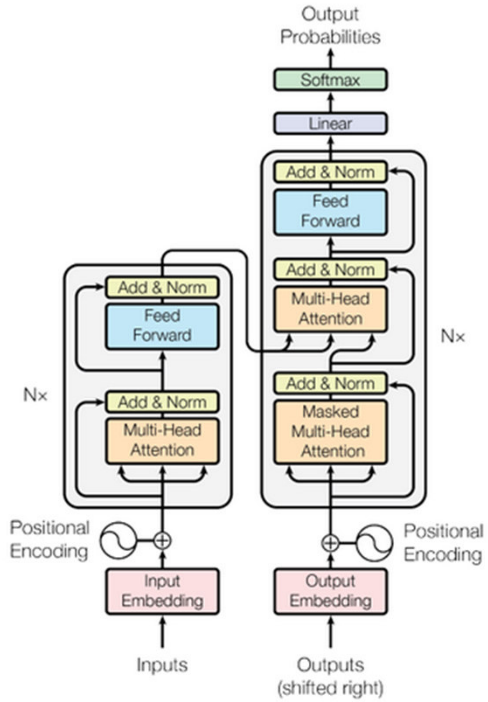
Representation matters



Images ⇨ **Convolutional** Neural Networks **(CNN)**



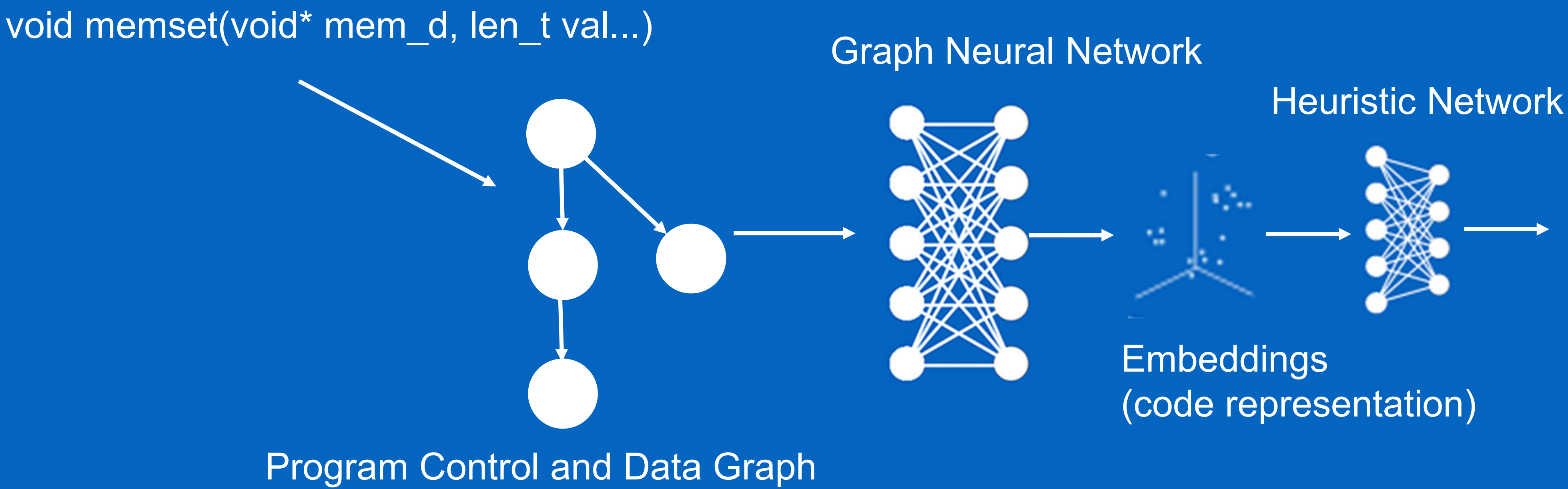
Social Networks ⇨ **Graph** Convolutional Networks **(GCN)**
Gated **Graph** Neural Networks **(GGNN)**
Graph Transformer Networks **(GTN)**



Text ⇨ **Transformer** based Models

Slides adapted from Saman Amarasinghe @ MIT

How to best represent programs?



G. Ye, Z. Tang, H. Wang, J. Fang, S. Huang, Z. Wang, Deep Program Structure Modeling Through Multi-Relational Graph-based Learning, PACT 2020.

How to best represent programs?

void memset(void* mem_d, len_t val...)

Small changes in the program graph (e.g., the loop trip count) can lead to a significant change in the program behaviour.

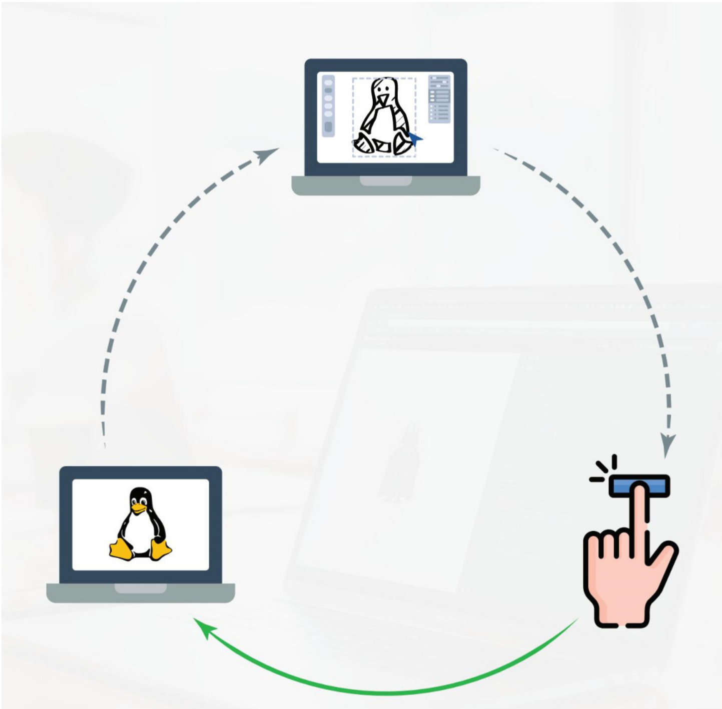
Semantic properties?

Cloud-based compilation

- Use cloud servers for large-scale compiler analysis and optimisation
  - E.g., pointer alias analysis on large program graphs

Build LLVM in 90 seconds

From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers, ATC 2019



Conclusions

- AutoML to lower entry barriers for ML in compilers
- General-purpose benchmark synthesis
- Low-cost profiling with active learning
- Many interesting problems ahead

Machine learning in compilers (papers, tools and datasets):
https://github.com/zwang4/awesome-machine-learning-in-compilers