Deep Learning for Compilers

Hugh Leather

University of Edinburgh

Overview

- Machine Learning for Compilers
- Generating Benchmarks 

- Deep Learned Heuristics 

- Deep Fuzzing Compiler Testing 

- Future Work

Chris Cummins University of Edinburgh

Pavlos Petoumenos University of Edinburgh

Zheng Wang Lancaster University

Hugh Leather University of Edinburgh

THE UNIVERSITY of EDINBURGH informatics

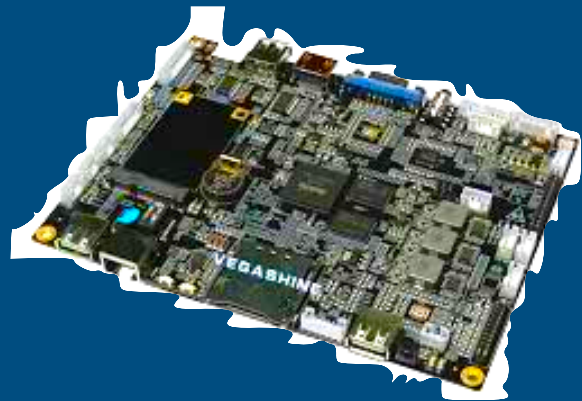
EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC Engineering and Physical Sciences Research Council

Overview

- **Machine Learning for Compilers**
- Generating Benchmarks — Best Paper Award
- Deep Learned Heuristics — Best Paper Award
- Deep Fuzzing Compiler Testing — Best Paper Award
- Future Work

Compilers are *hard*

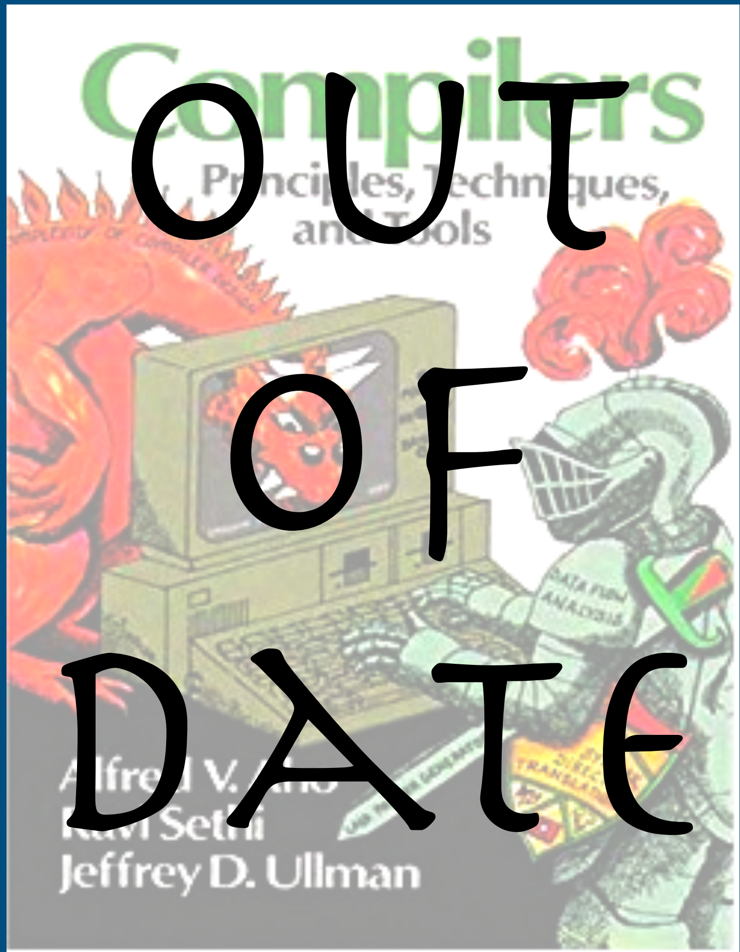


Huge number of variables
NP-hard or worse
Keep changing

Nondeterministic machines
Many components
Keep changing

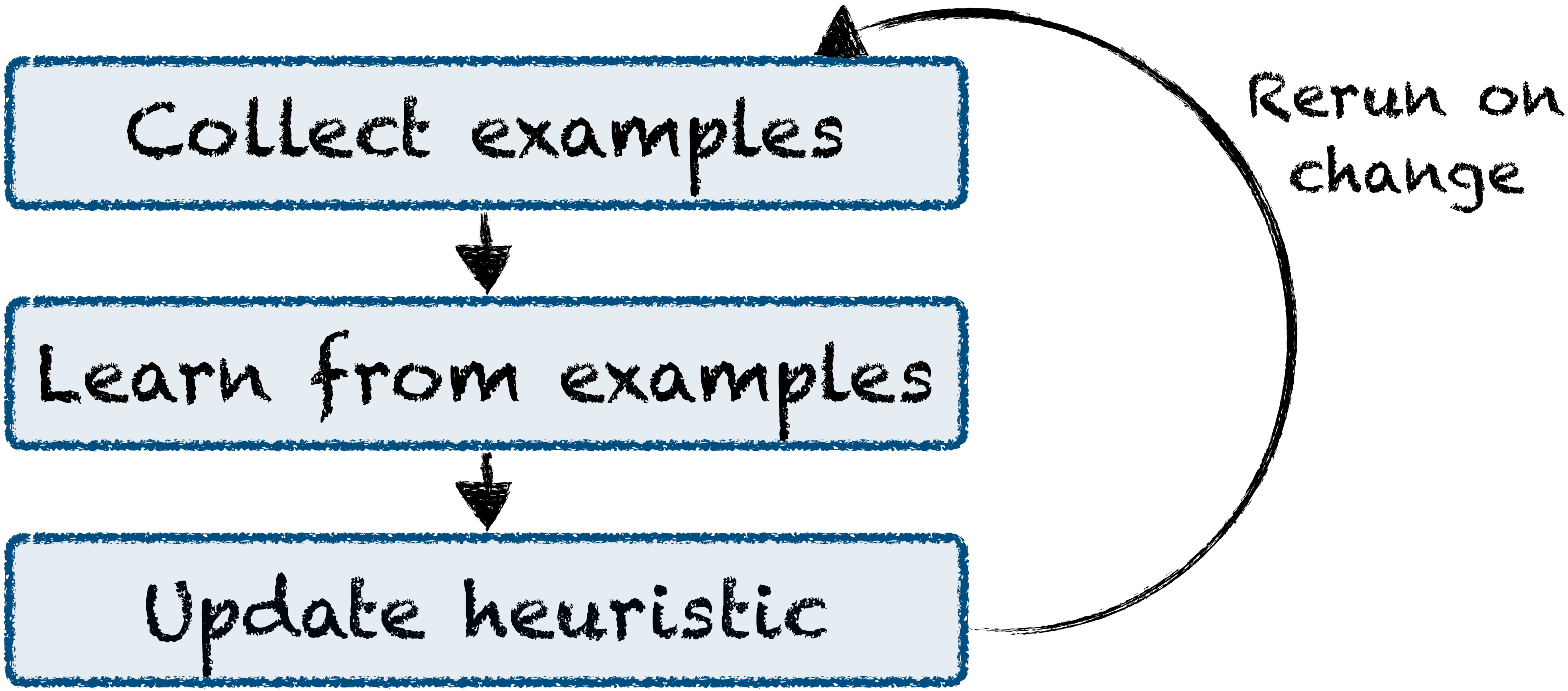
Compilers are *hard*

Slow programs

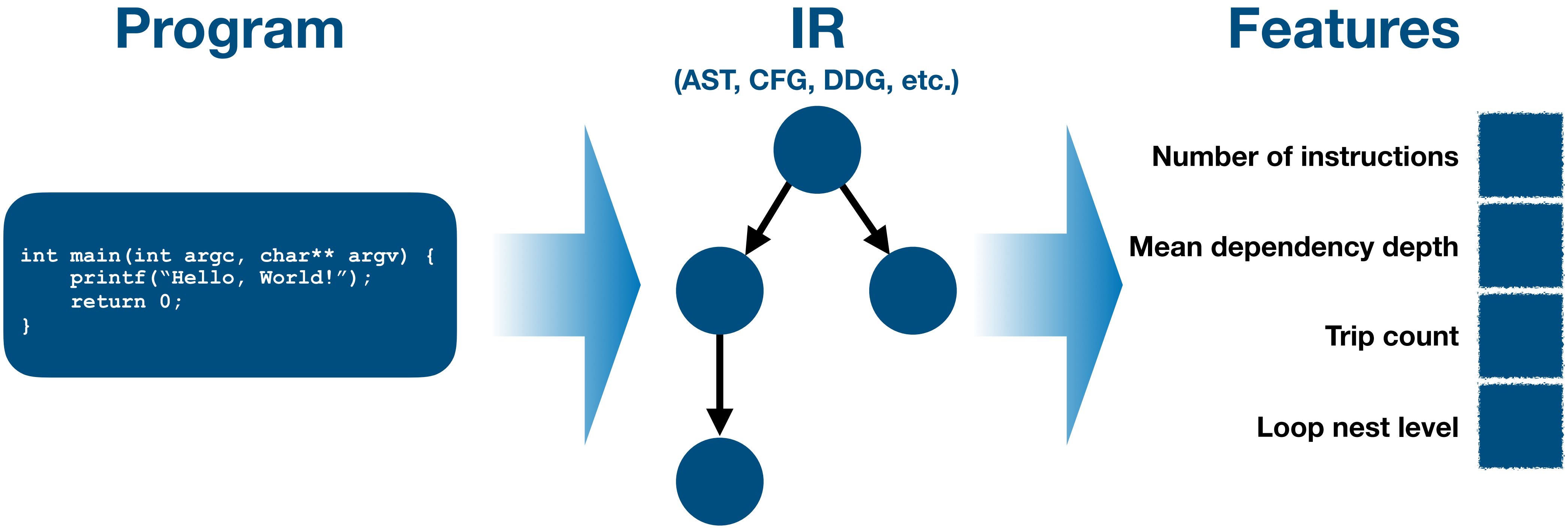


Energy waste

Machine Learning to the Rescue

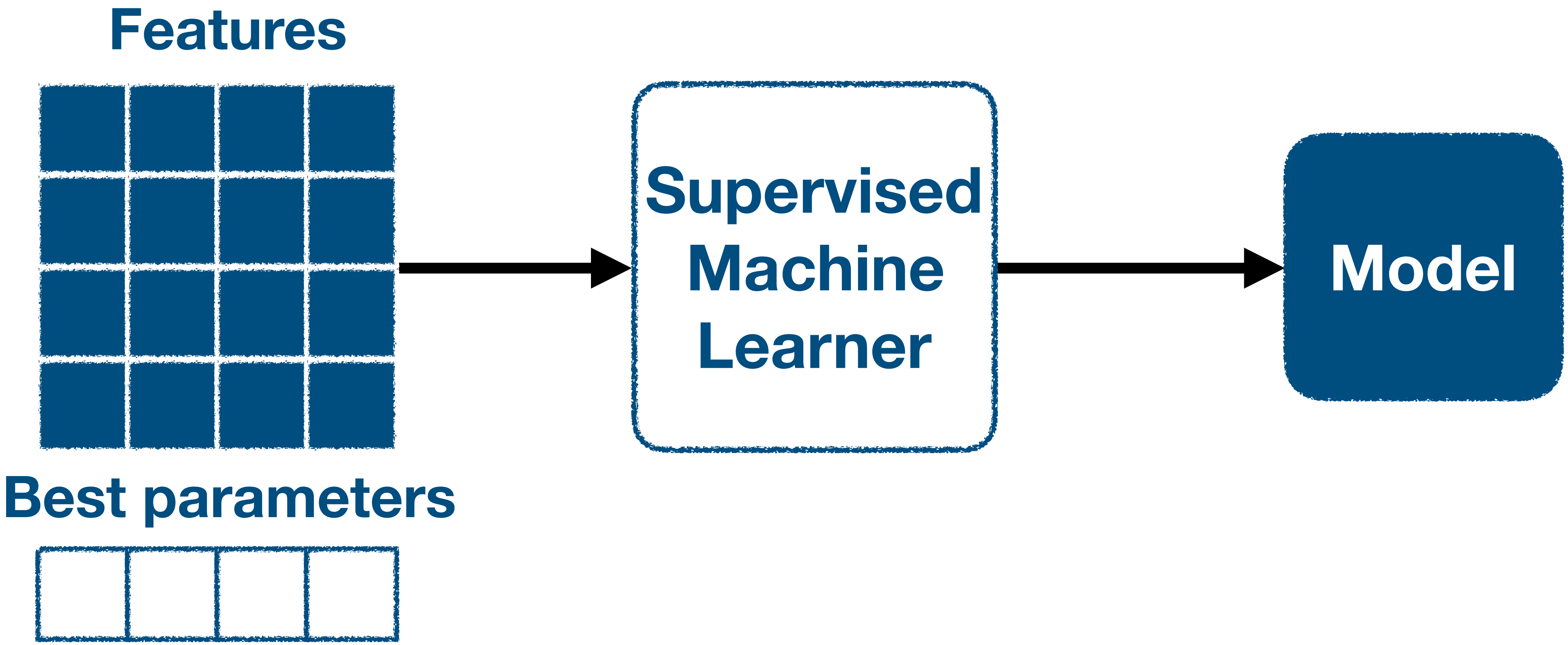


Summarise the Program

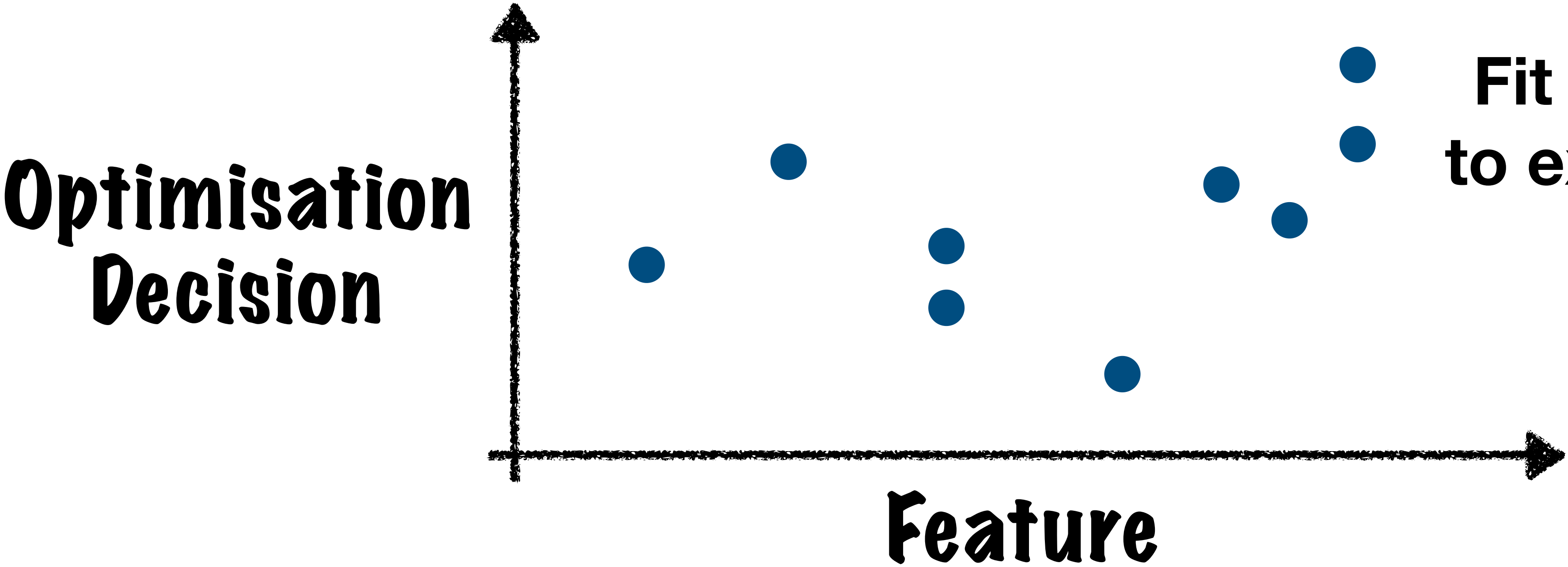


Gather Examples

Learn a Model

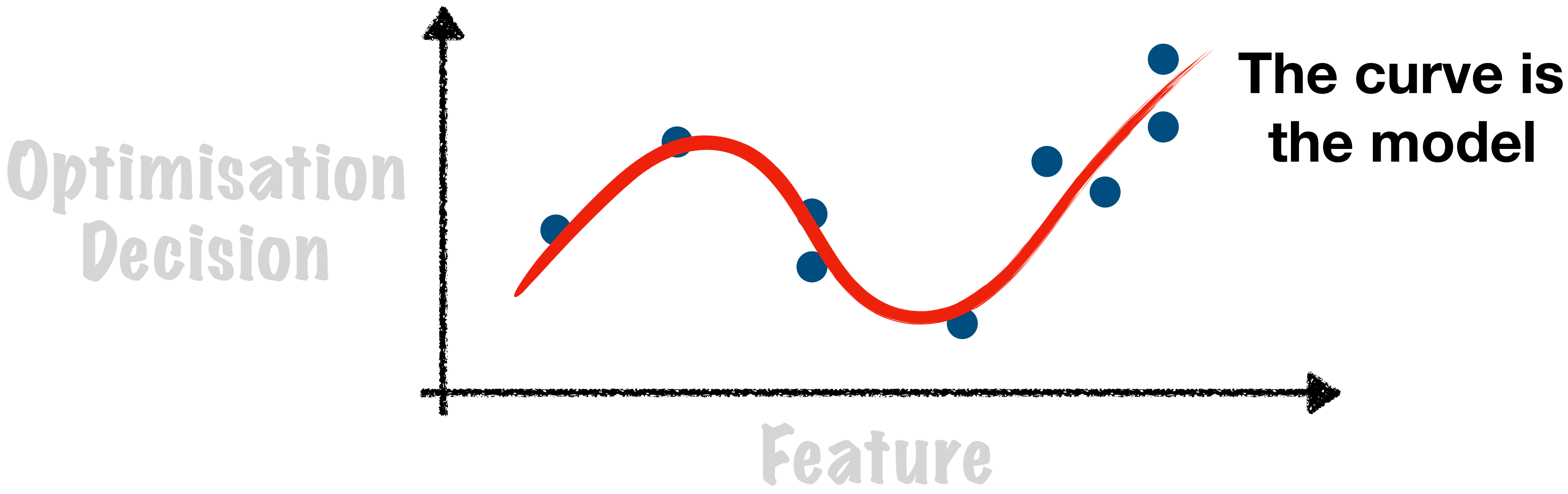


What is a Model?

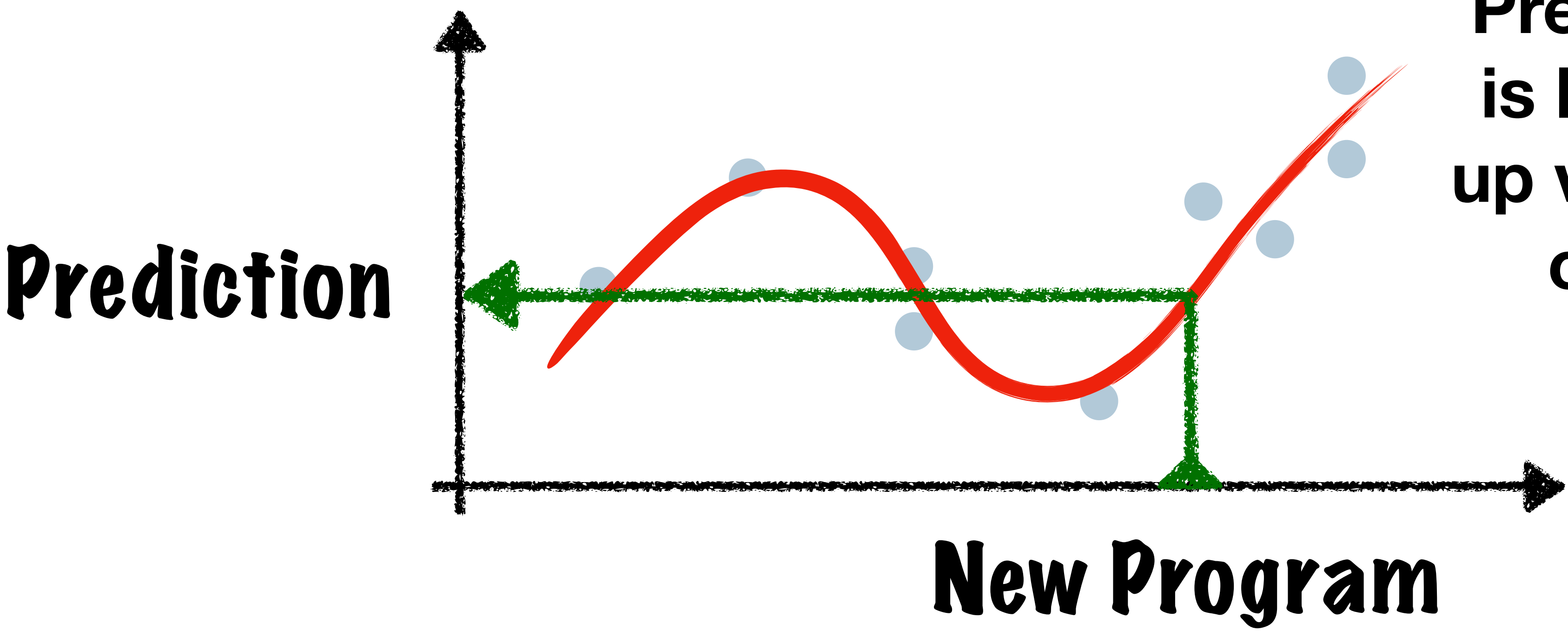


Fit a curve
to examples

What is a Model?

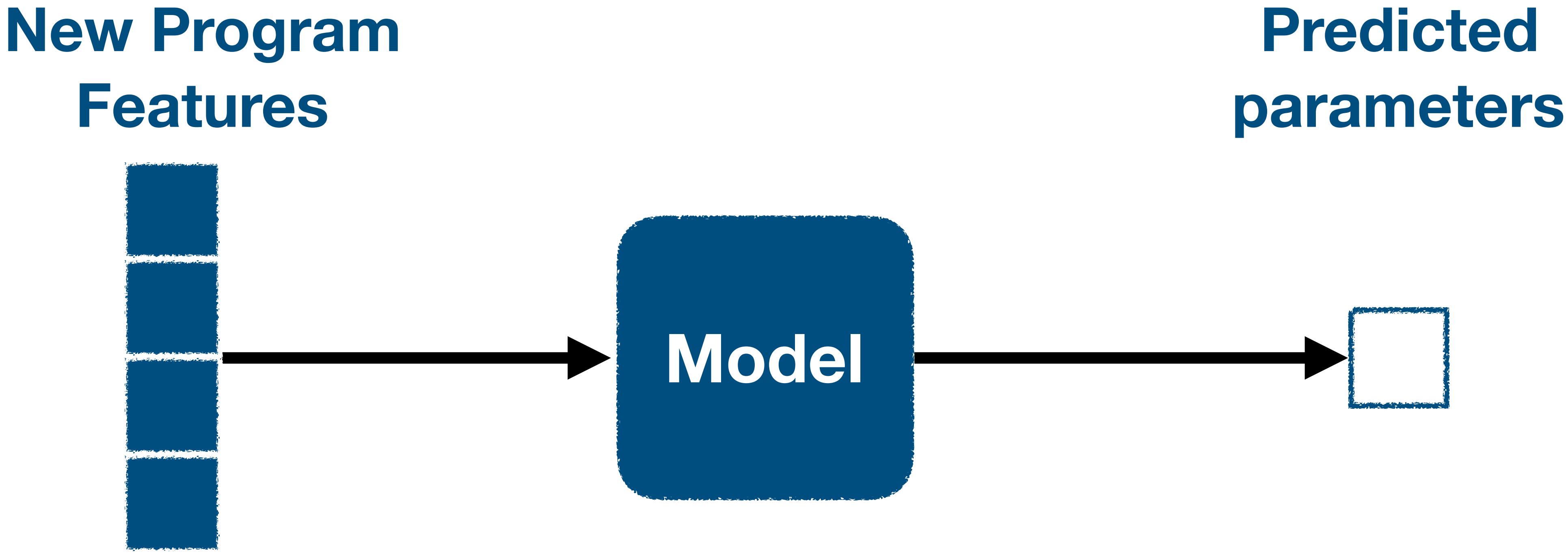


What is a Model?



Prediction is looking up value on curve

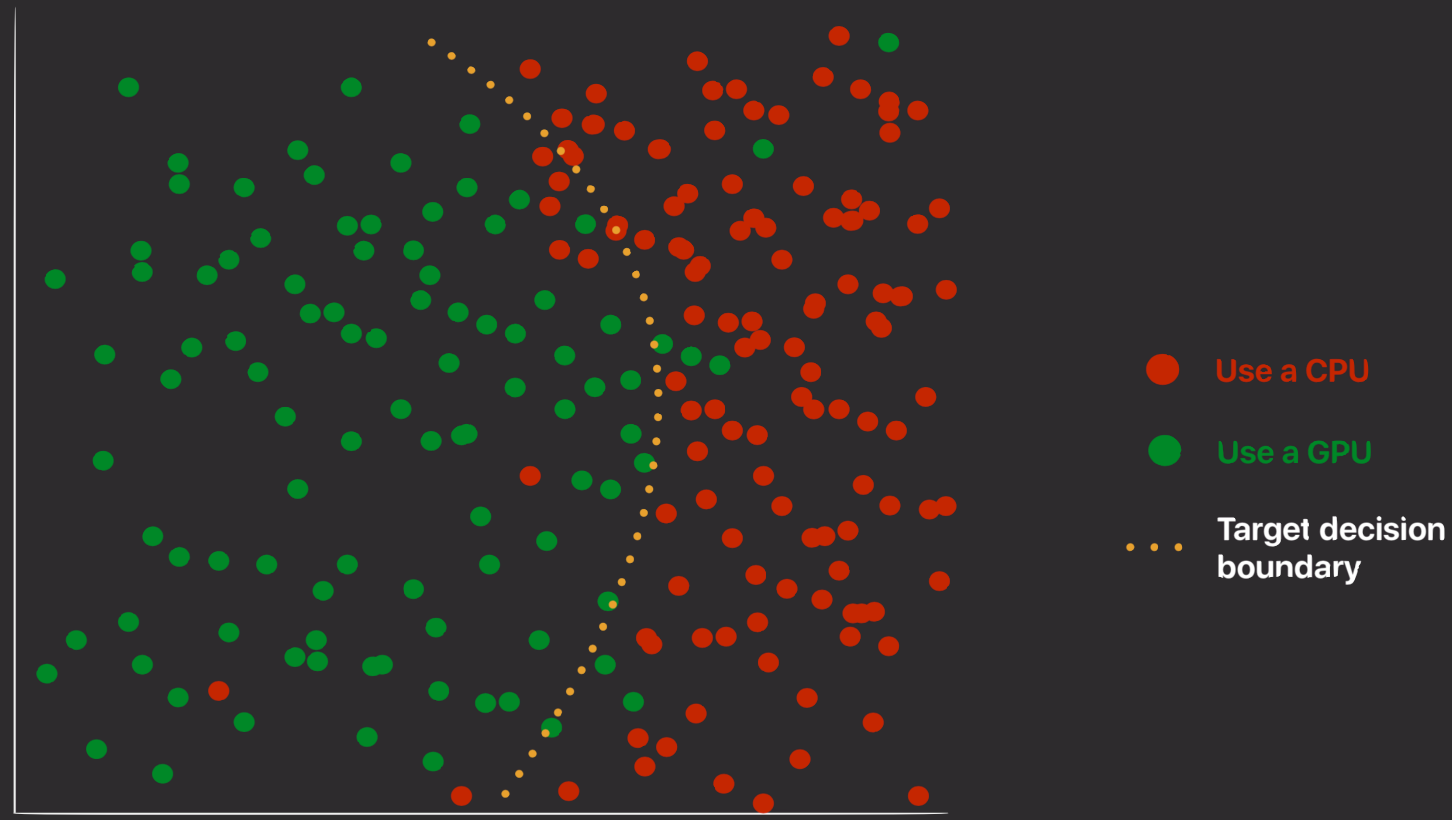
Use the Model



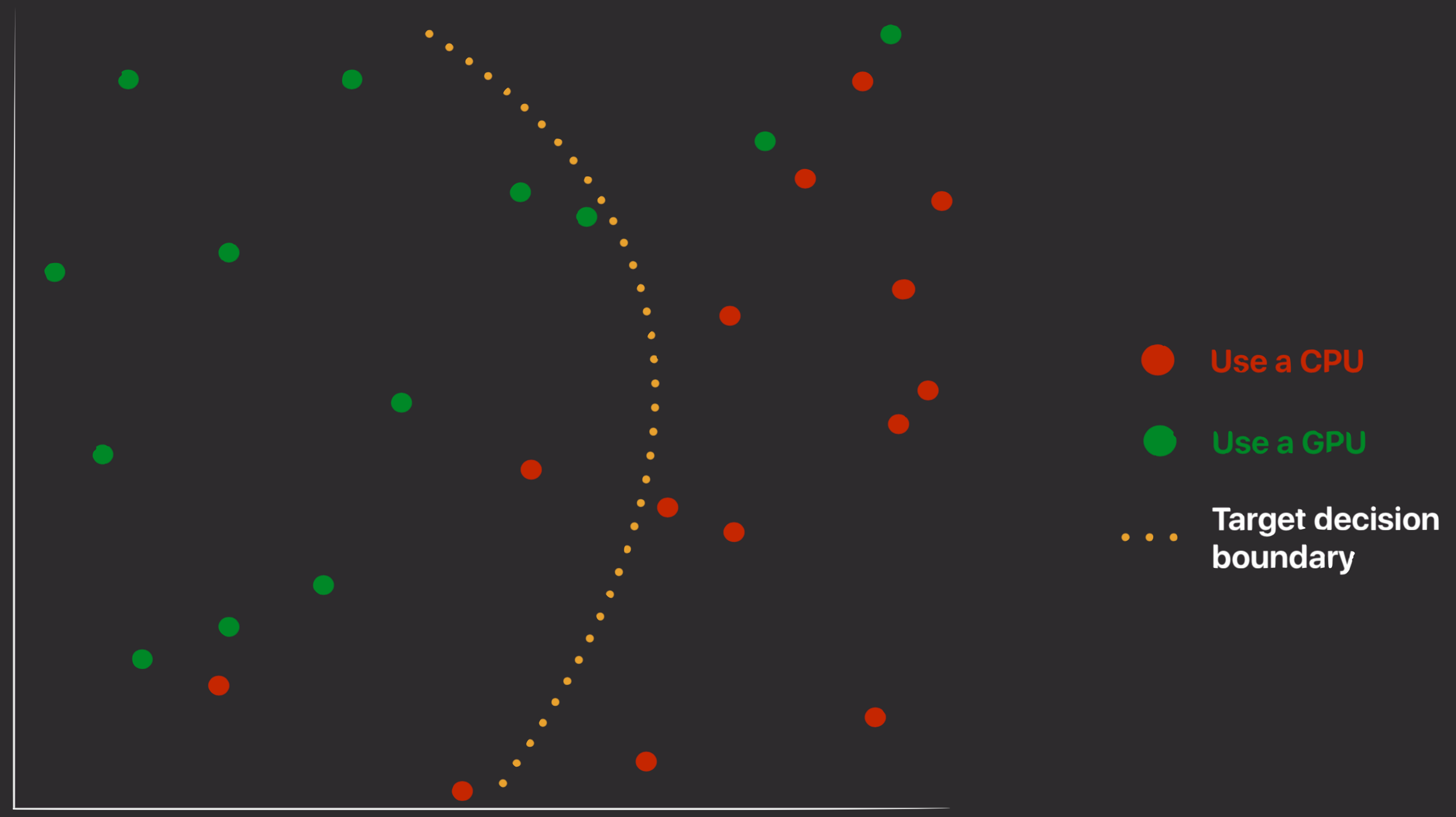
Overview

- Machine Learning for Compilers
- **Generating Benchmarks**
- Deep Learned Heuristics
- Deep Fuzzing Compiler Testing
- Future Work

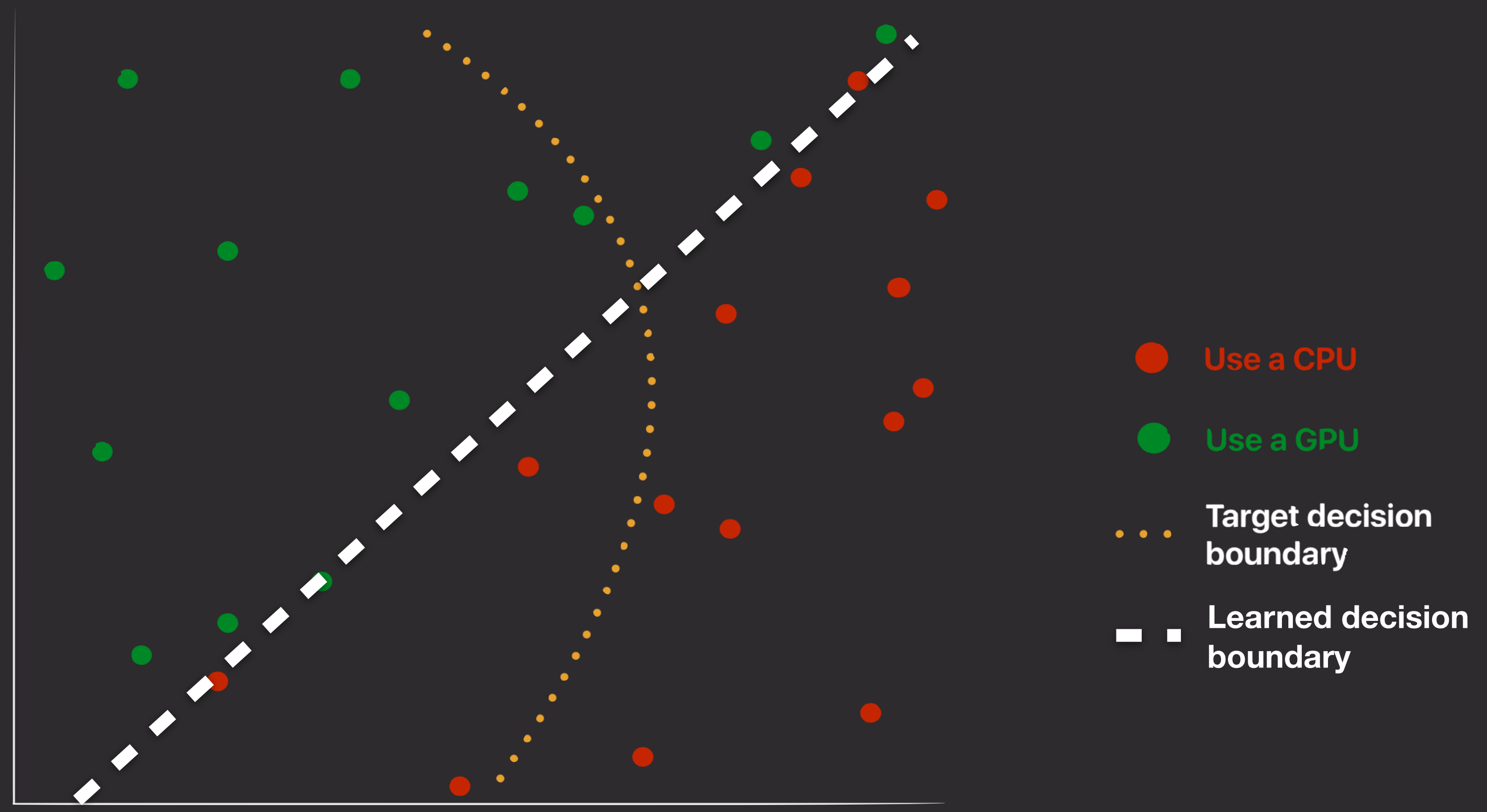
What we want.



What we get.

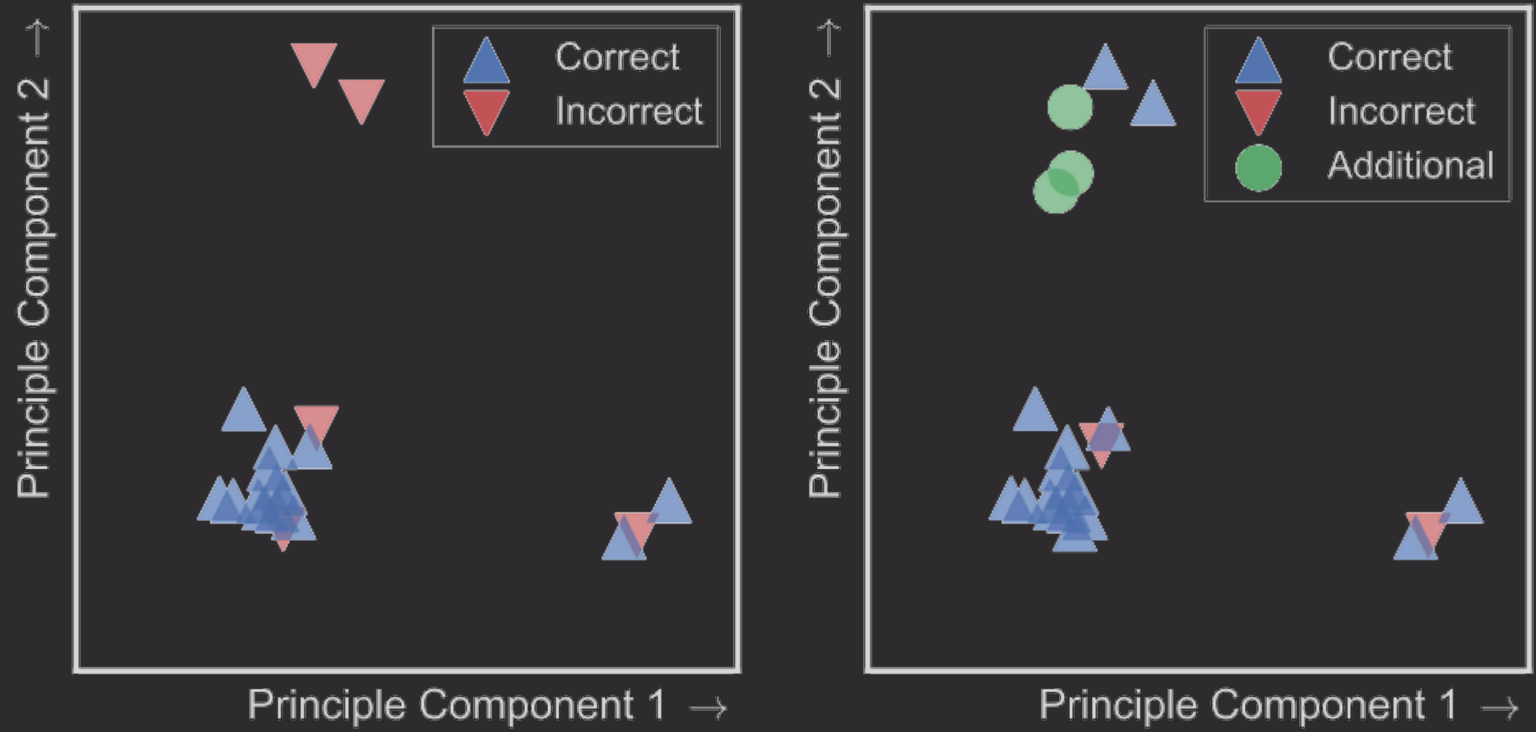


Learn the Wrong Thing!



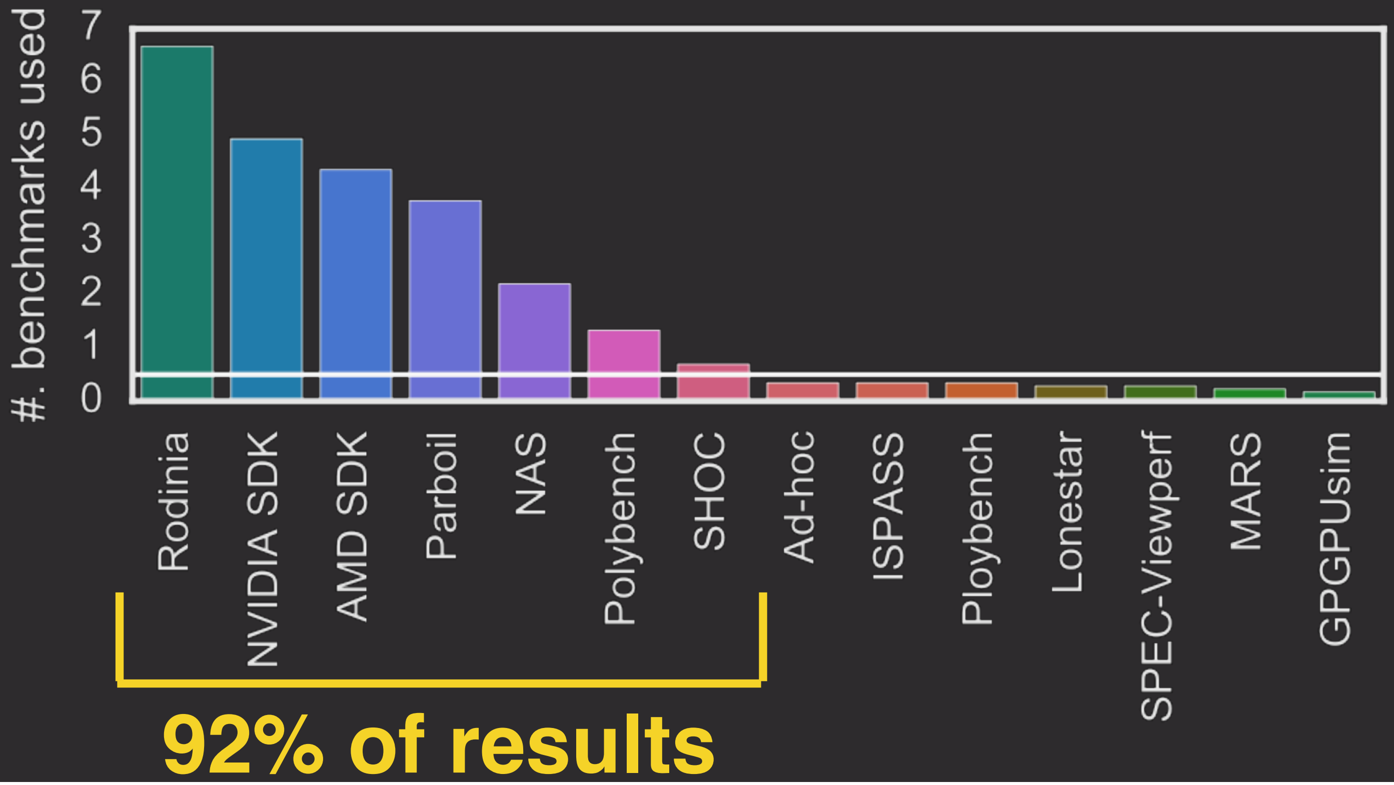
problem statement

1. more benchmarks = better models



problem statement

1. more benchmarks = better models
2. there aren't enough benchmarks



problem statement

1. more benchmarks = better models
2. there aren't enough benchmarks

avg compiler paper = 17
Iris dataset = 150
MNIST dataset = 60,000
ImageNet dataset = 10,000,000

problem statement

1. more benchmarks = better models
2. there aren't enough benchmarks
3. benchmarks must be diverse

	AMD	NPB	NVIDIA	Parboil	Polybench	Rodinia	SHOC
AMD	-	38.0%	74.5%	76.7%	21.7%	45.8%	35.9%
NPB	22.7%	-	45.3%	36.7%	13.4%	16.1%	23.7%
NVIDIA	29.9%	37.9%	-	21.8%	78.3%	18.1%	63.2%
Parboil	89.2%	28.2%	28.2%	-	41.3%	73.0%	33.8%
Polybench	58.6%	30.8%	45.3%	11.5%	-	43.9%	12.1%
Rodinia	39.8%	36.4%	29.7%	36.5%	46.1%	-	59.9%
SHOC	42.9%	71.5%	74.1%	41.4%	35.7%	81.0%	-

problem statement

1. more benchmarks = better models
2. there aren’t enough benchmarks
3. benchmarks must be diverse

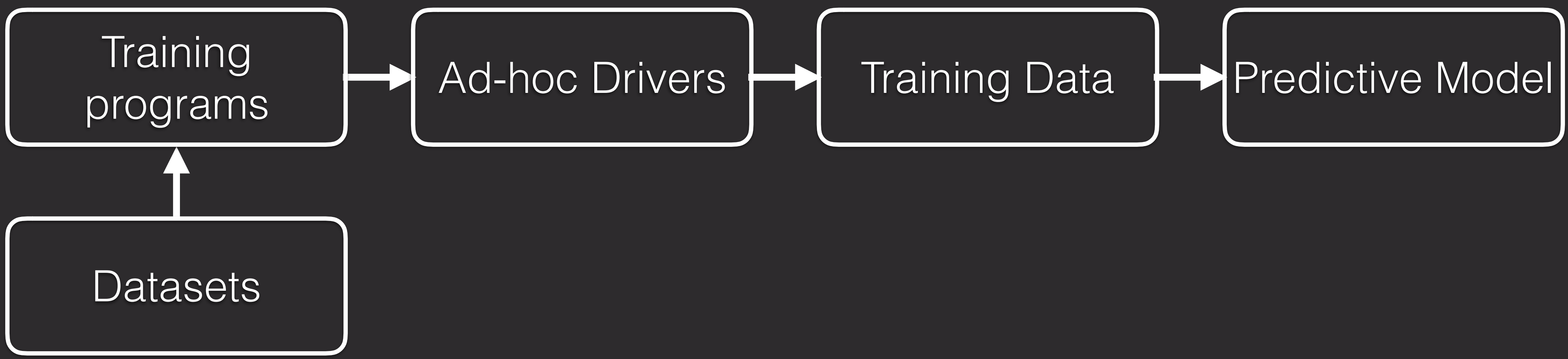
?

Contributions

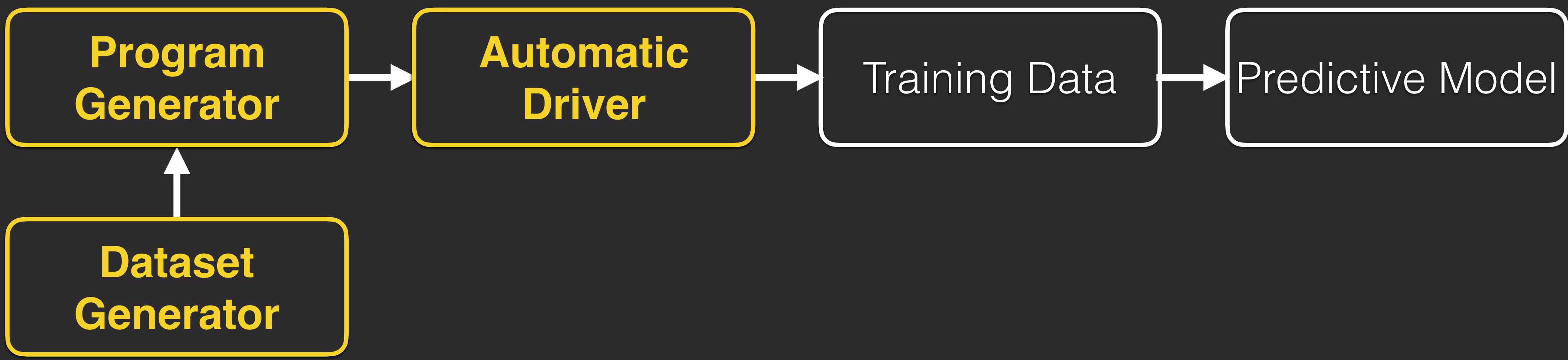
<u>Human-like</u> program generator

Model produces code
<u>4.3x faster</u>
than state of the art

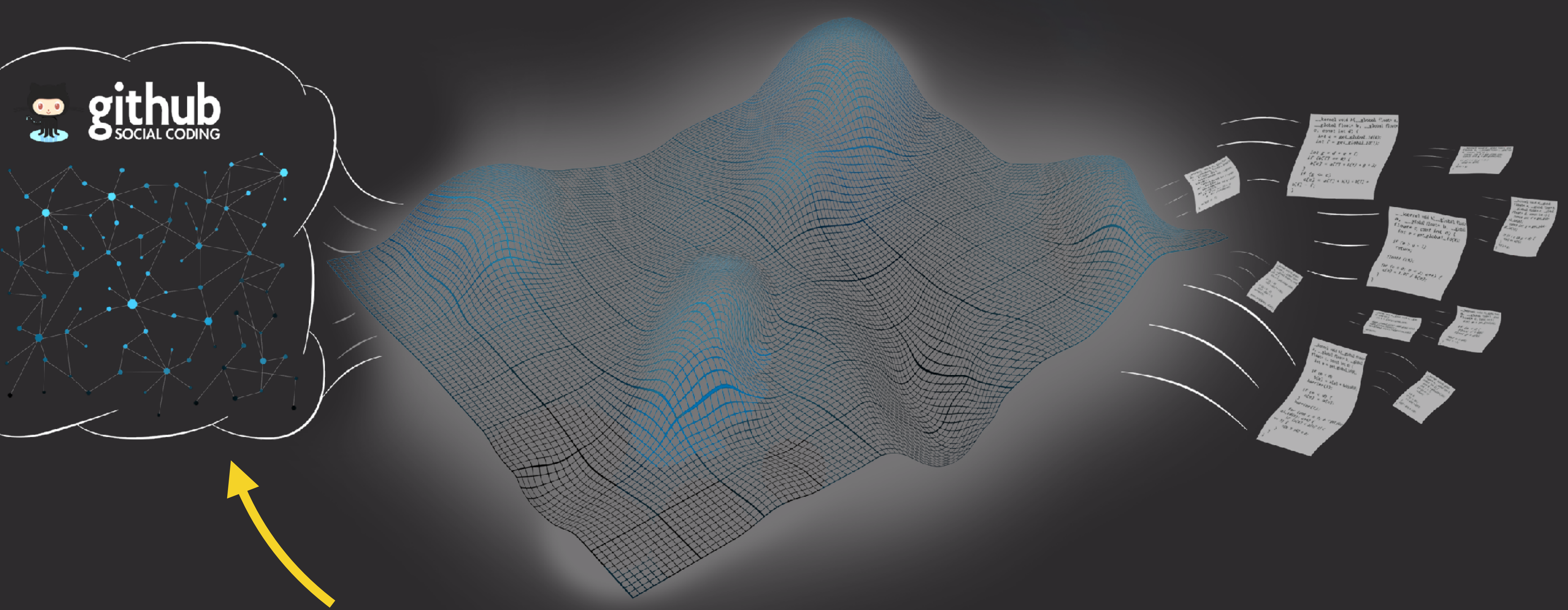
old approach



our approach

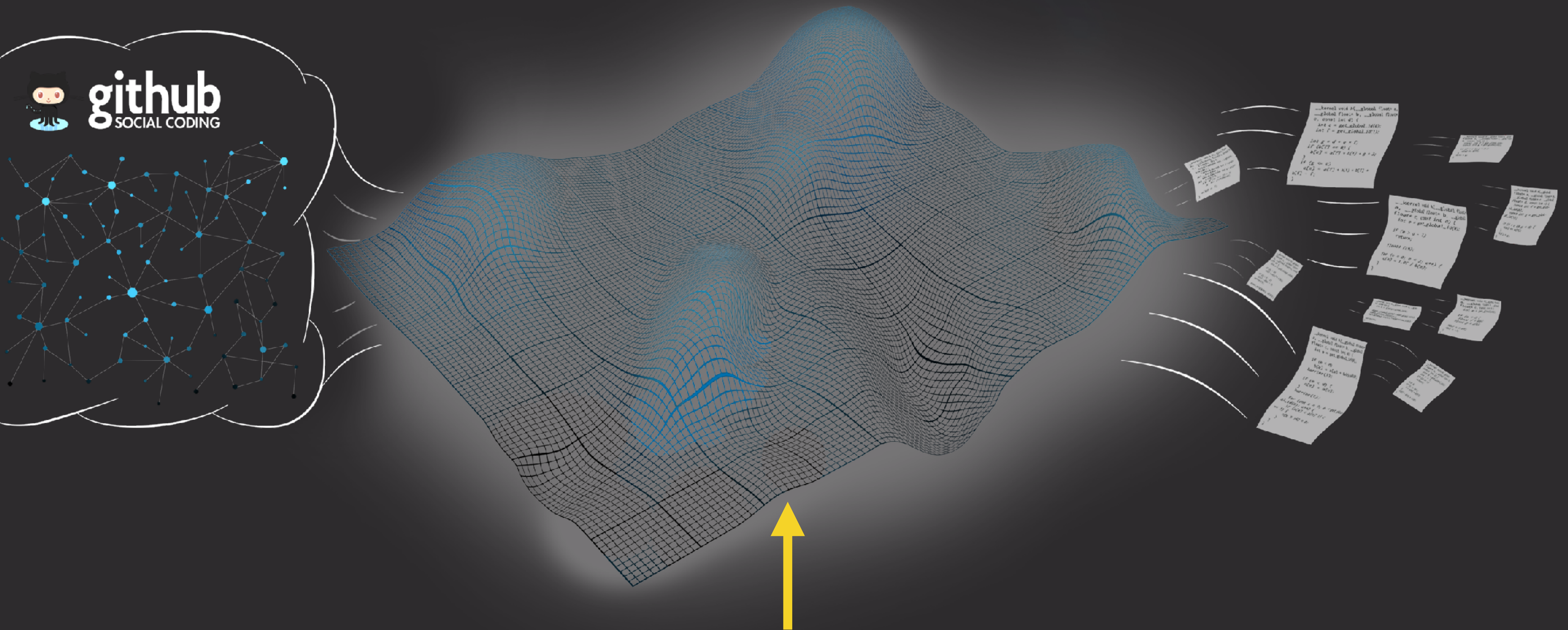


our approach



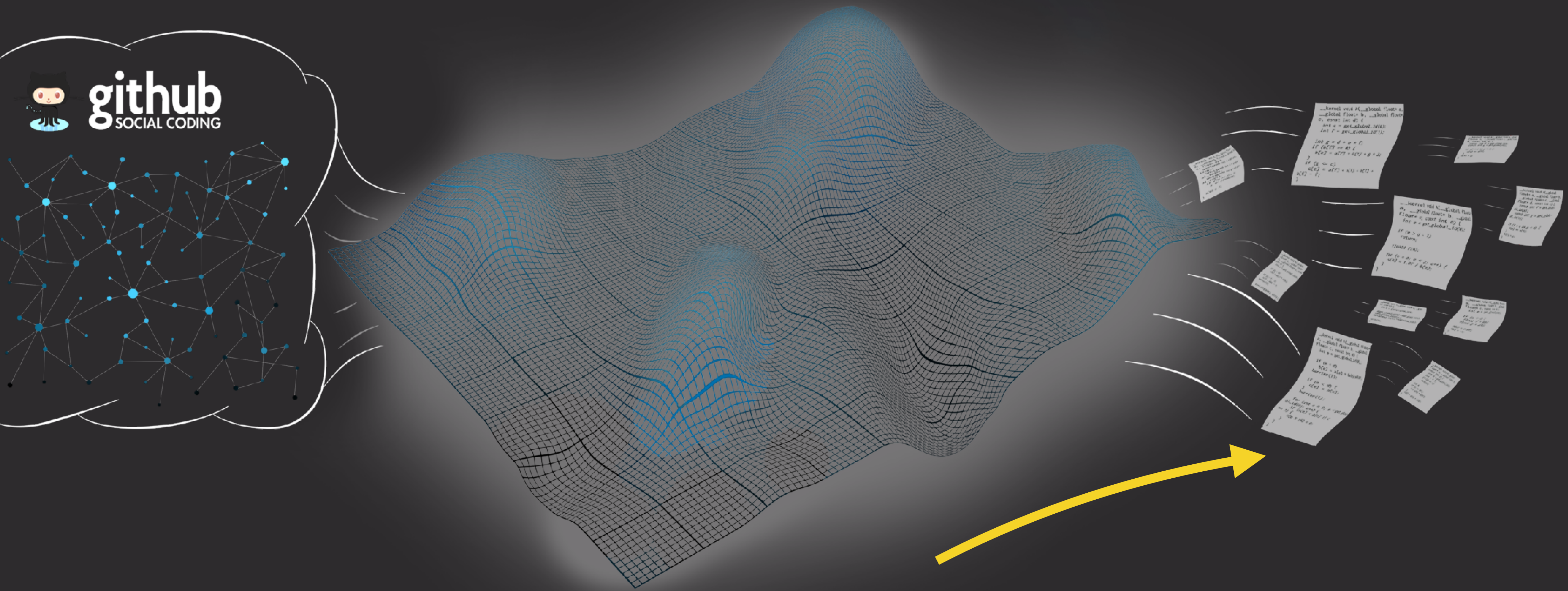
mine code from web

our approach

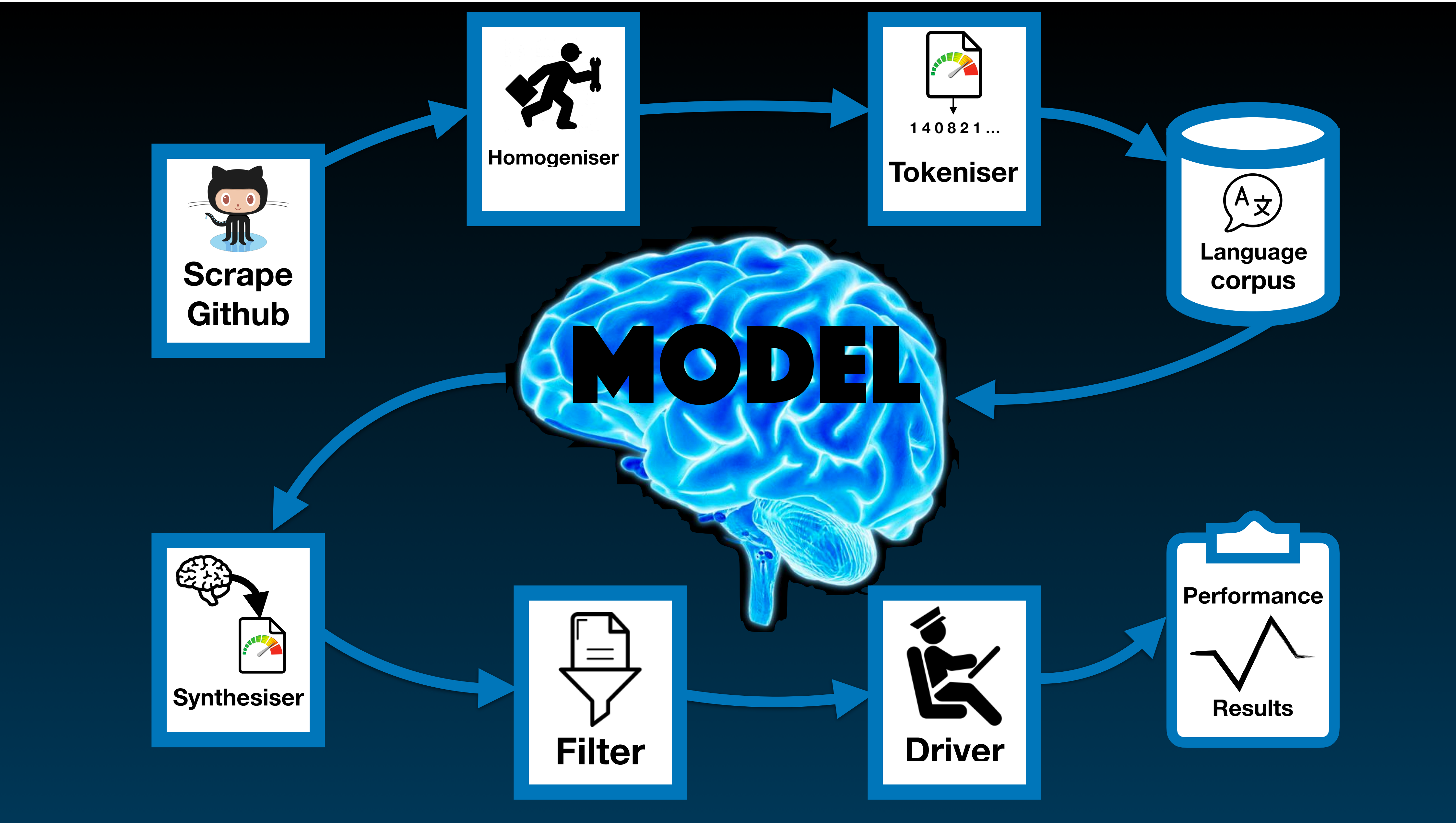


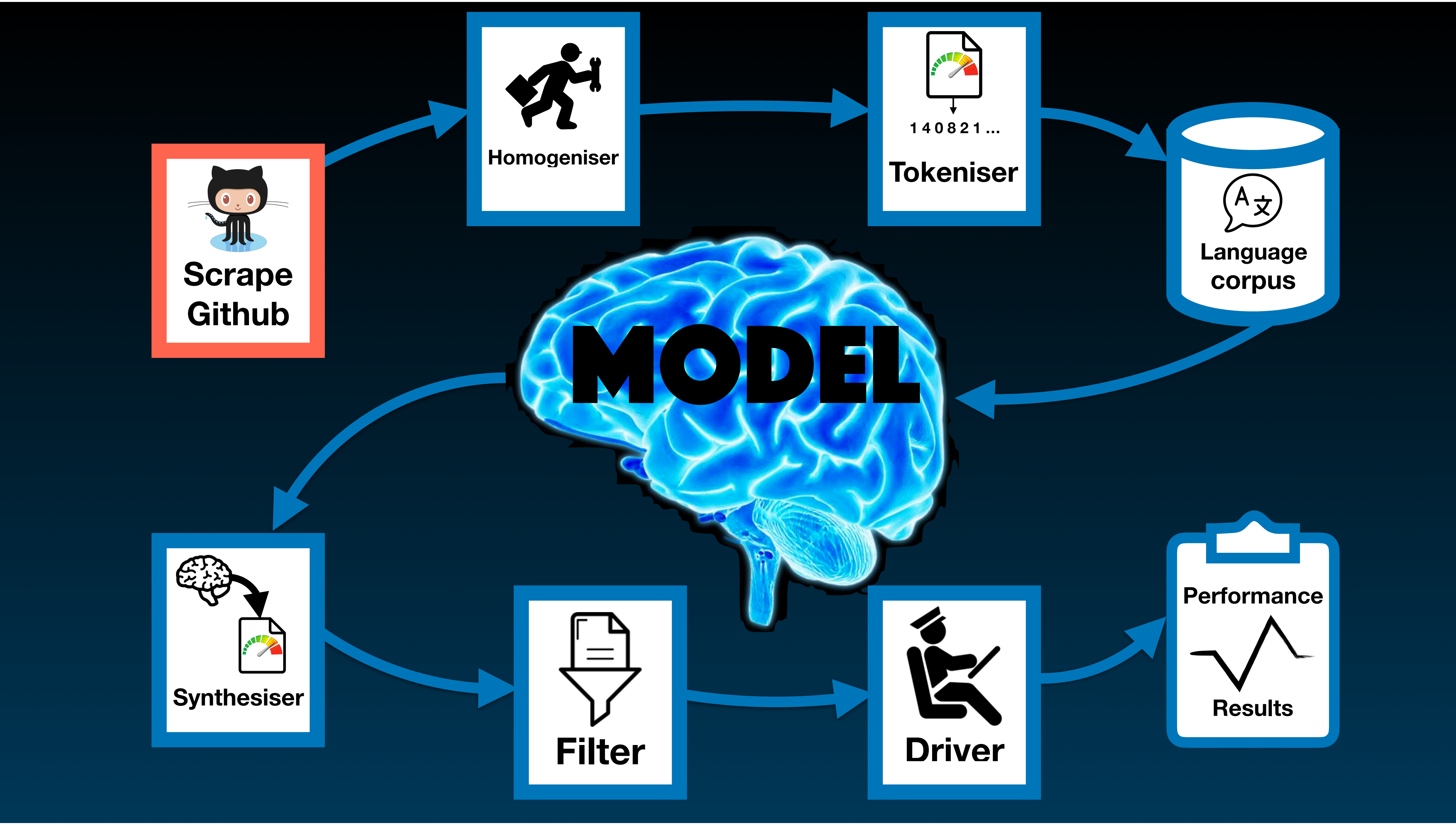
model source distr.

our approach



sample lang. model


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

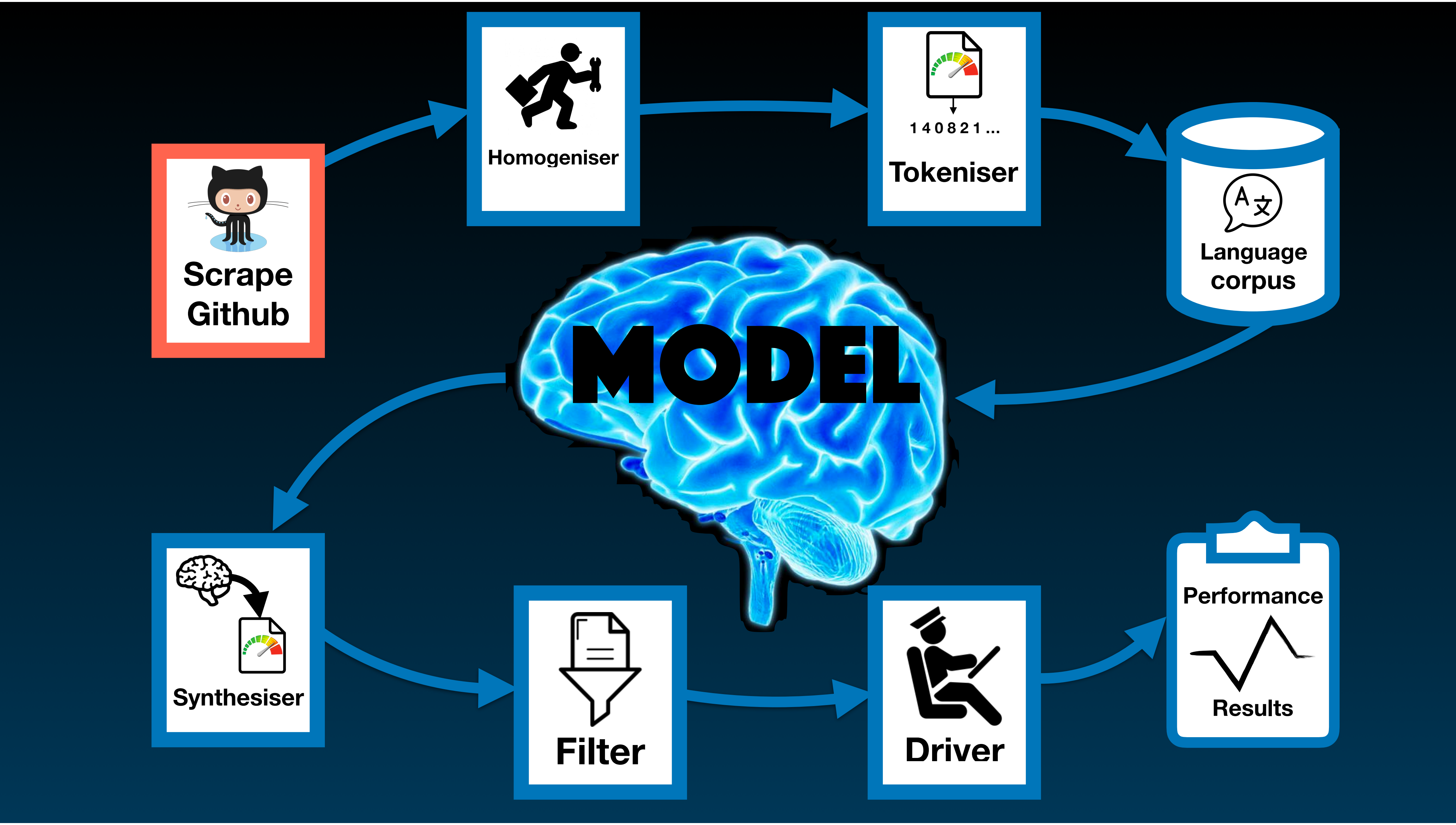
Infer the common usage of a PL from samples.

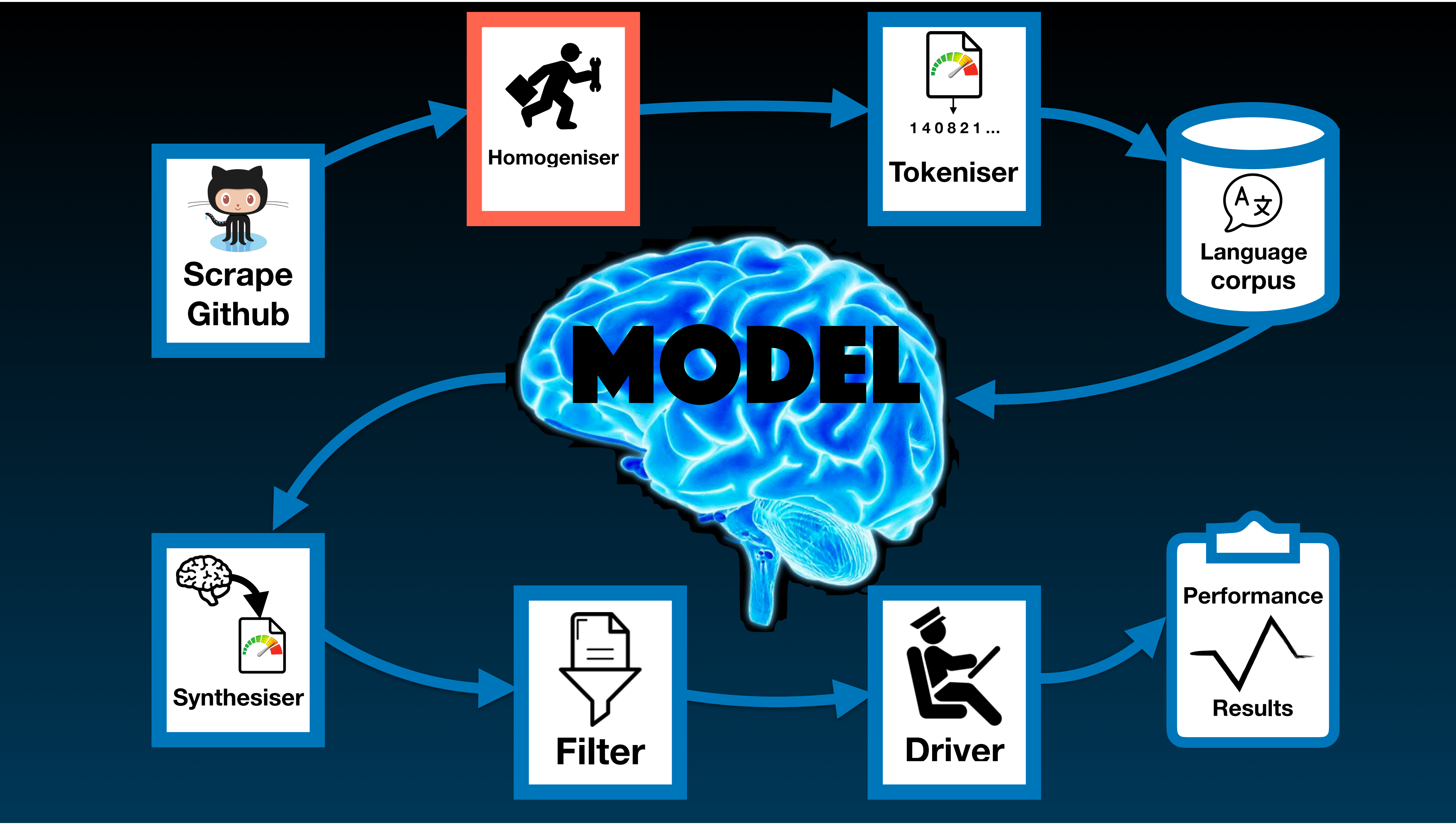
Huge repository of public knowledge:

And they have an API :-)

2.8 million lines of OpenCL

```
$ curl https://api.github.com/search/repositories\?
q\=opencl\&sort\=stars\&order\=desc
{
  "total_count": 3155,
  "incomplete_results": false,
  "items": [
    {
      "id": 7296244,
      "name": "lwjgl3",
      "full_name": "LWJGL/lwjgl3",
```


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

```
/* Copyright (C) 2014, Joe Blogs. */
#define CLAMPING
#define THRESHOLD_MAX 1.0f
float myclamp(float in) {
#ifdef CLAMPING
    return in > THRESHOLD_MAX ? THRESHOLD_MAX : in < 0.0f ? 0.0f : in;
#else
    return in;
#endif // CLAMPING
}
__kernel void findAllNodesMergedAabb(__global float* in, __global float* out,
                                     int num_elems)
{
   // Do something really flipping cool
   int id = get_global_id(0);
    if (id < num_elems)
    {
        out[id] = myclamp(in[id]);
    }
}
```

```
/* Copyright (C) 2014, Joe Blogs. */
#define CLAMPING
#define THRESHOLD_MAX 1.0f
float myclamp(float in) {
#ifdef CLAMPING
    return in > THRESHOLD_MAX ? THRESHOLD_MAX : in < 0.0f ? 0.0f : in;
#else
    return in;
#endif // CLAMPING
}
__kernel void findAllNodesMergedAabb(__global float* in, __global float* out,
                                     int num_elems)
{
   // Do something really flipping cool
   int id = get_global_id(0);
    if (id < num_elems)
    {
        out[id] = myclamp(in[id]);
    }
}
```

Strip comments

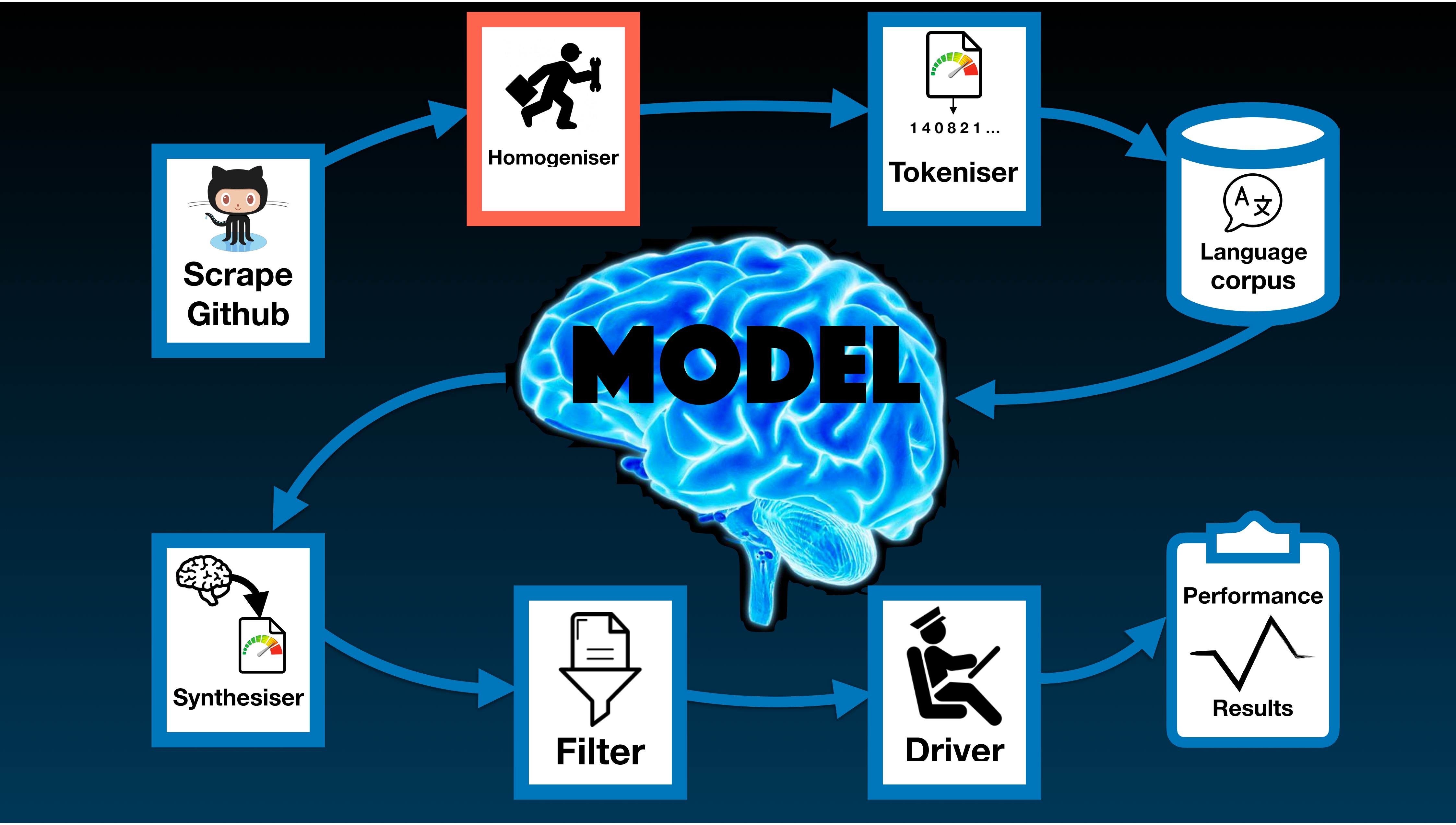
Preprocess

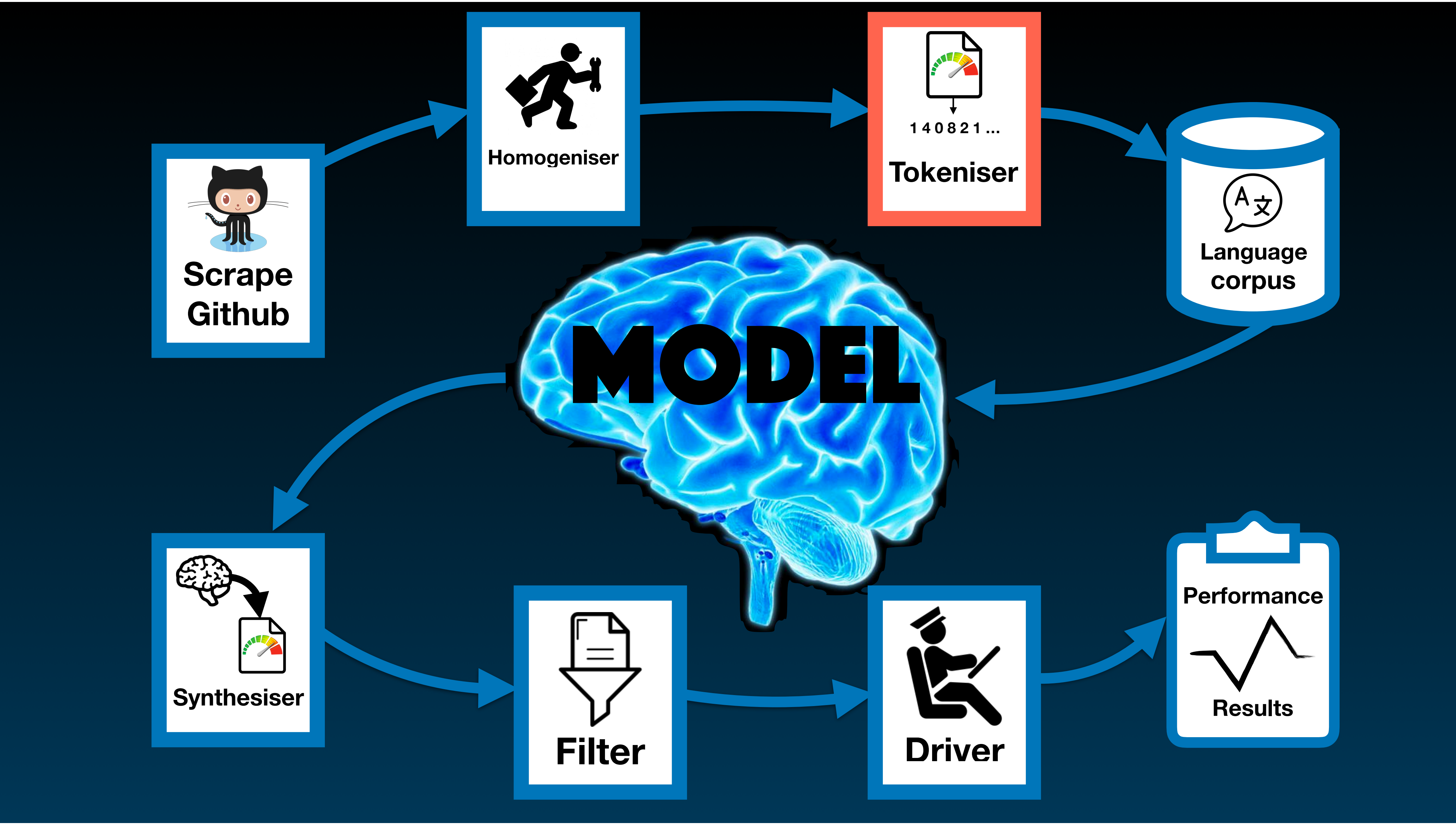
Rewrite names

Enforce code style

Strip comments
Preprocess
Rewrite names
Enforce code style

```
float A(float a) {
    return a > 1.0f ? 1.0f : a < 0.0f ? 0.0f : a;
}

__kernel void B(__global float* b, __global float* c, int d) {
    int e = get_global_id(0);
    if (e < d) {
        c[e] = A(b[e]);
    }
}
```


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Encoded:

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0

Encoded: 0

vocabulary encoding

```
kernel_void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1

Encoded: 0 1

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2

Encoded: 0 1 2

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2

Encoded: 0 1 2 1

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3

Encoded: 0 1 2 1 3

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4

Encoded:

0	1	2	1	3	4

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5

Encoded: 0 1 2 1 3 4 5

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5

Encoded: 0 1 2 1 3 4 5 1

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6

Encoded: 0 1 2 1 3 4 5 1 6

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6
*	7

Encoded: 0 1 2 1 3 4 5 1 6 7

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6
*	7

Encoded: 0 1 2 1 3 4 5 1 6 7 1

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6
*	7
a	8

Encoded: 0 1 2 1 3 4 5 1 6 7 1 8

vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

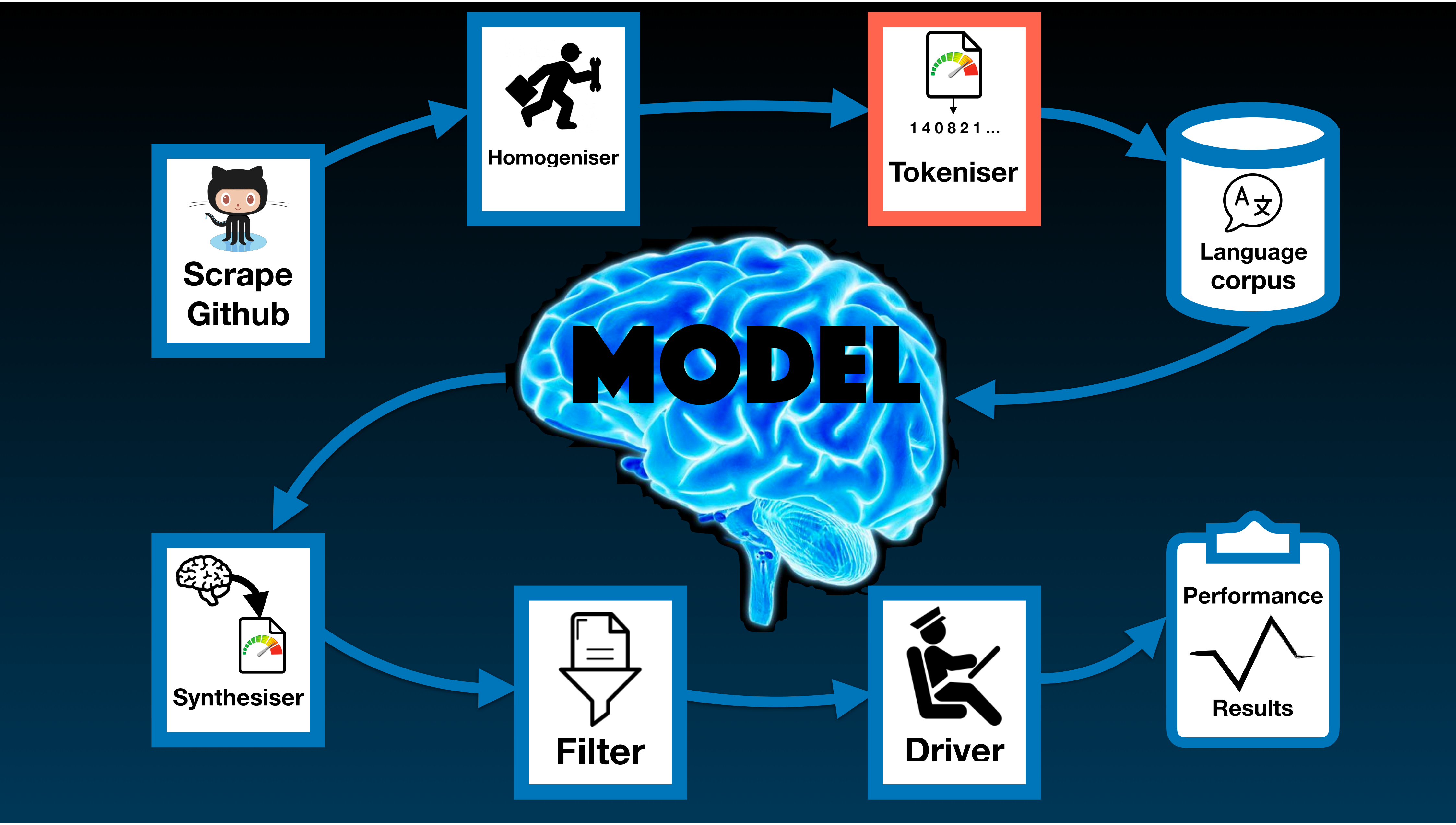
Vocab:

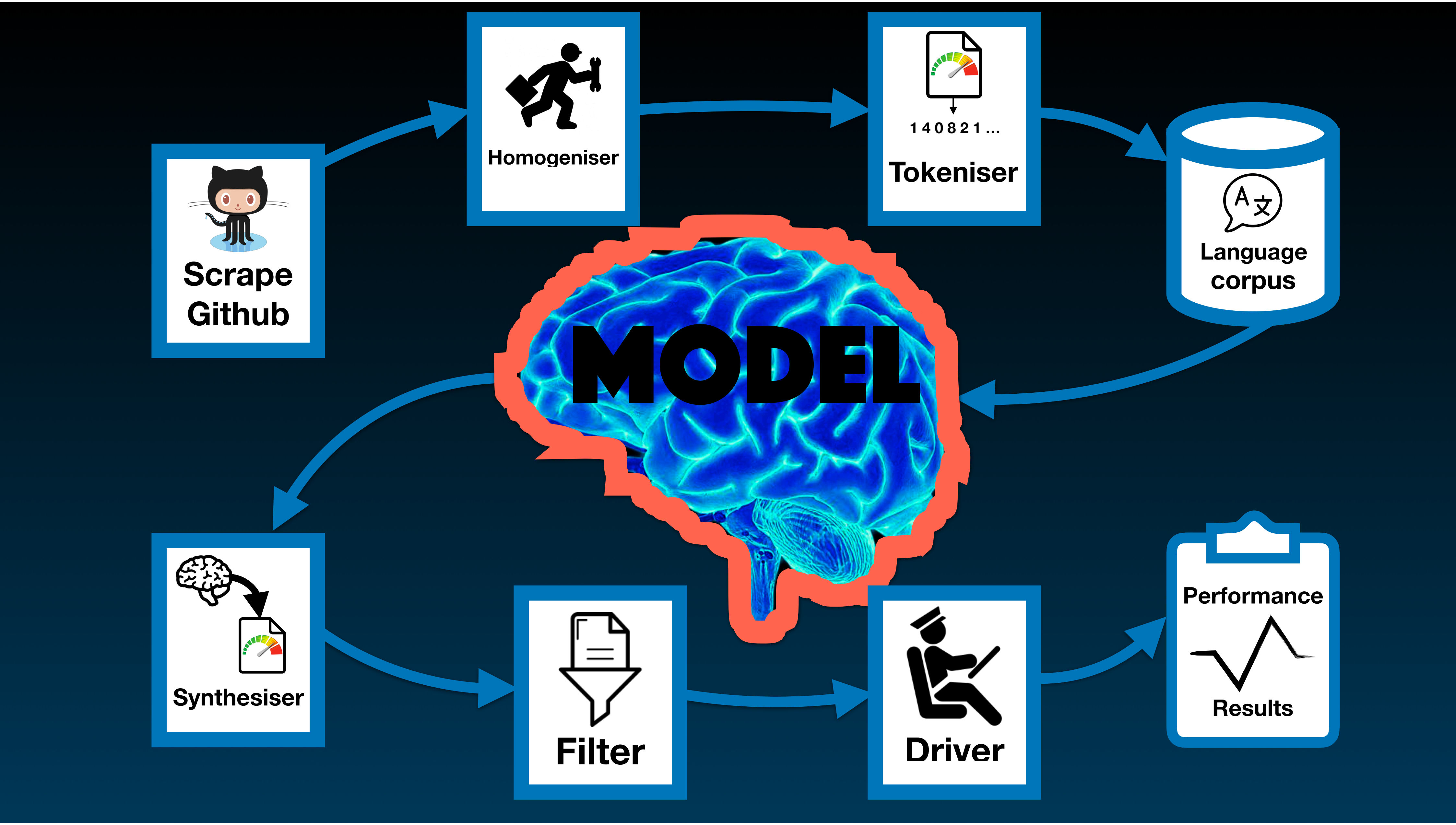
Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6
*	7
a	8

Token	Index
,	9
const	10
b	11
)	12
{	13
\n	14
[	15
get_global_id	16
0	17

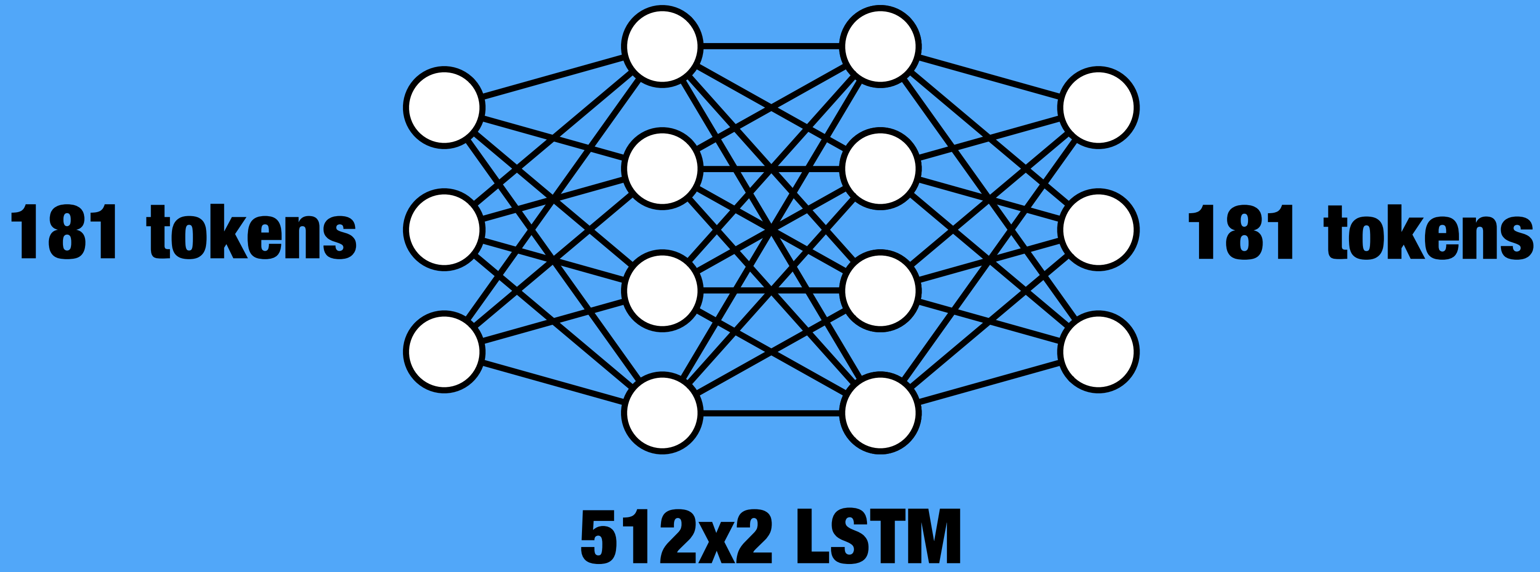
Token	Index
]	18
=	19
3	20
.	21
1	22
4	23
+	24
;	25

Encoded: 0 1 2 1 3 4 5 1 6 7 1 8 …


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

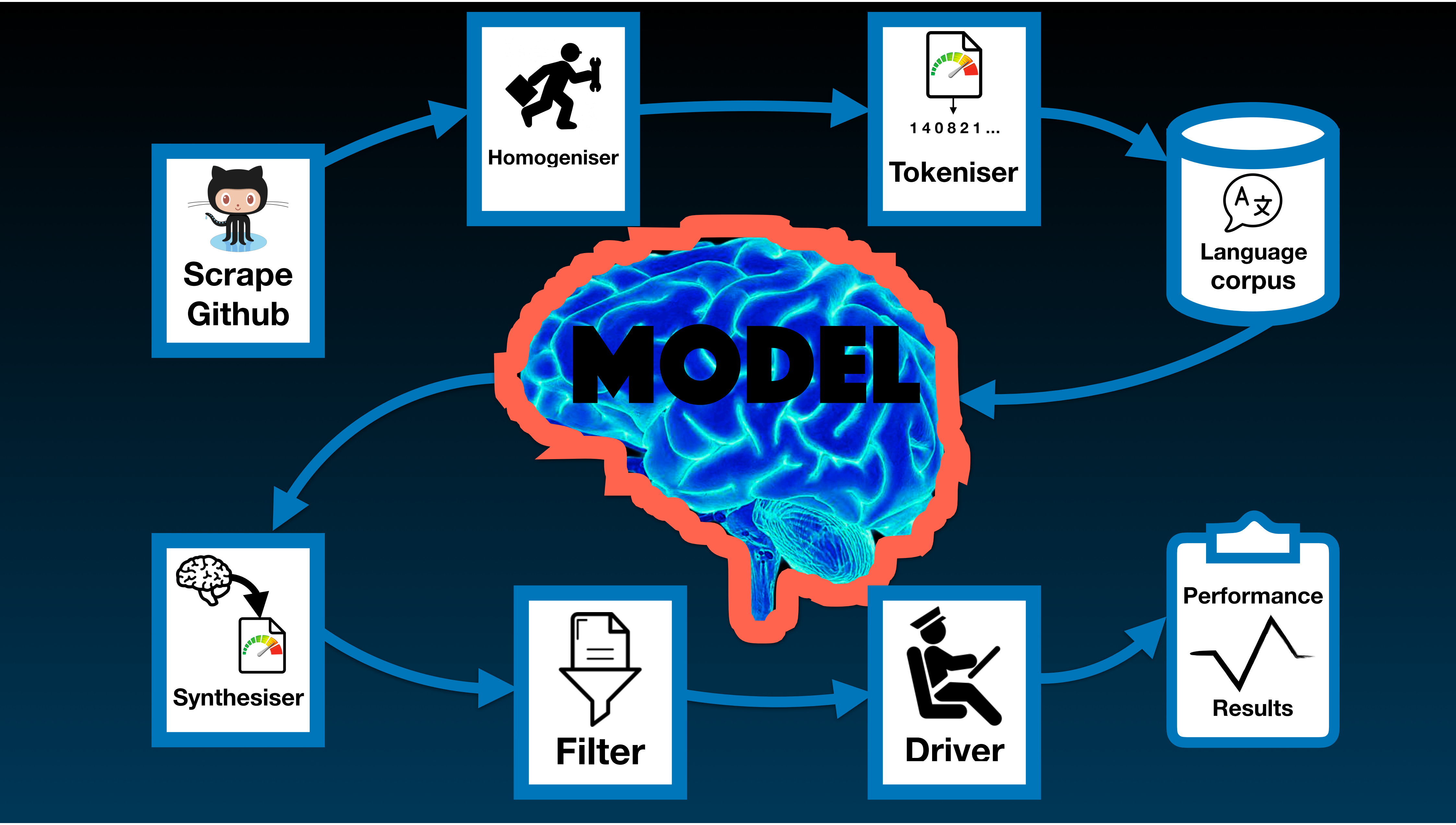
neural network

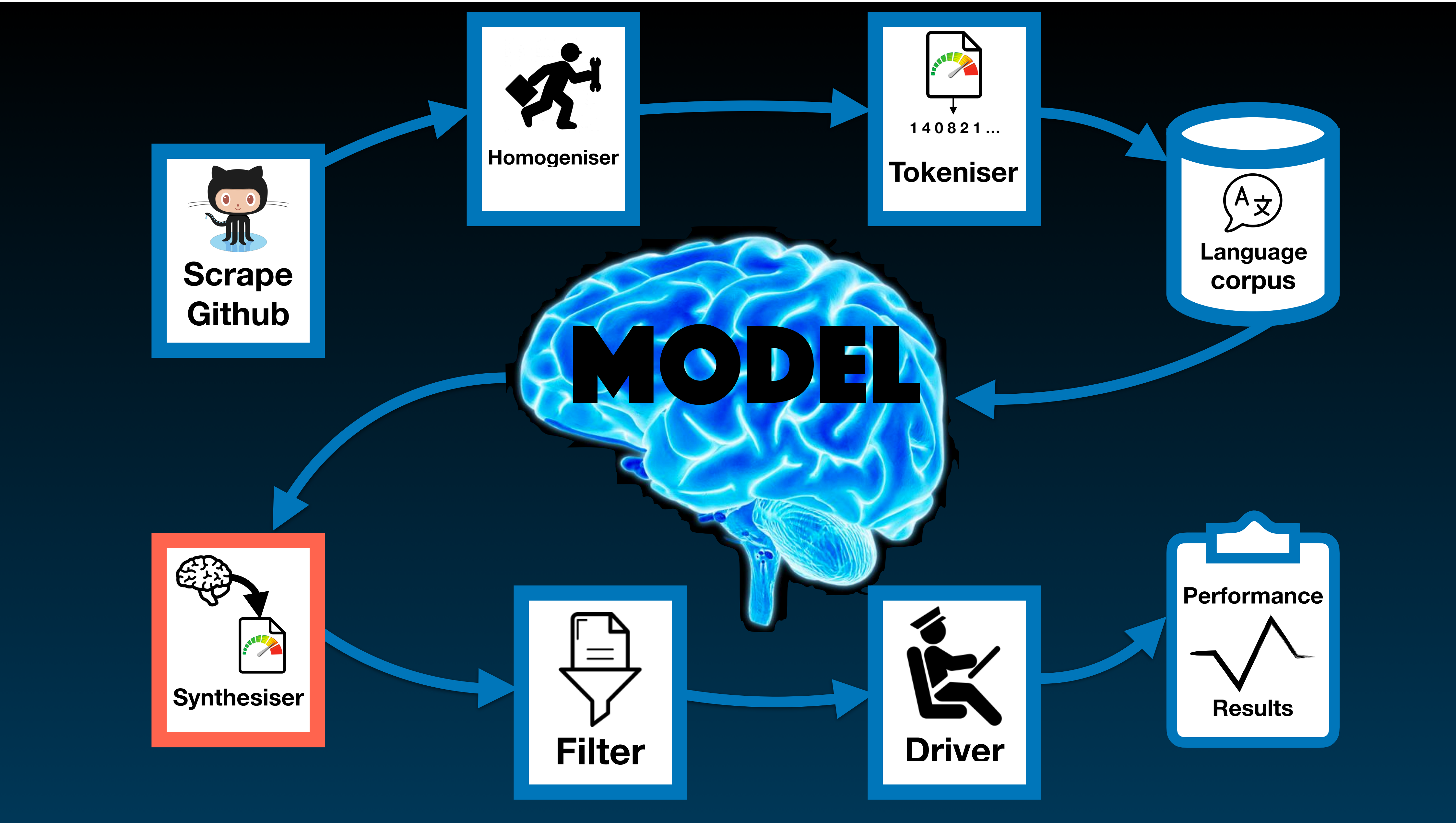


Input: 30M token corpus 0 1 2 ...

Learns probability distribution over corpus.

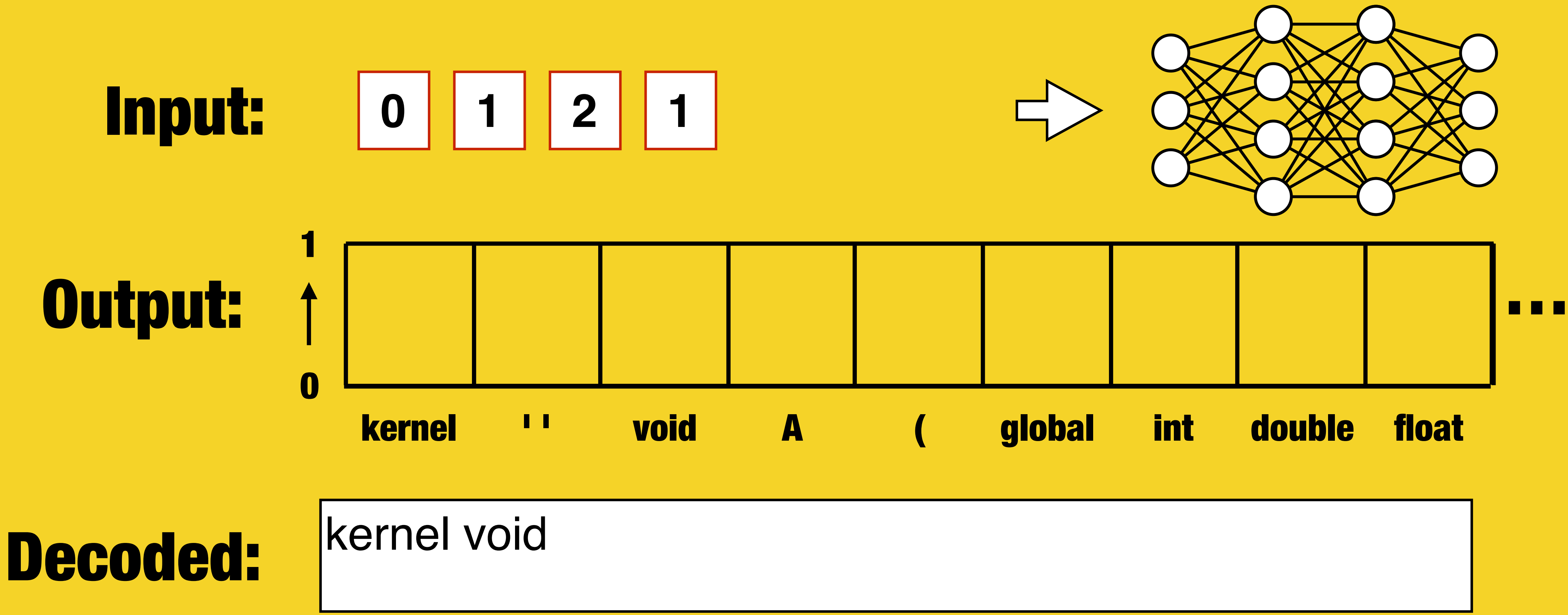
< 500 lines of code, 12 hours training on GPU.


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

synthesizer + harness

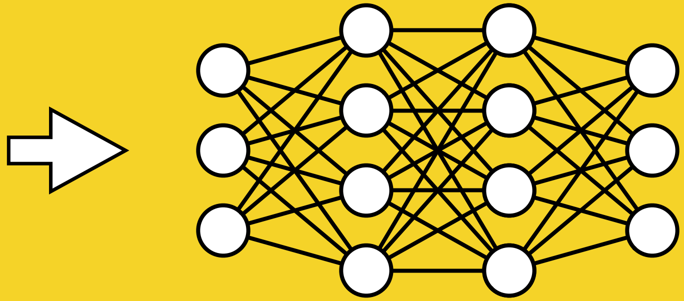
1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.



synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: 0 1 2 1

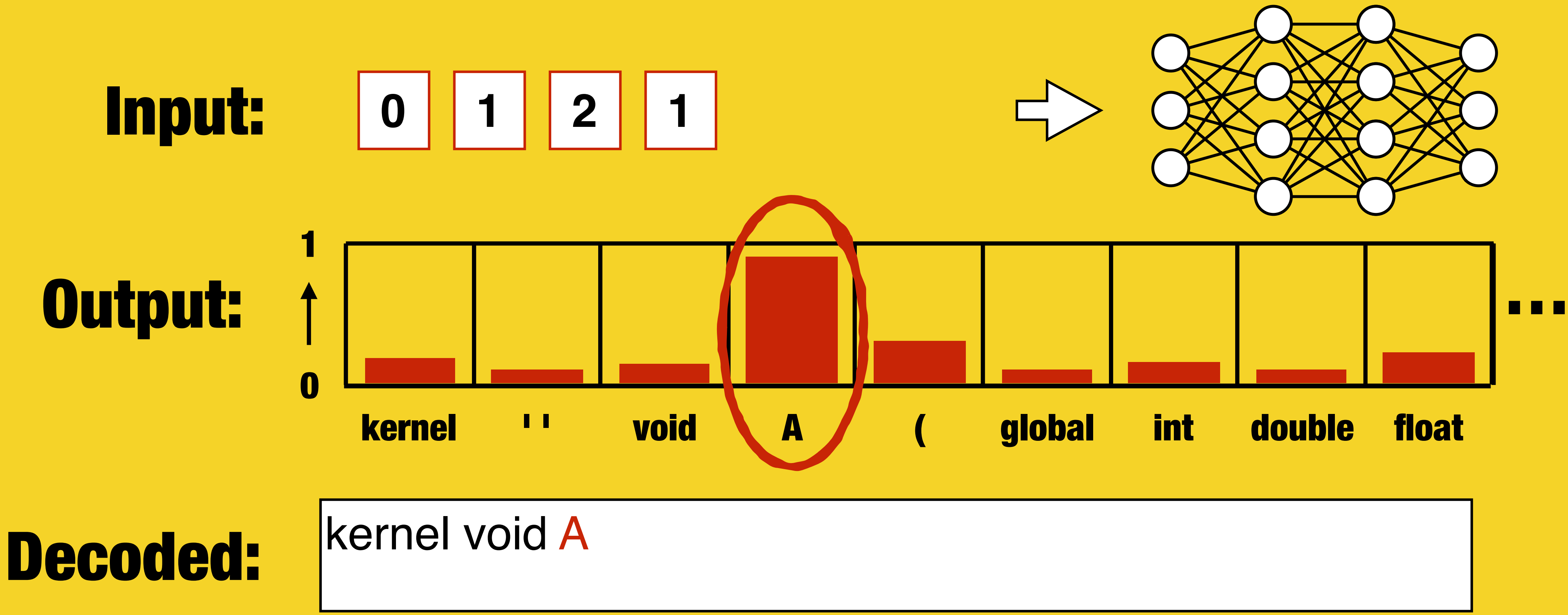


Output:

Decoded: kernel void

synthesizer + harness

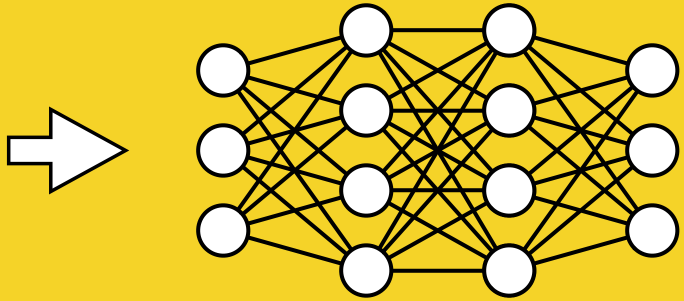
1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.



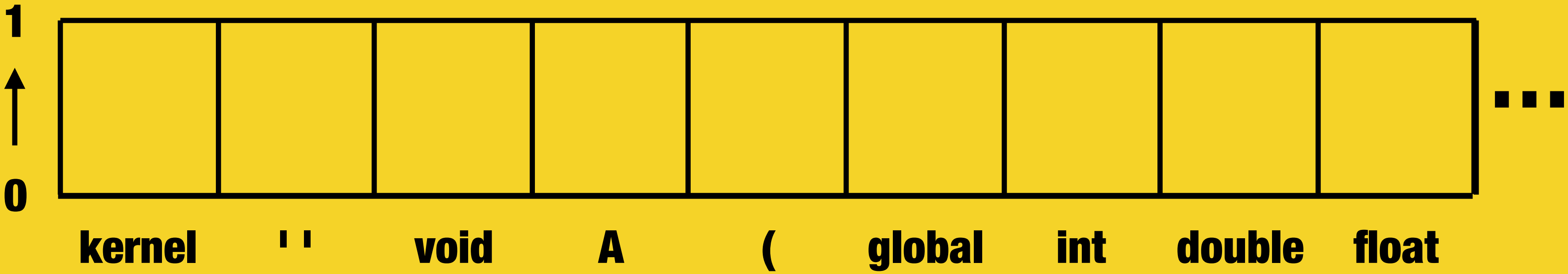
synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: 0 1 2 1 3



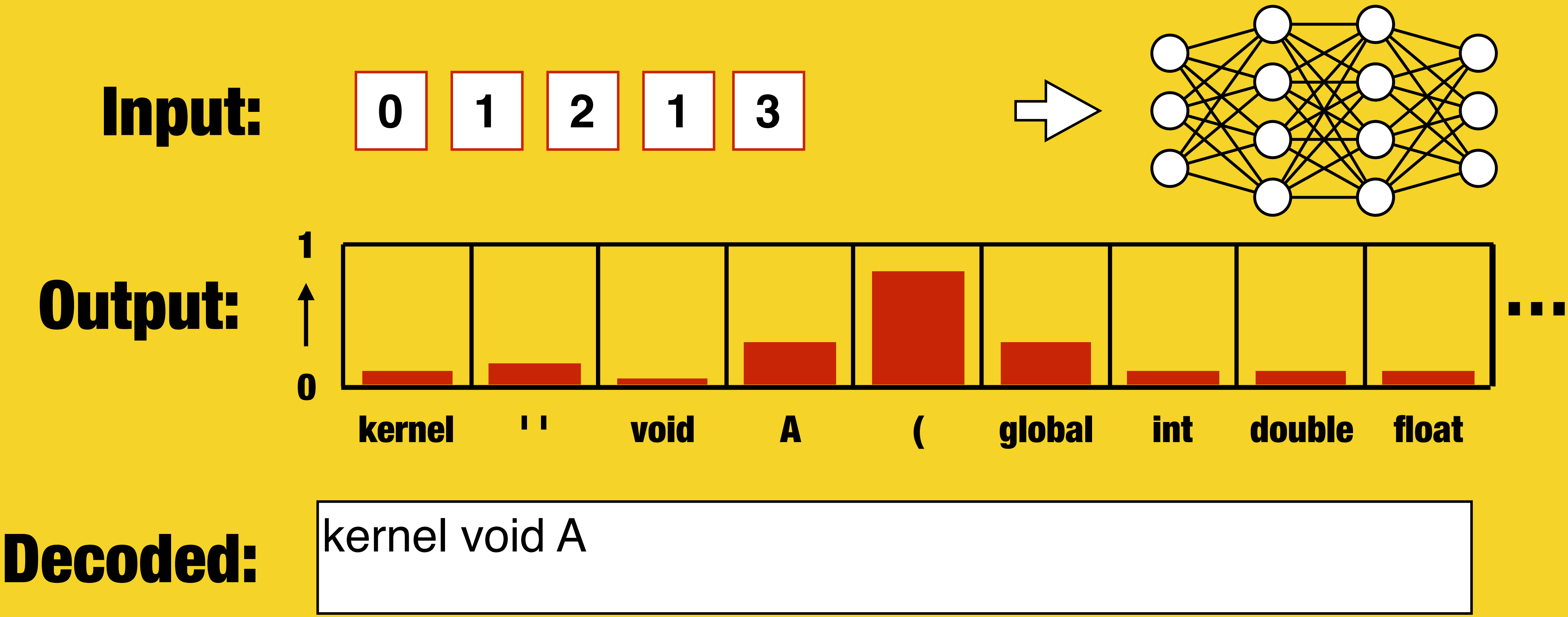
Output:



Decoded: kernel void A

synthesizer + harness

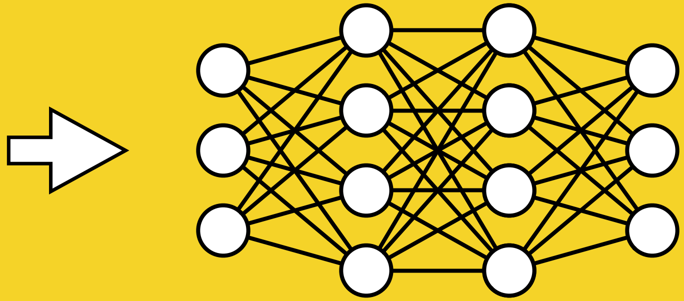
1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.



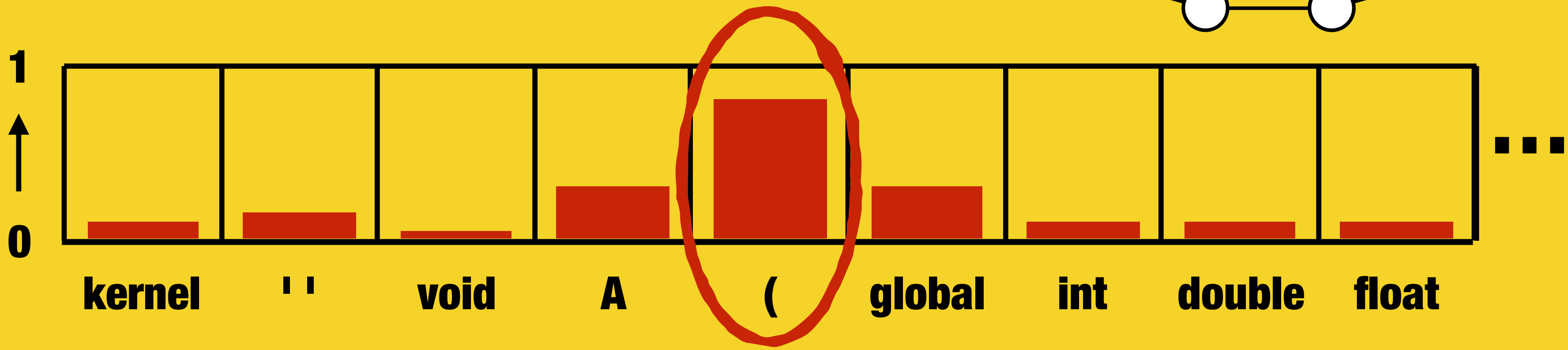
synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: 0 1 2 1 3



Output:



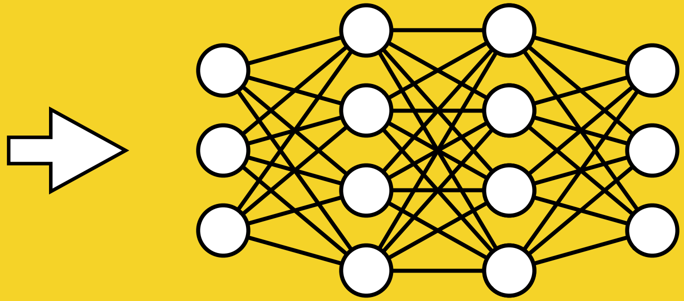
Decoded:

kernel void A(

synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: 0 1 2 1 3 4



Output:

1

0

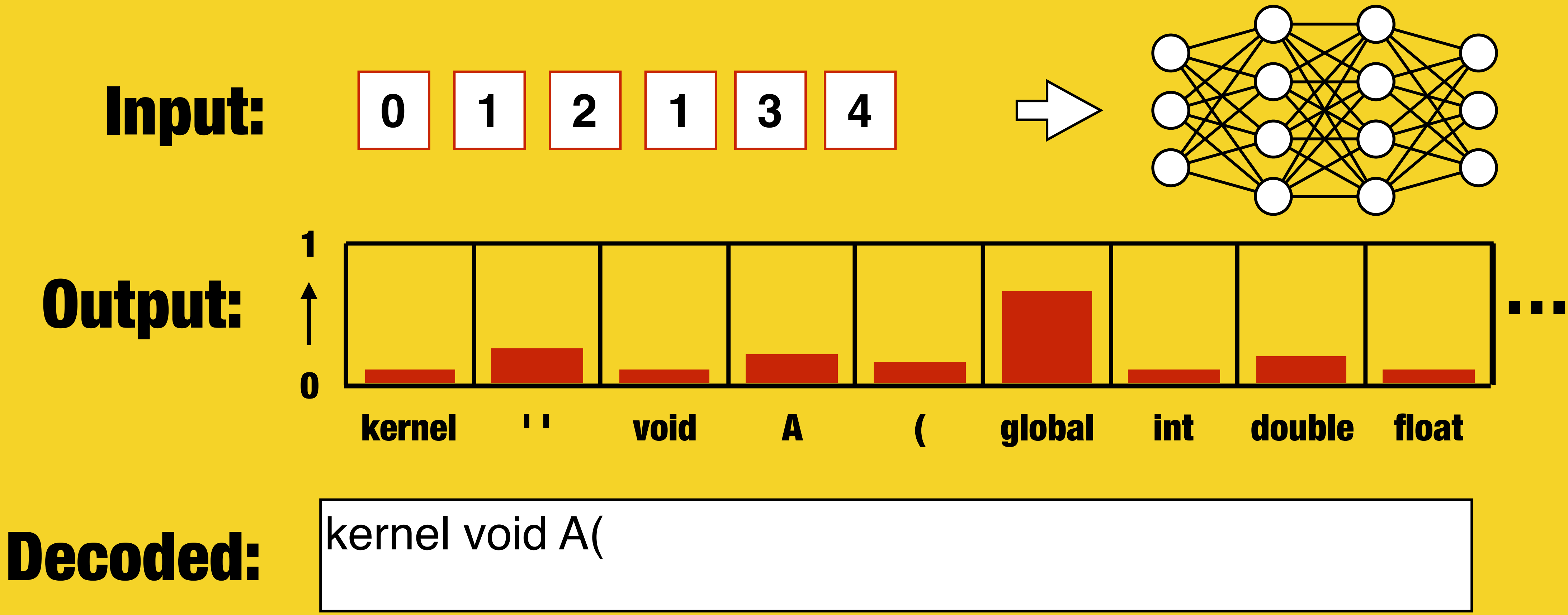
kernel ' ' void A (global int double float ...

Decoded:

kernel void A(

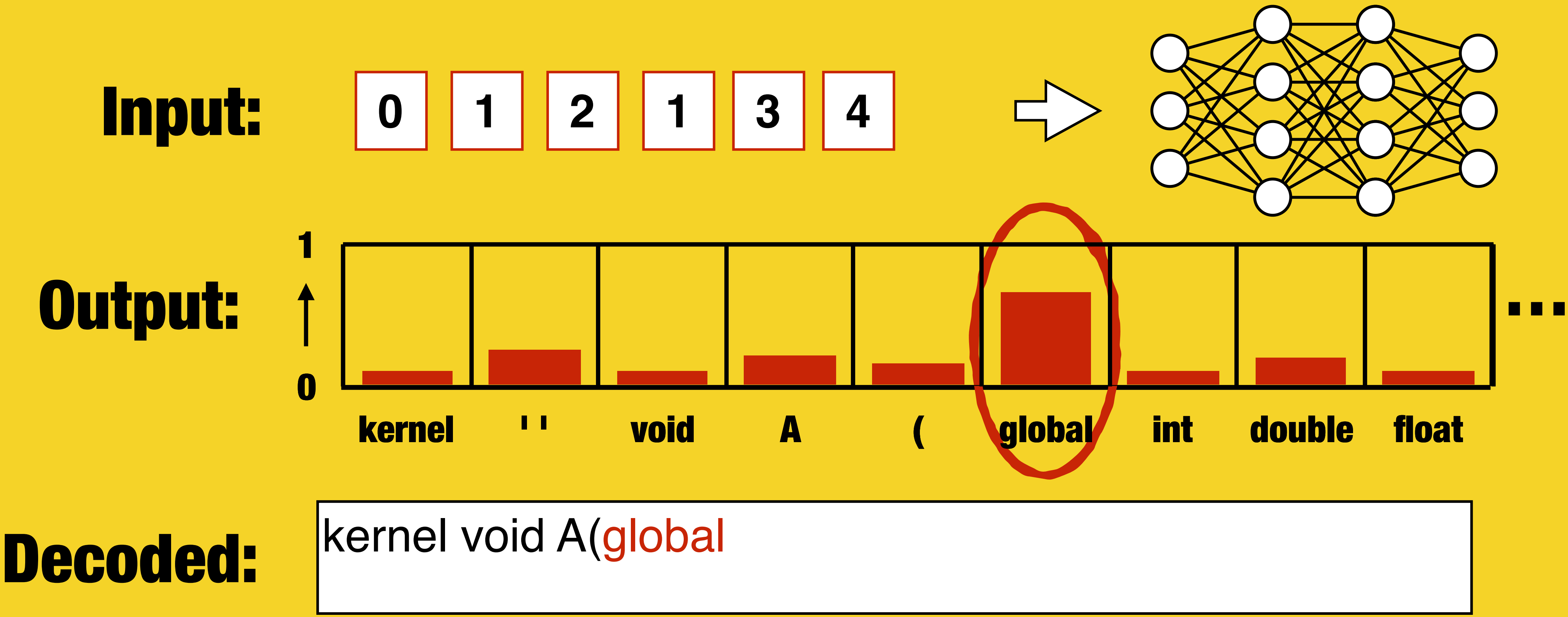
synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.



synthesizer + harness

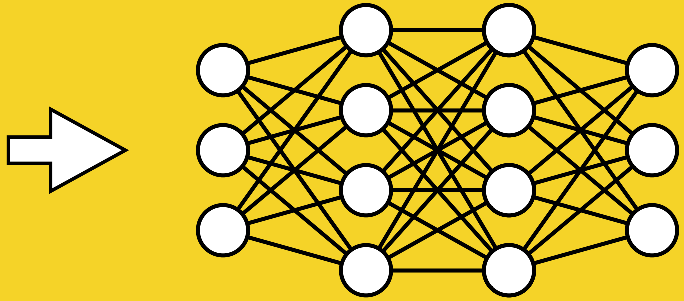
1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.



synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: ... 2 1 3 4 5



Output: 1 ↑ 0

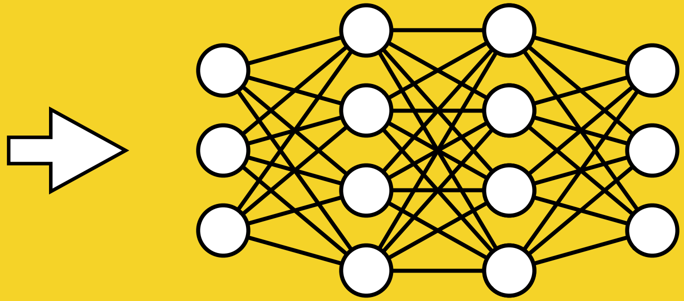
kernel ' ' void A (global int double float ...

Decoded: kernel void A(global

synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.

Input: … 2 1 3 4 5



Output:

Decoded:

kernel void A(global

synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.
3. Can we parse signature?

    Yes: Generate input data, compile and run it.

    No: Compile it but don't run it.

Decoded:

```
kernel void A(global int* a) {
```

synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.
3. Can we parse signature?

    Yes: Generate input data, compile and run it.

    No: Compile it but don't run it.

Decoded:

```
kernel void A(global int* a) {
```

Examples

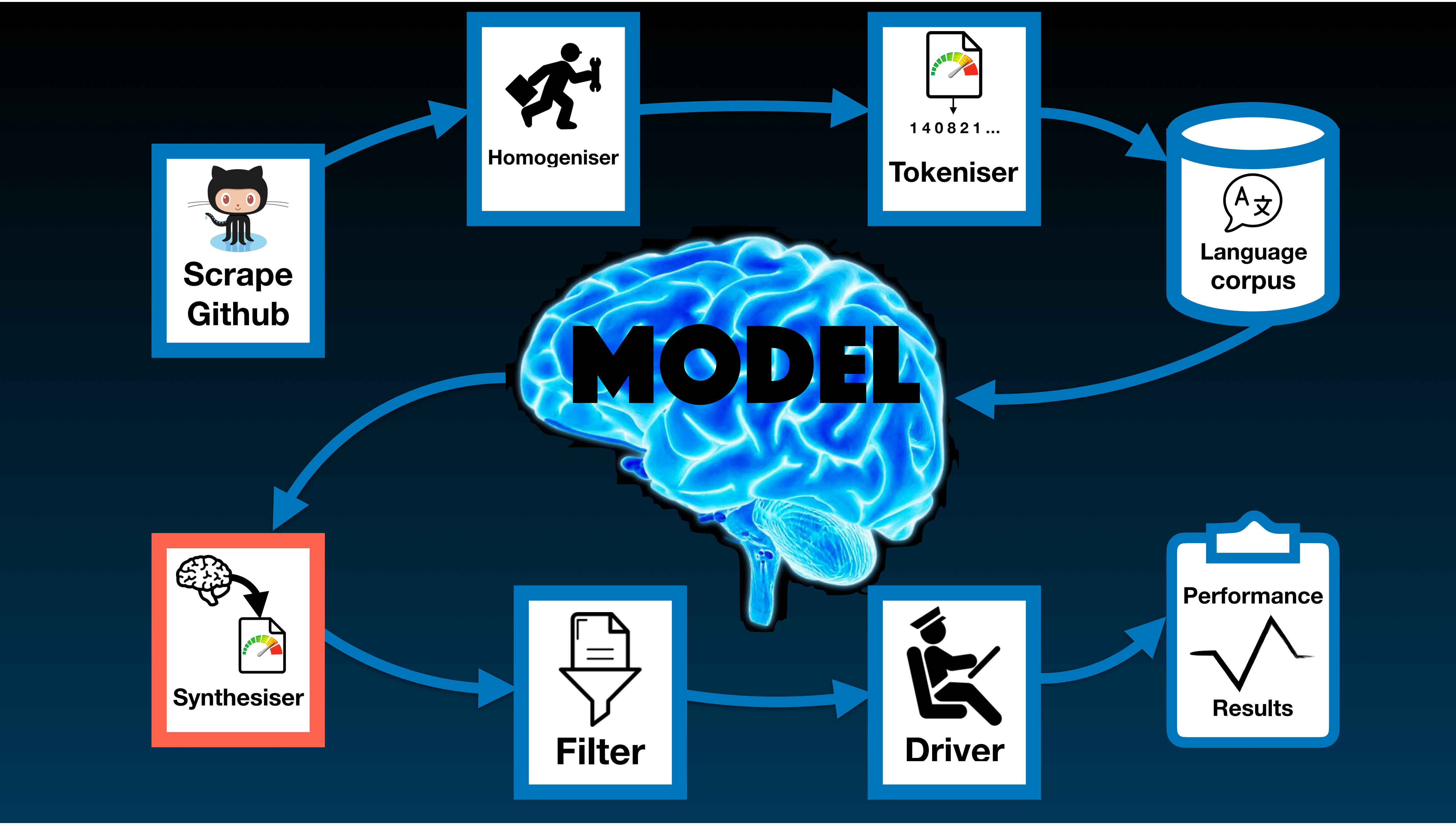
```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const int d) {
  int e = get_global_id(0);
  float f = 0.0;
  for (int g = 0; g < d; g++) {
    c[g] = 0.0f;
  }
  barrier(1);

  a[get_global_id(0)] = 2*b[get_global_id(0)];
}
```

Examples

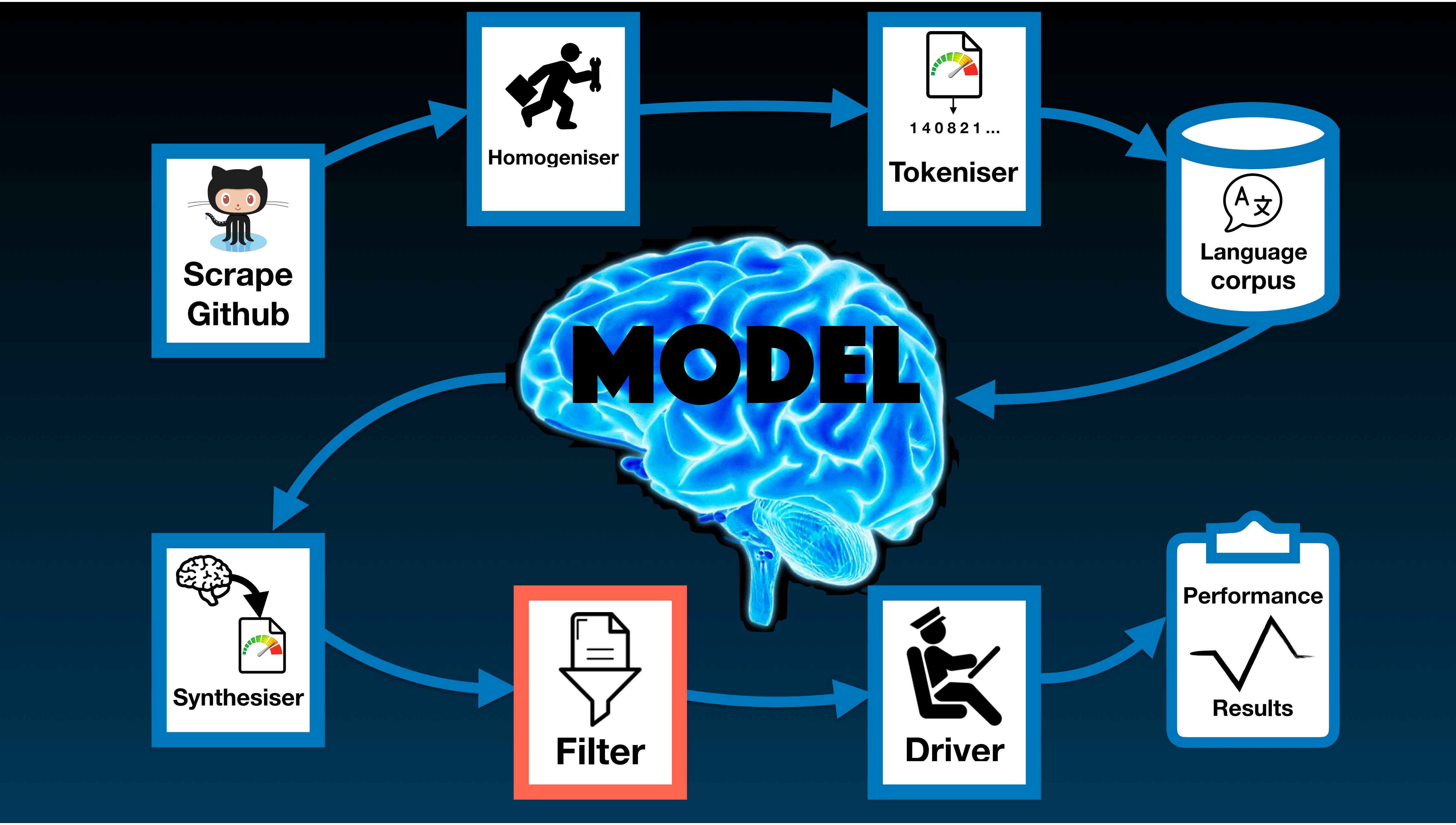
```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const int d) {
  int e = get_global_id(0);
  if (e >= d) {
    return;
  }
  c[e] = a[e] + b[e] + 2 * a[e] + b[e] + 4;
}
```

Examples

```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const int d) {
  unsigned int e = get_global_id(0);
  float16 f = (float16)(0.0);
  for (unsigned int g = 0; g < d; g++) {
    float16 h = a[g];
    f.s0 += h.s0;
    f.s1 += h.s1;
    /* snip ... */
    f.sE += h.sE;
    f.sF += h.sF;
  }
  b[e] = f.s0 + f.s1 + f.s2 + f.s3 + f.s4 +
     f.s5 + f.s6 + f.s7 + f.s8 + f.s9 + f.sA +
     f.sB + f.sC + f.sD + f.sE + f.sF;
}
```



Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results


Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

Does it compile?

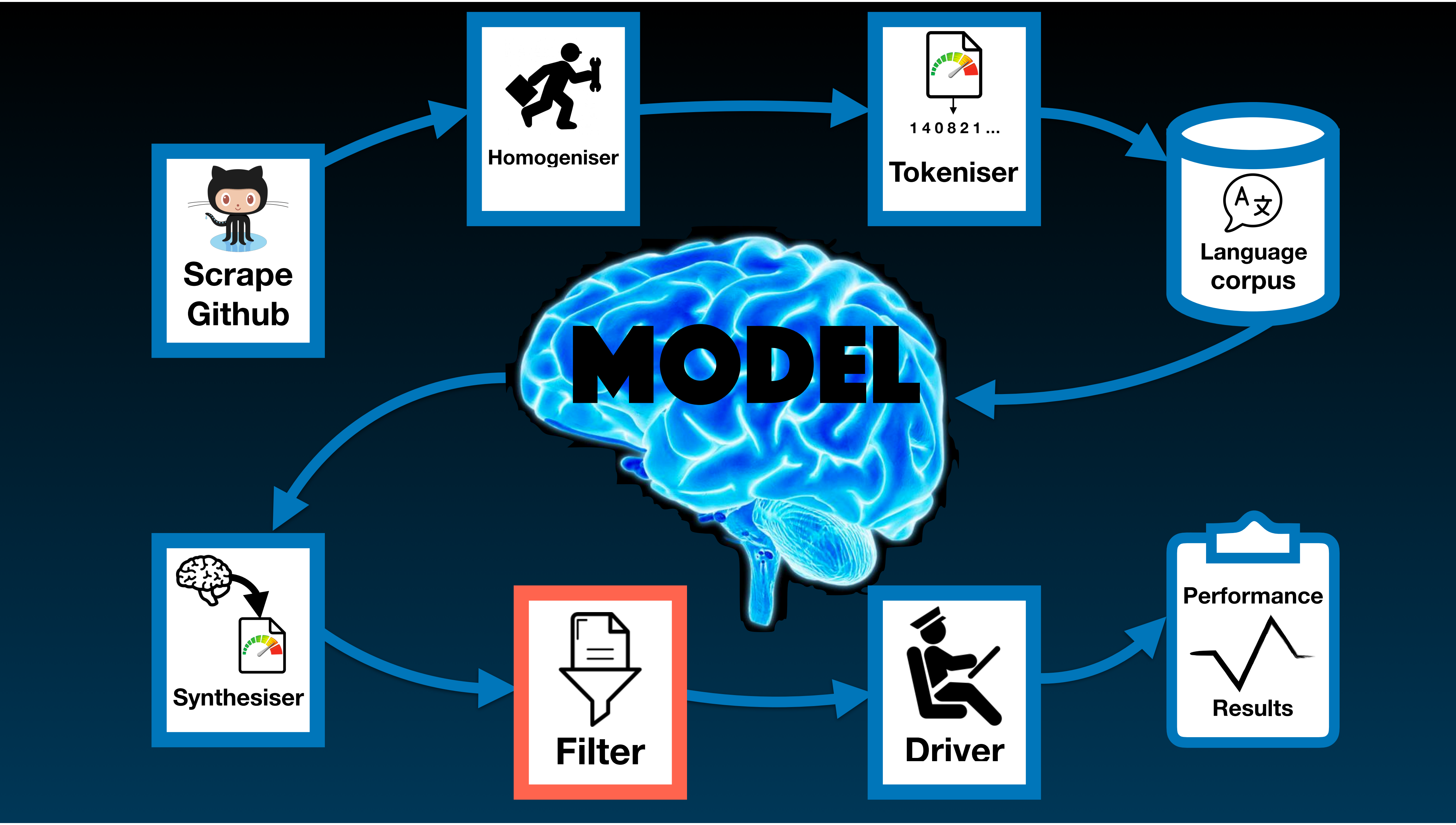
70% fail

Does it do anything?

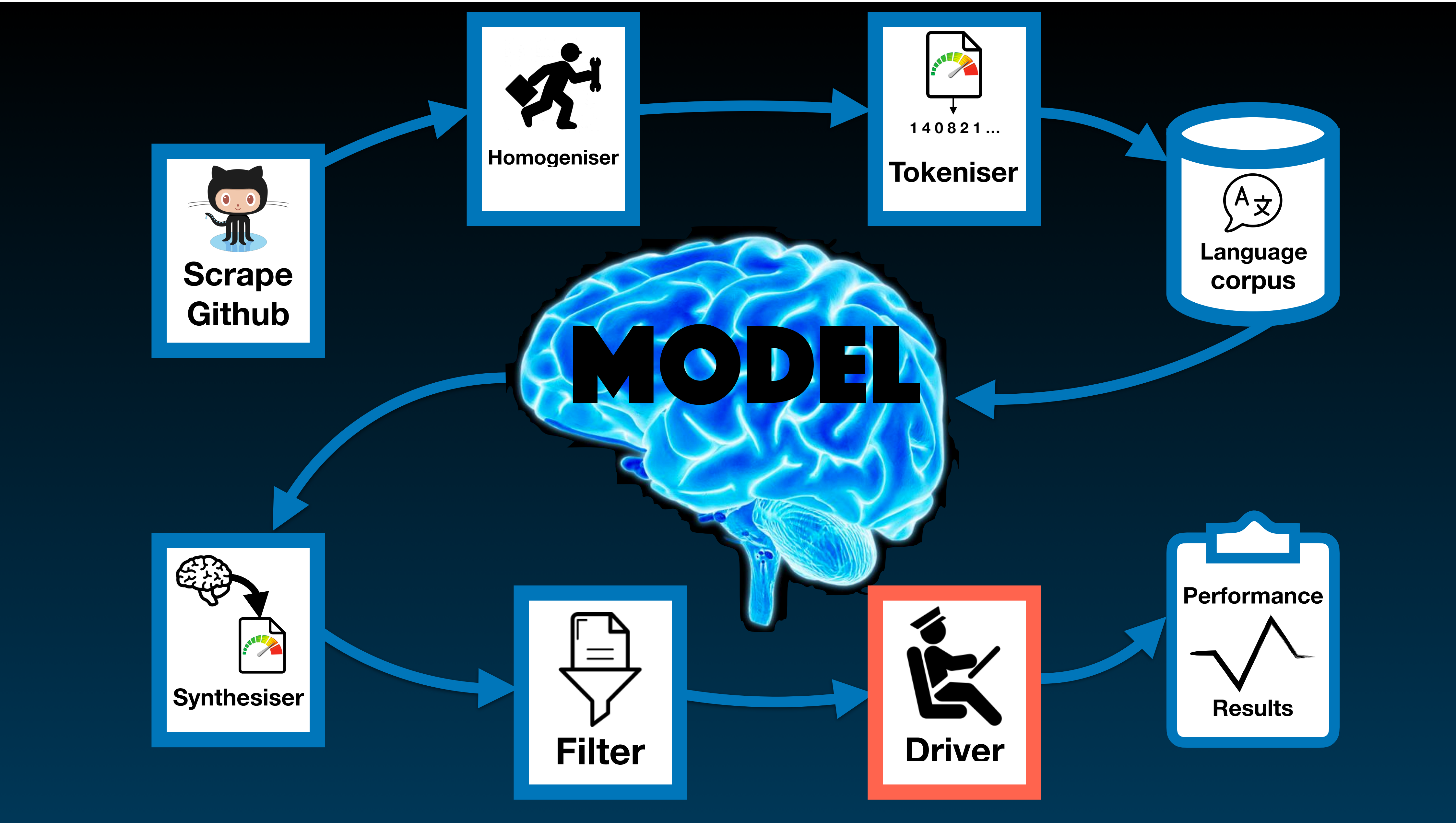
Dynamic checks

- ***has output***
- ***input dependent***
- ***deterministic***

Yield 20-25%



Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

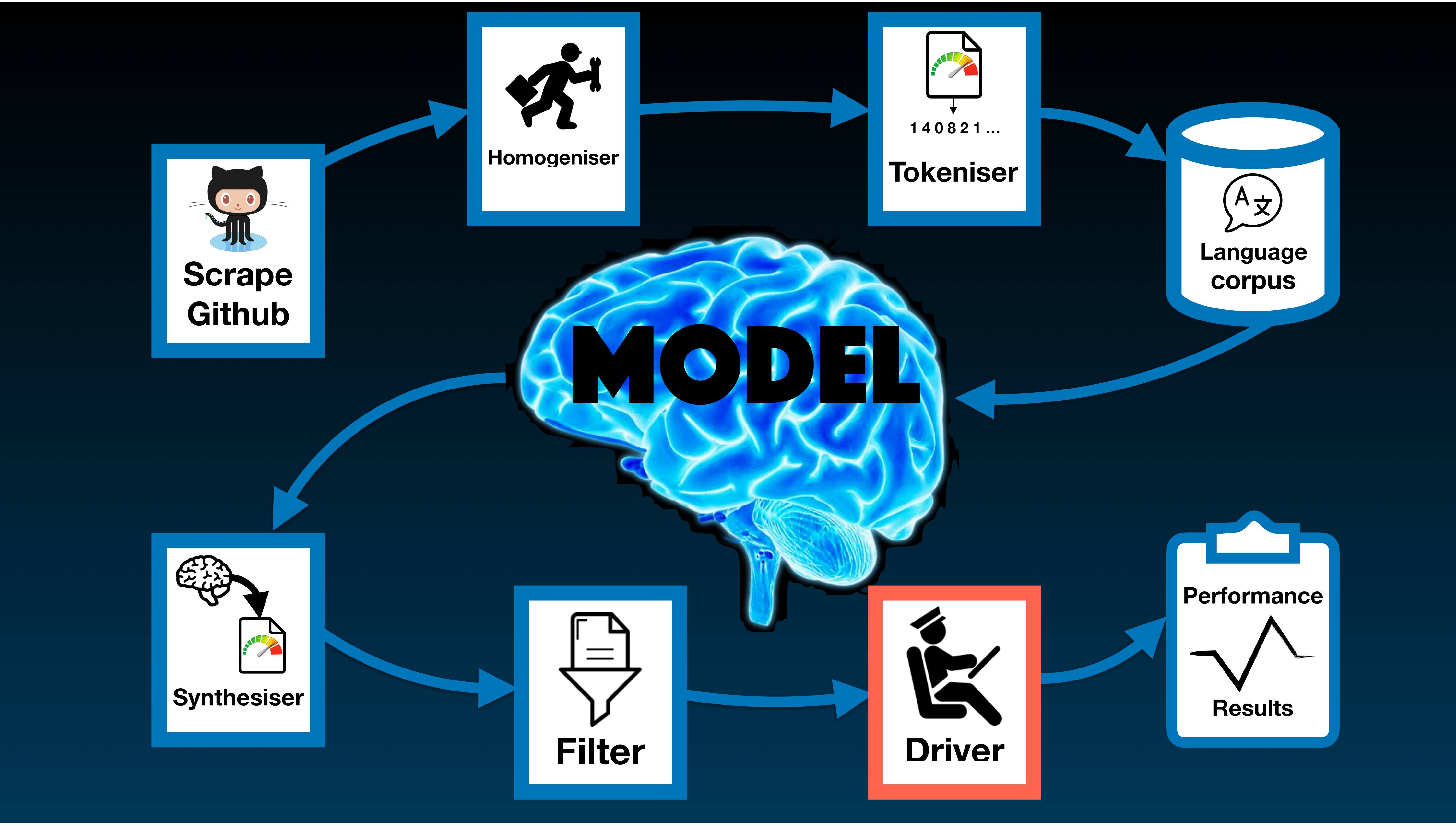

Scrape
Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language
corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const float d,
                const int e) {
  int f = get_global_id(0);
  if (f >= e) {
    return;
  }
  c[f] = a[f] + b[f] + 2 * c[f] + d + 4;
}
```

```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const float d,
                const int e) {
  int f = get_global_id(0);
  if (f >= e) {
    return;
  }
  c[f] = a[f] + b[f] + 2 * c[f] + d + 4;
}
```

Payload for size S:

[rand()] * S	[rand()] * S	[rand()] * S	rand()	S



Scrape Github
Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Language corpus
MODEL
Synthesiser
Filter
Driver
Performance
Results

How well does it work?

52%
blind test

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[illegible] 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) [illegible] (0.5[illegible]f [illegible] g * 1.0f / 18.0f + e / 2.0f);
  }

  for (c = 0; c < 30; c++) {
    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;
```

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;
      [illegible]onst f[illegible]t u = d[3 * r + 2] - q;
      [illegible]onst [illegible]at v = (d[3 * r] - o) * (d[3 * r + 1] - [illegible]) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r + 2] - q) + g;
      co[illegible] float w = e[r] / (((d[3 * r] - o) * (d[3 * r] - o) + (d[3 * [illegible]1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[illegible]r + 2] - q) + g) * sqrt((d[3 * r] - o) * (d[3 * r] - o[illegible]) [illegible]d[3 * r + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2][illegible] - [illegible] * (d[3 * r + 2] - q) + g));
      l = l [illegible] * r] - o) * w;
      m = m + (d[3 * r + 1] - p) * w;
      n = n + (d[3 * r + 2] - q) * w;
    }
    [illegible] + h)] = l;
    [illegible] + h) + 1] = m;
    f[[illegible] + 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);
```

humanorrobot.uk

Games About

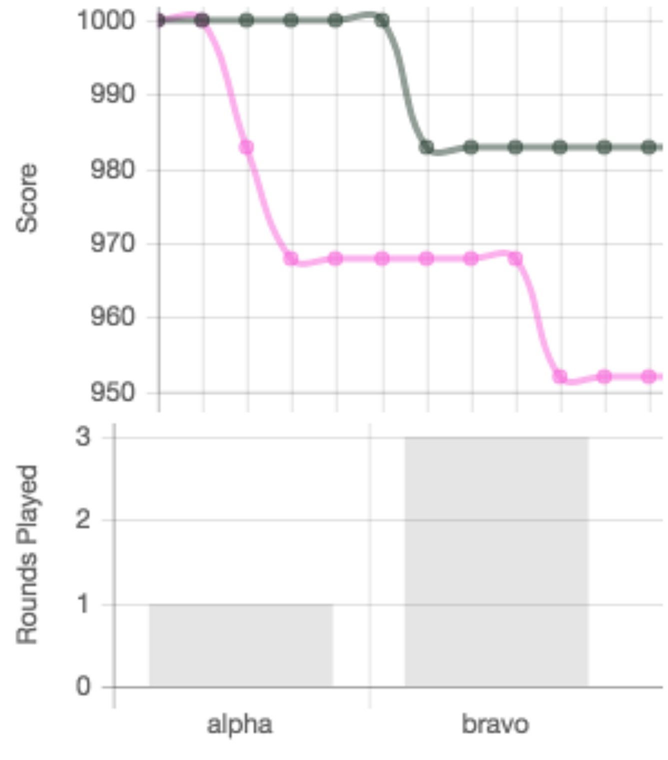
Round 1

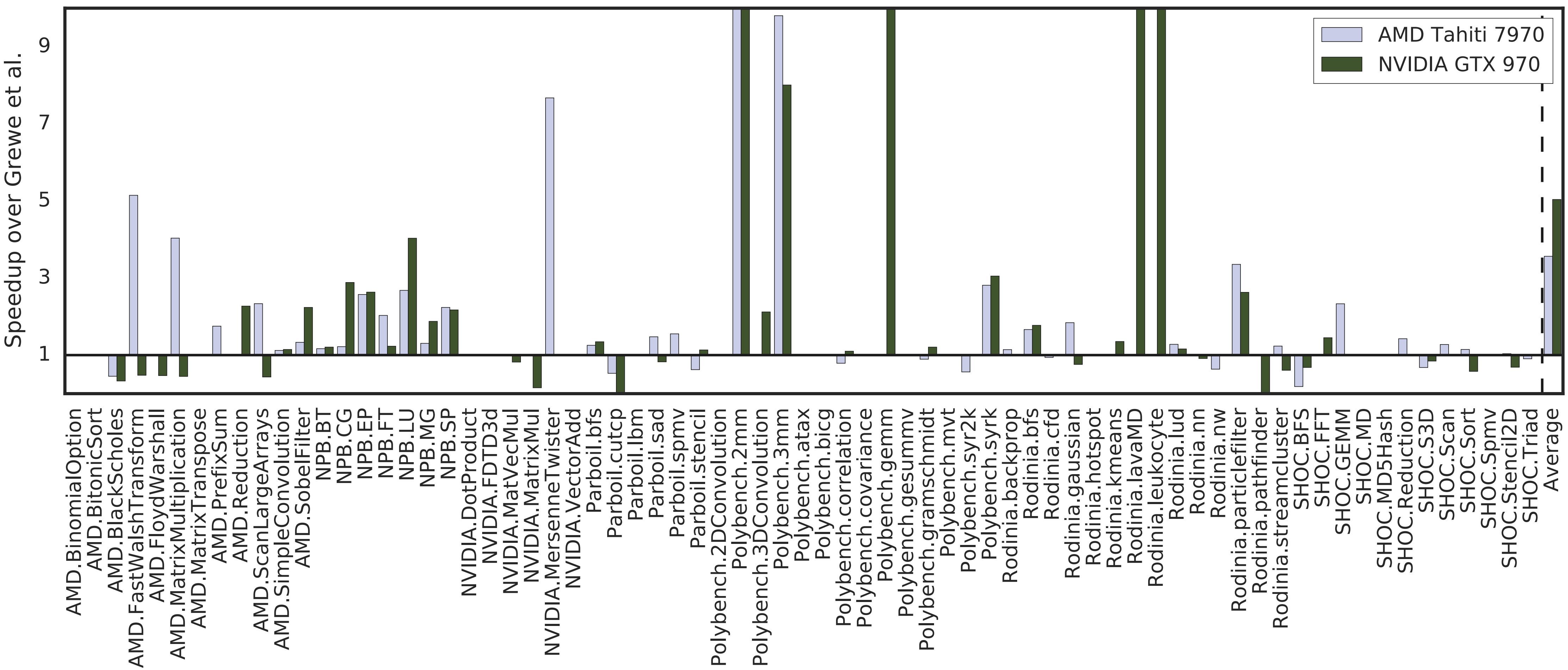
Player: **1010**, Robot: **938**

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

  if (e >= d) {
    return;
  } else {
    a[e] = a[e];
  }
  b[d] = e;
}
```

Try it

Human Robot



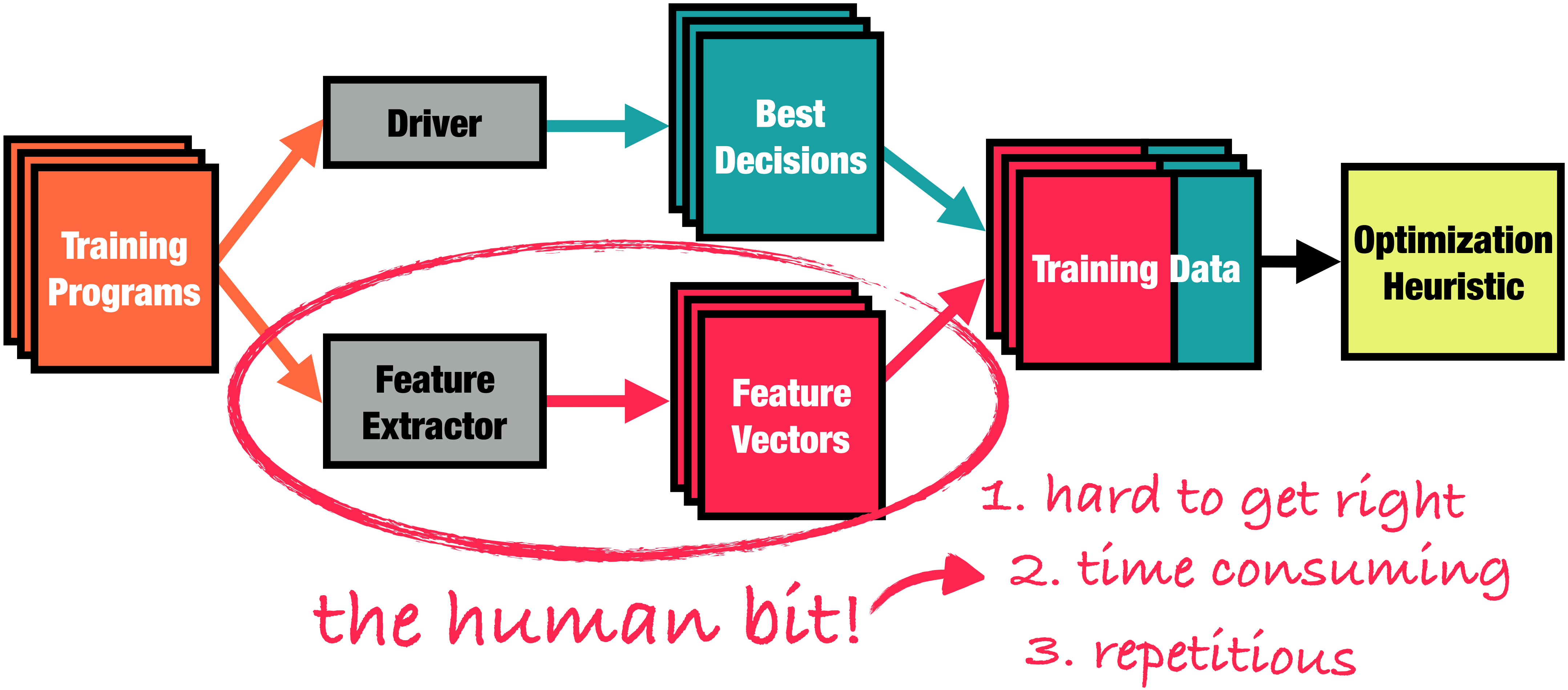


71 programs, 1,000 synthetic benchmarks. 4.30x faster

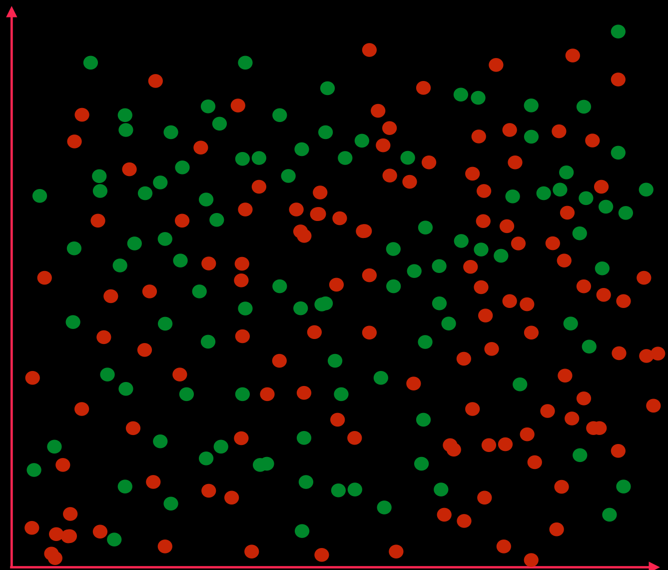
Overview

- Machine Learning for Compilers
- Generating Benchmarks
- **Deep Learned Heuristics**
- Deep Fuzzing Compiler Testing
- Future Work

Machine learning in compilers

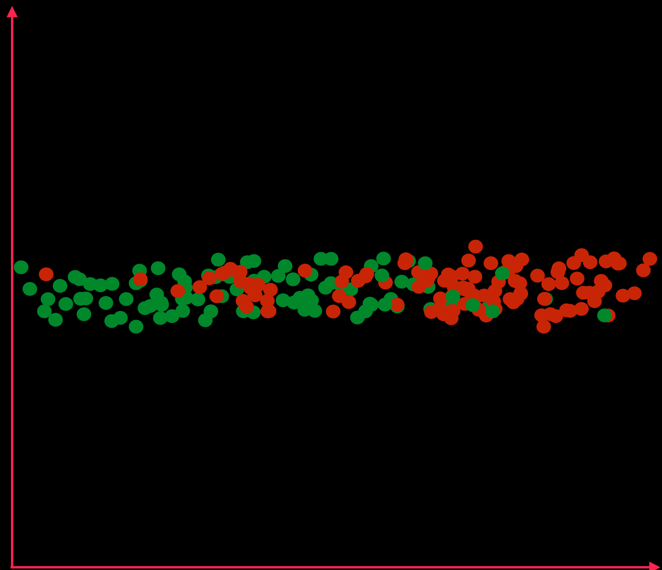


Ways to fail



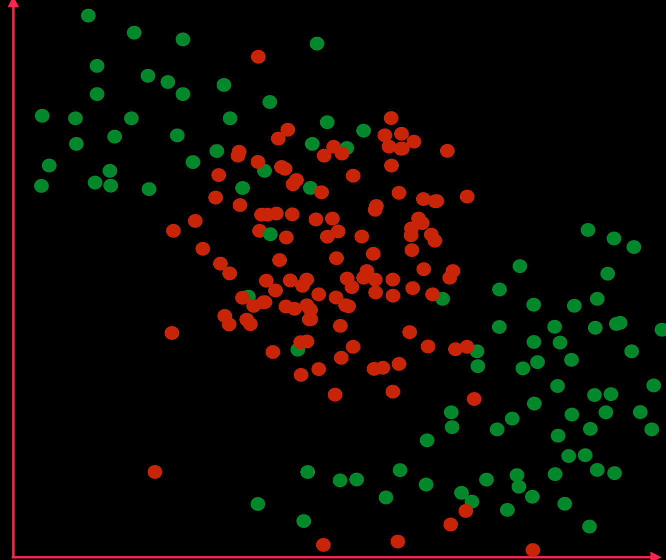
irrelevant

e.g. not capturing the right information



incomplete

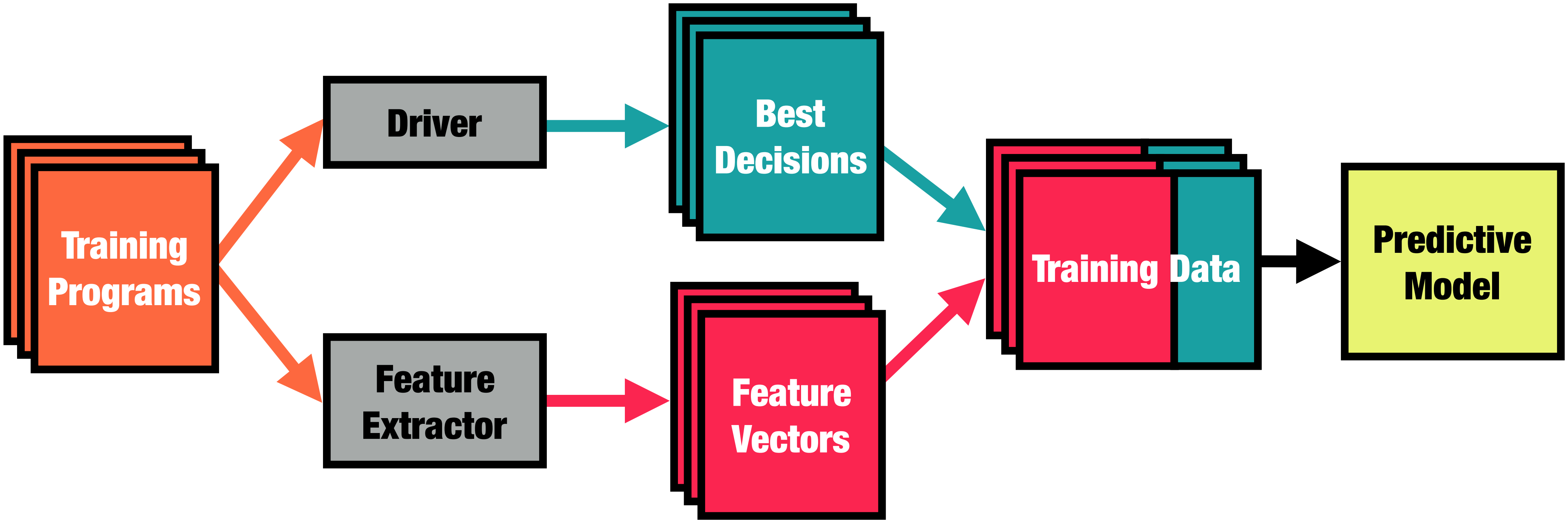
e.g. missing critical information



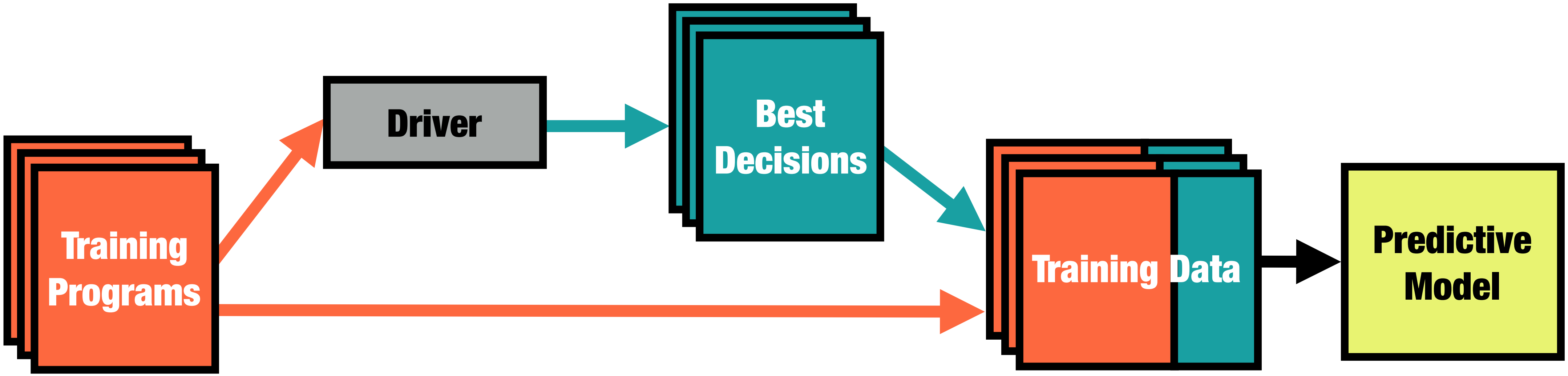
unsuitable

e.g. wrong combination of features / model

What we have


Training Programs
Driver
Feature Extractor
Best Decisions
Feature Vectors
Training Data
Predictive Model

What we need


Training Programs
Driver
Best Decisions
Training Data
Predictive Model

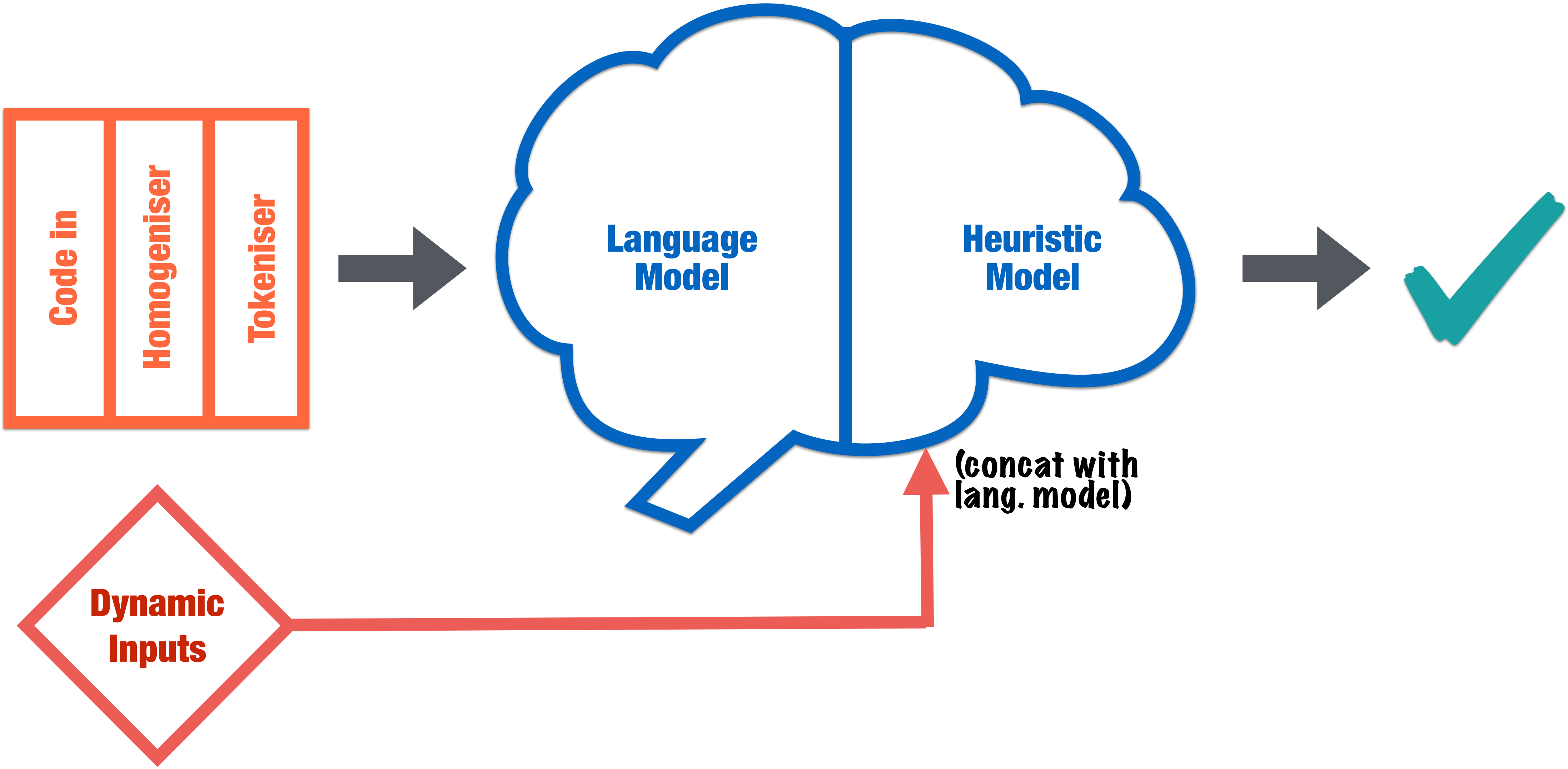
Contributions

Heuristics <u>without features</u>

<u>Beats</u> expert approach

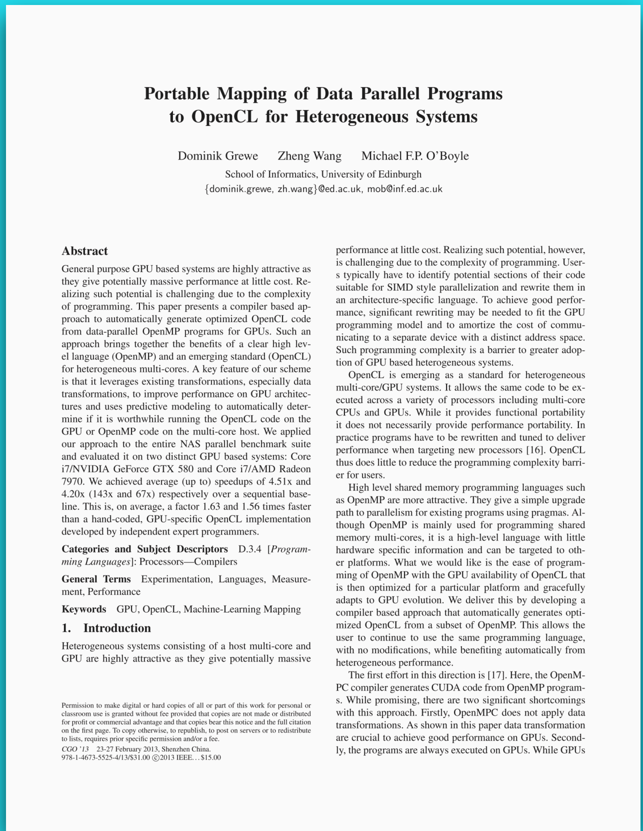
Learning <u>across heuristics</u>

Dynamic Inputs



Prior Art

Heterogeneous Mapping



Portable Mapping of Data Parallel Programs to OpenCL for Heterogeneous Systems

Dominik Grewe Zheng Wang Michael F.P. O'Boyle
School of Informatics, University of Edinburgh
{dominik.grewe, zh.wang}@ed.ac.uk, mob@inf.ed.ac.uk

CGO'13
Grewe et. al

Thread Coarsening

Automatic Optimization of Thread-Coarsening for Graphics Processors

Alberto Magni, School of Informatics, University of Edinburgh, United Kingdom, a.magni@sms.ed.ac.uk
Christophe Dubach, School of Informatics, University of Edinburgh, United Kingdom, christophe.dubach@ed.ac.uk
Michael O'Boyle, School of Informatics, University of Edinburgh, United Kingdom, mob@inf.ed.ac.uk

PACT'14
Magni et. al

Prior Art

Heterogeneous Mapping	Thread Coarsening

Decision Space

Binary classification {CPU, GPU}	**One-of-six classification** {1, 2, 4, 8, 16, 32}

Model

Decision Tree	Cascading **Neural Networks**

CGO'13

PACT'14

Prior Art

Heterogeneous Mapping

Thread Coarsening

Features

4 features

7 features

Combined from 7 raw values.

Principle Components of 34 raw values.

2 papers!

Instruction counts / ratios.

Instruction counts / ratios / relative deltas.

CGO'13

PACT'14

Prior Art

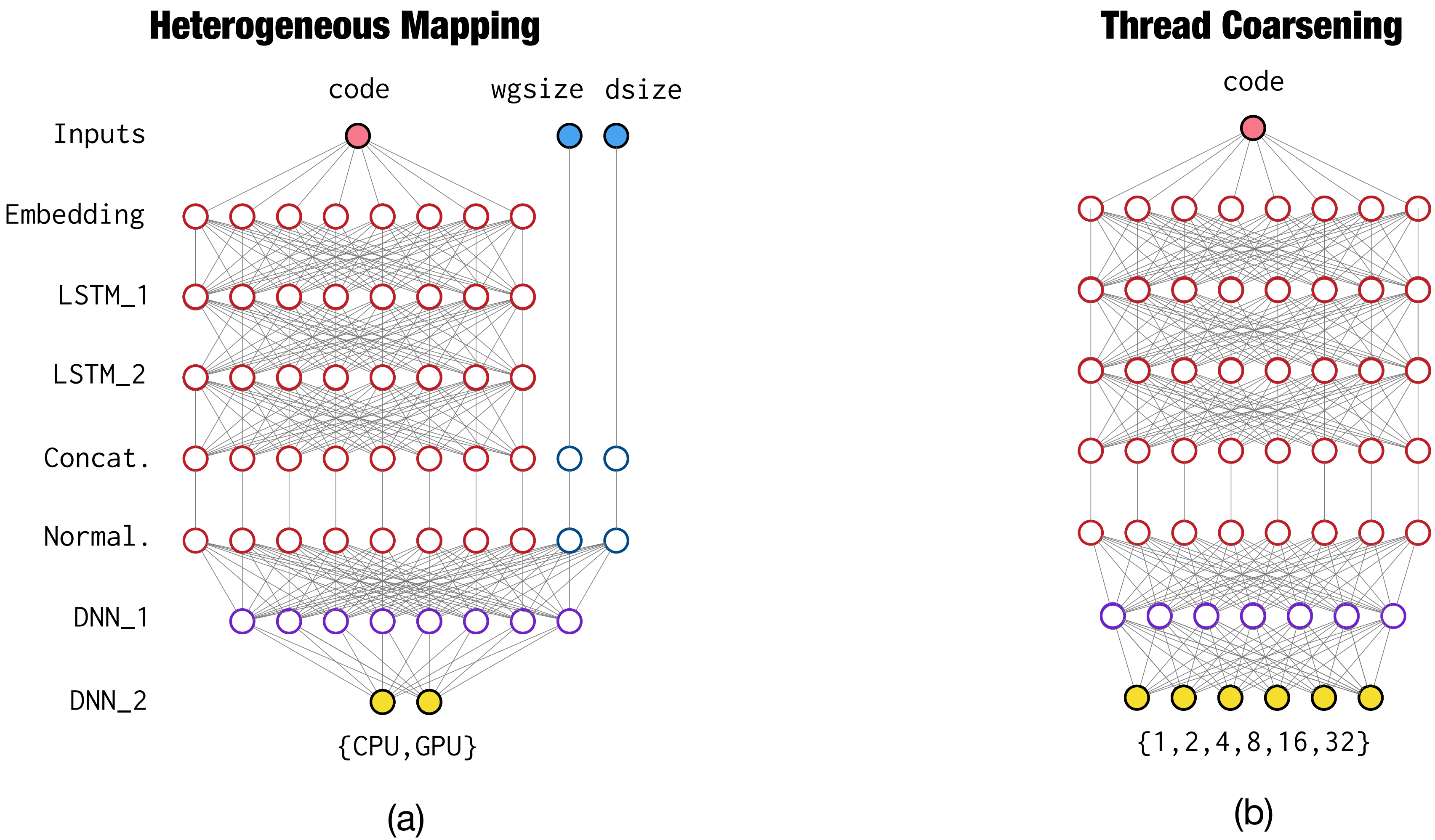
Heterogeneous Mapping	Thread Coarsening
Hardware	
2x CPU-GPU architectures	4x GPU architectures
Training Programs	
7 Benchmark Suites	3 Benchmark Suites
CGO'13	PACT'14

Our Approach

Heterogeneous Mapping | **Thread Coarsening**

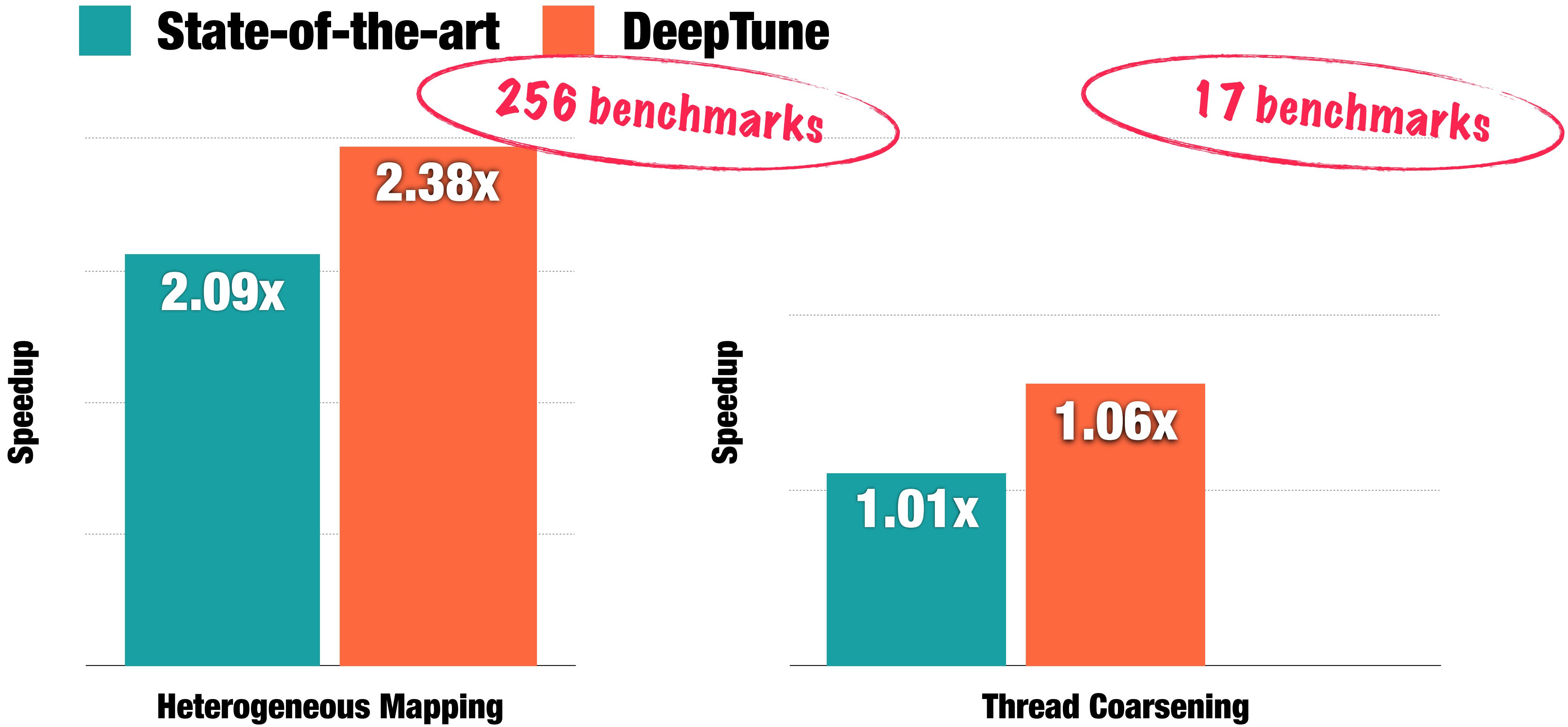
1. Use the same model design for both
2. No tweaking of parameters
3. Minimum change - 3 line diff

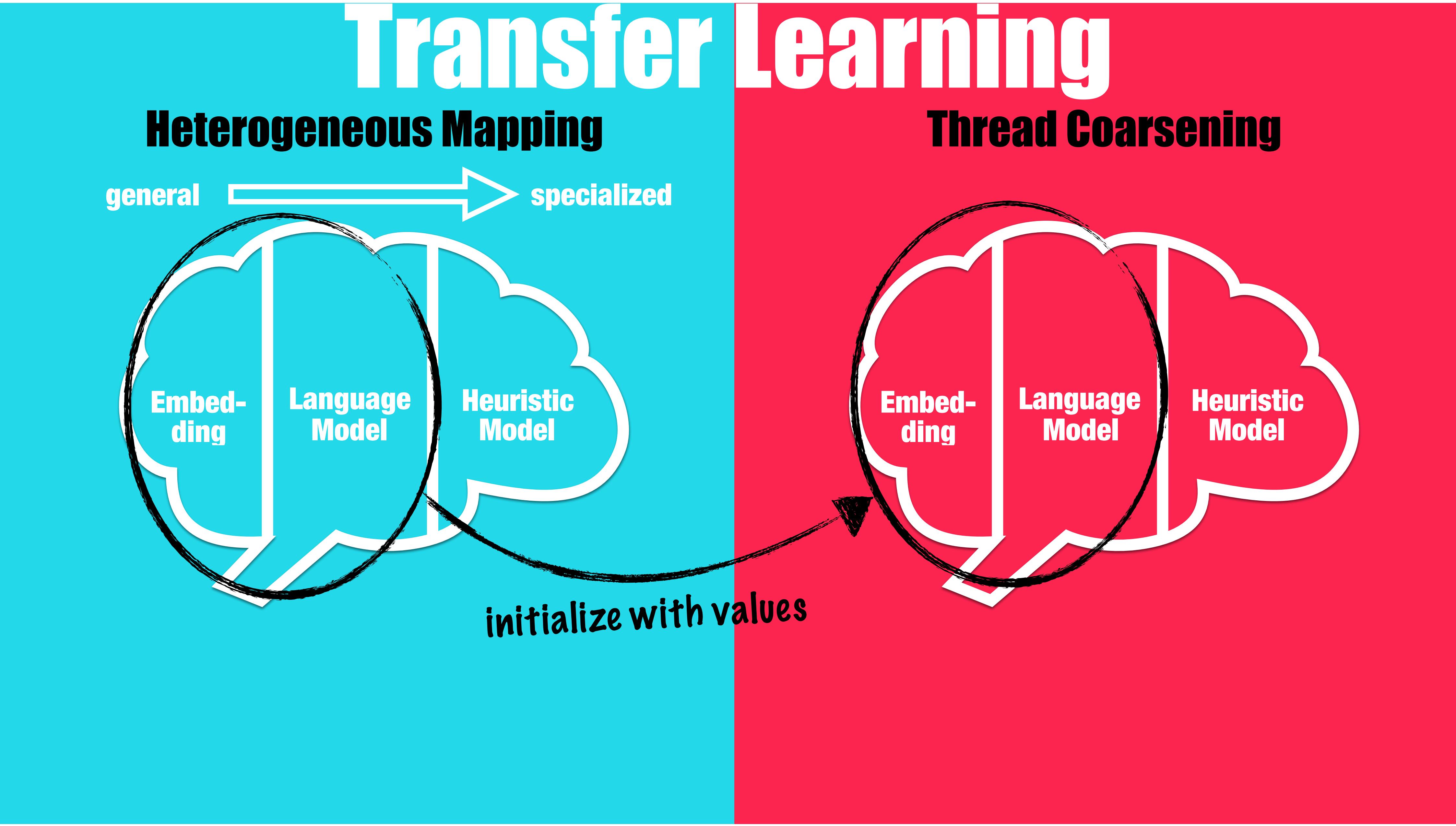
Neural Networks



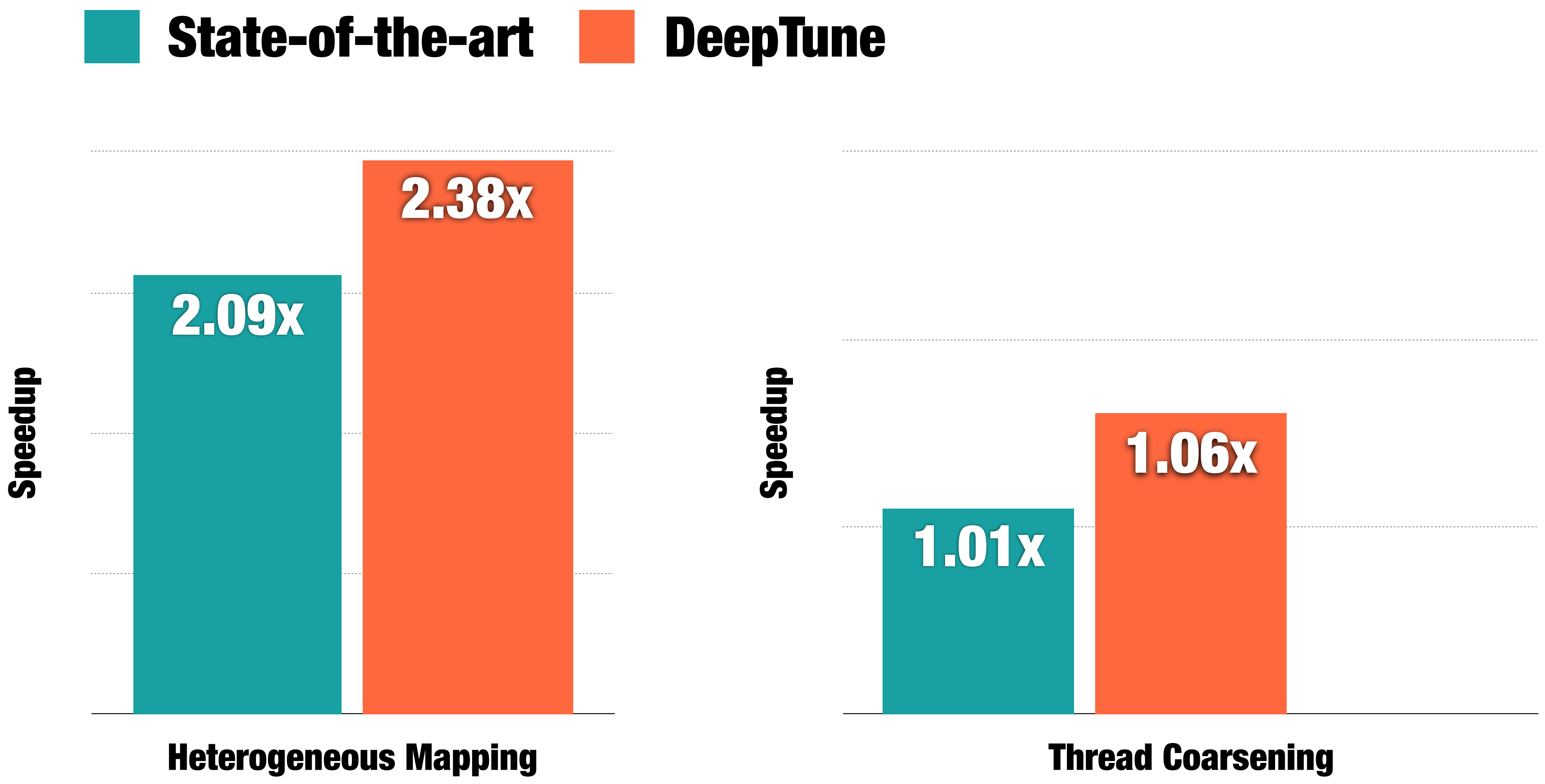
How well does it work?

14% and 5% improvements over state-of-the-art

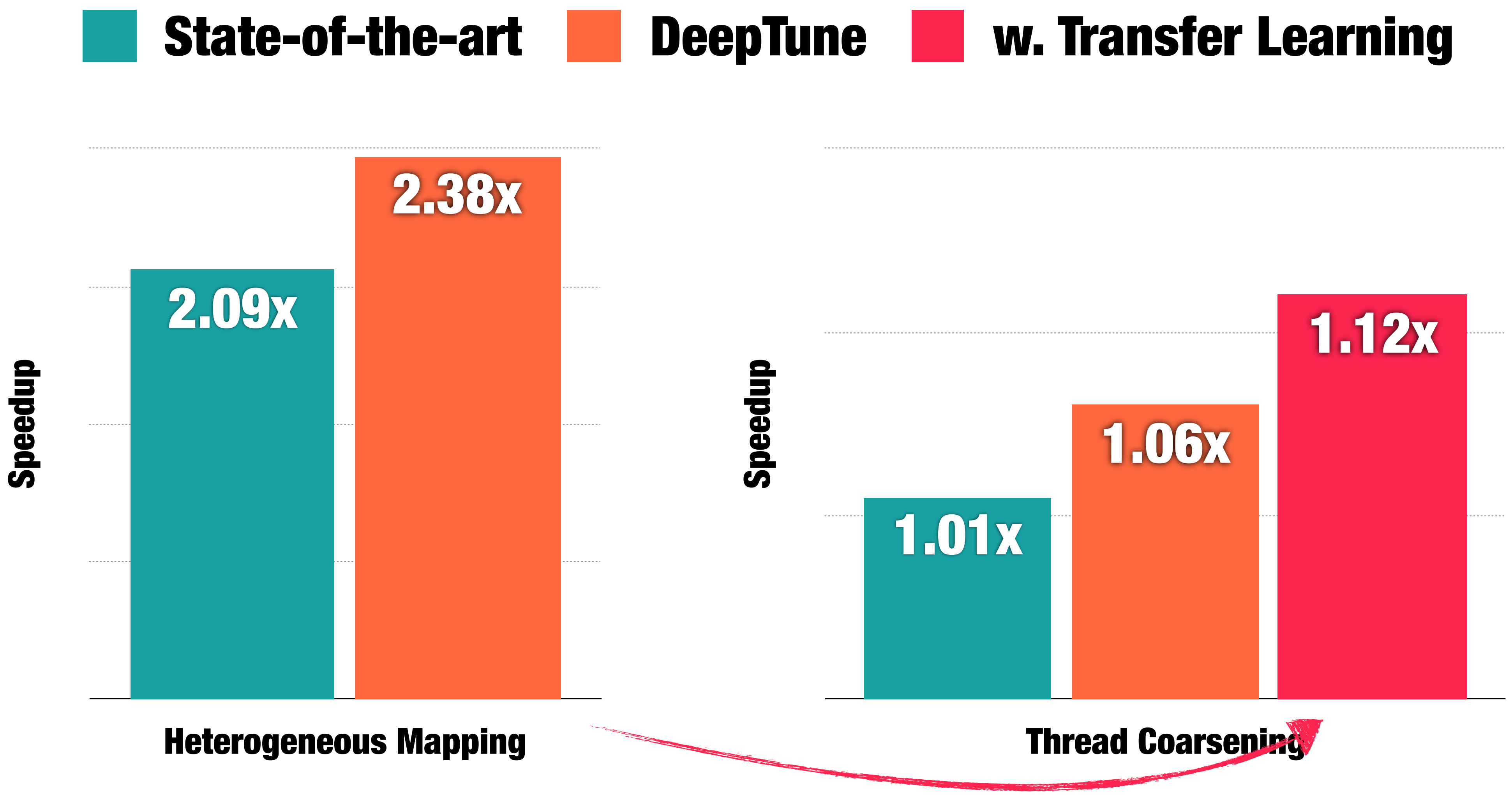



Transfer Learning
Heterogeneous Mapping
Thread Coarsening
general
specialized
Embed-
ding
Language
Model
Heuristic
Model
Embed-
ding
Language
Model
Heuristic
Model
initialize with values

14% and 5% improvements over state-of-the-art



14% and 11% improvements over state-of-the-art



Overview

- Machine Learning for Compilers
- Generating Benchmarks
- Deep Learned Heuristics
- **Deep Fuzzing Compiler Testing**
- Future Work

compilers break

Compiler crash

Rewrite code around bug

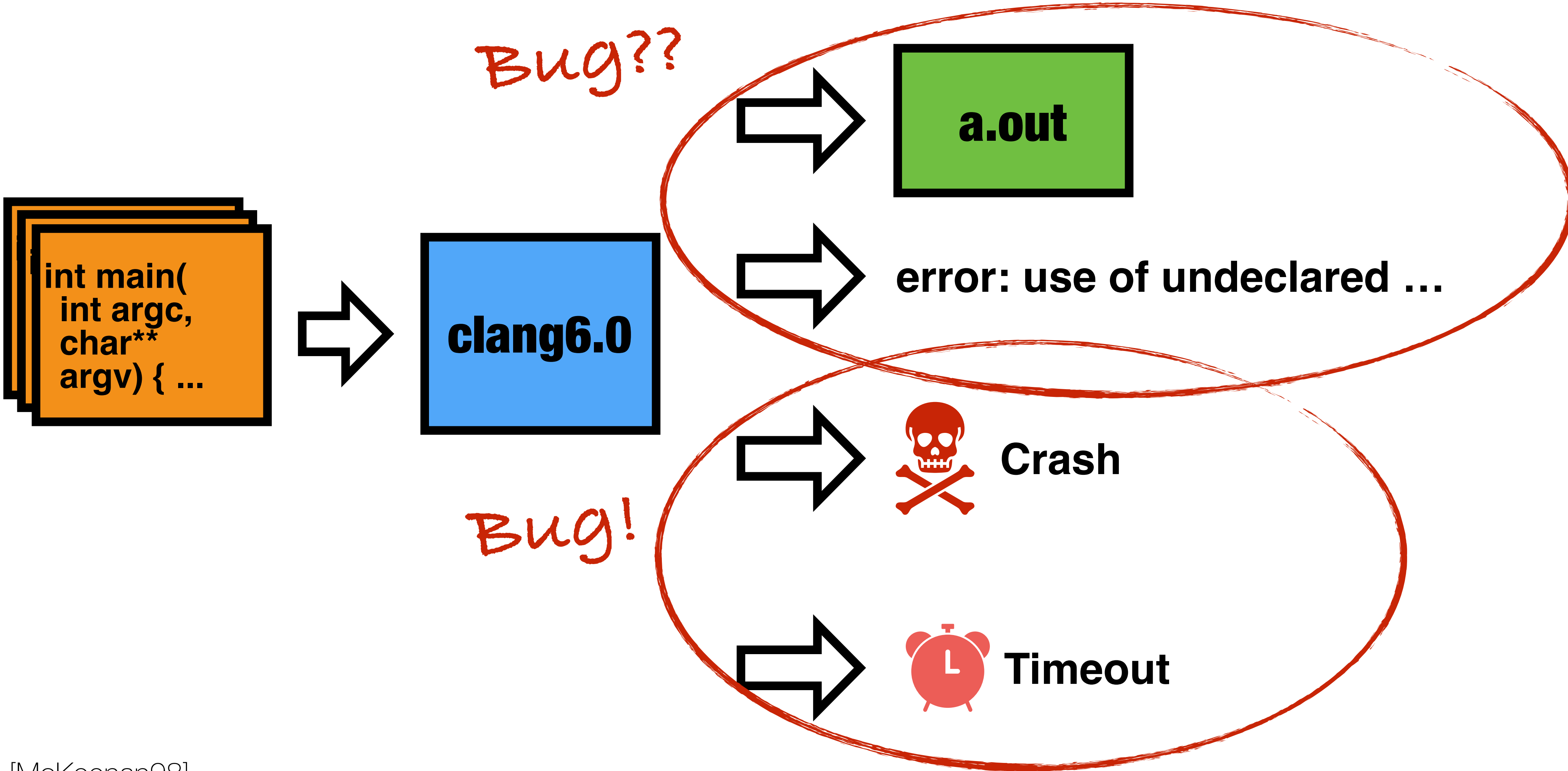
Semantics change

Security risk

Regression suites

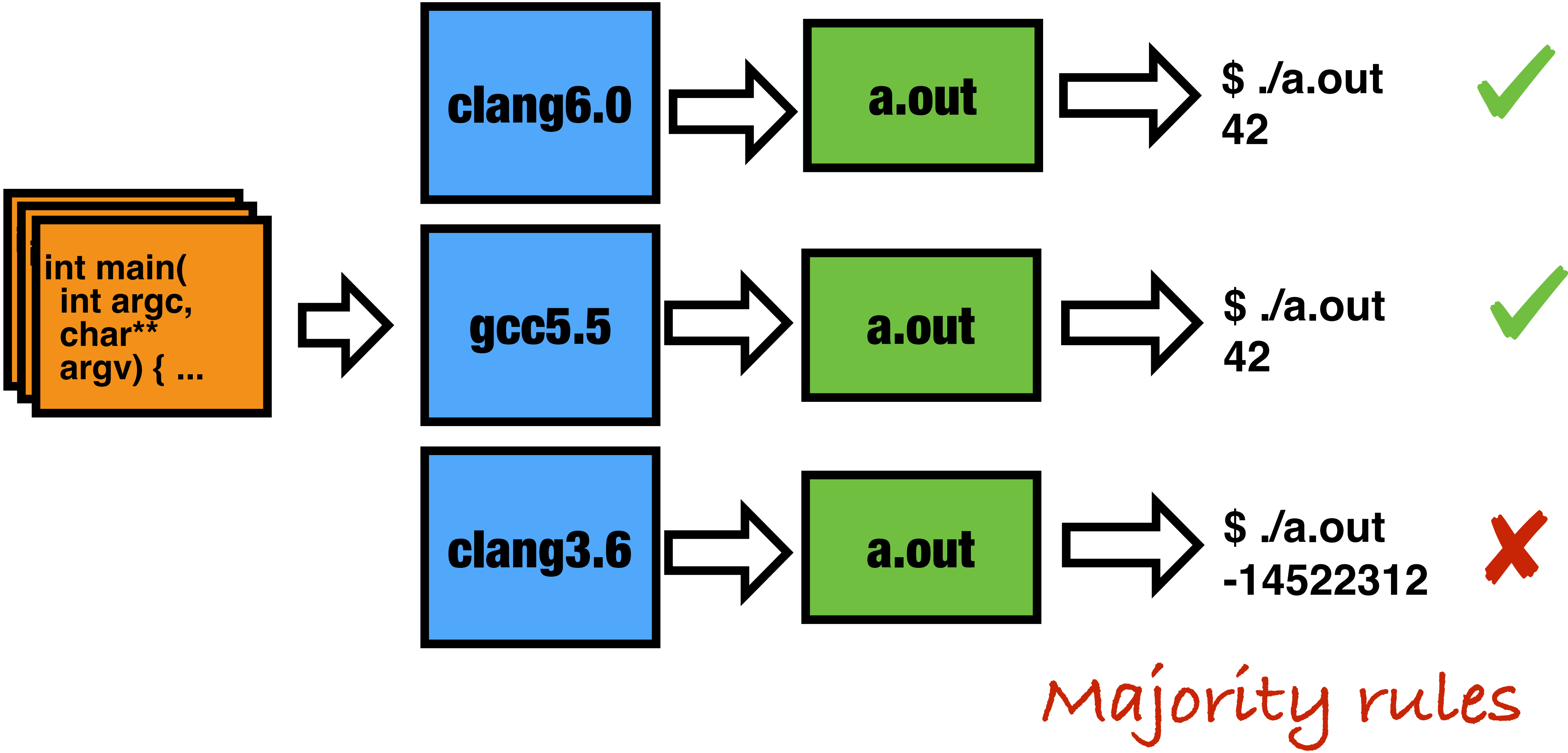
- Slow
- Late
- Expensive
- Incomplete

fuzzing a compiler



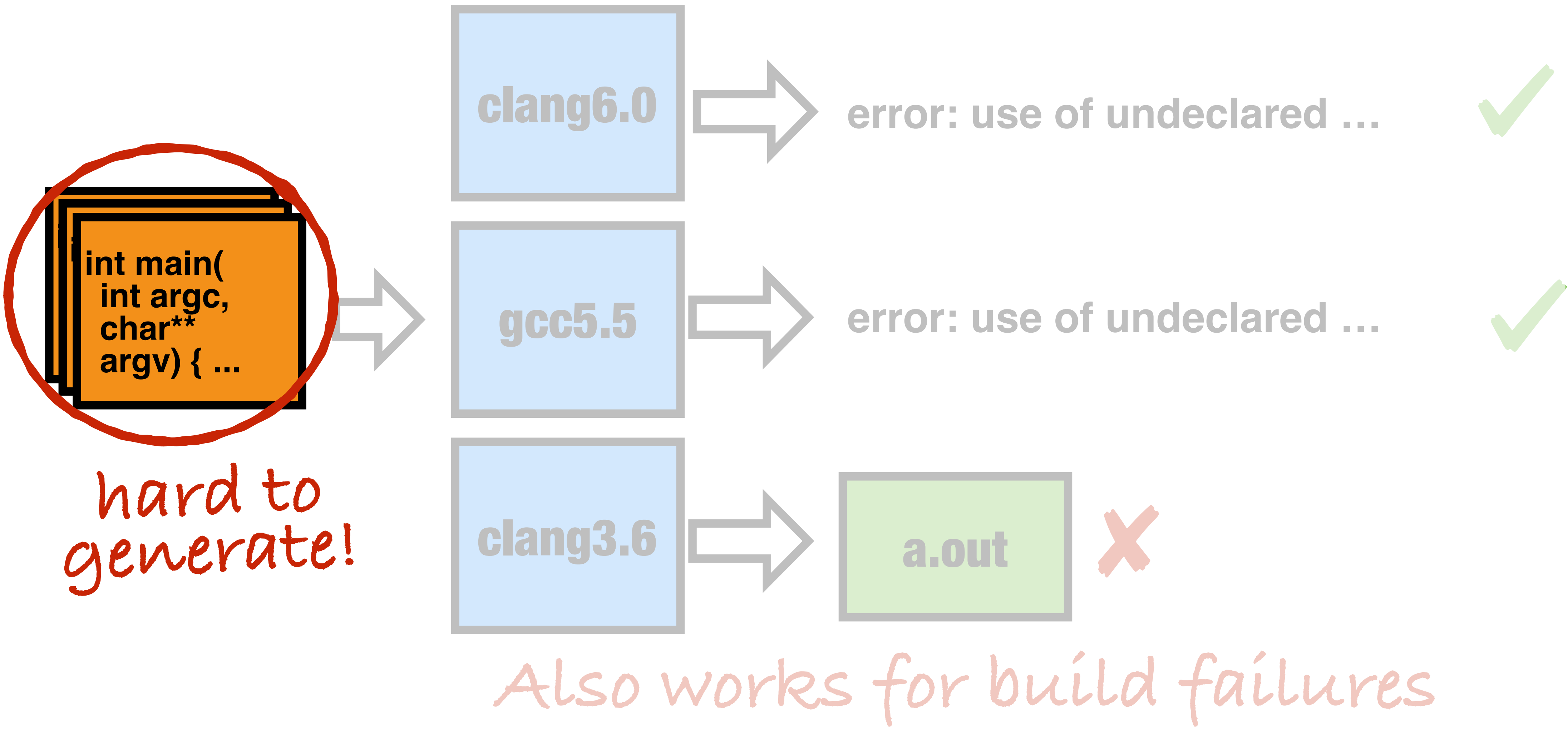
circa [McKeenan98]

differential testing compilers



circa [McKeenan98]

differential testing compilers



circa [McKeenan98]

an ideal fuzzer

1. **Cheap**

   Easy to implement and extend

   (Languages and features grow quickly)

2. **Interpretable Testcases**

   Necessary for triage

   (i.e. 45 lines or less [Sun2016])

3. **Plausible Output**

   Representative of handwritten code

   (So that bugs gets fixed)

state-of-the-art: CLSmith

https://github.com/ChrisLidbury/CLSmith

```
#include "CLSmith.h"

struct S0 {
  int32_t g_4[4][10];
  ...
};

kernel void A(global ulong *r) {
  int i, j, k;
  struct S0 c_1856;
  struct S0* p_1855 = &c_1856;
  c_1856 = c_1857;
  func_1(p_1855);
  barrier(CLK_LOCAL_MEM_FENCE
| CLK_GLOBAL_MEM_FENCE);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 10; j++)
...>g_4[i][j], "p_1855->g_4[i][j]",
print_hash_value);
  result[get_linear_global_id()] =
crc64_context ^
```

Random grammar enumeration.

Extensive static analyses support subset of OpenCL features.

Targets compiler middle ends.

Incredibly effective!
100s of bugs to date.

state-of-the-art: CLSmith

https://github.com/ChrisLidbury/CLSmith

1. **Cheap ✖ nope!**
   Years to develop! 50k lines of C++.
   Each PL feature engineered by hand.

2. **Interpretable Testcases ✖ nope!**
   Avg. 1200 lines (excluding headers).
   Requires reduction: ~4 hours / test.

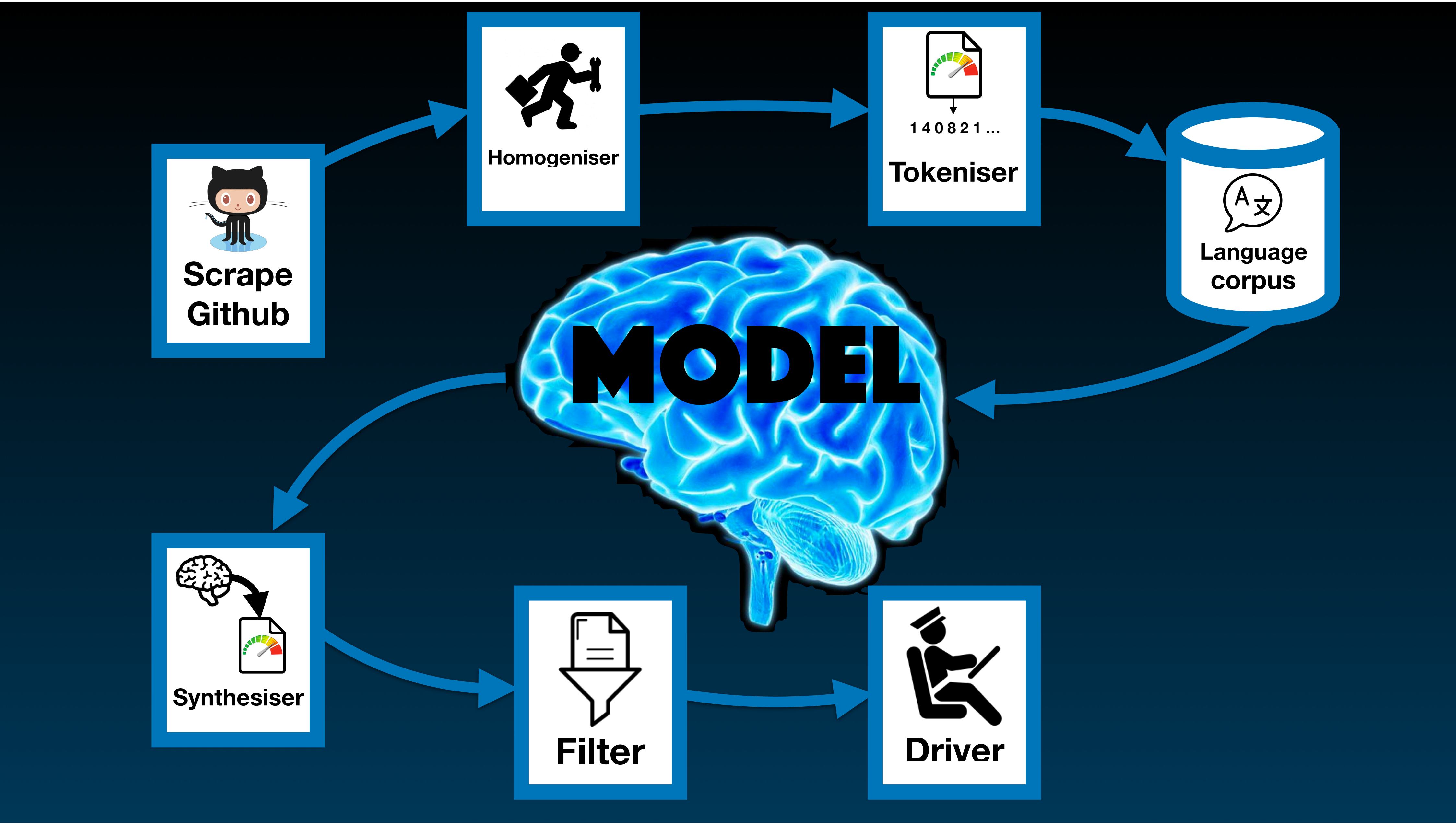
3. **Plausible Output ✖ nope!**
   Unusual and restricted combinations of PL features.
   87 dials control "shape" of output - hand tuned.

contributions

Automatic inference of fuzzers from examples.

102x less code than state-of-art.

Similar bug finding power, simpler test cases.


Homogeniser
1 4 0 8 2 1 ...
Tokeniser
Scrape
Github
Language
corpus
MODEL
Synthesiser
Filter
Driver

how well does it work?

testing campaign

10 OpenCL compilers

- 3 GPUs, 5 CPUs, Xeon Phi, Emulator

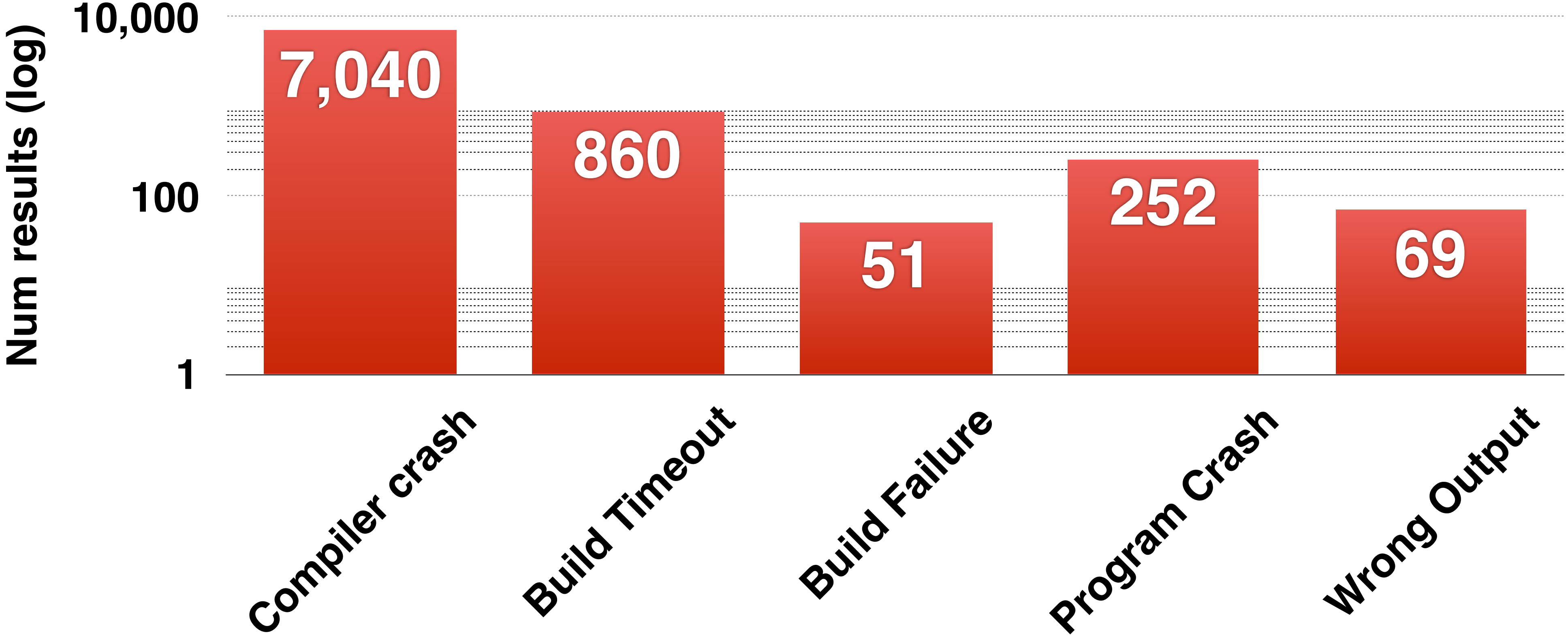
Test with optimizations on / off

- Treat as separate testbeds

48 hours per testbed

results overview

Errors in every compiler!



67 bug reports to date…

… crashes during parsing / compilation

```
void A() {void* a; uint4 b=0; b=(b>b)?a:a }
```

Affects: Intel OpenCL SDK 1.2.0.25

```
kernel void A(global int* a) {
  int b = get_global_id(0);
  a[b] = (6 * 32) + 4 * (32 / 32) + a;
}
```

Affects: Beignet 1.3

“Bad code” finds bugs in error handling

67 bug reports to date…

… crashes during type checking

```
kernel void A() {
  __builtin_astype(d, uint4);
}
```

Affects: 6 / 10 compilers we tested

Unexpected outcome: Learning from handwritten code leads to bugs found in compiler builtins!

67 bug reports to date…

… errors in optimizers

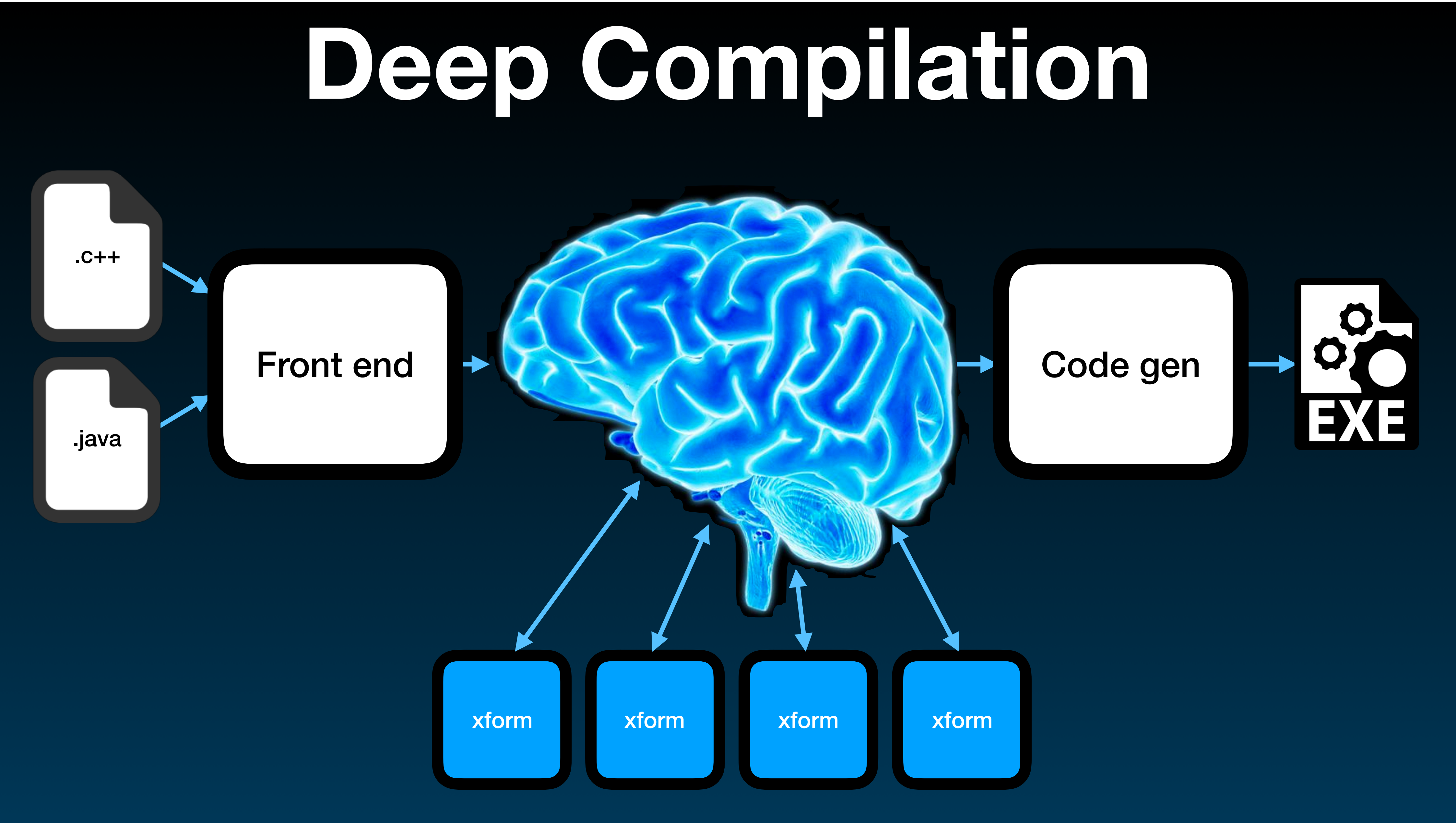
```
kernel void A(global double* a, global double* b,
              global double* c, int d, int e) {
  double f;
  int g = get_global_id(0);
  if (g < e - d - 1)
    c[g] = (((e) / d) % 5) % (e + d);
}
```

Affects: Intel OpenCL SDK 1.2.0.25

CLSmith doesn't allow thread-dependent control flow.

Overview

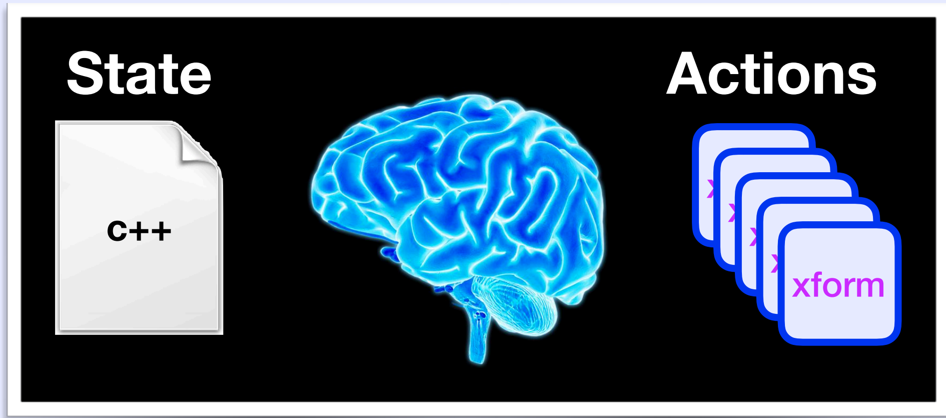
- Machine Learning for Compilers
- Generating Benchmarks
- Deep Learned Heuristics
- Deep Fuzzing Compiler Testing
- **Future Work**


Deep Compilation
.c++
.java
Front end
Code gen
EXE
xform
xform
xform
xform

Deep Reinforced Super Optimisation

Super optimisation

- Brute force search for optimal code
- Excellent results
- **Slow**
- Need smart search



- Use reinforcement learning
- DNN chooses actions
- Actions are xform or change focus
- Stop when predicts no improvement

Deep Data Flow

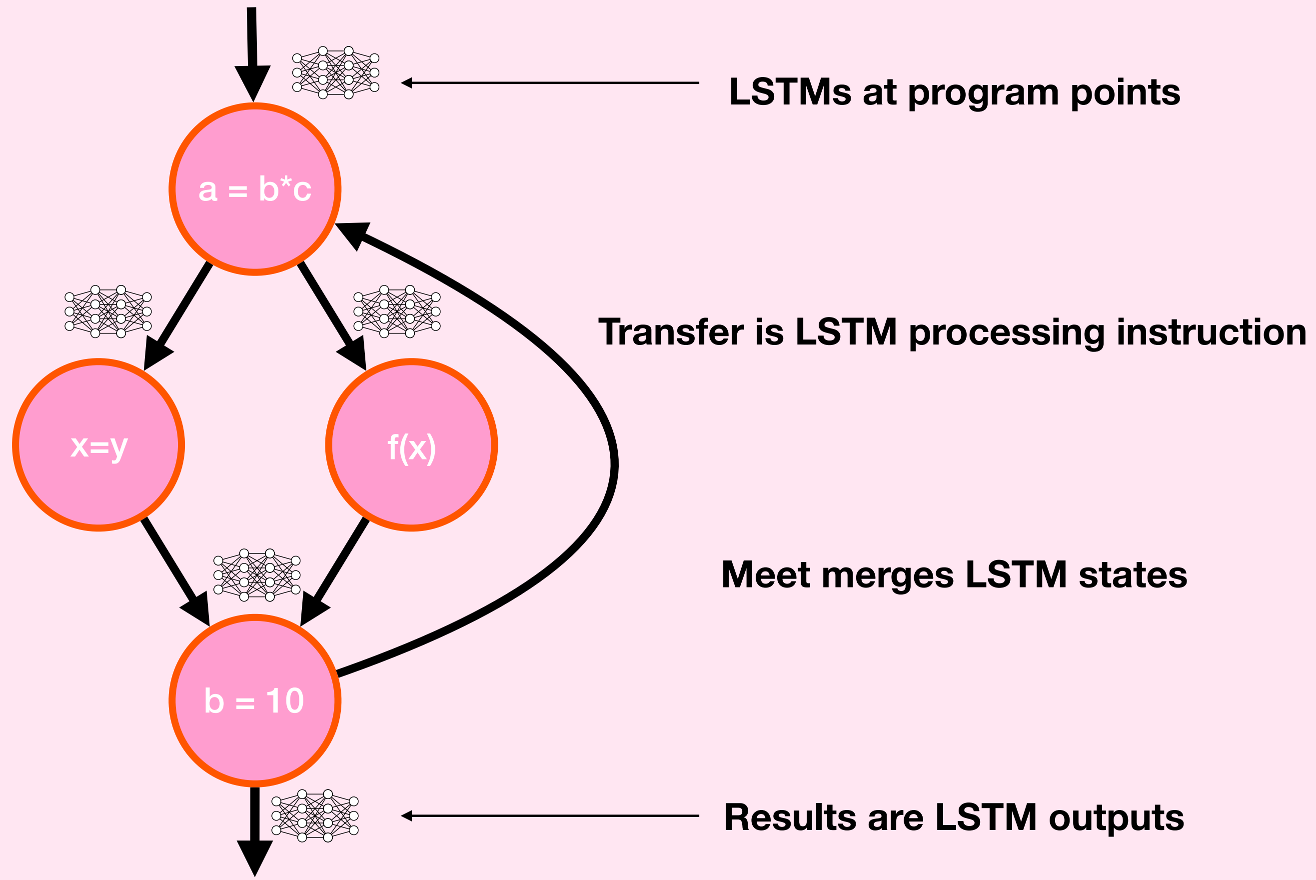
Learn analyses for heuristics not correctness

DNN struggle with data flow

LSTM cannot analyse even reachability on CFG

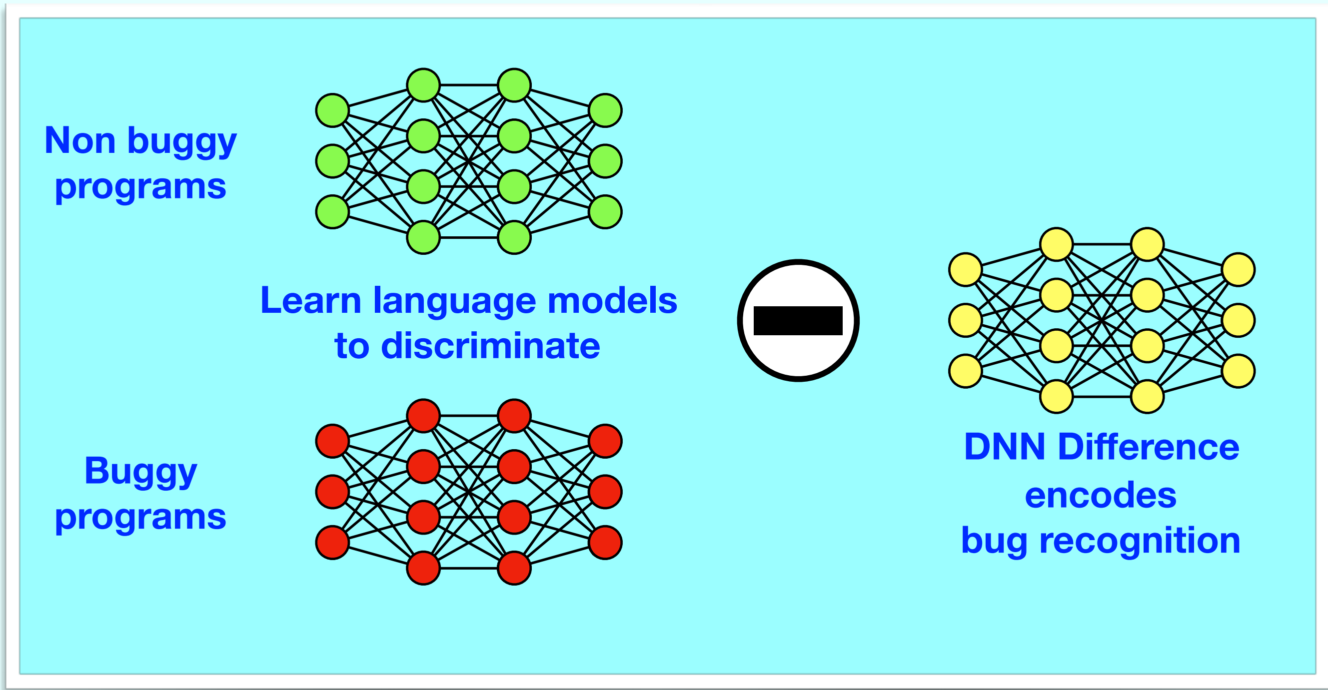
But can learn if given traces

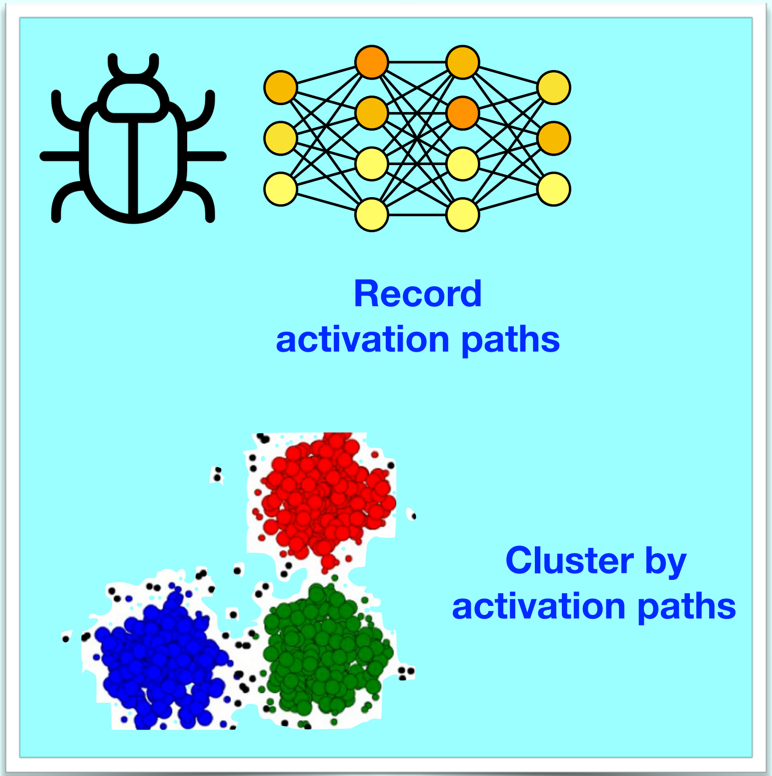
Can we extend to abstract interpretation?



Automatic Bug Triage

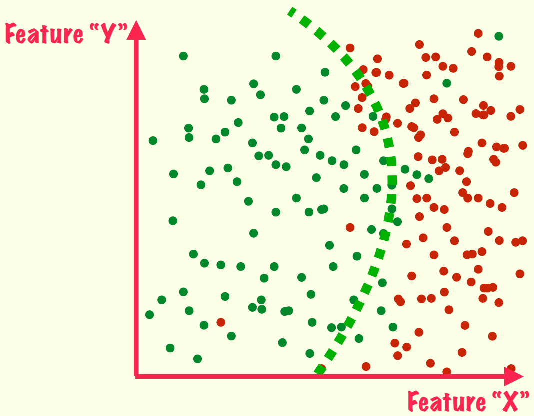
Fuzzers make thousands of bug cases too quickly



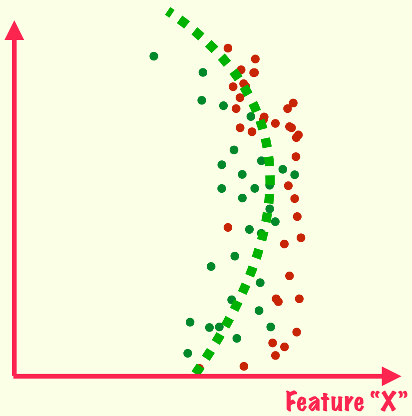


Deep Active Learning

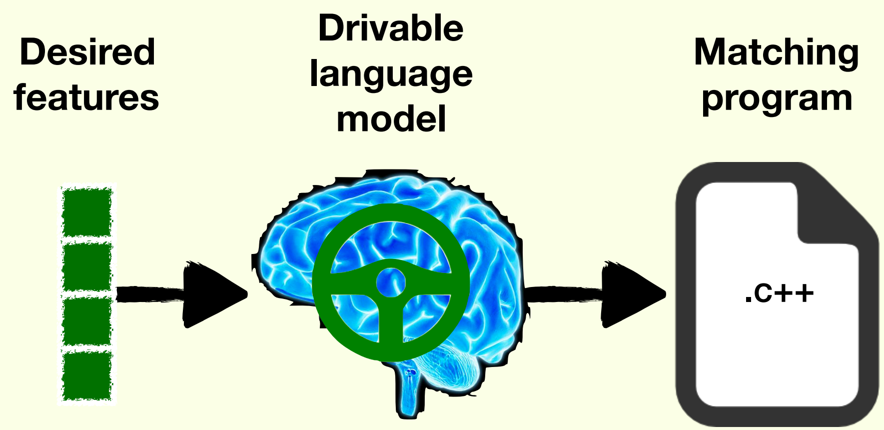
Most points uninteresting



Good ones do just as well



Active learning directly selects useful points



Conclusion

- **Deep learning = better compilers**
- **Deep learning = lower cost**
- **Fun stuff still to do**