Deep Learning for Compilers, Fun, & Profit

Chris Cummins
https://chriscummins.cc
cummins@fb.com

overview

1. What I used to do
2. What I now do
3. What you should do

overview

1. What I used to do
2. What I now do
3. What you should do

⚠️ I'm not qualified to give advice

Every PhD is different

Sample size = 1

My PhD credentials:

- I finished
- I achieved my goals
- I had fun

Good reasons to do a PhD

✅ You don't want a real job

✅ You don't want to grow up

✅ You want money

What is a PhD

- "on-the-job training for being a scientist"
- pure transferable skills
- a thesis

What is a thesis

- Thesis = 3 papers 📄 📄 📄
- 1 paper 📄 = 1 idea 💡
- ergo, PhD = 3 ideas

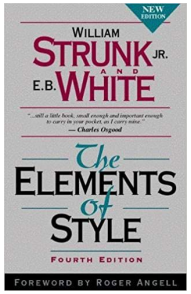
What is a PhD NOT

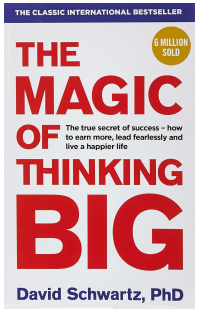
- an excuse to be a loner
- validation of your intellect
- a grind

What you should do

1. Figure out what success looks like:
    - to your viva panel
    - to yourself
2. Don't think of a PhD as a job
3. Travel!

Reading list

1. 

2. 

2. **1500+ research papers**

What you should do: industry transition

1. **Take internships**

2. **Start practicing interviews early & often**

3. **Don't get 1 job offer, aim for 3**

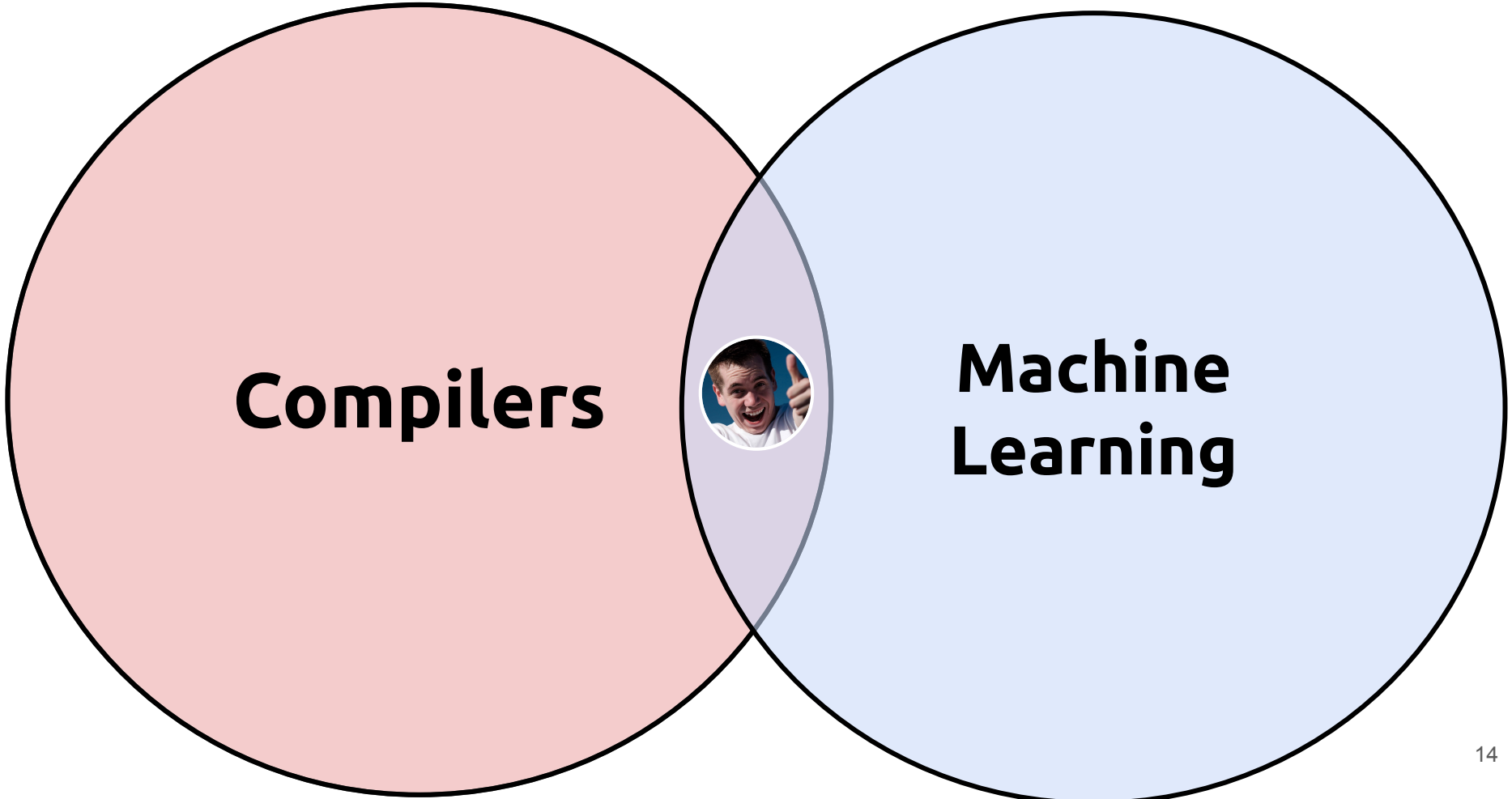
overview

1. What I used to do
2. What I now do
3. What you should do

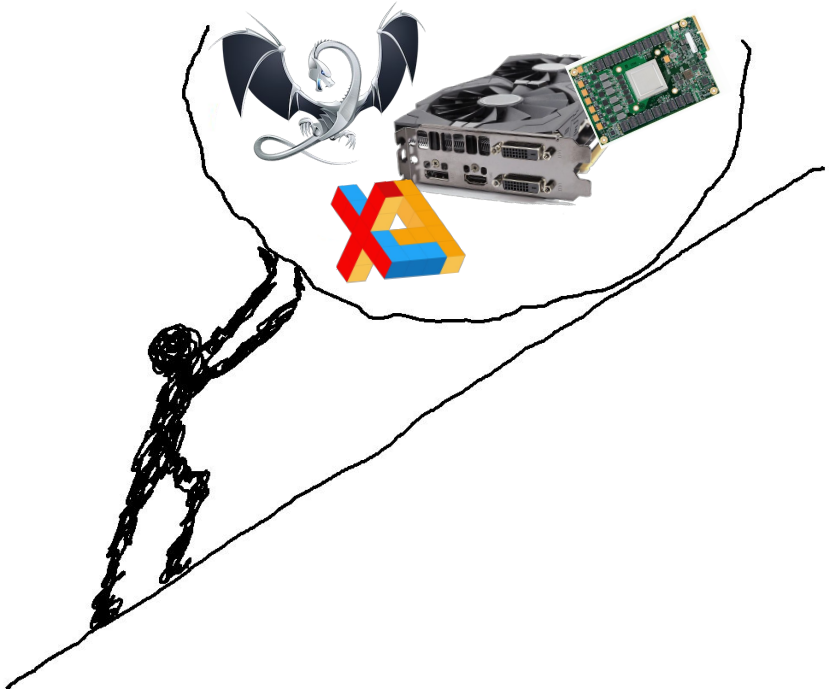
overview

1. What I used to do
2. What I now do
3. What you should do

"machine learning for compilers for machine learning"



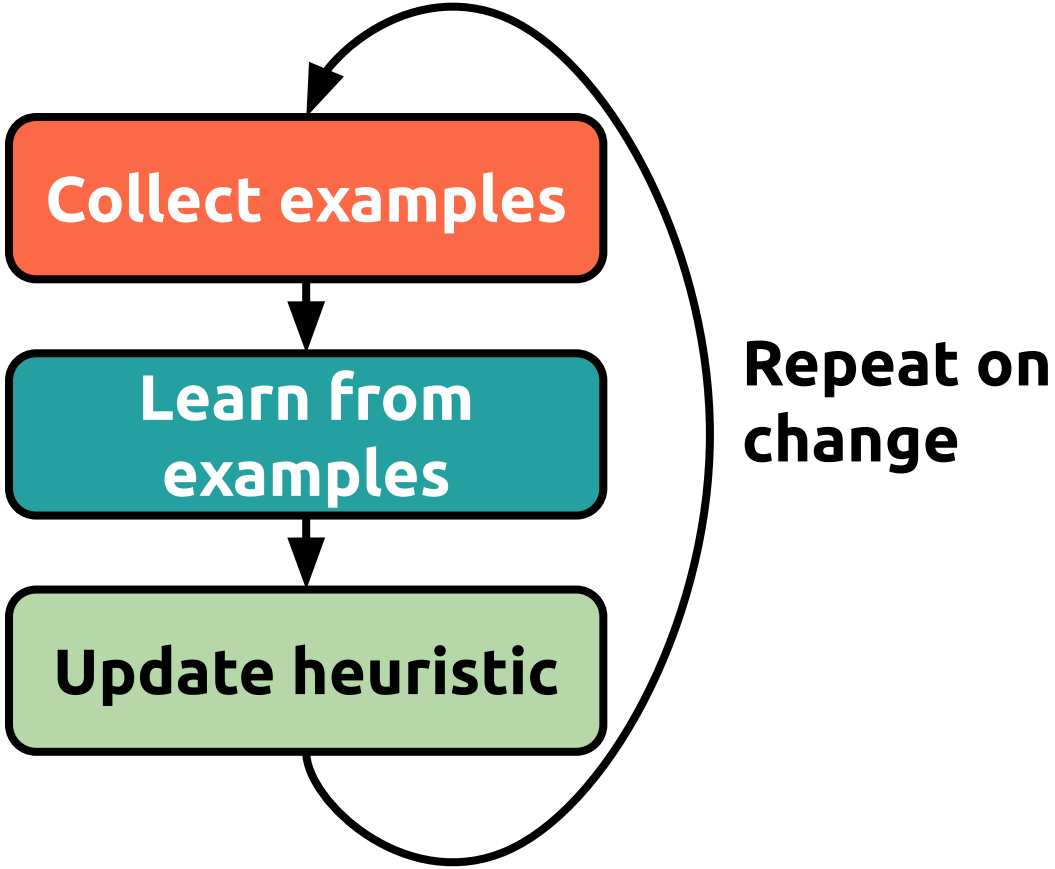
Building compilers... a job for life



- Compiler / HW keeps changing
- 1000s of variables
- NP-hard or worse

- Bad heuristics
- Wasted energy
- Widening performance gap

"**Build** an optimizing compiler, your code will be fast for a day.
Teach a compiler to optimize ... "



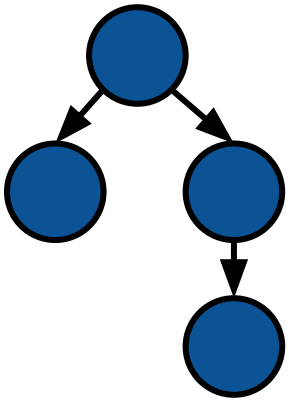
Summarize the program

Program

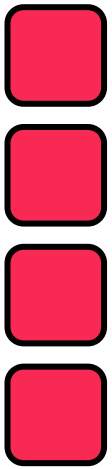
```
void LinearAlgebraOp<InputScalar,
OutputScalar>::AnalyzeInputs(
    OpKernelContext* context, TensorInputs* inputs,
    TensorShapes* input_matrix_shapes, TensorShape*
batch_shape) {
  int input_rank = -1;
  for (int i = 0; i < NumMatrixInputs(context); ++i) {
    const Tensor& in = context->input(i);
    if (i == 0) {
      input_rank = in.dims();
      OP_REQUIRES(
          context, input_rank >= 2,
          errors::InvalidArgument(
                  "Input tensor ", i,
                  " must have rank >= 2"));
```

IR

(CFG, DFG, AST,...)



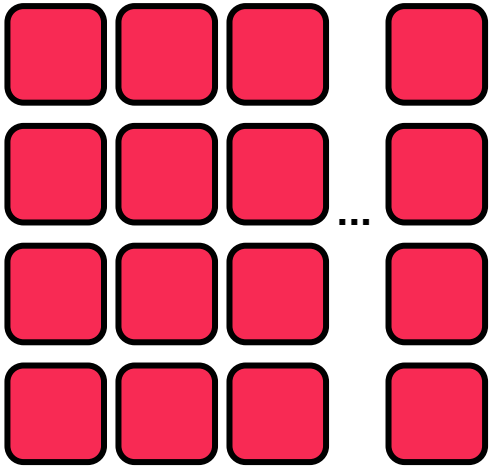
Features



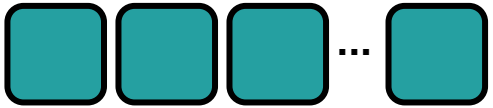
- #. instructions
- loop nest level
- arithmetic intensity
- trip counts

Collect examples

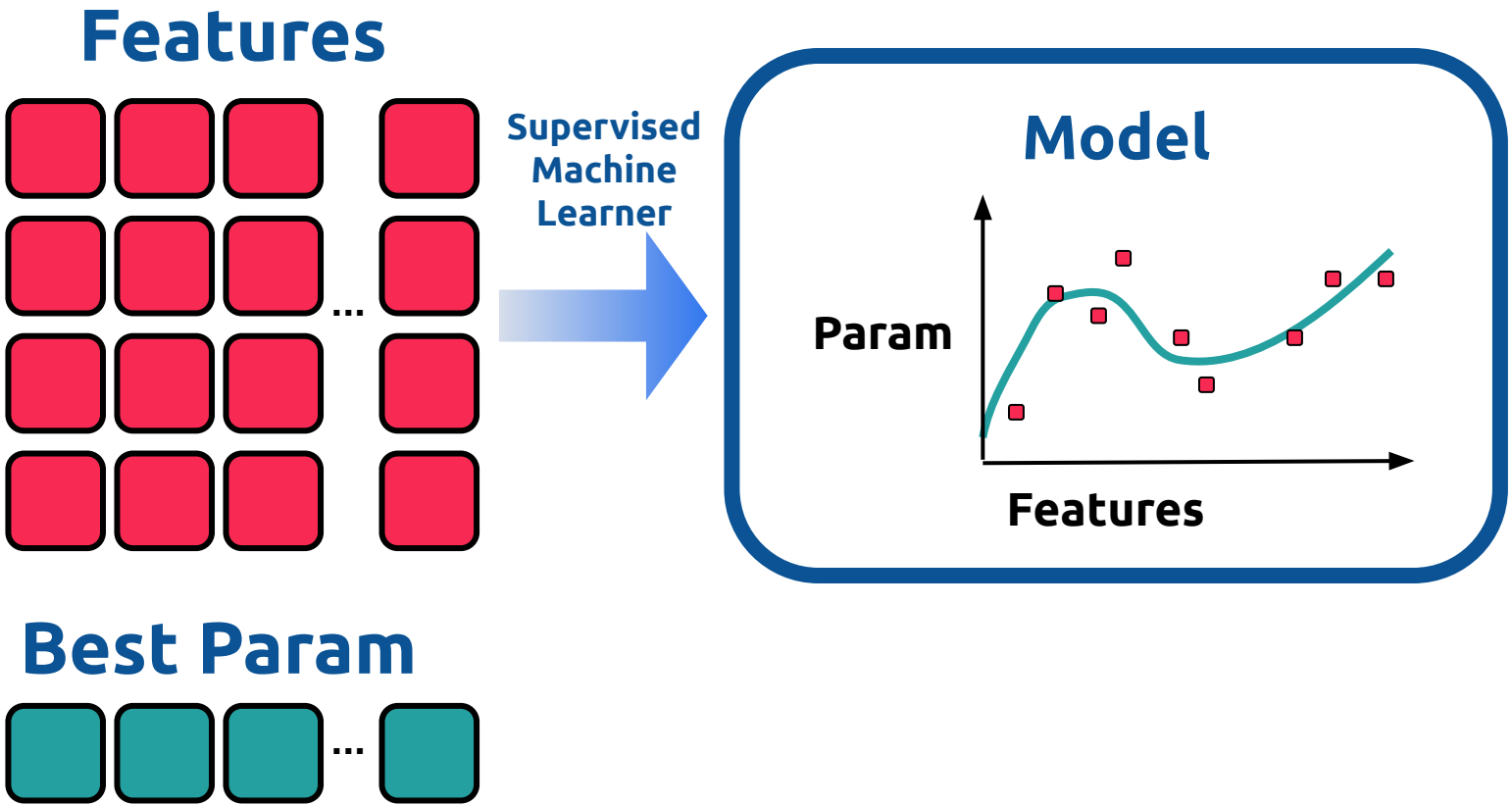
Features



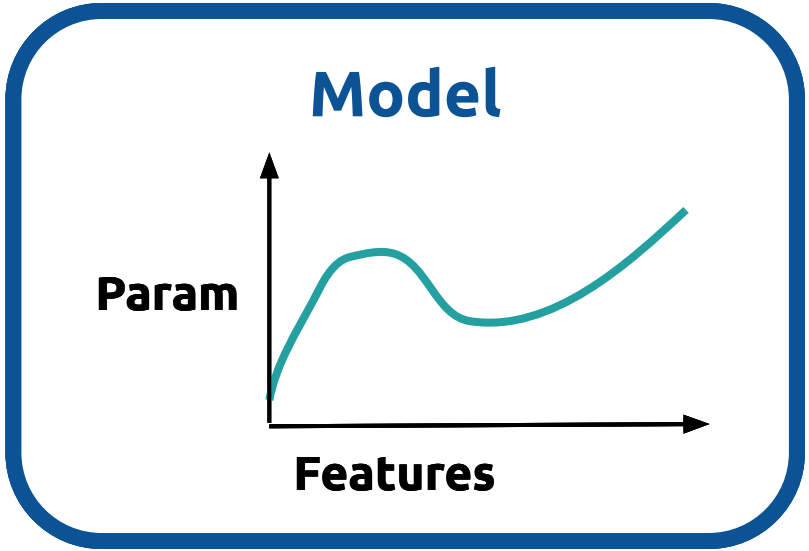
Best Param



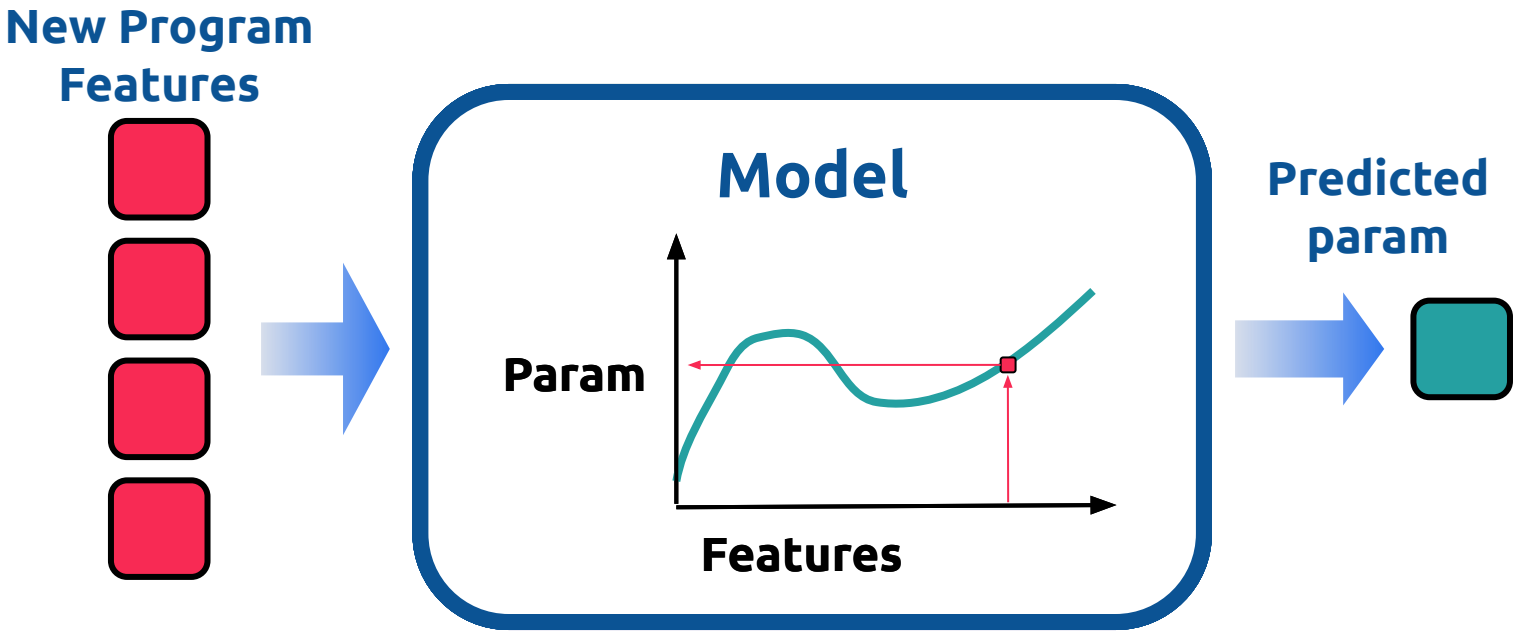
Learn from examples



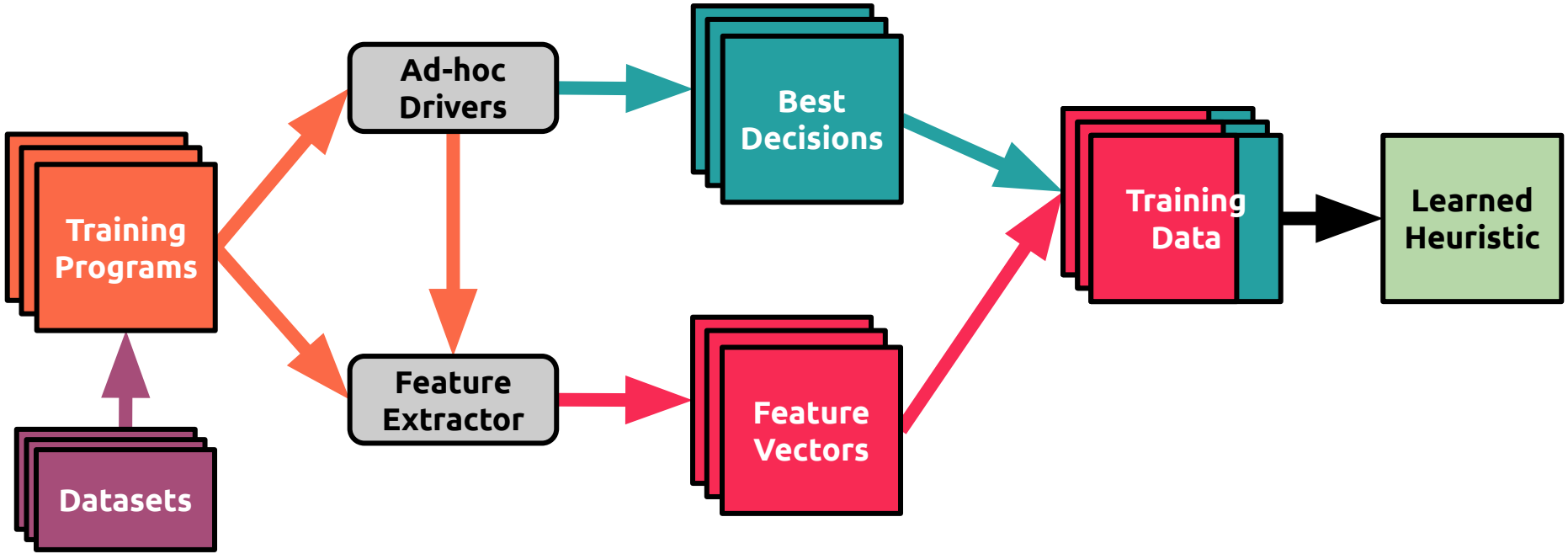
The model is the heuristic



The model is the heuristic

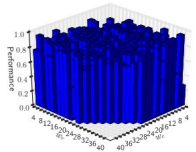


Machine learning for compilers



Machine learning for compilers

E.g. OpenCL Workgroup Size [Cummins2016a], [Cummins2016b]



Stencil Benchmarks

Training Programs

Datasets

Ad-hoc Drivers

Blocking API wrappers

Feature Extractor

Ad-hoc LLVM analysis pass

Best Decisions

2D WG size

Feature Vectors

114 numeric features

Training Data

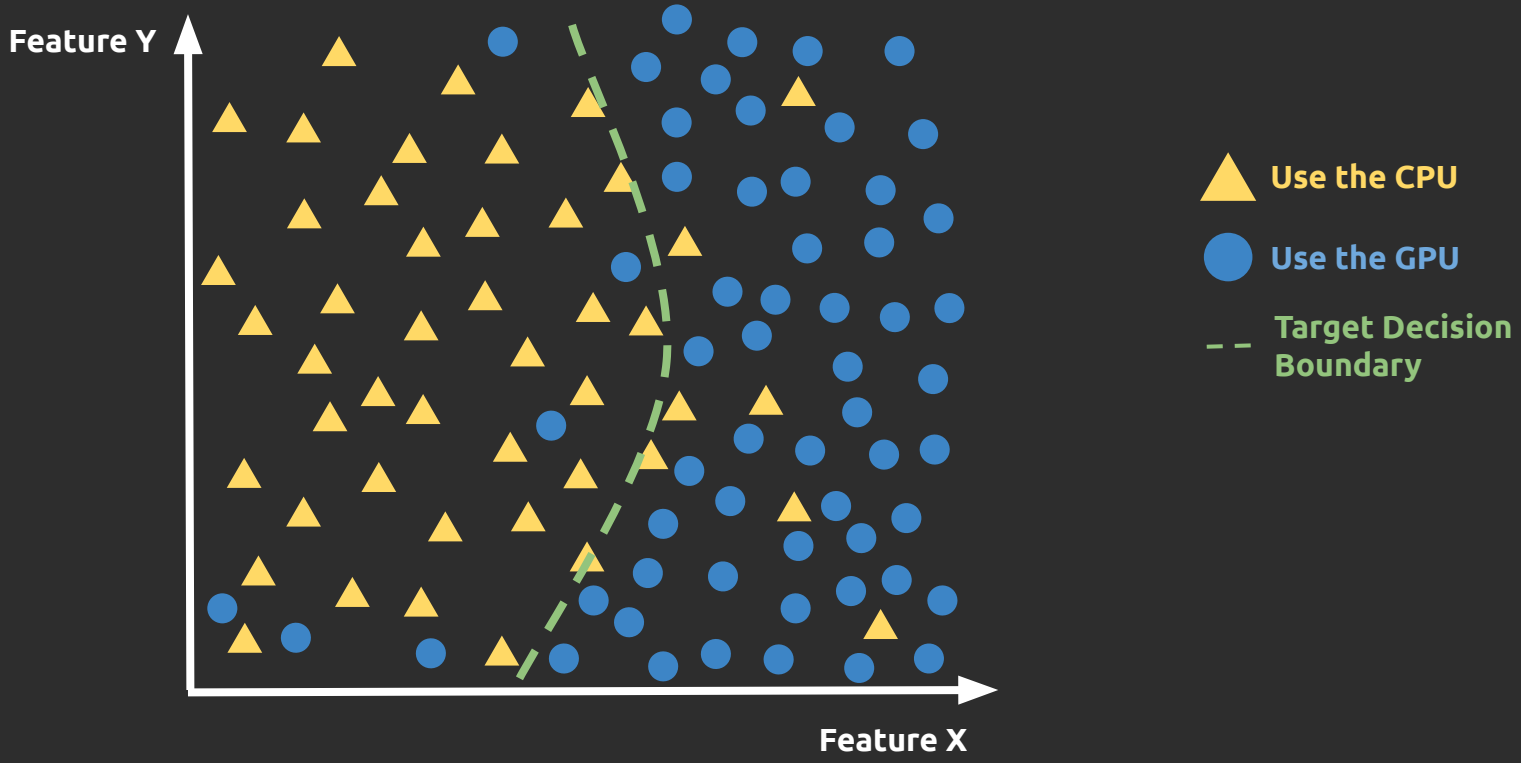
Learned Heuristic

Decision Tree, SVM, NN, Regressors, ...

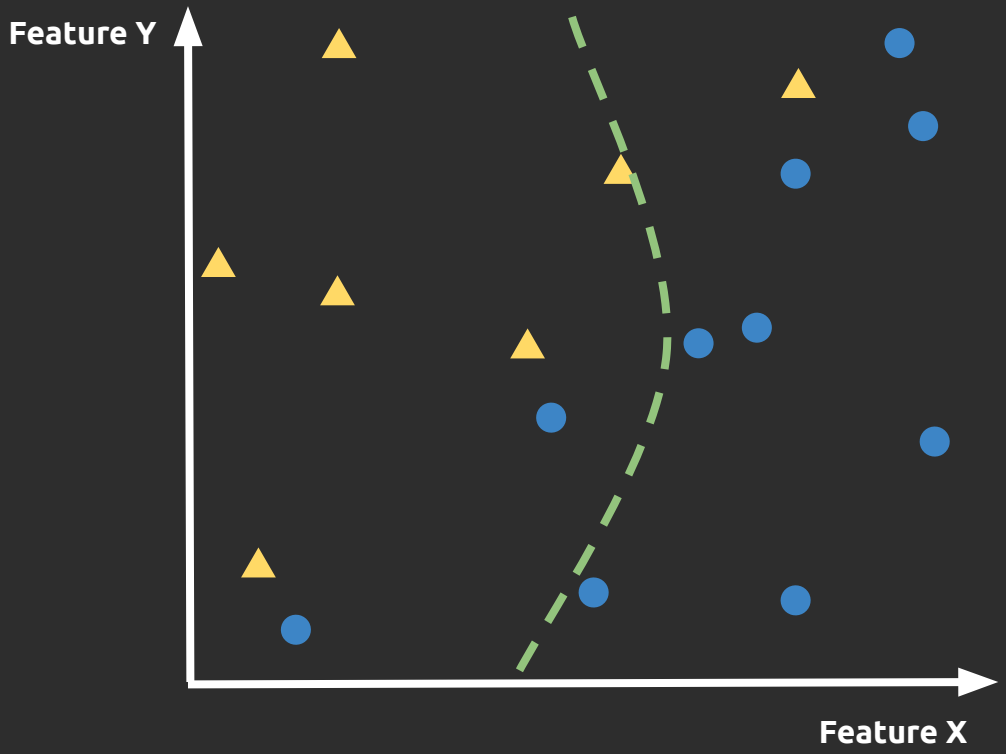
3.34× better than default heuristic

18% better than domain experts

What we want


Feature Y
Feature X
Use the CPU
Use the GPU
Target Decision Boundary

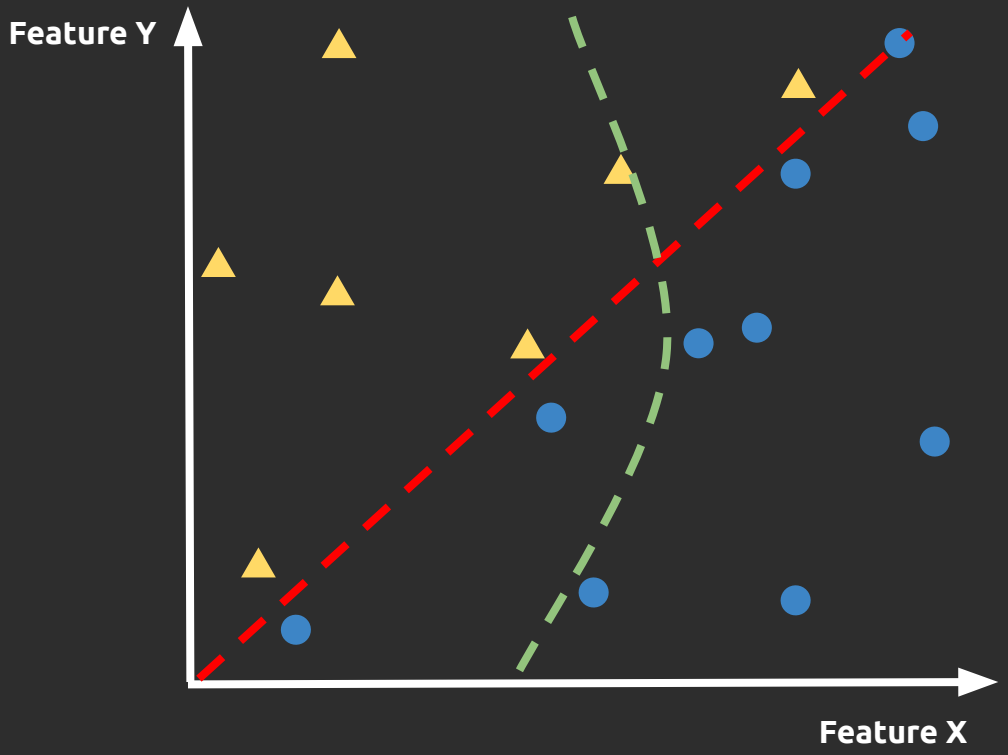
What we have



avg. compiler paper uses 17 benchmarks

source: 25 GPGPU papers from 2013-2016 in CGO, PACT, PPopp HiPC

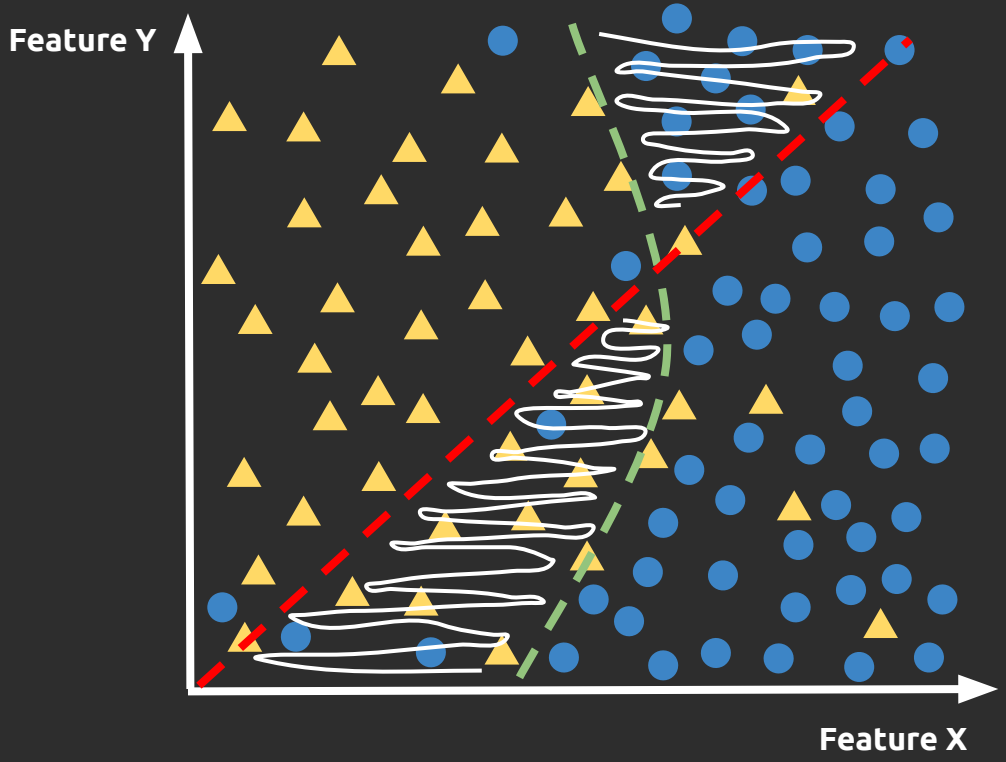
What we have



avg. compiler paper uses 17 benchmarks

source: 25 GPGPU papers from 2013-2016 in CGO, PACT, PPopp HiPC

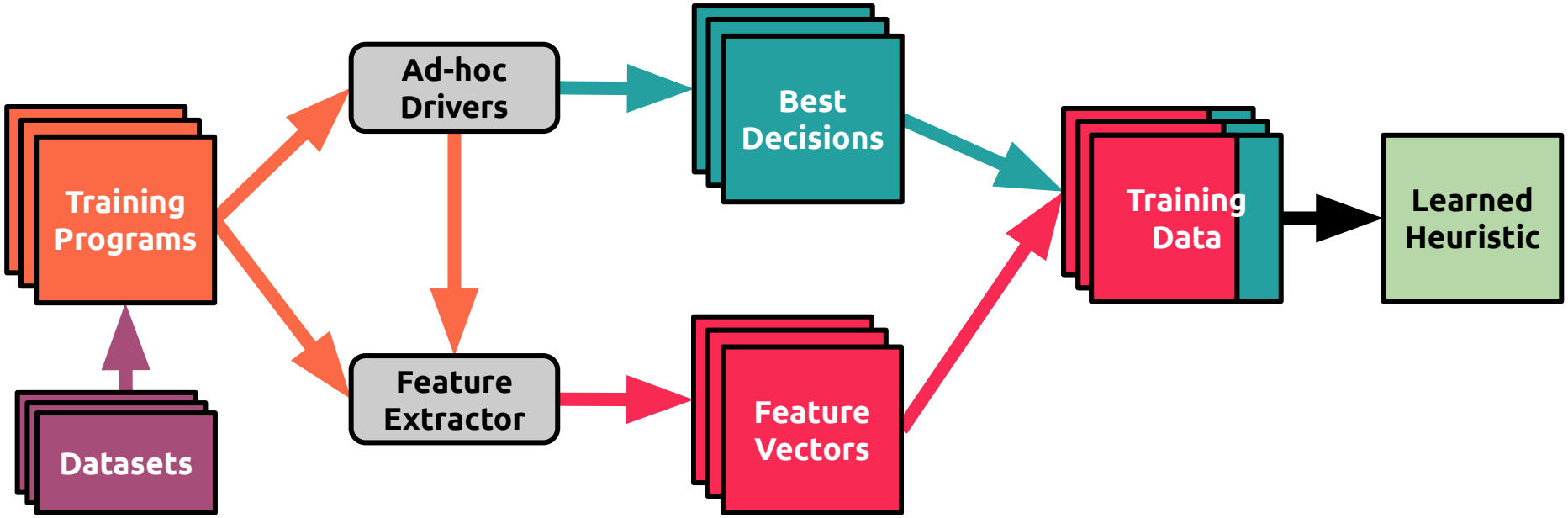
What we have



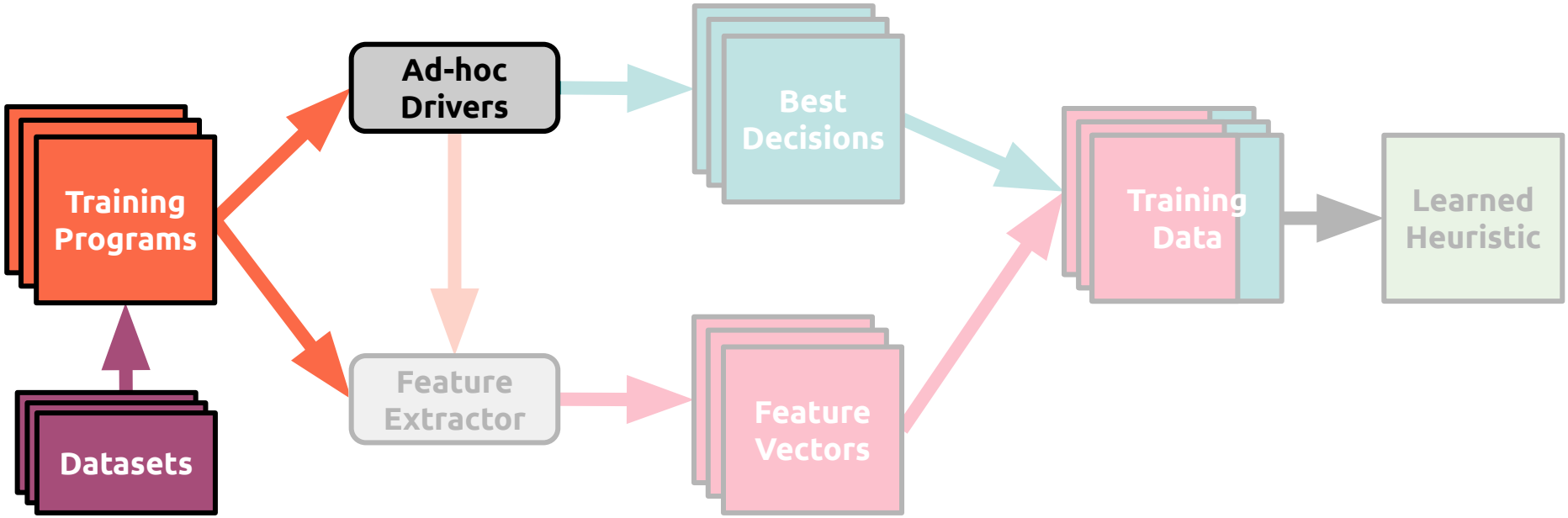
avg. compiler paper uses 17 benchmarks

source: 25 GPGPU papers from 2013-2016 in CGO, PACT, PPopp HiPC

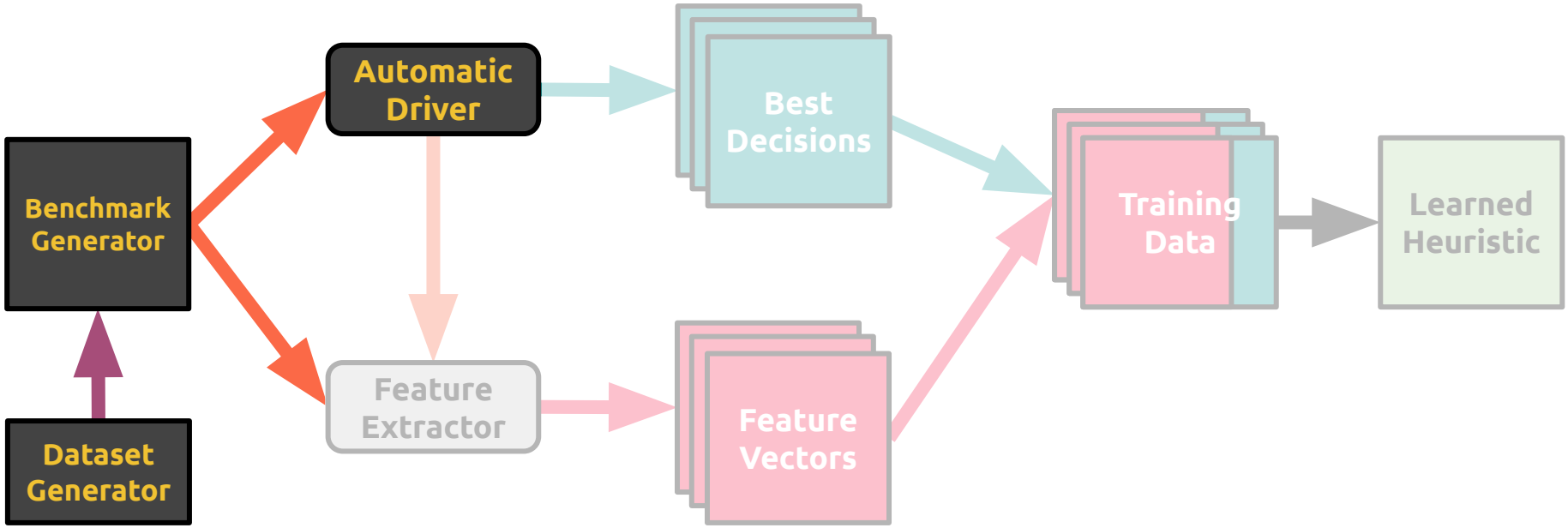
Machine learning for compilers



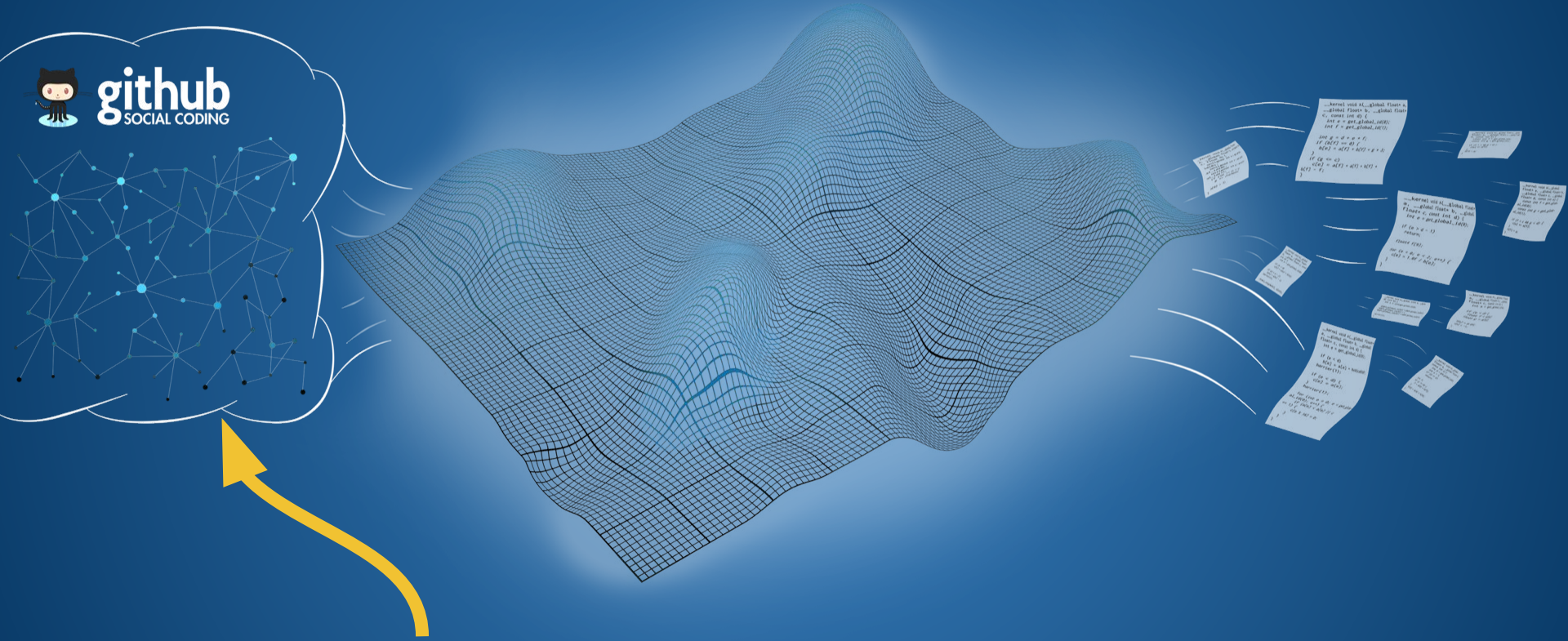
What we have


Training Programs
Datasets
Ad-hoc Drivers
Feature Extractor
Best Decisions
Feature Vectors
Training Data
Learned Heuristic

What we *need*

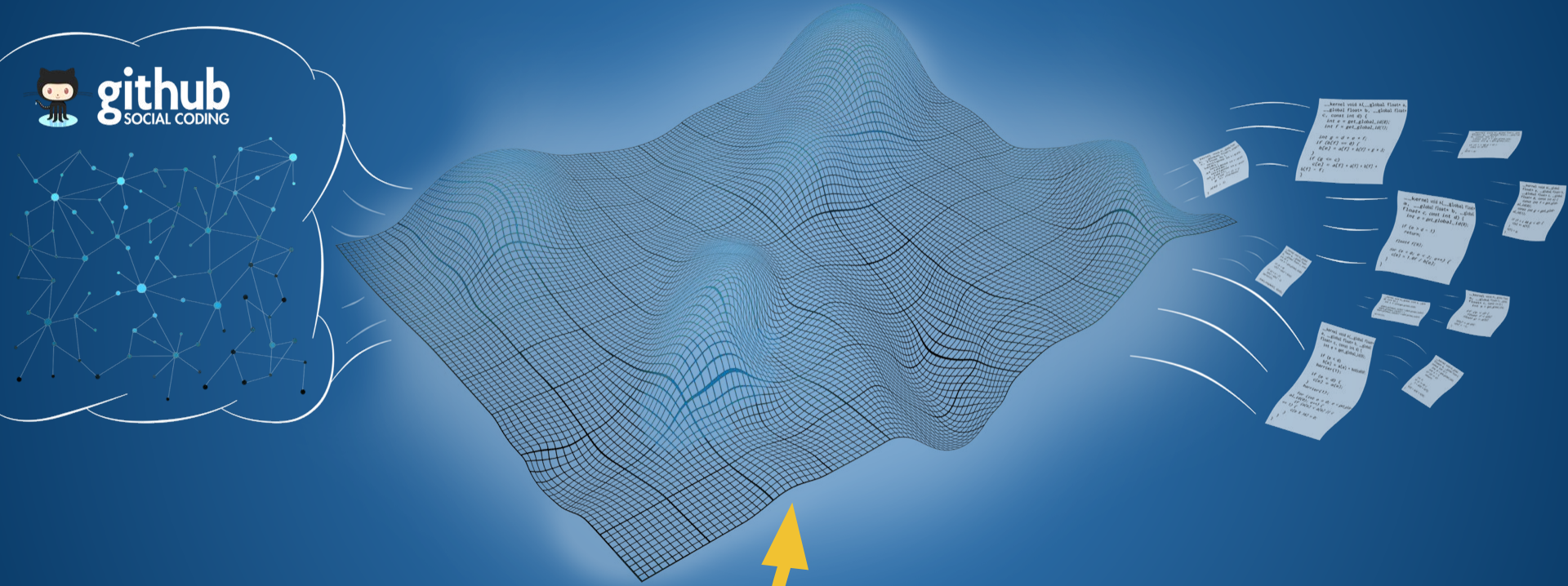

Automatic Driver
Benchmark Generator
Dataset Generator
Feature Extractor
Best Decisions
Feature Vectors
Training Data
Learned Heuristic

Generating compiler benchmarks



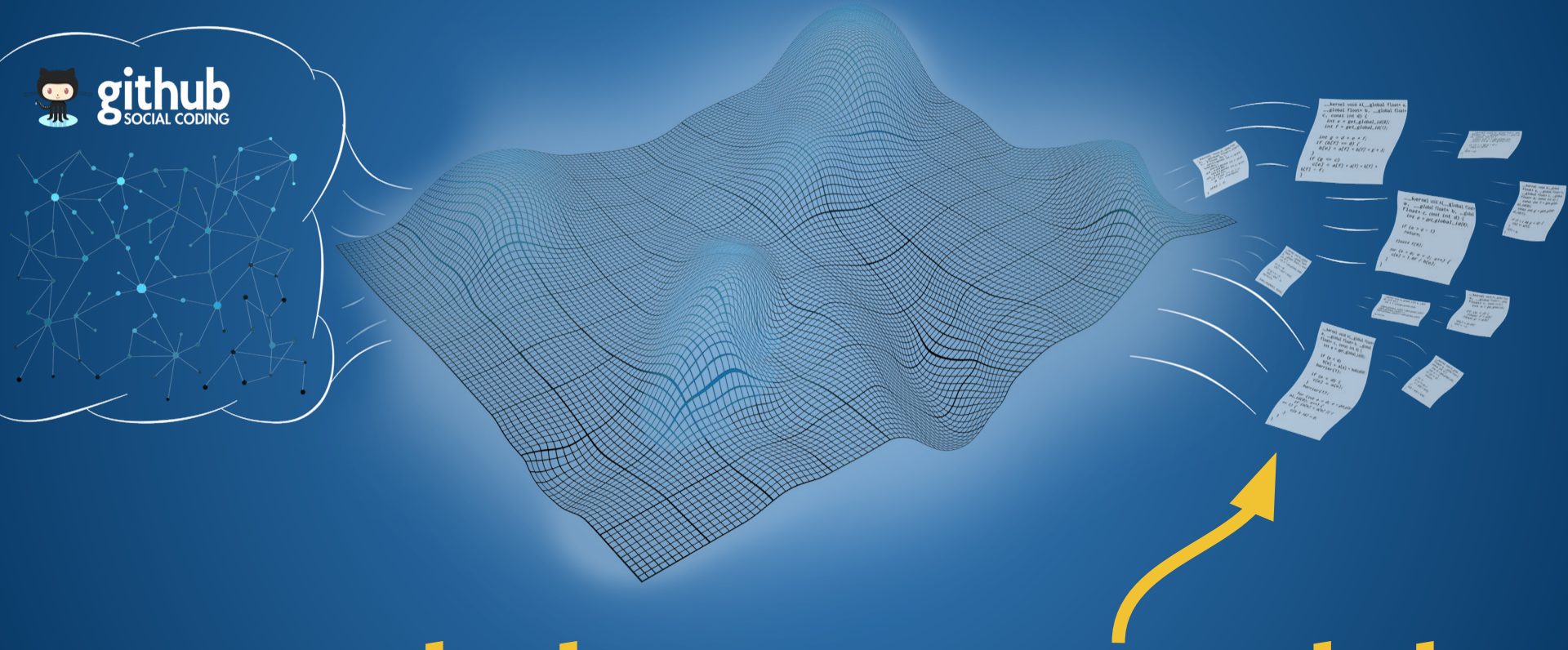
mine code from the web

Generating compiler benchmarks

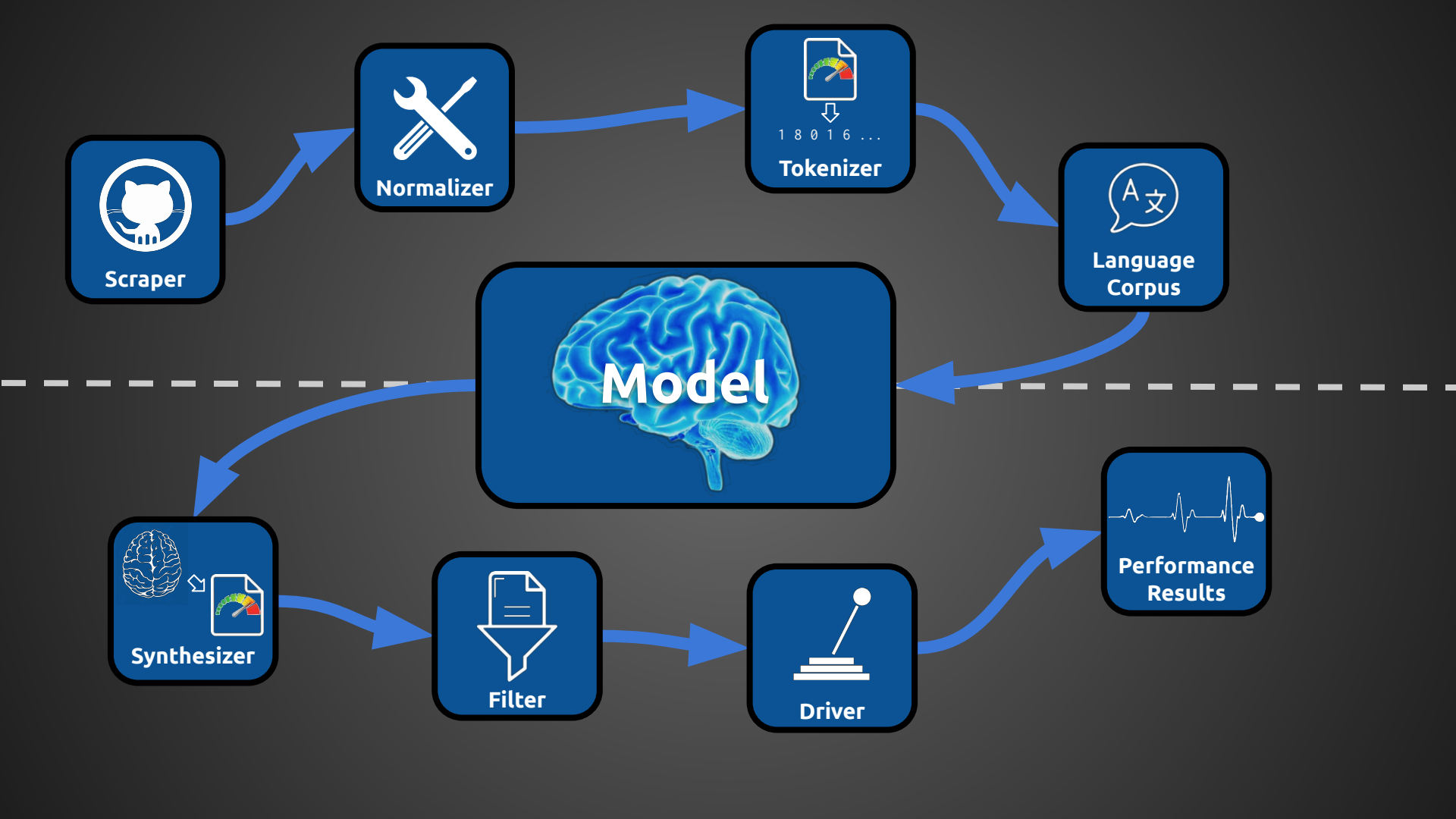


model source distribution

Generating compiler benchmarks



sample language model


Scraper
Normalizer
1 8 0 1 6 ...
Tokenizer
Language Corpus
Model
Synthesizer
Filter
Driver
Performance Results

Listing 3: Sample 3

```
__kernel void A(__global int* a, __global int* b, __global int* c,
    ↪ __global int* d, const uint e) {
  const uint f = get_global_id(0);

  if (e == 0 && f == 0)
    *d = 0;
  else if (f < e) {
    int g = b[f];
    uint h = c[f];
    if (g > 0) {
      a[h] = f;
      h++;
    }
    if (f == e - 1)
      *d = h;
  }
}
```

Listing 4: Sample 4

```
__kernel void A(__global float* a, __global float* b, __global float* c,
    ↪ const int d) {
  int e = get_global_id(0);

  if (e < d) {
    float f = b[e];
    float g = a[e];
    a[e] = f * 3.141592f / (f + 1.0f + e * 1024 - f) - (0.55f * g * 1.0f
        ↪ / 18.0f + e / 2.0f);
  }

  for (e = 0; e < 30; e++) {
    c[e] = 0;
  }
}
```

Listing 4: Sample 4

```
__kernel void A(int a, __global float* b, __global int* c, __global int*
    ↪ d, __local int* e, int f) {
  int g = get_local_id(0);
  e[get_local_id(0)] = 0;
  barrier(1);
  while (g < f) {
    int h = c[g];

    if (h != -1) {
      __global float* i = b + g * a;
      float j = 0;
      int k = 0;

      for (int l = 0; l < a; ++l) {
        float m = i[l];
```

Listing 7: Sample 7

```
__kernel void A(int a, int b, int c, __global const float* d, __global
    ↪ const float* e, __global float* f, float g) {
  const int h = get_local_id(0);
  const int i = get_group_id(0);

  const int j = 4 * i + h;
  const int k = 4 * i + h + a;
  if (4 * i + h + a < c) {
    float l = 0.0;
    float m = 0.0;
    float n = 0.0;
    const float o = d[3 * (4 * i + h + a)];
    const float p = d[3 * (4 * i + h + a) + 1];
    const float q = d[3 * (4 * i + h + a) + 2];
    for (int r = 0; r < c; r++) {
      const float s = d[3 * r] - o;
      const float t = d[3 * r + 1] - p;
      const float u = d[3 * r + 2] - q;
      const float v = (d[3 * r] - o) * (d[3 * r] - o) + (d[3 * r + 1] -
          ↪ p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r +
          ↪ 2] - q) + g;
      const float w = e[r] / (((d[3 * r] - o) * (d[3 * r] - o) + (d[3 *
          ↪ r + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d
          ↪ [3 * r + 2] - q) + g) * sqrt((d[3 * r] - o) * (d[3 * r] - o
          ↪ ) + (d[3 * r + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2]
          ↪ - q) * (d[3 * r + 2] - q) + g));
      l = l + (d[3 * r] - o) * w;
      m = m + (d[3 * r + 1] - p) * w;
      n = n + (d[3 * r + 2] - q) * w;
    }
    f[3 * (4 * i + h)] = l;
    f[3 * (4 * i + h) + 1] = m;
    f[3 * (4 * i + h) + 2] = n;
  }
}
```

Listing 10: Sample 10

```
__kernel void A(__global ulong *a) {
    int i, j;
    struct S0 c_8;
    struct S0* p_7 = &c_8;
    struct S0 c_9 = {
        {{0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
         {0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
         {0x43250E6DL,2UL},{0x43250E6DL,2UL}},
        0x4BF90EDCAD2086BDL,
    };
    c_8 = c_9;
    barrier(0 | 1);
    B(d);
    barrier(0 | 1);
    uint64_t e = 0xFFFFFFFFFFFFFFFFUL;
    a[get_linear_global_id()] = e ^ 0xFFFFFFFFFFFFFFFFUL;
}
```

52%

blind test

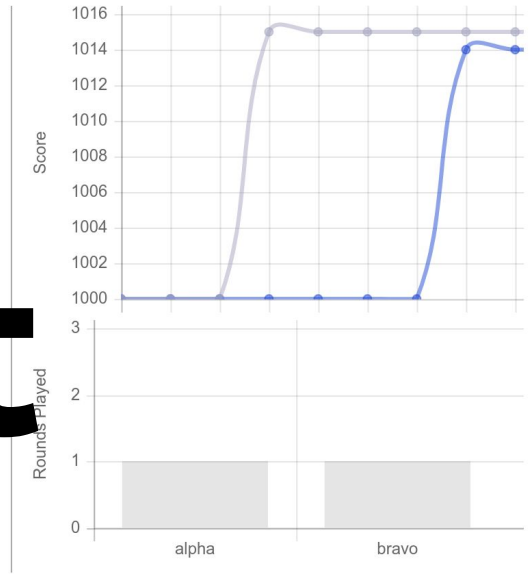
96% control group

Round 1

Player: **973**, Robot: **1028**

```
__kernel void A(__global __write_only int *a, __global __read_only int *b,
                const float c, const float d, const float e, const float f,
                const int g, const int h, const int i) {
  int j = get_global_id(0);
  int k = get_global_id(1);
  float l = c + j * e / g;
  float m = d + k * f / h;
  int n = 0;
  float o = 0;
  float p = 0;
  while (n < i) {
    float q = o * o;
    float r = p * p;
    if ((q + r) > 4.0) break;
    float s = q - r + l;
    p = 2 * o * p + m;
    o = s;
    ++n;
  }
  if (n < i)
    a[k * g + j] = b[n];
  else
    a[k * g + j] = 0xff000000;
}
```

Try it



Human Robot

Synthesizing Benchmarks for Predictive Modeling

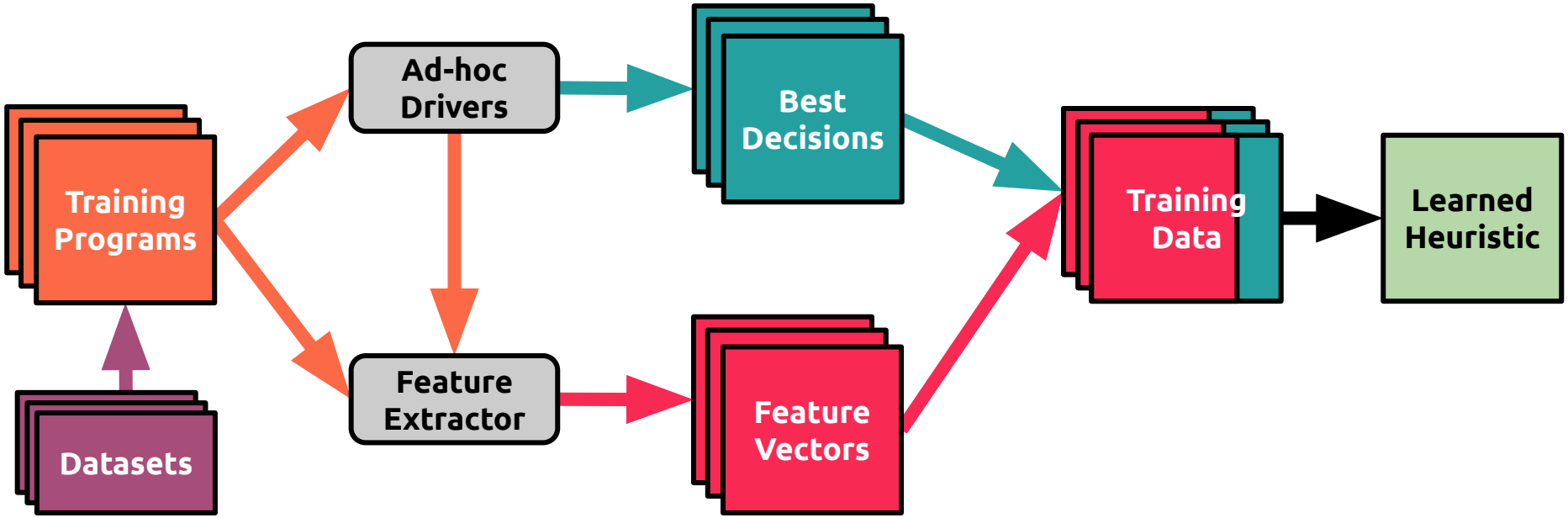
C. Cummins, P. Petoumenos, Z. Wang, H. Leather

DL approach to model PL **syntax** & **usage**

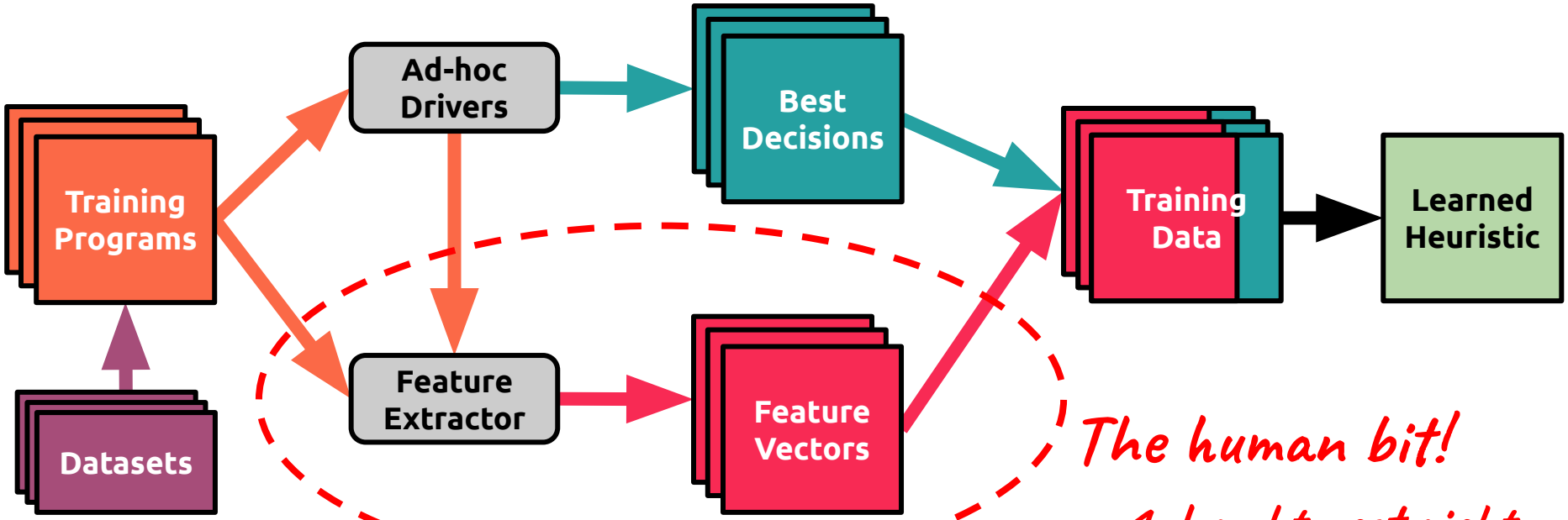
First **general purpose** benchmark generator

4.30x better model **performance**

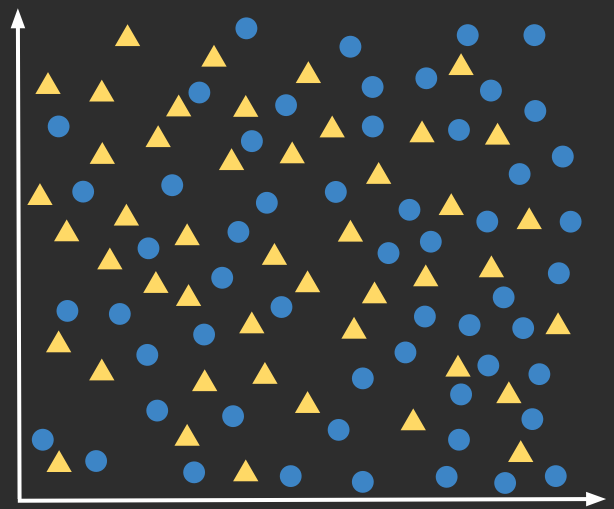
Machine learning for compilers



Machine learning for compilers

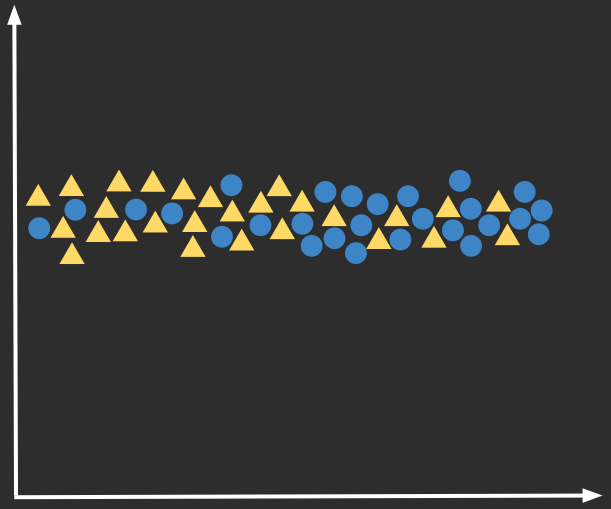


Ways to fail



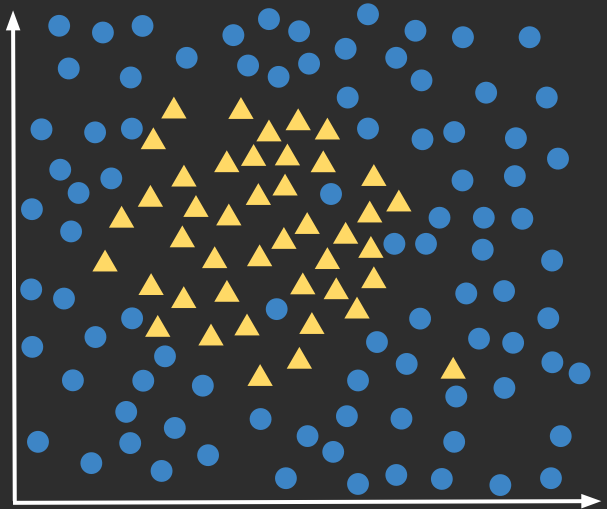
Irrelevant

e.g. not capturing the right information



Incomplete

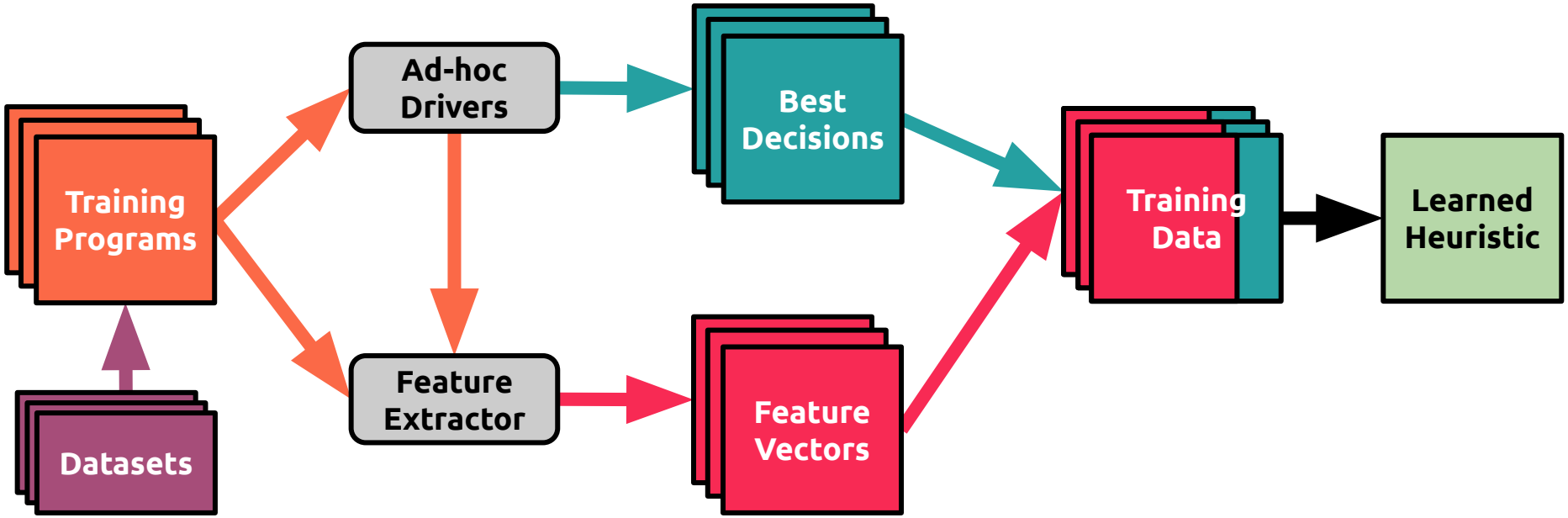
e.g. missing critical information



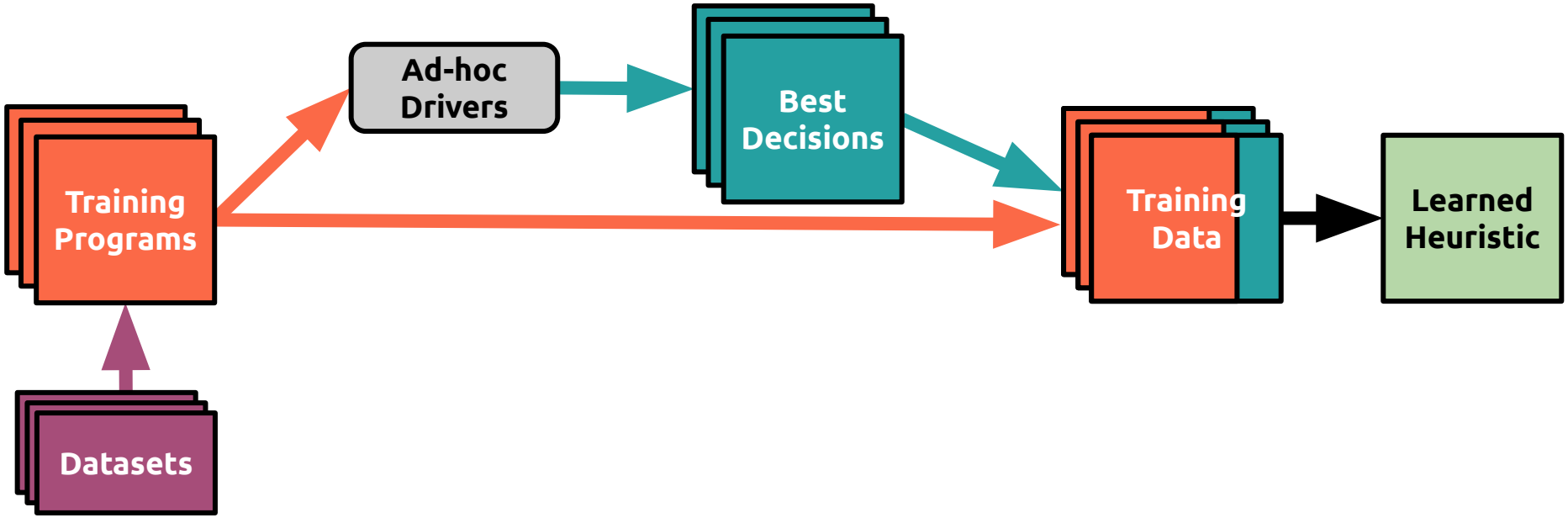
Unsuitable

e.g. wrong combination of features+model

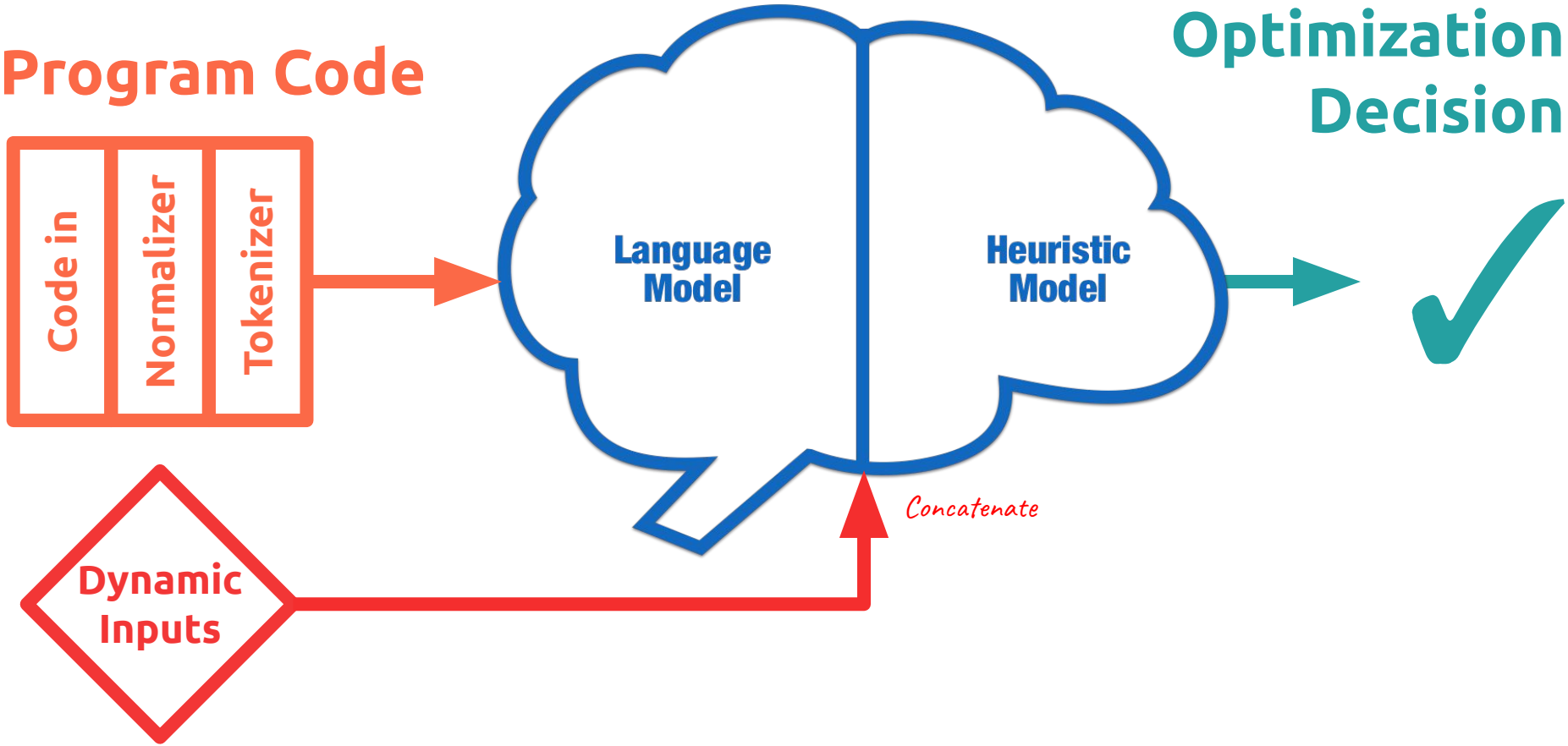
What we have


Training Programs
Datasets
Ad-hoc Drivers
Feature Extractor
Best Decisions
Feature Vectors
Training Data
Learned Heuristic

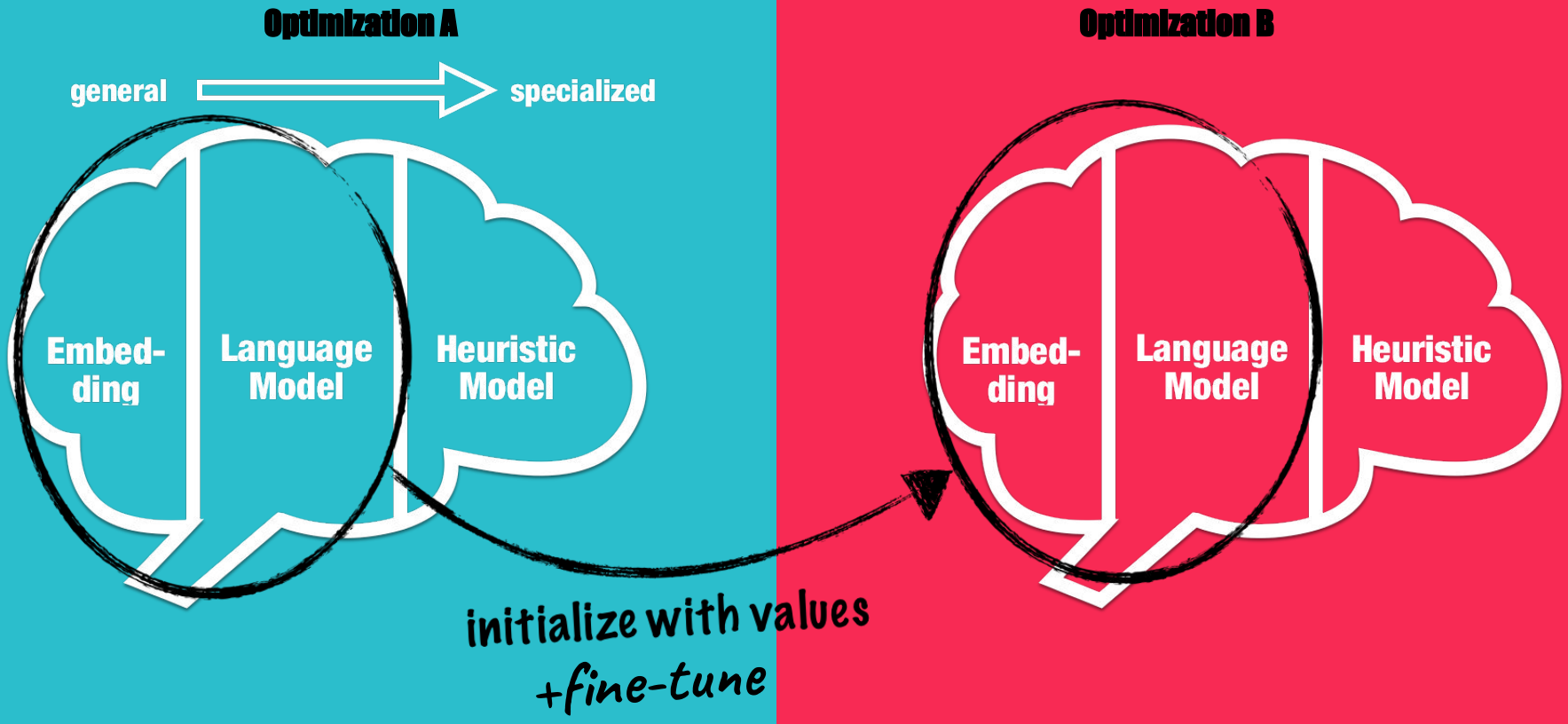
What we *need*


Training Programs
Datasets
Ad-hoc Drivers
Best Decisions
Training Data
Learned Heuristic

End-to-end optimization learning



Learning across optimizations



End-to-end Deep Learning of Optimization Heuristics

C. Cummins, P. Petoumenos, Z. Wang, H. Leather

Heuristics **without** features

12.5% **better** than using features

Learning **across** heuristics

Compilers break

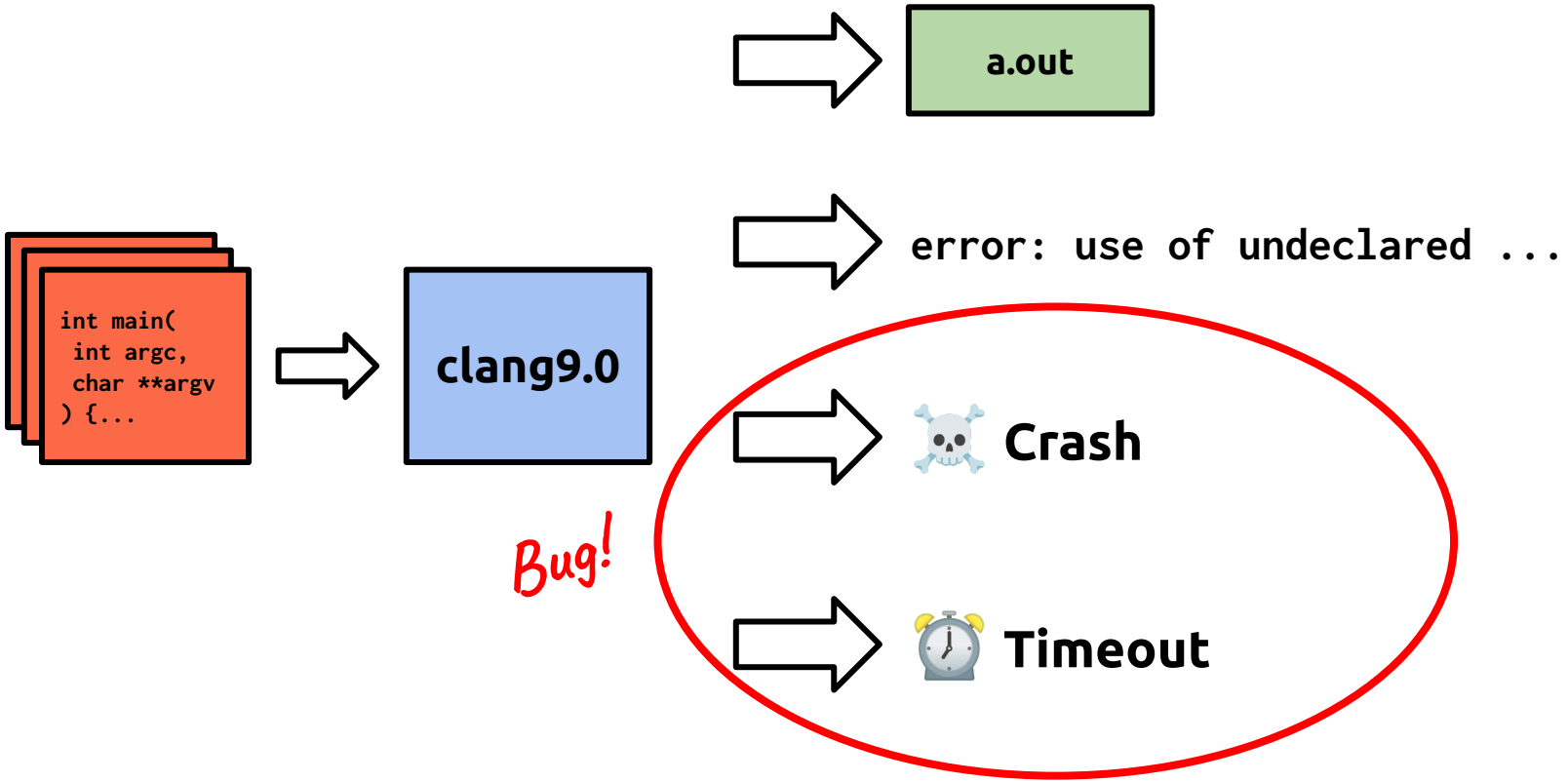
😤 **Compile-time failure**

😱 **Broken software**

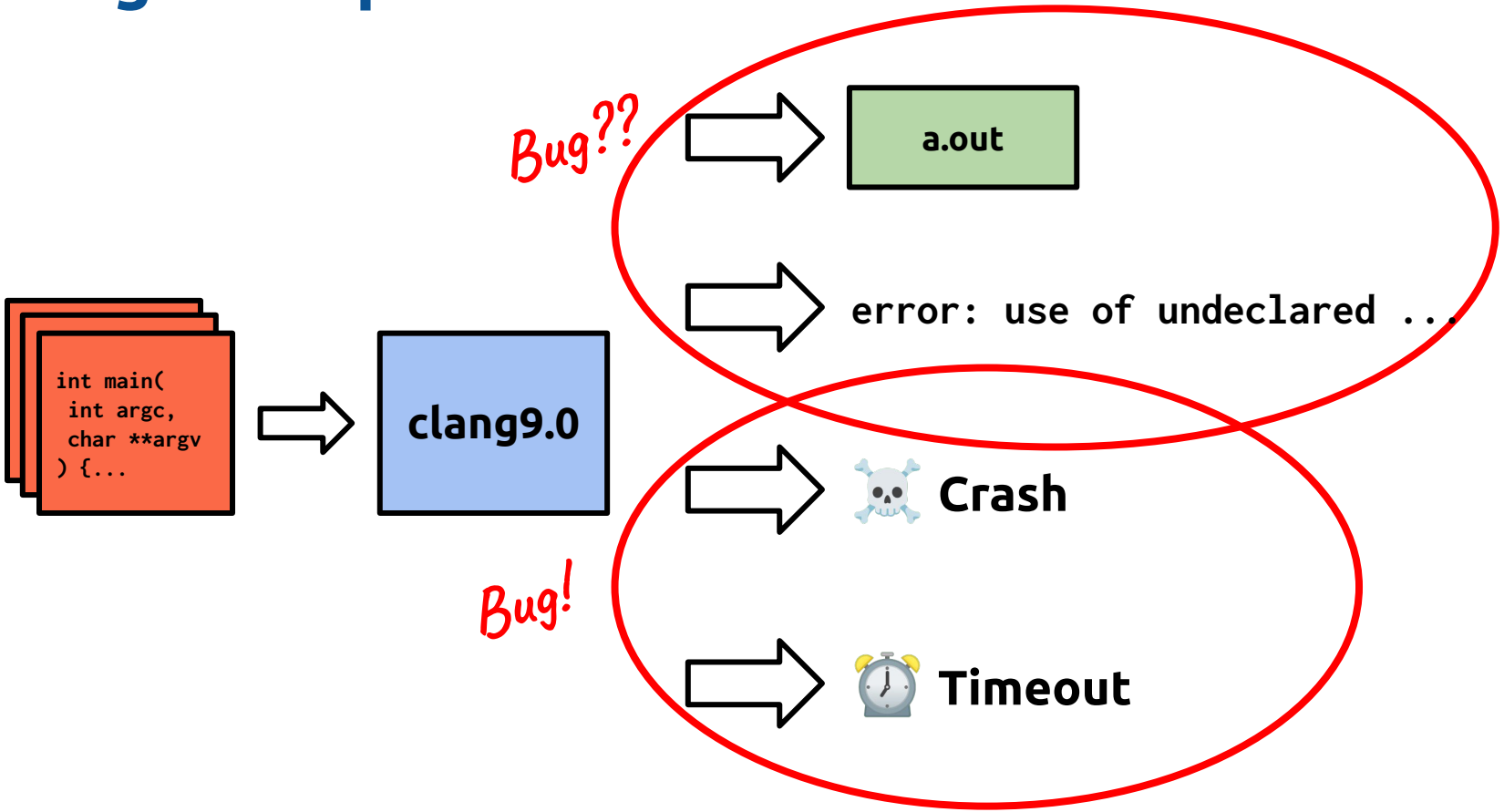
Regression suites

1. Slow
2. Late
3. Expensive
4. Incomplete

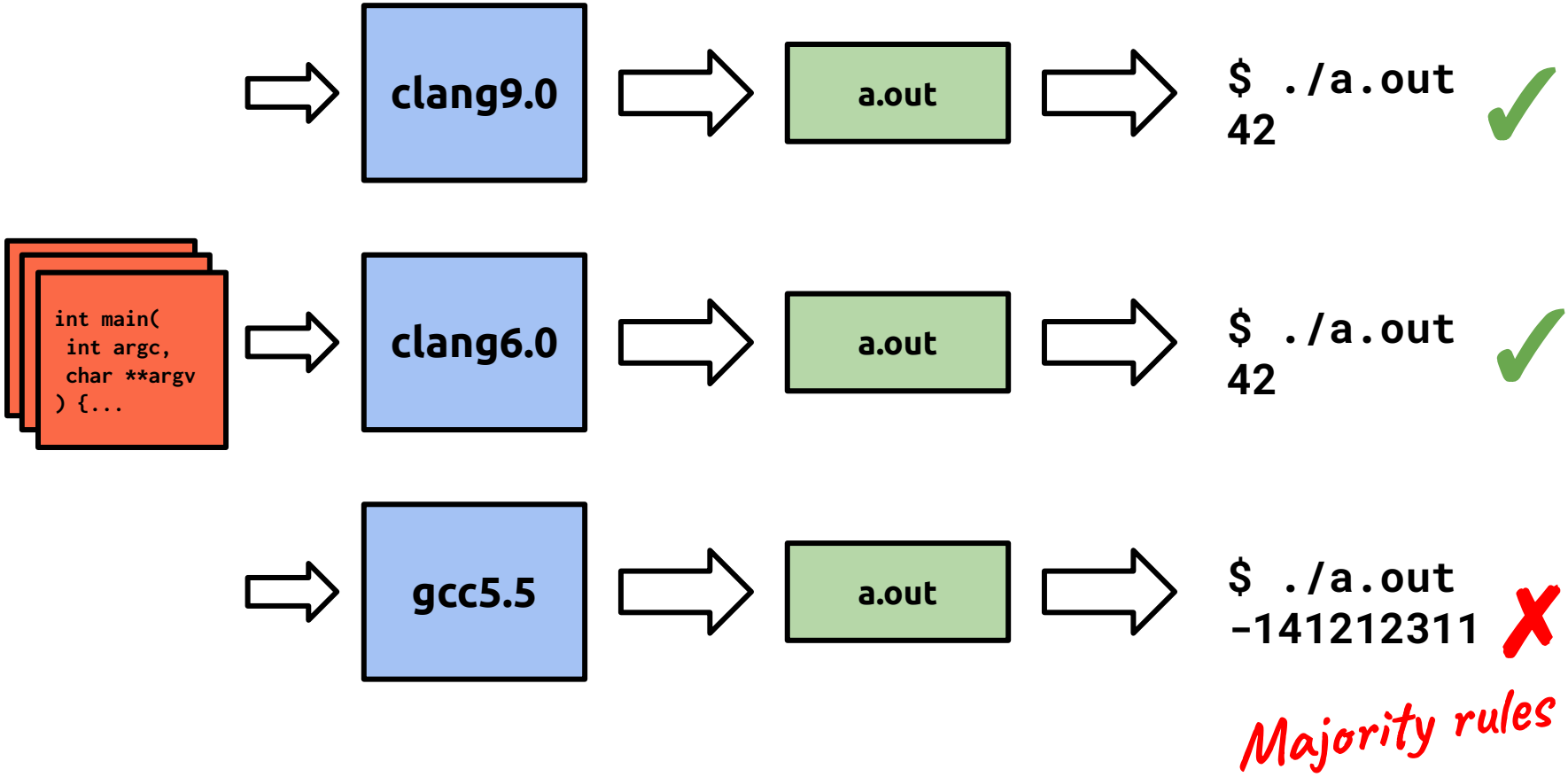
Fuzzing a compiler



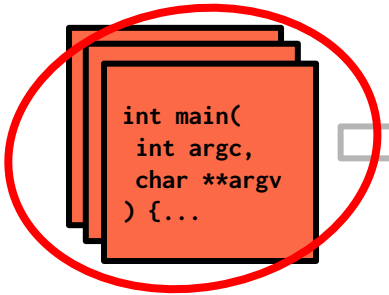
Fuzzing a compiler



Differential testing compilers



Differential testing compilers



Hard to generate!

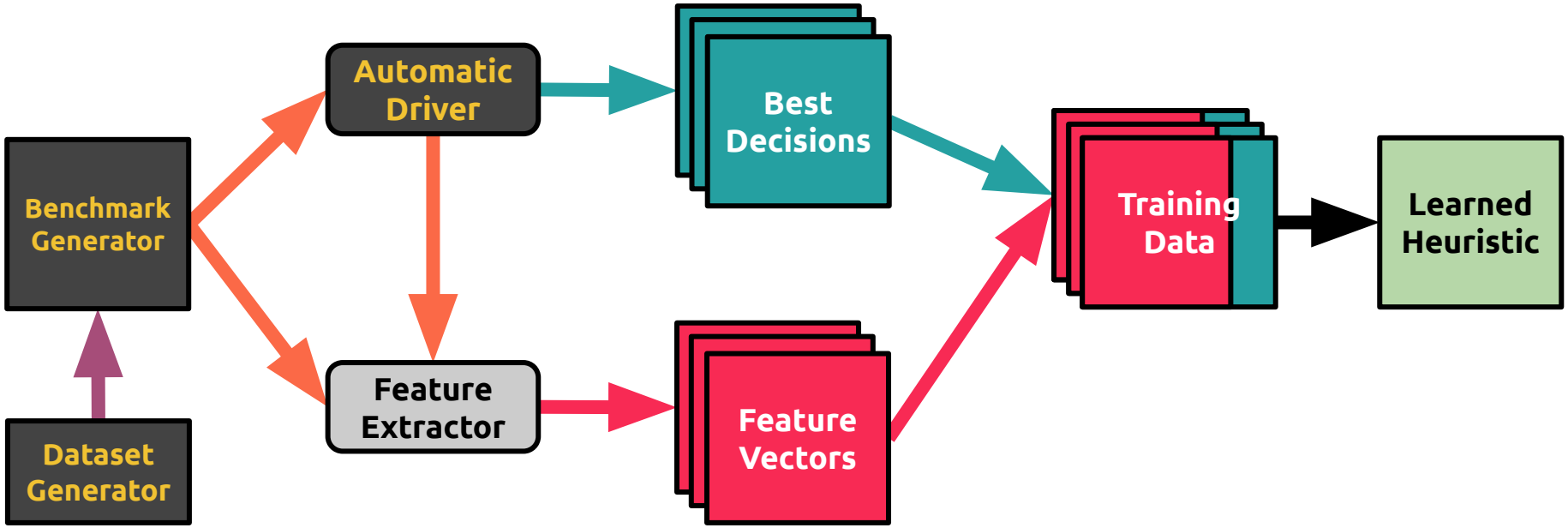
clang9.0 → a.out → `$ ./a.out` `42` ✓

clang6.0 → a.out → `$ ./a.out` `42` ✓

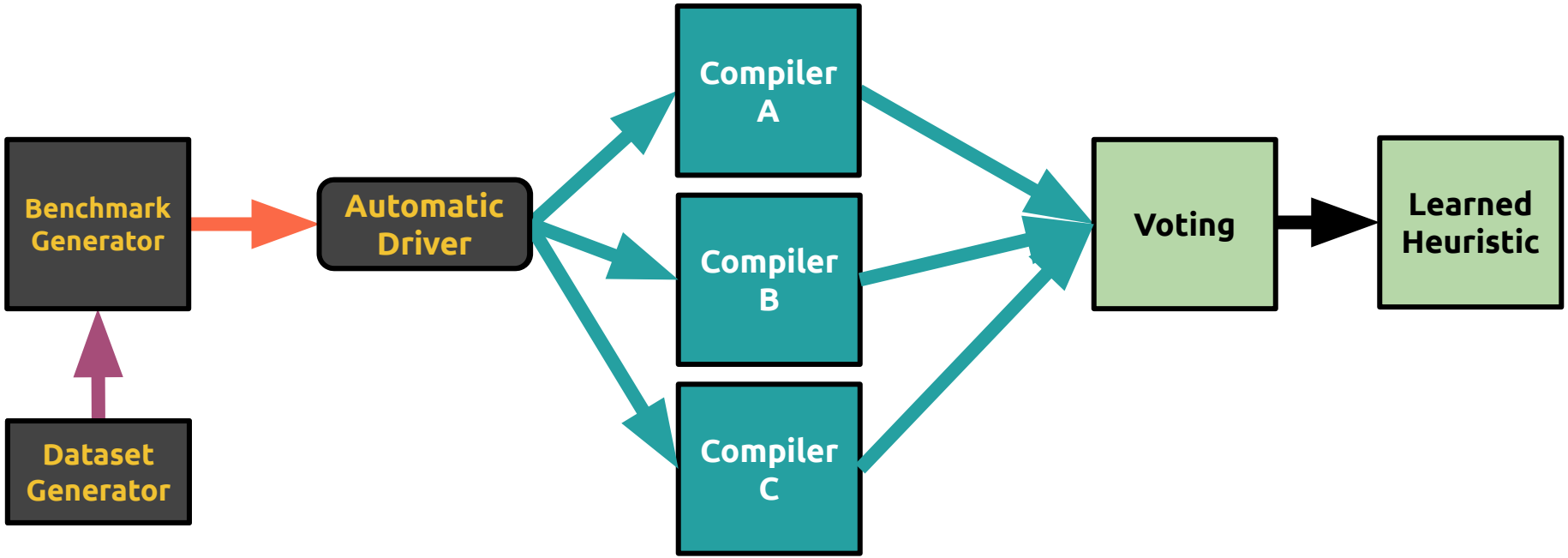
gcc5.5 → a.out → `$ ./a.out` `-141212311` ✗

Majority rules

What we have


Automatic Driver
Best Decisions
Benchmark Generator
Training Data
Learned Heuristic
Feature Extractor
Feature Vectors
Dataset Generator

What we *need*


Benchmark Generator
Dataset Generator
Automatic Driver
Compiler A
Compiler B
Compiler C
Voting
Learned Heuristic

Compiler Fuzzing through Deep Learning

C. Cummins, P. Petoumenos, Z. Wang, H. Leather

Automatic inference of fuzzers

102× less code than state-of-art

Small, interpretable test cases

In production

codeplay

In prototype

IBM

Thesis

1. 💡 Can we automate benchmark engineering?
2. 💡 Can we automate feature engineering?
3. 💡 Can we use (1) to automate testing?

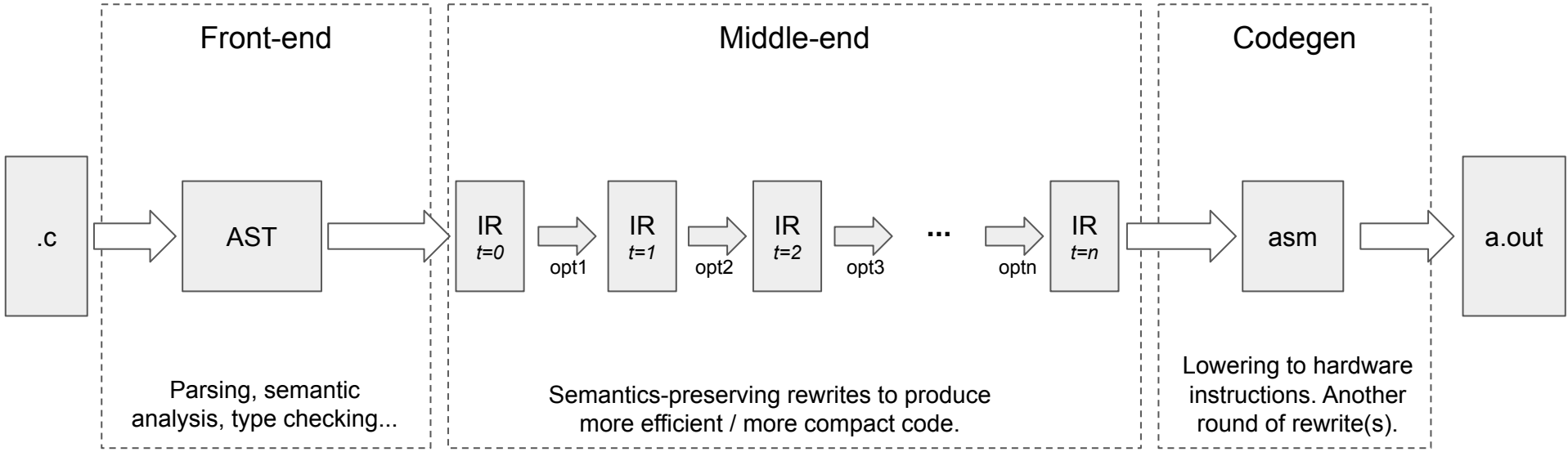
overview

1. What I used to do
2. What I now do
3. What you should do

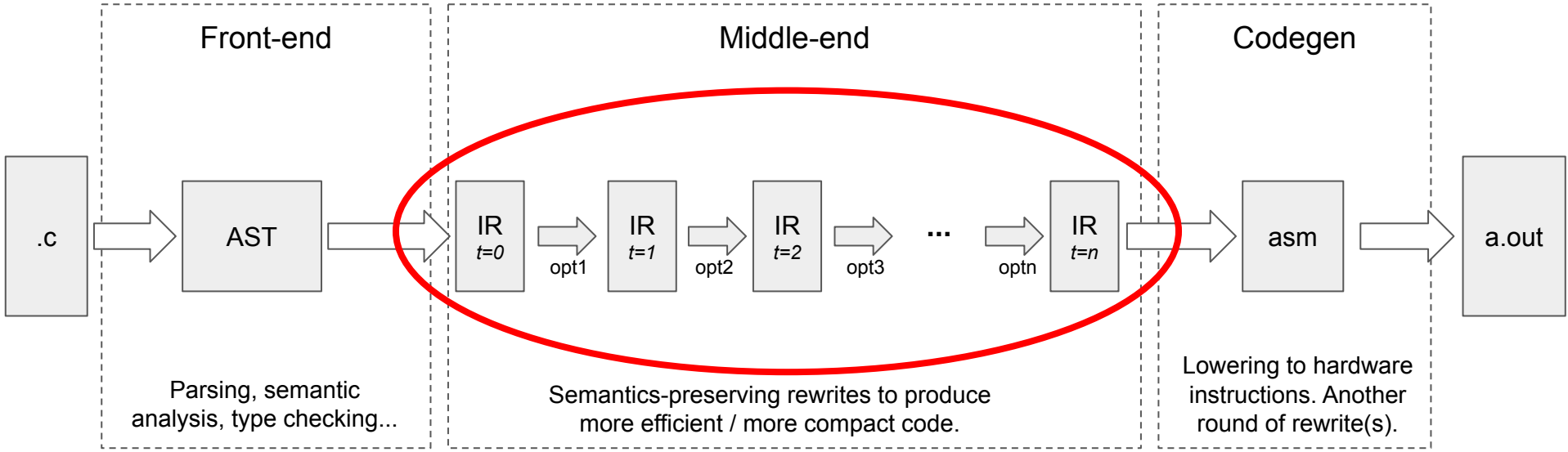
overview

1. What I used to do
2. What I now do
3. What you should do

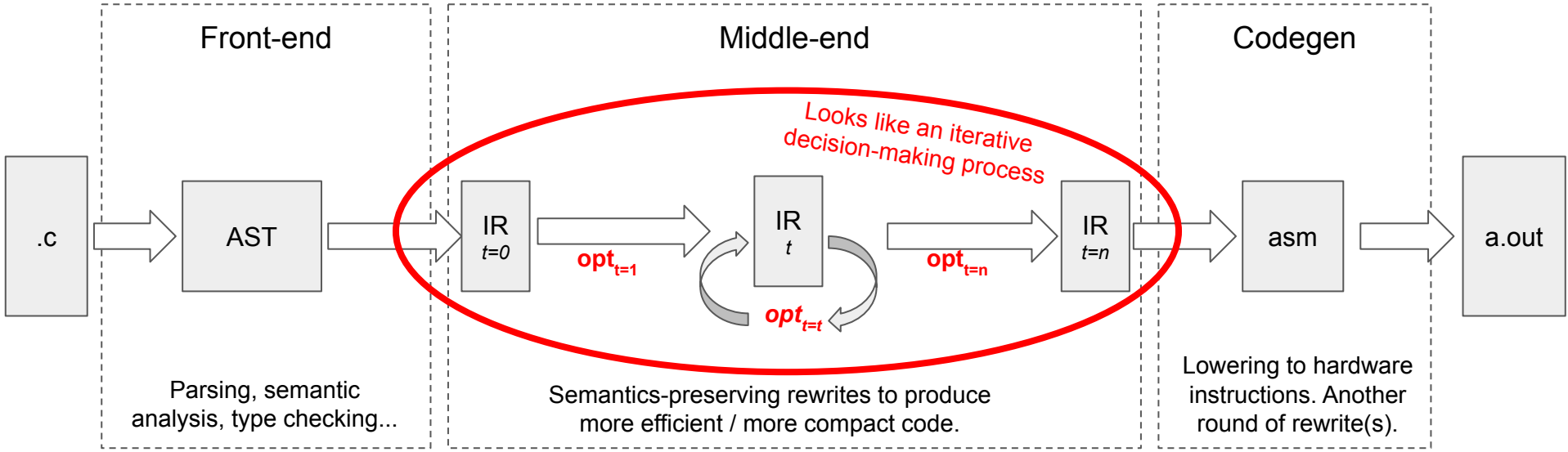
High-level overview of compiler



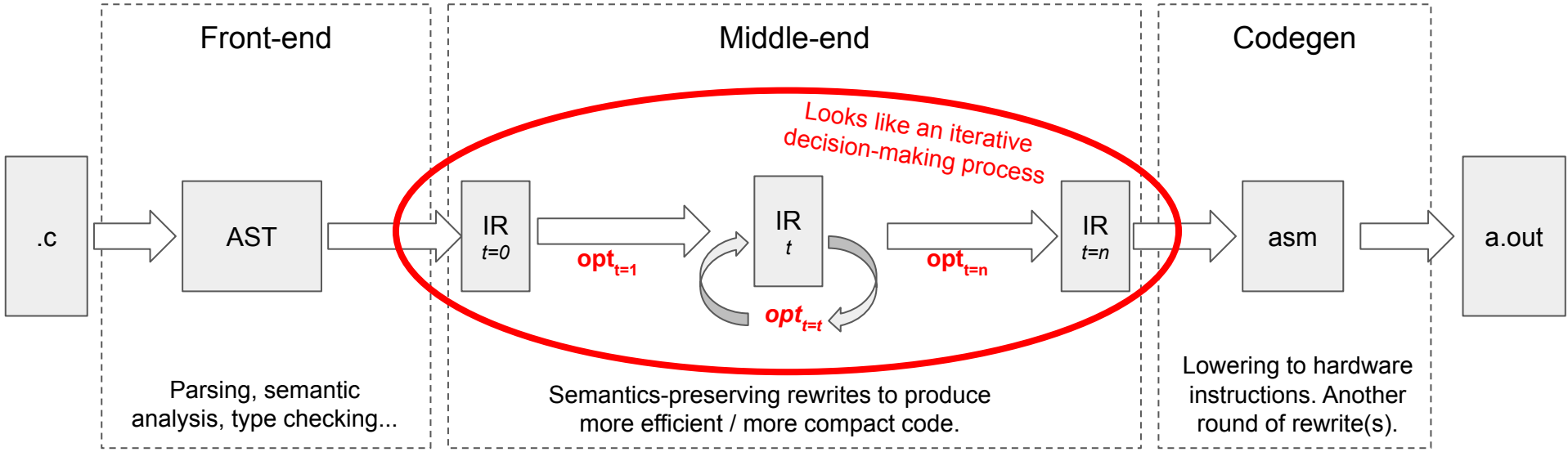
Optimizations are applied in a fixed order



Choosing the best is known as "Phase Ordering"



Choosing the best is known as "Phase Ordering"



Search shows large performance wins.
(up to 2.23x)

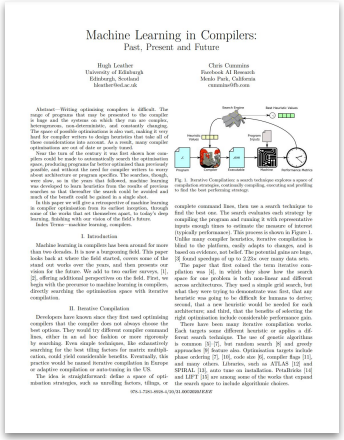
But search is slow, and must be repeated for each program, input, hardware config, compiler version.


facebook
1 Hacker Way

ML for Compilers, recap

Large wins to be had!

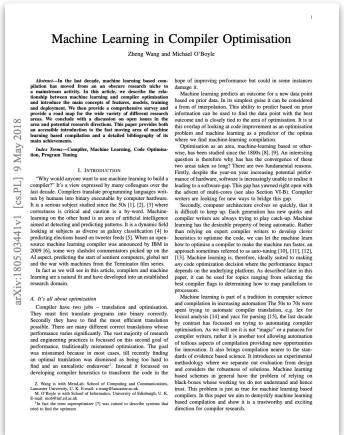
Months of work to set up compiler infrastructure + analyses



Machine Learning in Compilers:
Past, Present and Future

Short, high-level overview

https://chriscummins.cc/pub/2020-fdl.pdf



Machine Learning in Compiler Optimisation

Zheng Wang and Michael O'Boyle

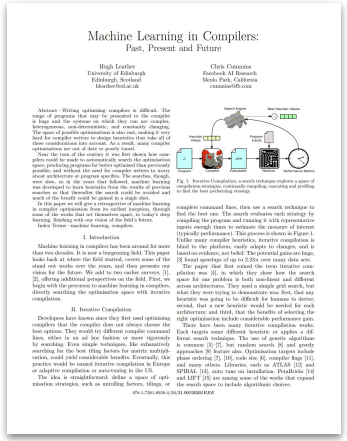
More comprehensive survey

https://arxiv.org/pdf/1805.03441.pdf

ML for Compilers, recap

Large wins to be had!

~~Months of work to set up compiler infrastructure + analyses~~



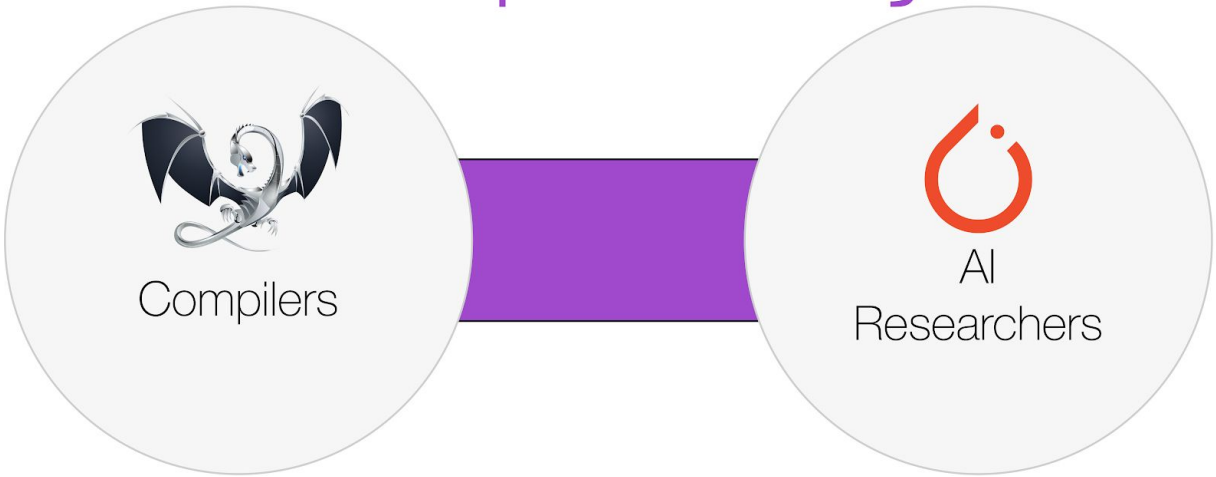
Short, high-level overview

https://chriscummins.cc/pub/2020-fdl.pdf

More comprehensive survey

https://arxiv.org/pdf/1805.03441.pdf

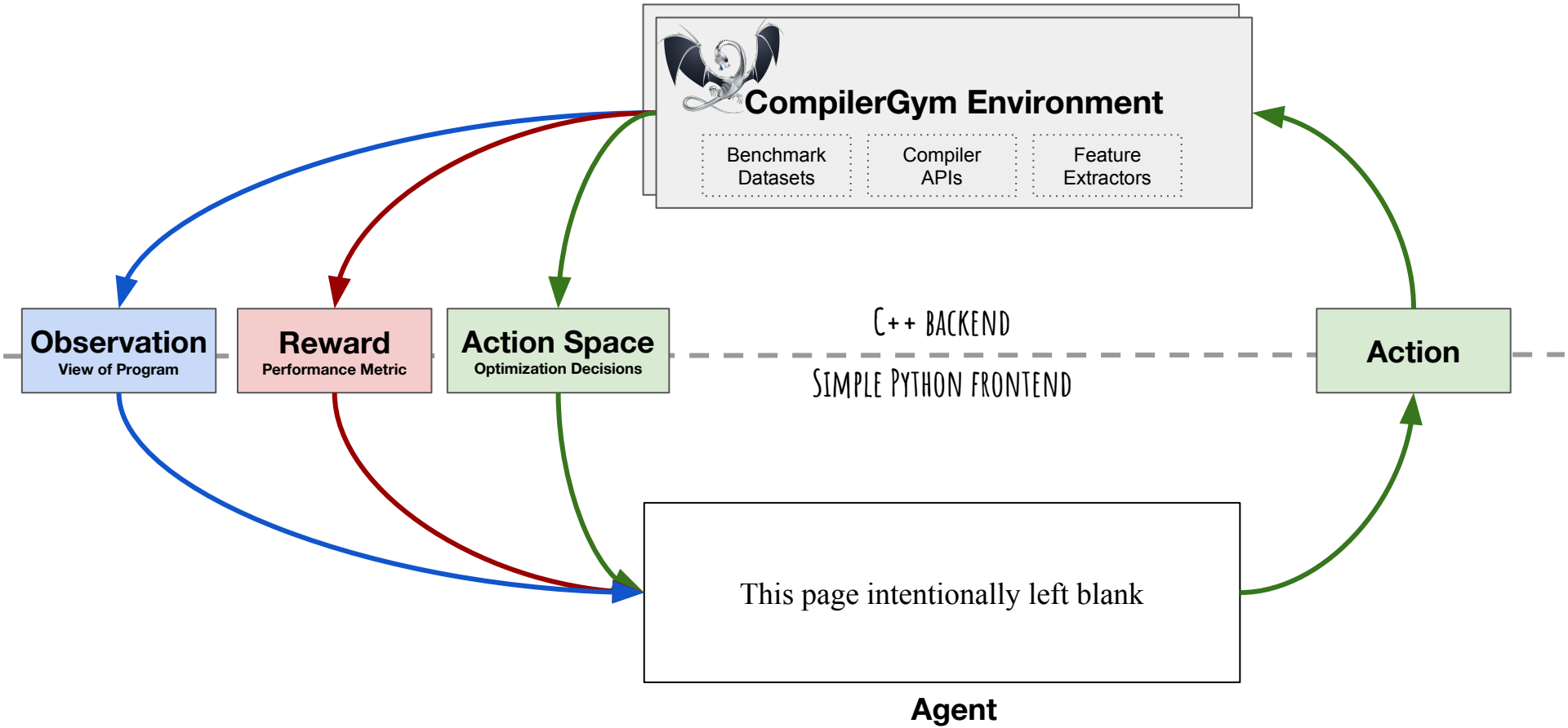
CompilerGym

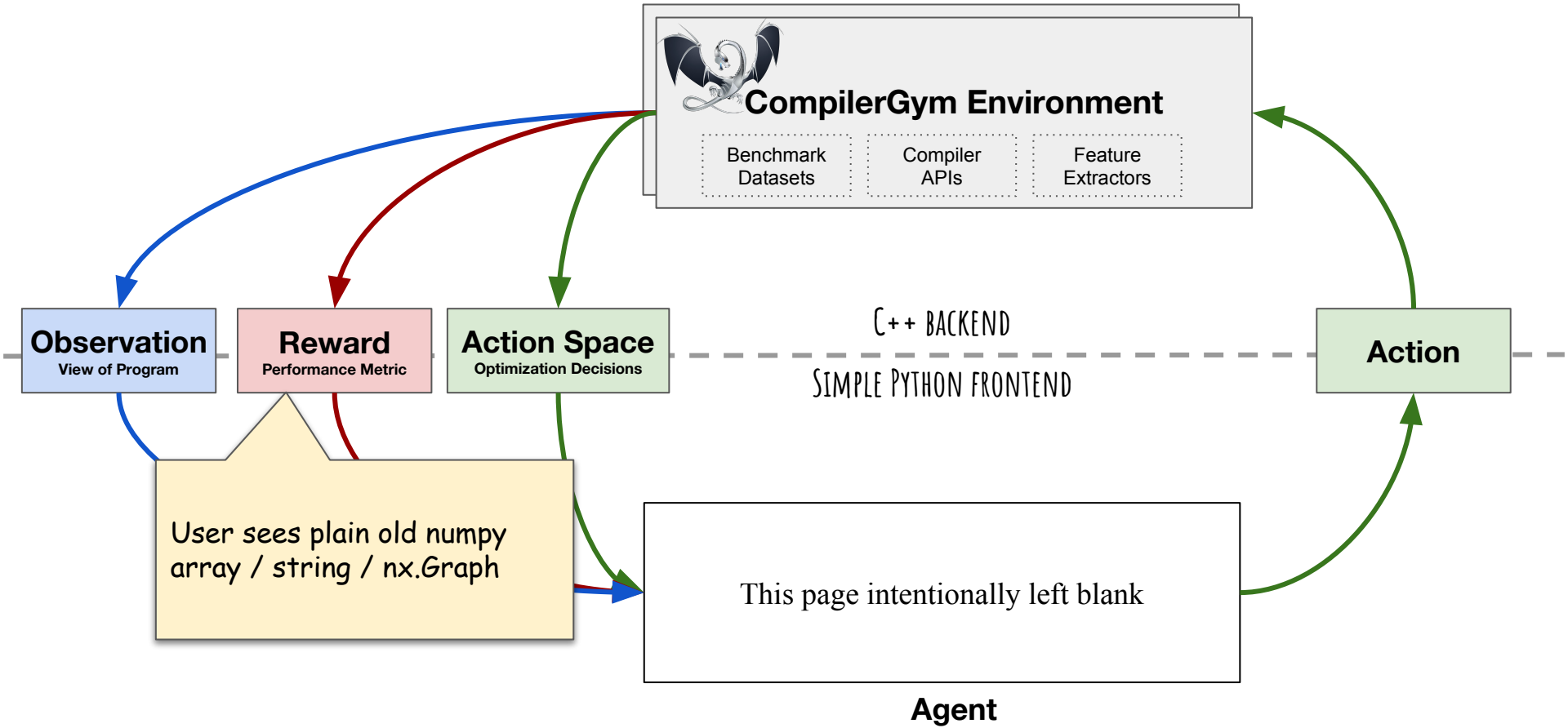


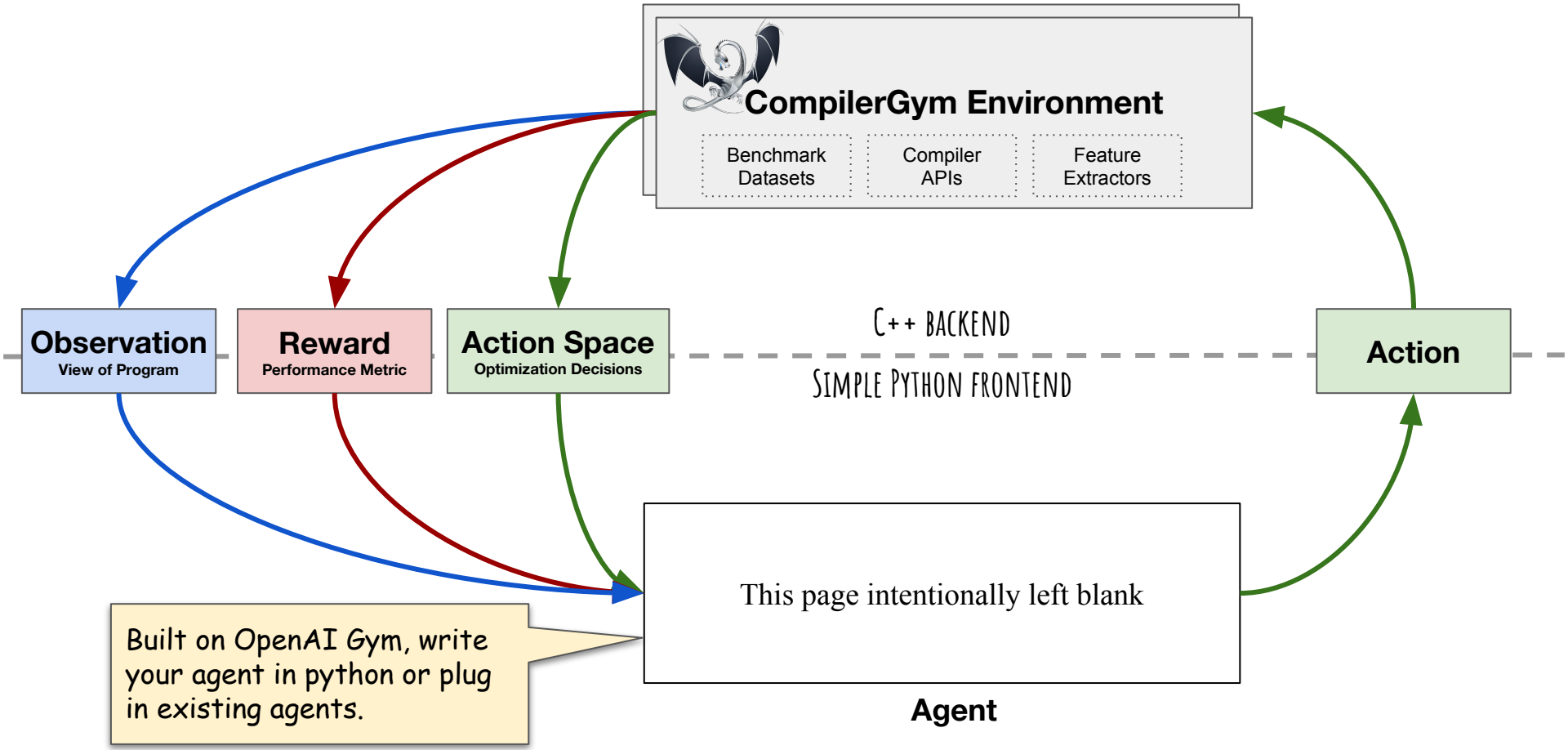
Making Compiler Optimization Accessible to All.

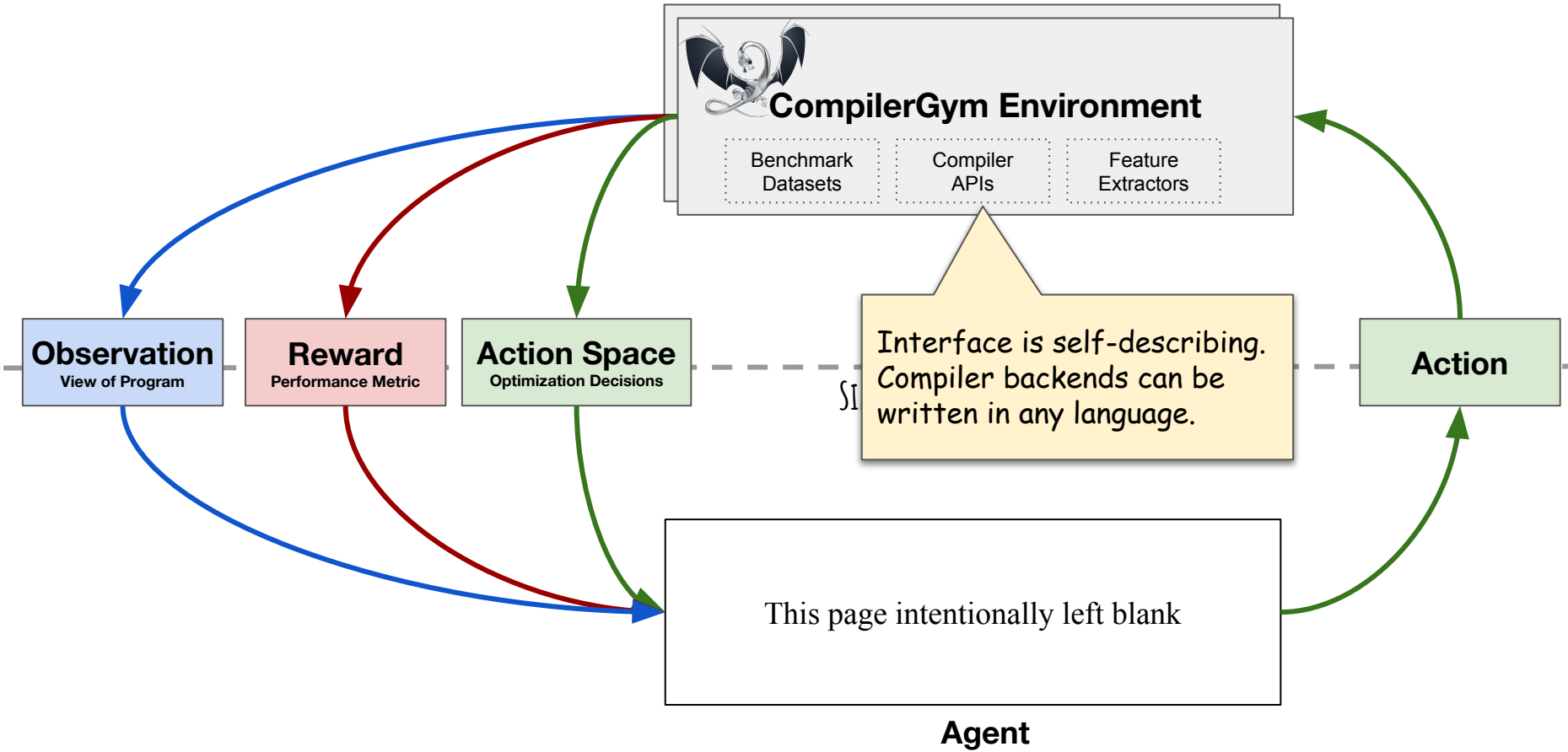
CompilerGym Goals

1. **Lower the barrier to entry** to AI for compilers
2. **Advance the state-of-the-art** in AI for compilers
3. Provide **common benchmarks** for compiler tasks


CompilerGym Environment
Benchmark Datasets
Compiler APIs
Feature Extractors
Observation
View of Program
Reward
Performance Metric
Action Space
Optimization Decisions
C++ BACKEND
SIMPLE PYTHON FRONTEND
Action
This page intentionally left blank
Agent


CompilerGym Environment
Benchmark Datasets
Compiler APIs
Feature Extractors
Observation
View of Program
Reward
Performance Metric
Action Space
Optimization Decisions
C++ BACKEND
SIMPLE PYTHON FRONTEND
Action
User sees plain old numpy array / string / nx.Graph
This page intentionally left blank
Agent


CompilerGym Environment
Benchmark Datasets
Compiler APIs
Feature Extractors
Observation
View of Program
Reward
Performance Metric
Action Space
Optimization Decisions
C++ BACKEND
SIMPLE PYTHON FRONTEND
Action
This page intentionally left blank
Agent
Built on OpenAI Gym, write your agent in python or plug in existing agents.


CompilerGym Environment
Benchmark Datasets
Compiler APIs
Feature Extractors
Interface is self-describing. Compiler backends can be written in any language.
Observation
View of Program
Reward
Performance Metric
Action Space
Optimization Decisions
Action
This page intentionally left blank
Agent

```
$ pip install compiler_gym                    # no compilation or config
```

```
$ pip install compiler_gym                # no compilation or config
$ python
>>> import gym
>>> import compiler_gym                   # ships with compiler environments
```

```
$ pip install compiler_gym                  # no compilation or config
$ python
>>> import gym
>>> import compiler_gym                     # ships with compiler environments
>>> env = gym.make(                         # creates a new environment
...     "llvm-v0",                          # selects a compiler to use
...     benchmark="cbench-v1/qsort",        # the program to compile
...     observation_space="Autophase",      # the observation space
...     reward_space="IrInstructionCountOz",  # the optimization target
... )
```

```
$ pip install compiler_gym                  # no compilation or config
$ python
>>> import gym
>>> import compiler_gym                     # ships with compiler environments
>>> env = gym.make(                         # creates a new environment
...     "llvm-v0",                          # selects a compiler to use
...     benchmark="cbench-v1/qsort",        # the program to compile
...     observation_space="Autophase",      # the observation space
...     reward_space="IrInstructionCountOz",  # the optimization target
... )
>>> observation = env.reset(benchmark)
>>> for _ in range(1000):
...     action = env.action_space.sample()    # your idea here!
...     observation, reward, done, info = env.step(action)
...     if done:
...         observation = env.reset(benchmark)
```

```
$ pip install compiler_gym                  # no compilation or config
$ python
>>> import gym
>>> import compiler_gym                     # ships with compiler environments
>>> env = gym.make(                         # creates a new environment
...     "llvm-v0",                          # selects a compiler to use
...     benchmark="cbench-v1/qsort",        # the program to compile
...     observation_space="Autophase",      # the observation space
...     reward_space="IrInstructionCountOz",  # the optimization target
... )
>>> observation = env.reset(benchmark)
>>> for _ in range(1000):
...     action = env.action_space.sample()    # your idea here!
...     observation, reward, done, info = env.step(action)
...     if done:
...         observation = env.reset(benchmark)
>>> env.write_bitcode("benchmark.bc")        # save program to disk for later analysis
```

CompilerGym Features

- Three compiler problems included:
  - LLVM phase ordering
  - CUDA code generation
  - GCC flag tuning
- Batteries included
  - **Millions of programs** for training, 41x more than prior works
  - Huge optimization spaces
  - Several observation spaces from published work
  - Optimizing for **reward** or **codesize**
- Battle tested
- **27x faster** than prior works

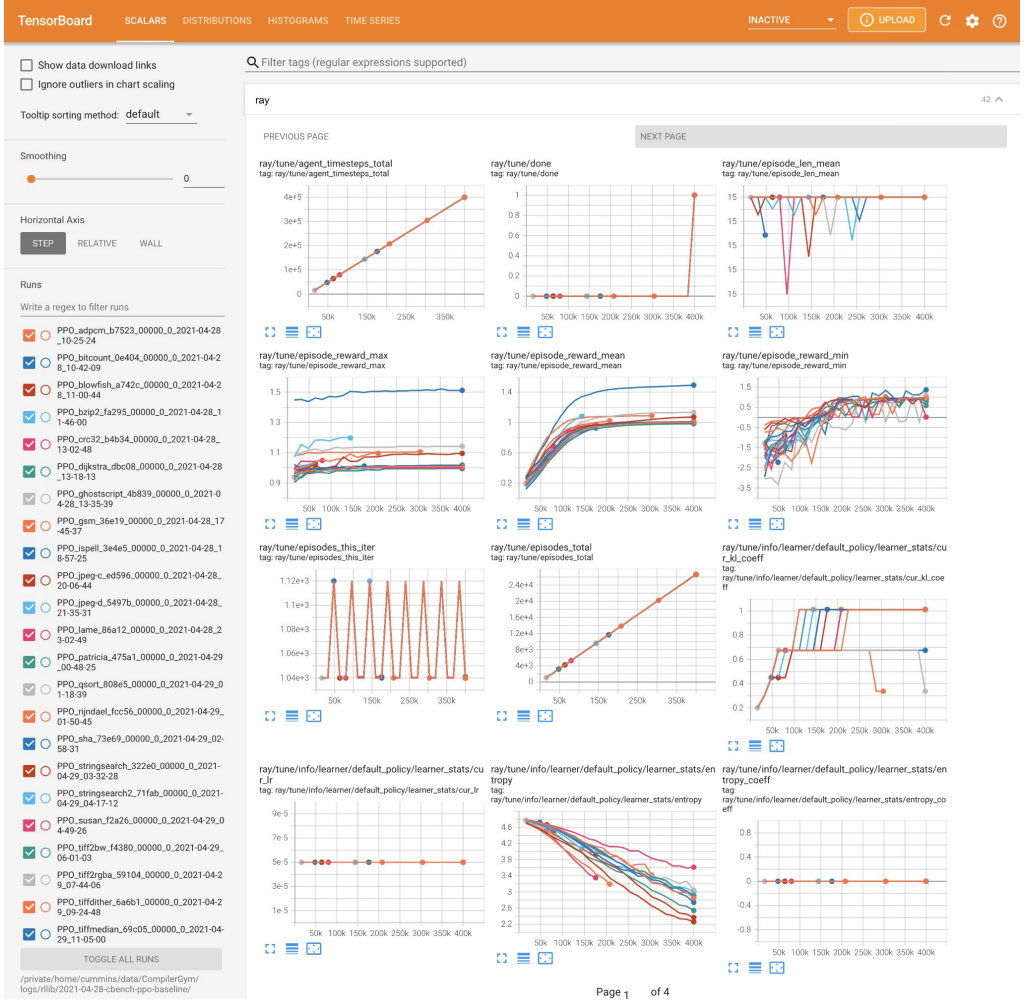
Why I think its cool

Repro state-of-the-art in < 15 LOC

```
import gym
import compiler_gym
from ray import tune
from ray.rllib.agents.ppo import PPOTrainer

tune.register_env(
    "CompilerGym",
    lambda _: gym.wrappers.TimeLimit(
        max_episode_steps=45,
        env=compiler_gym.make(
            "llvm-autophase-ic-v0",
            benchmark="cbench-v1/qsort",
        ),
    ),
)

tune.run(PPOTrainer, config={"env": "CompilerGym"})
```



Public Leaderboards 🏆

Author	Algorithm	Date	Walltime (lower is better)	Code size reduction (higher is better)
Facebook	Nevergrad (default optimizer)	2021-05	3,545.635s	1.074x
Facebook	Random agent (t=10800)	2021-03	10,512.356s	1.062x
Facebook	Greedy Search	2021-03	169.237s	1.055x
Facebook	Tabular Q	2021-04	2,534.305	1.036x
Facebook	Random agent (t=10)	2021-03	42.939s	1.031x
Your name here	**Your technique here**			

Encourage friendly competition across common benchmarks.
Provide a platform for promoting (and validating) people's work.
Low effort submission via pull requests.

Making Compiler Optimization Accessible to All

1. **Compiler research is fun 🙂**
   - Challenging, {high-dimensional, noisy} {action, state, reward} spaces
   - Impact! Lots of room to innovate. Publications and potential for real world use
2. **CompilerGym reduces barrier to entry to "pip install."**
   - Lets AI researchers focus on solving the problems
   - Lets compiler developers focus on the compiler
3. **Fast pace of development, lots to look forward to!**
   - New compilers + runtime optimization for LLVM
   - Collaborations with academia to push state-of-art

https://github.com/facebookresearch/CompilerGym

overview

1. What I used to do
2. What I now do
3. What you should do

Deep Learning for Compilers, Fun, & Profit

- **A PhD is a lot of fun!**
- **My research is on using ML to improve compilers.**
- **Now working on democratizing the field.**

Chris Cummins
https://chriscummins.cc
cummins@fb.com