

Synthesizing Benchmarks for Predictive Modeling

<http://chriscummins.cc/cgo17>



Chris Cummins

University of Edinburgh



Pavlos Petoumenos

University of Edinburgh



Zheng Wang

Lancaster University



Hugh Leather

University of Edinburgh

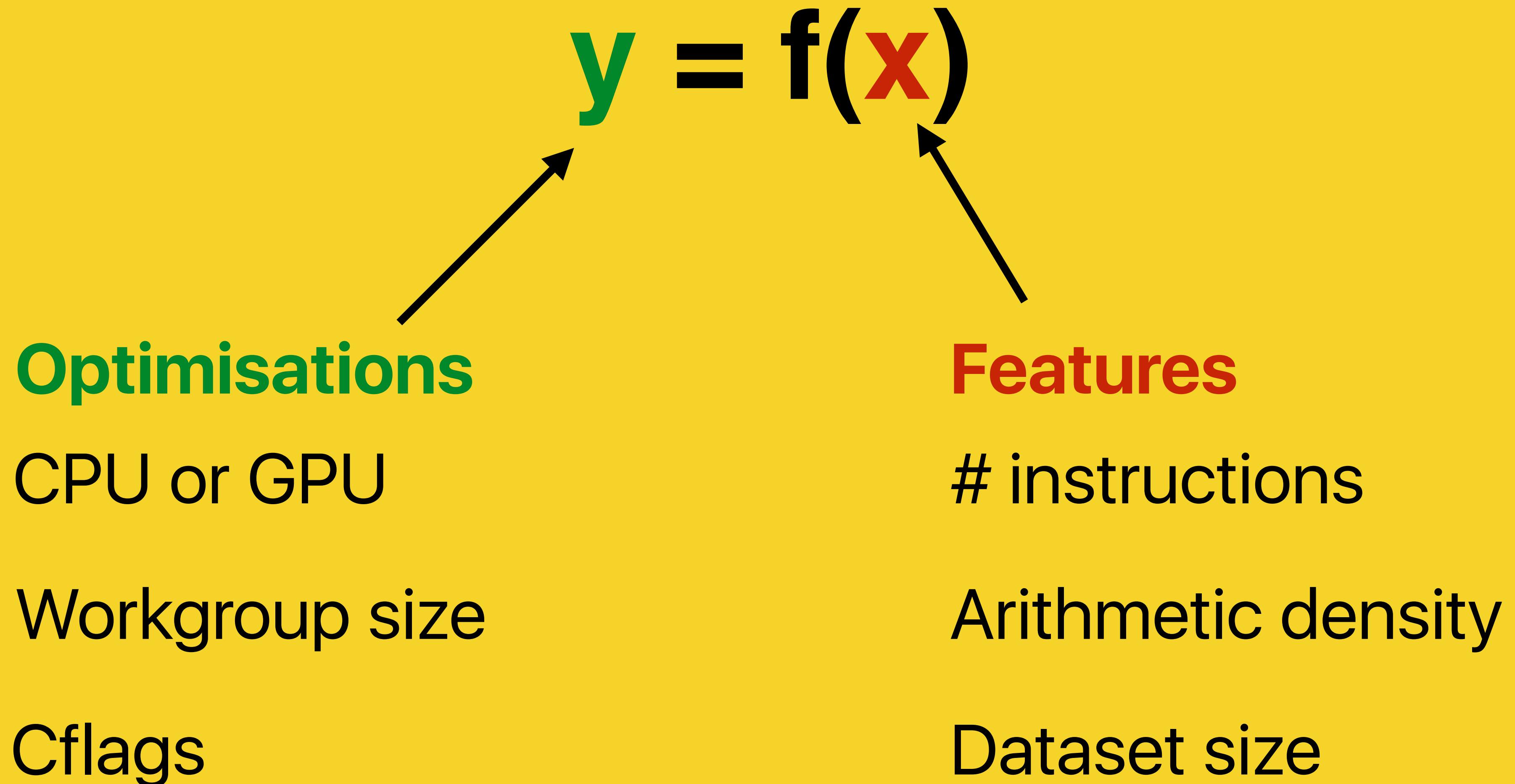


THE UNIVERSITY of EDINBURGH
informatics

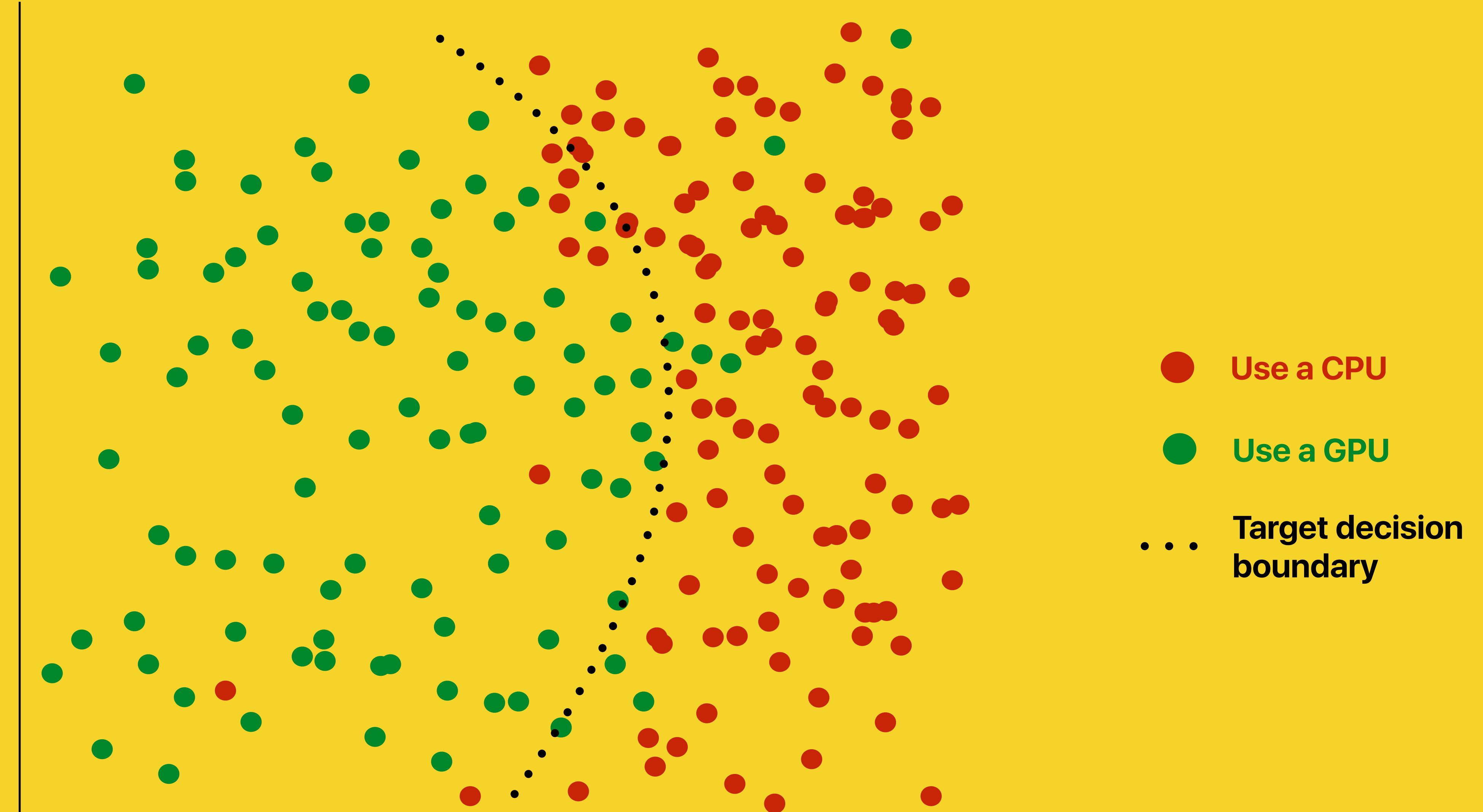
EPSRC Centre for Doctoral Training in
Pervasive Parallelism

EPSRC
Engineering and Physical Sciences
Research Council

machine learning for compilers

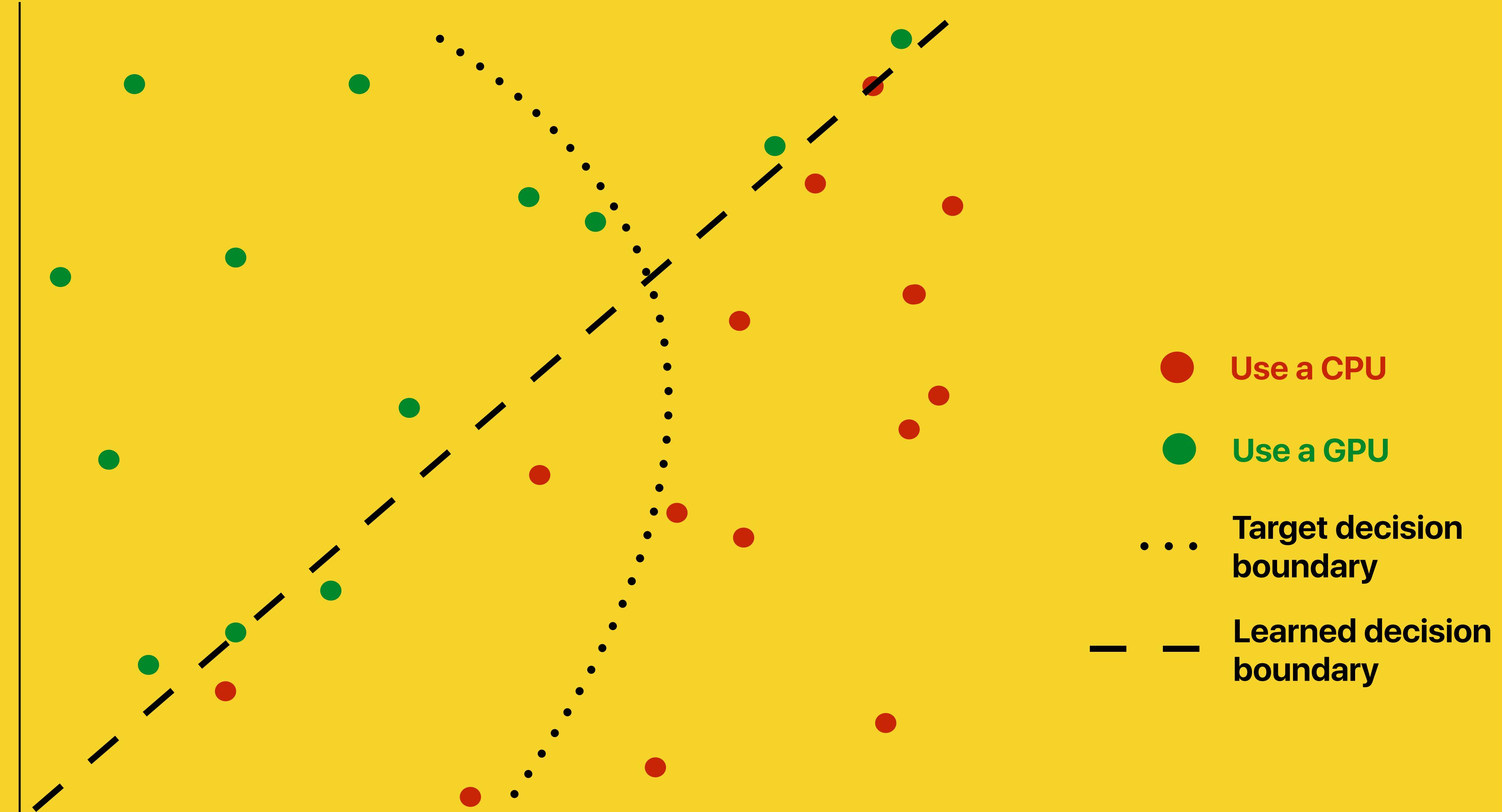


machine learning for compilers



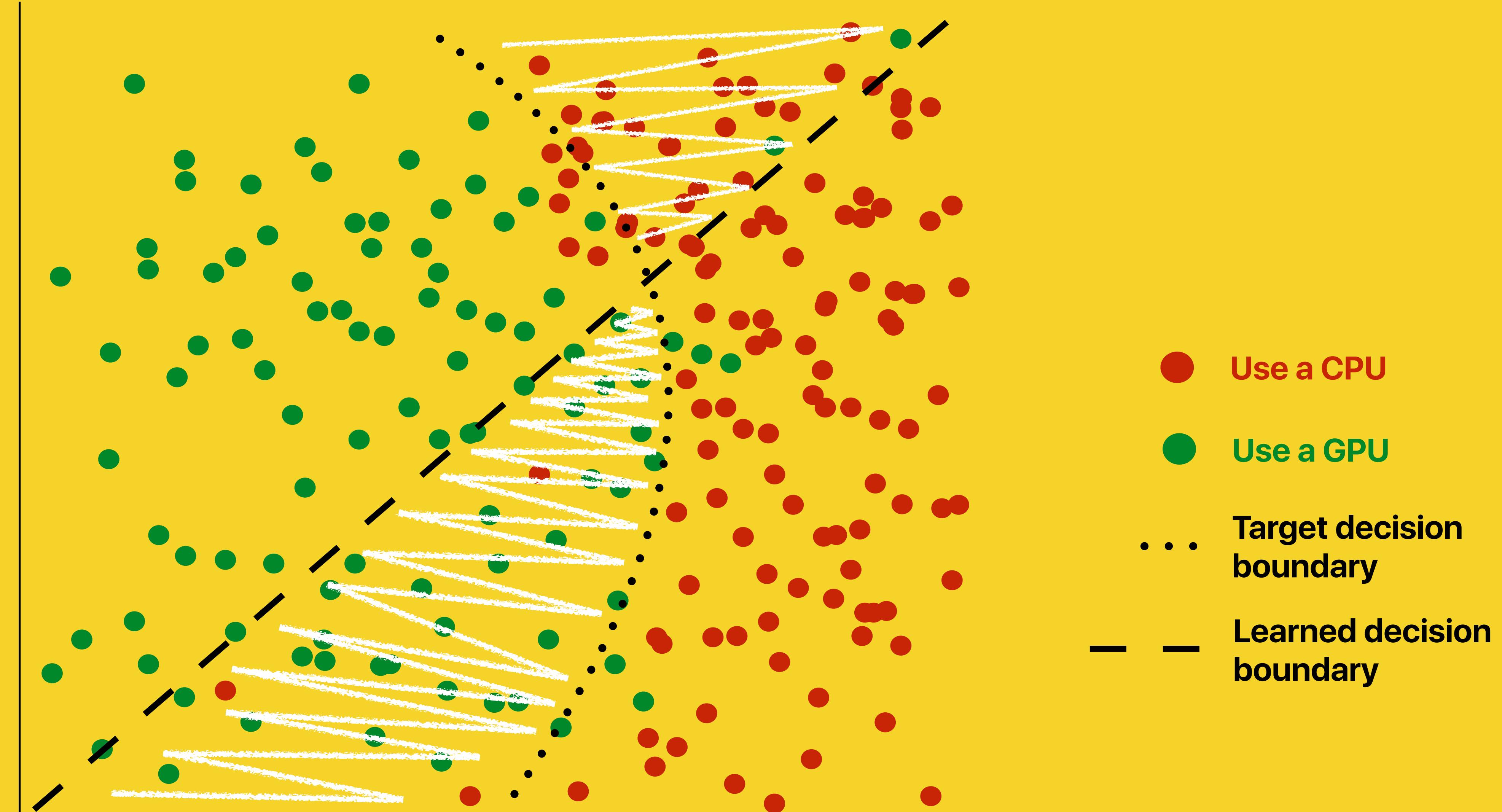
The idea.

machine learning for compilers



The reality.

machine learning for compilers



sparse data leads to inaccurate models!

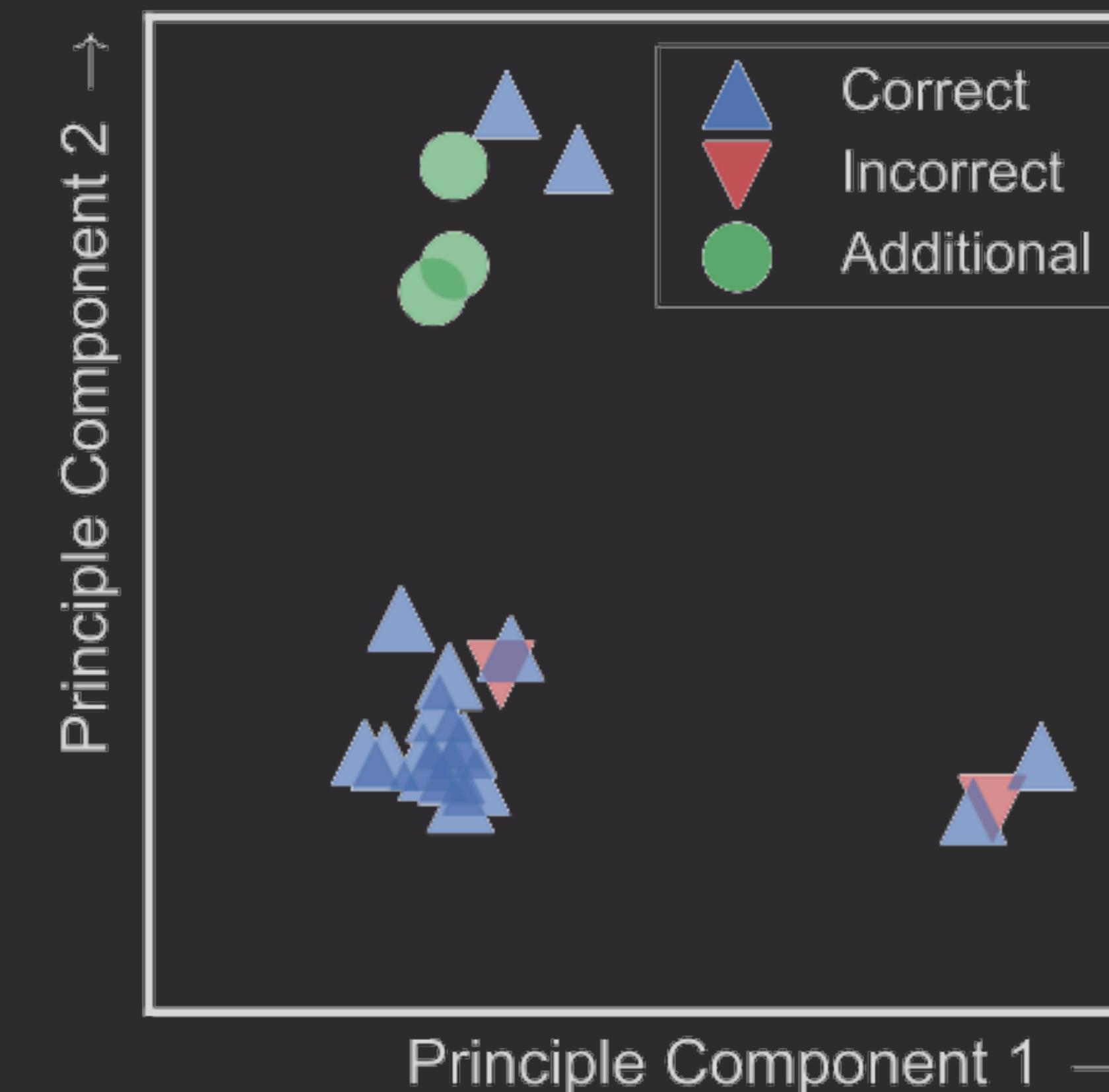
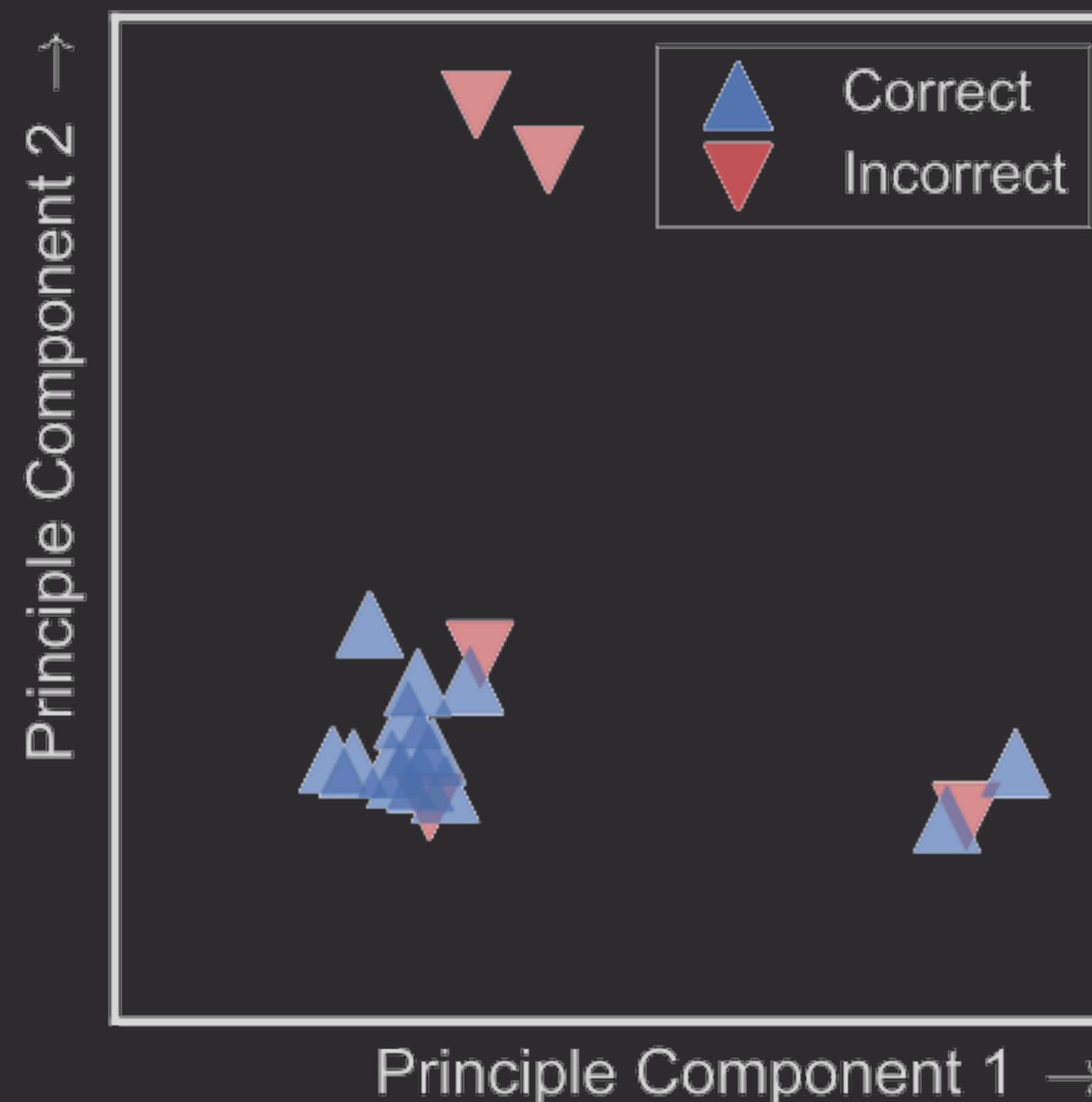
problem statement

1. there aren't enough benchmarks

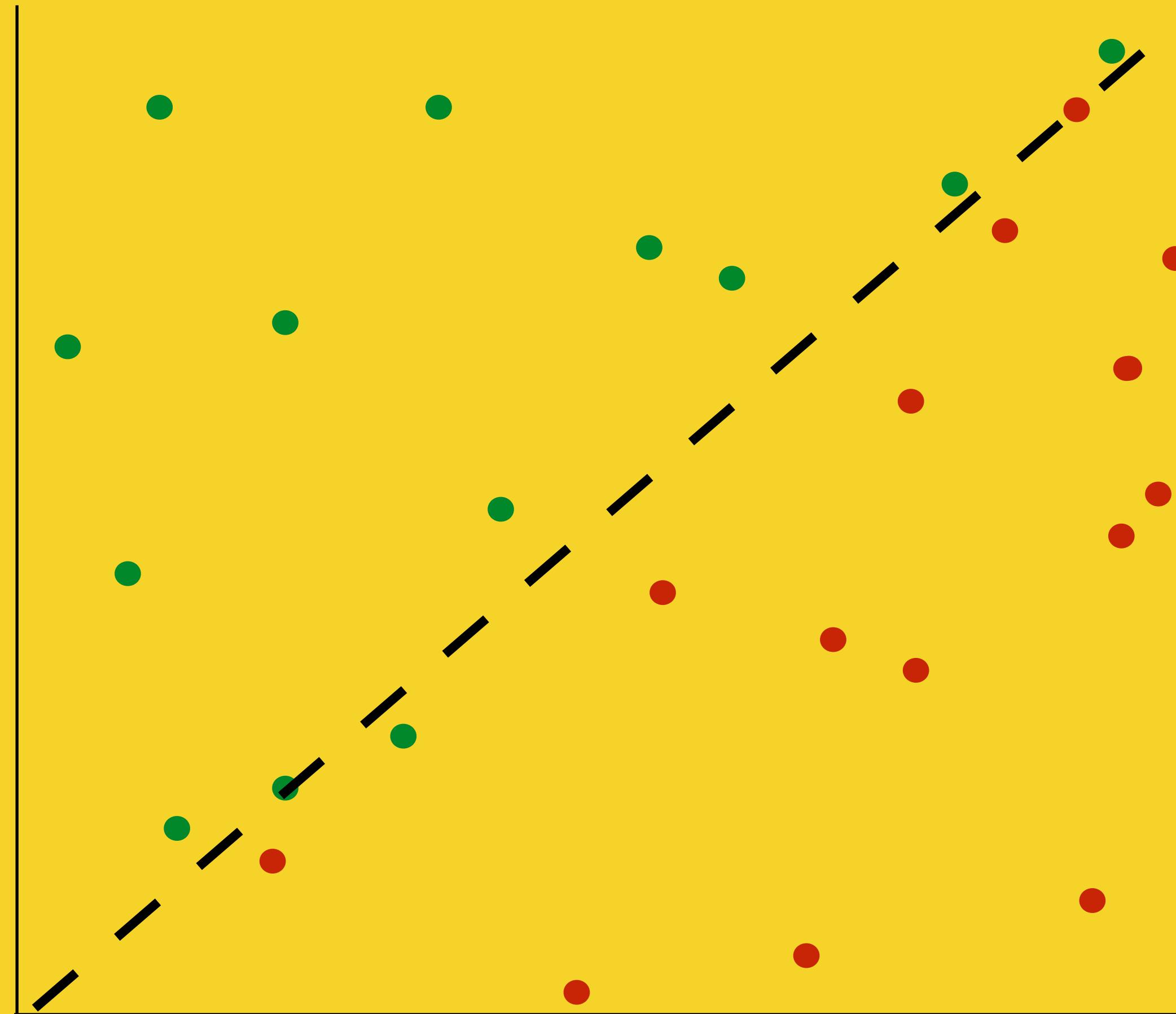
avg compiler paper	17	
Iris dataset	150	10x
MNIST dataset	60,000	10^3 x
ImageNet dataset	10,000,000	10^6 x

problem statement

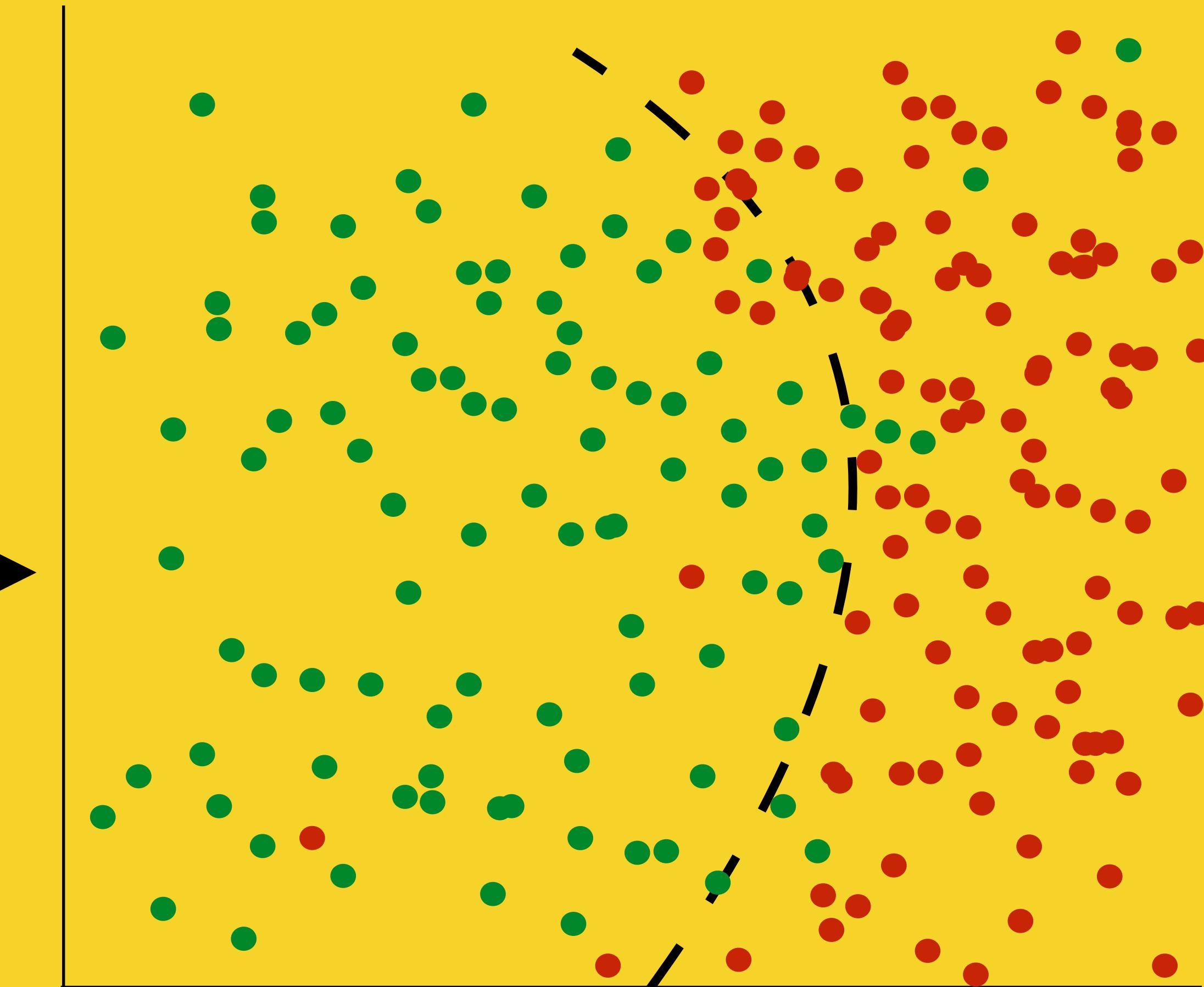
1. there aren't enough benchmarks
2. more benchmarks = better models



what we need

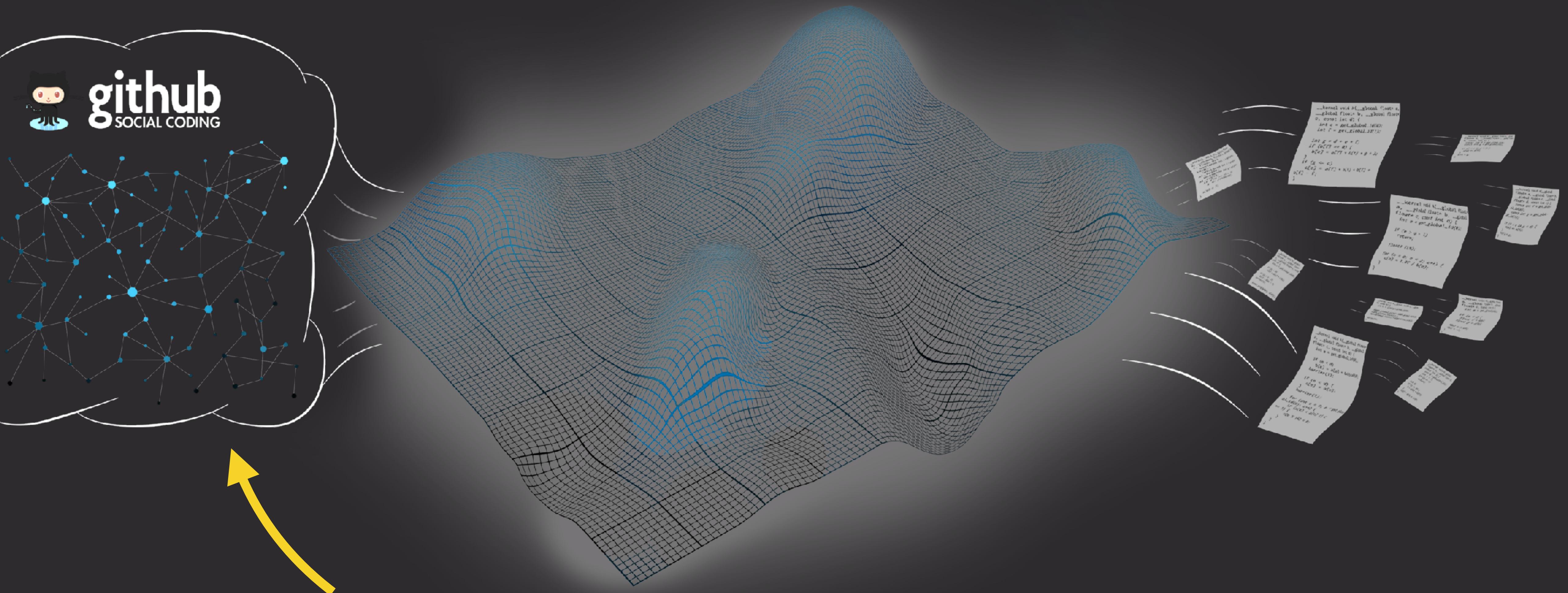


from this



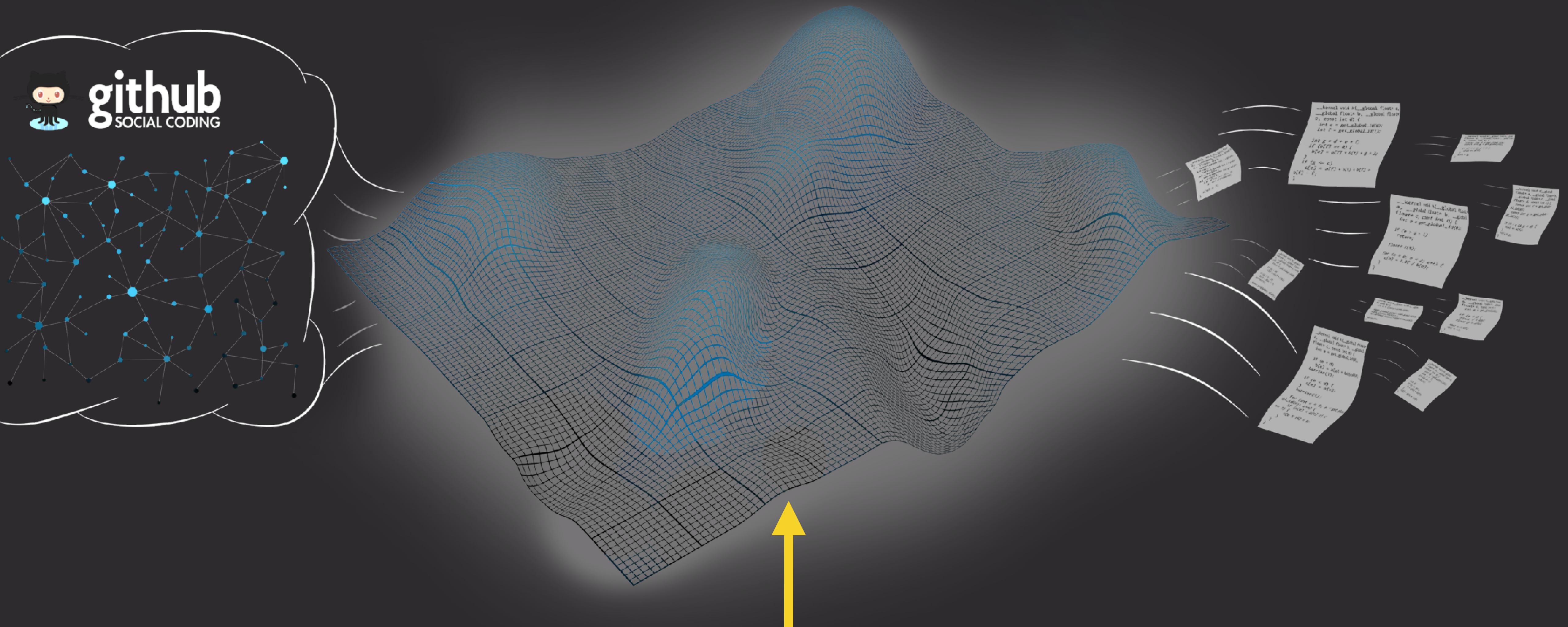
to this

our approach



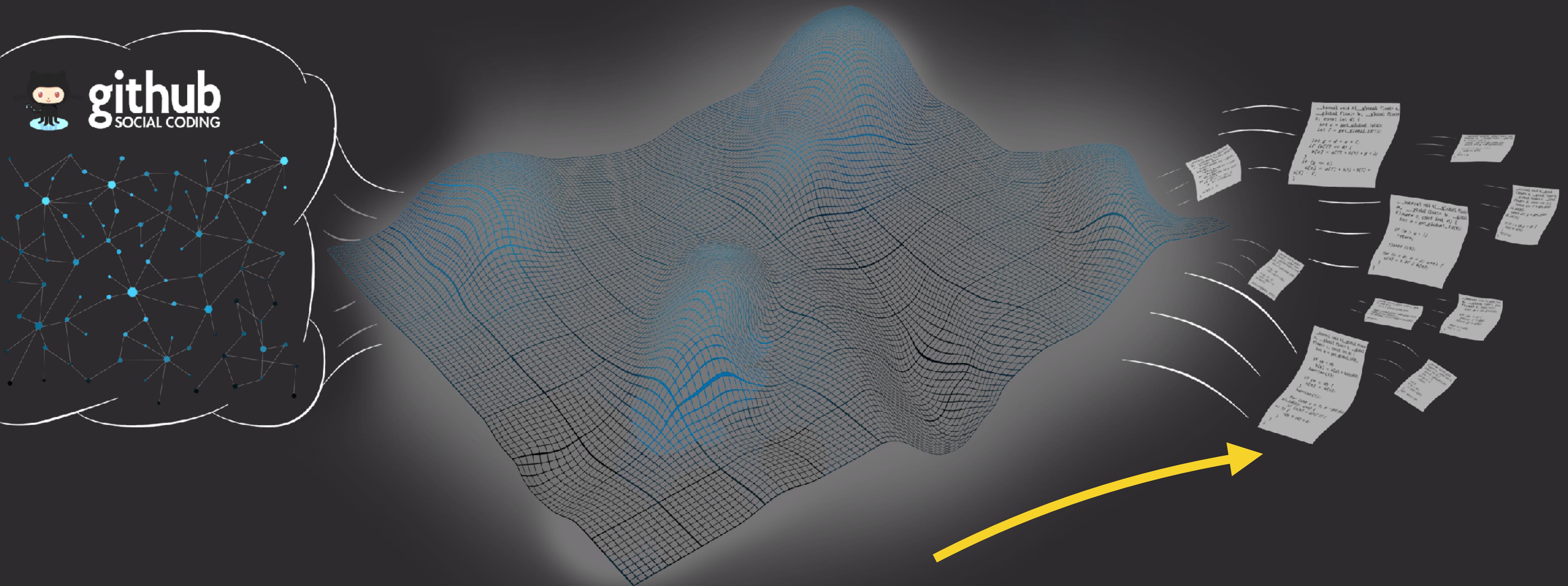
mine code from web

our approach



model source distr.

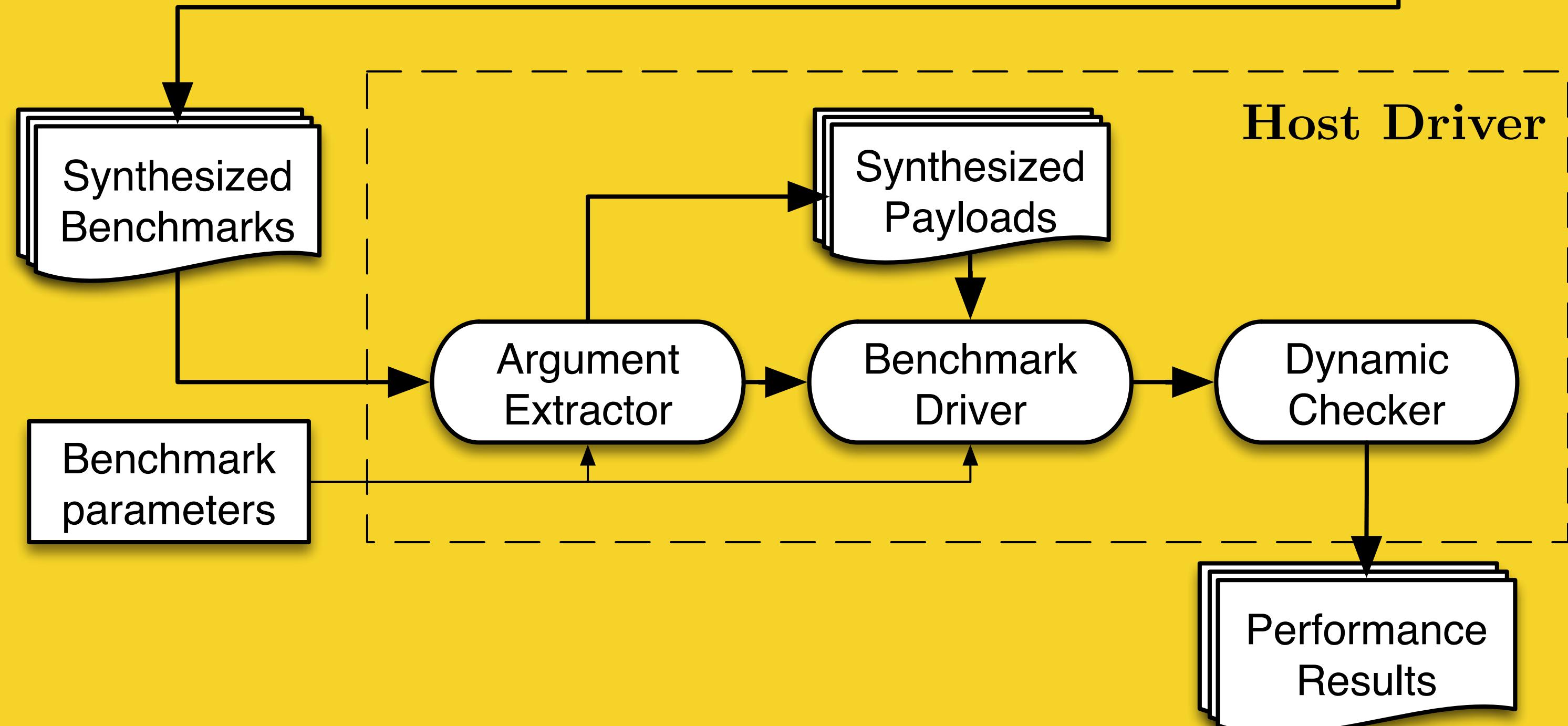
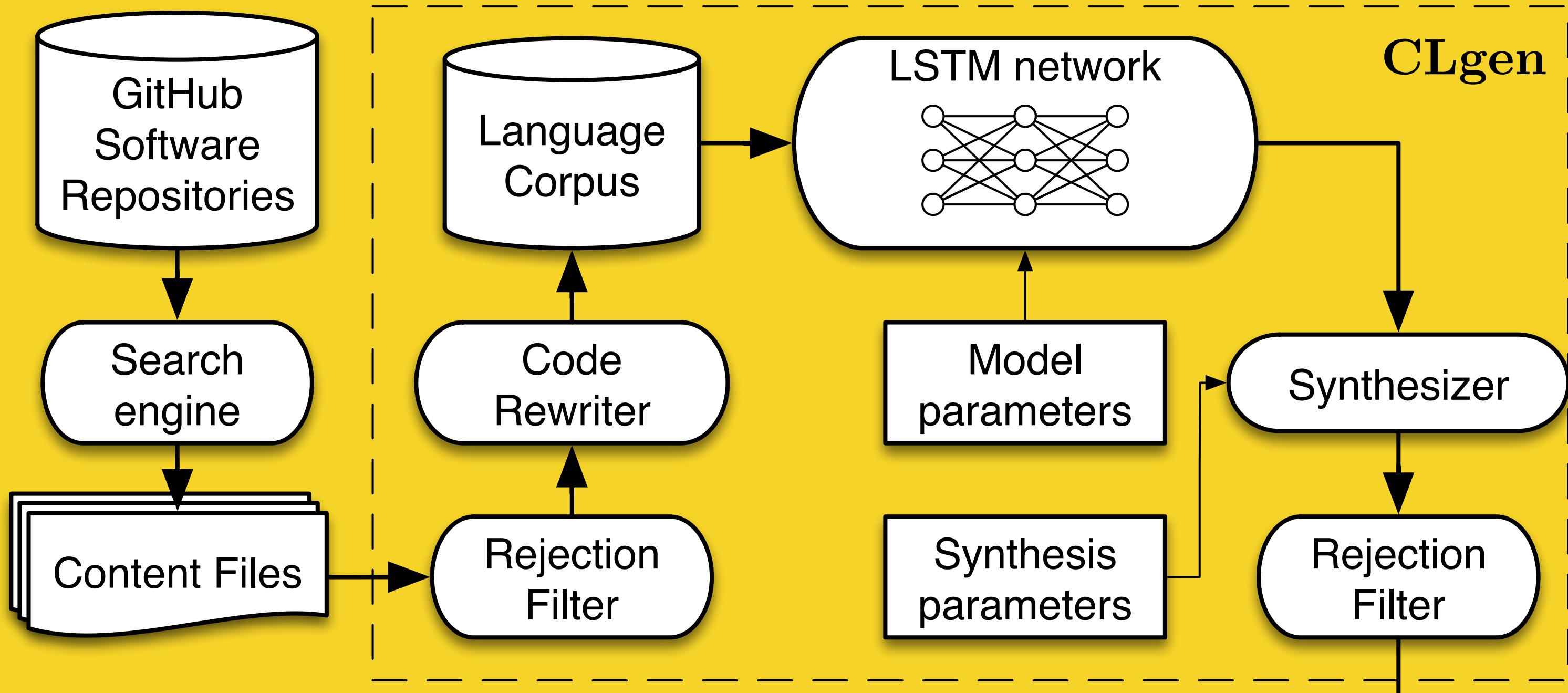
our approach



sample lang. model

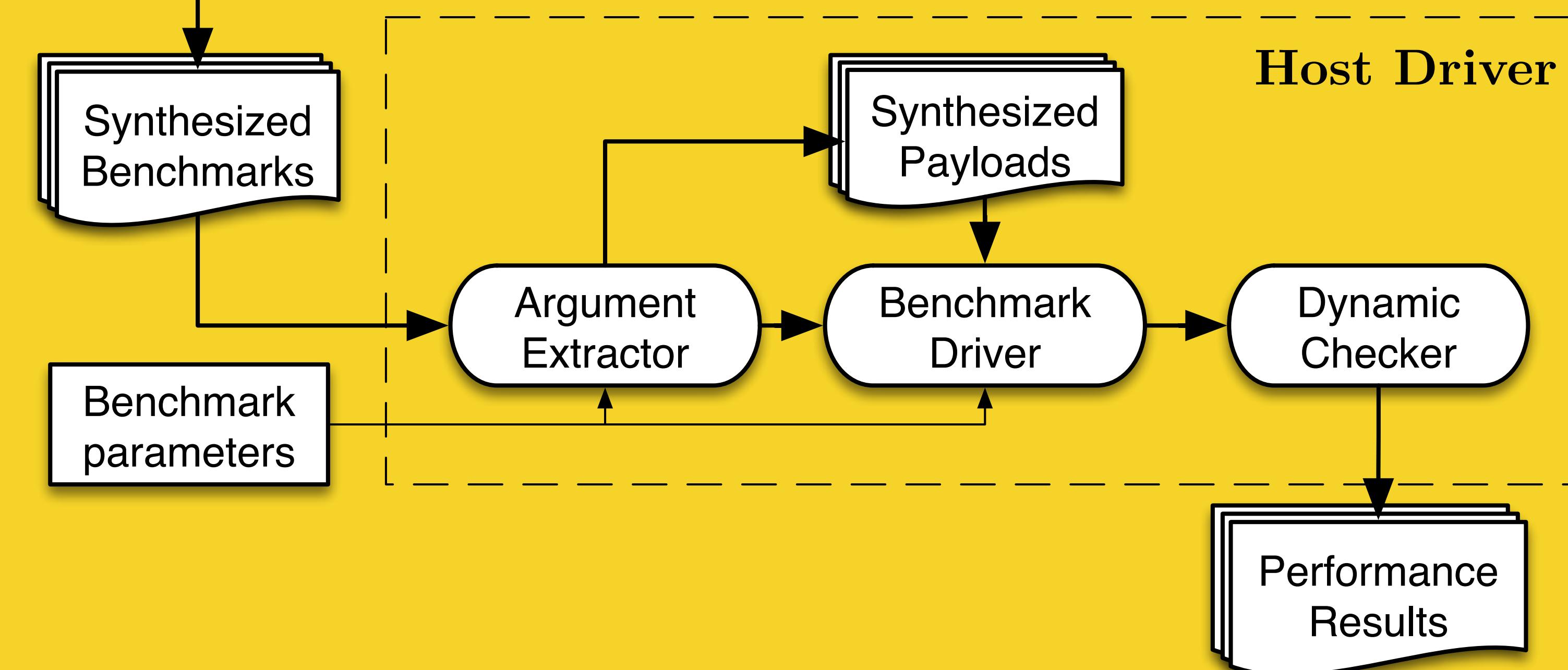
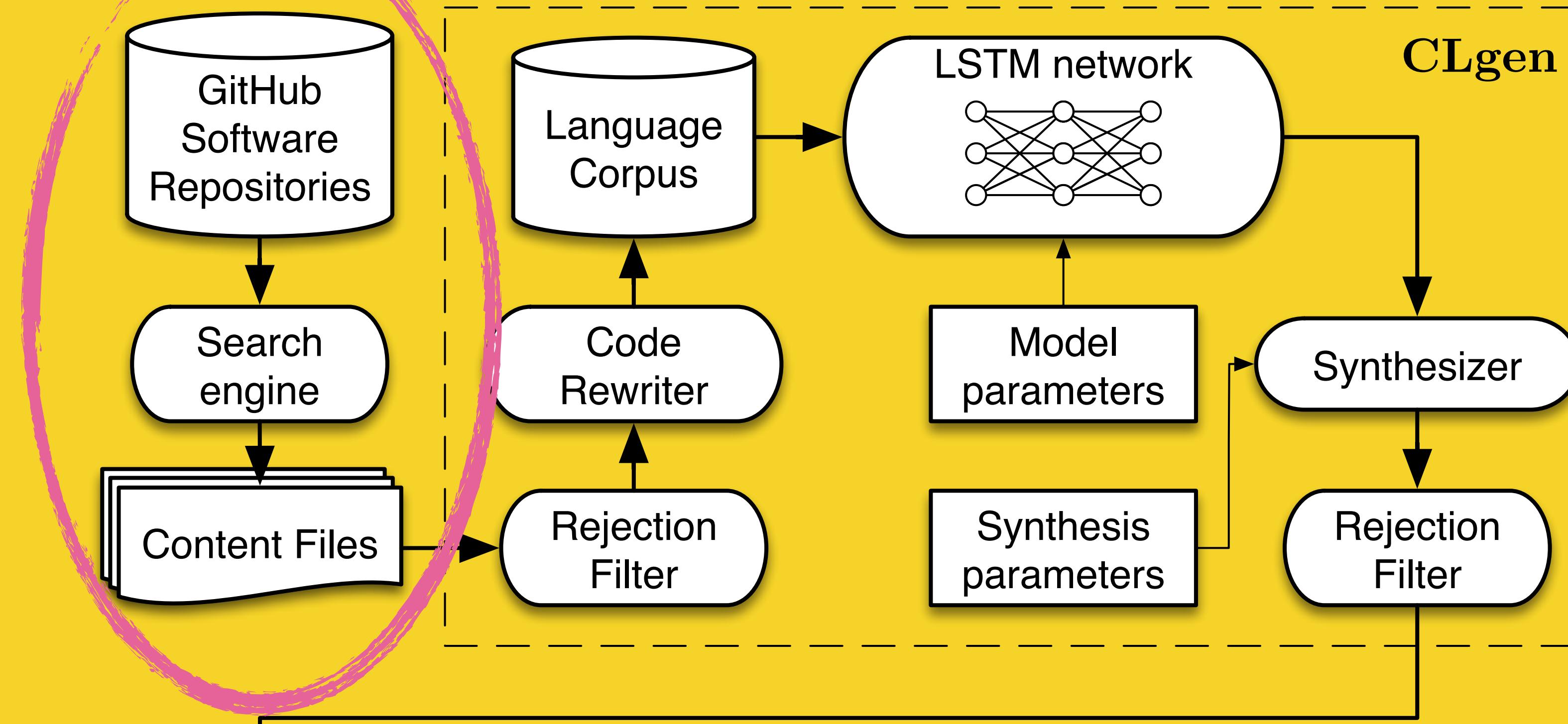
contributions

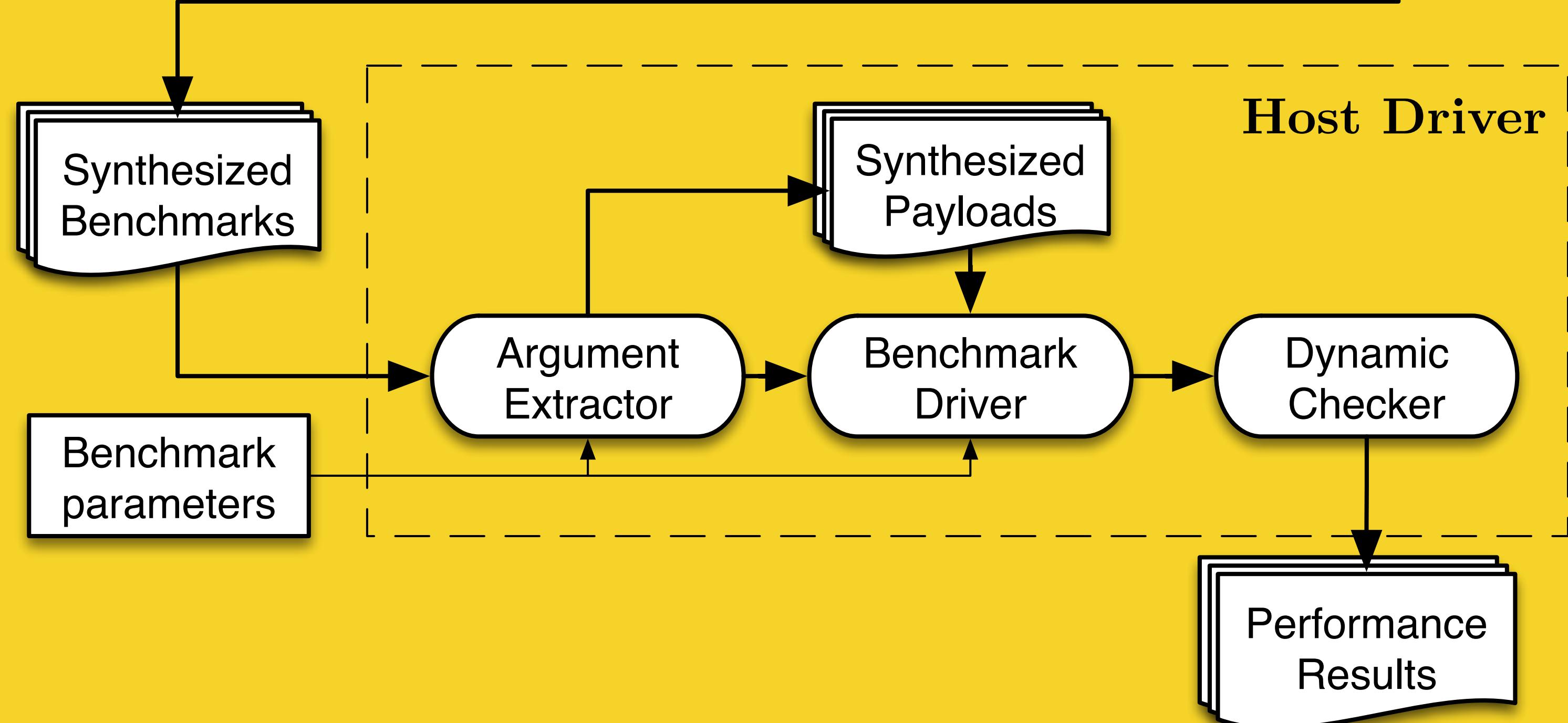
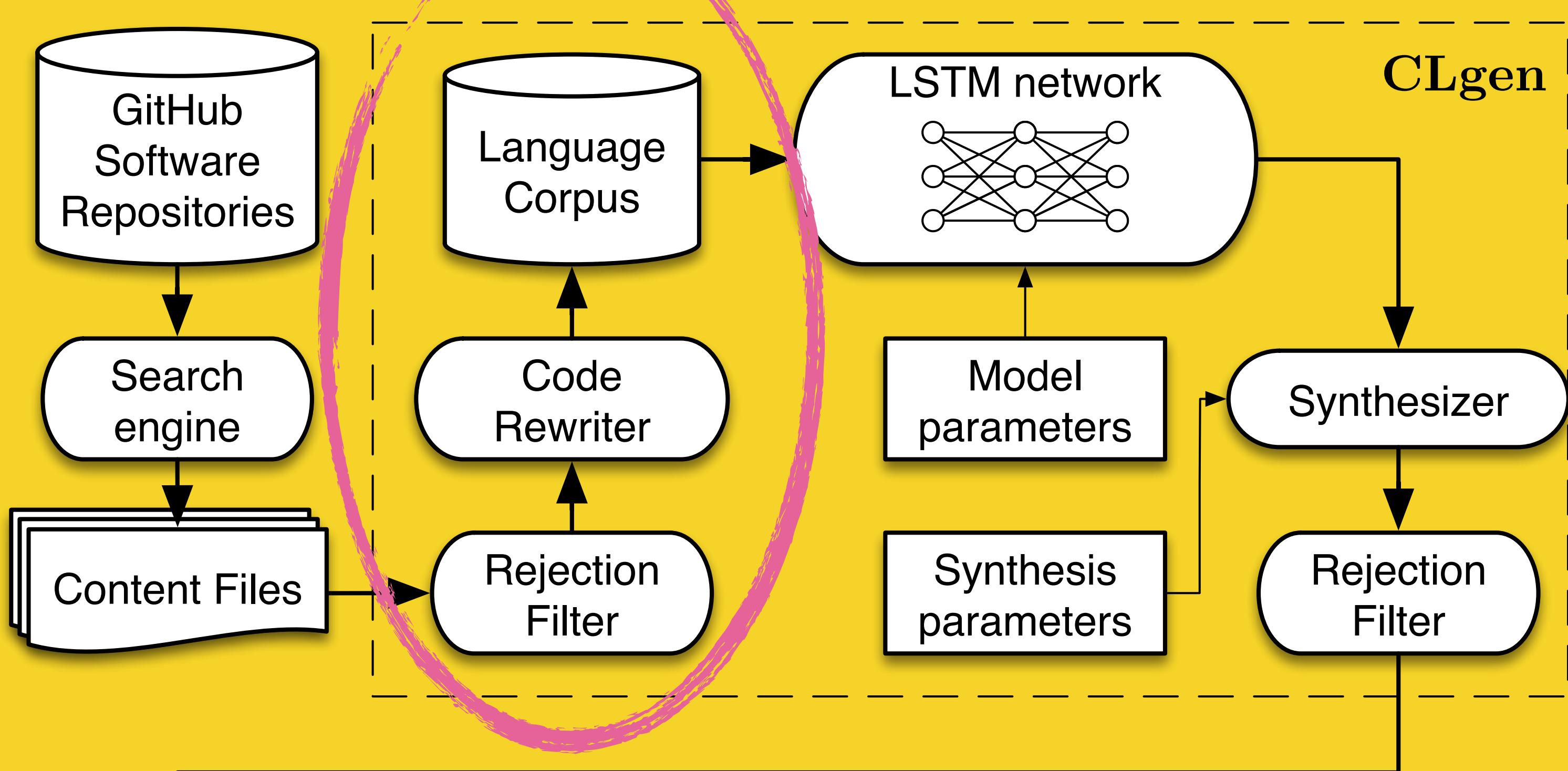
- 1. a deep learning approach to modeling PL semantics & usage**
- 2. first solution for general purpose benchmark synthesis**
- 3. automatic $1.27\times$ speedup**
- 4. improved model designs for further $4.30\times$ speedup**



8078
files

2.8M
lines





```
/* 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]);
    }
}
```

Is this real, valid OpenCL?
Can we minimise non-functional variance?

~~/* Copyright (C) 2011, 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

~~Strip comments~~
Preprocess

```
#define CLAMPING
#define THRESHOLD_MAX 1.0f

float myclamp(float in) {
#ifndef CLAMPING
    return in > THRESHOLD_MAX ? THRESHOLD_MAX : in < 0.0f ? 0.0f : in;
#else
    return in;
#endif}

__kernel void findAllNodesMergedAabb(__global float* in, __global float* out,
                                    int num_elems)
{
    int id = get_global_id(0);
    if (id < num_elems)
    {

        out[id] = myclamp(in[id]);
    }
}
```

~~Strip comments~~
~~Preprocess~~

```
float myclamp(float in) {
    return in > 1.0f ? 1.0f : in < 0.0f ? 0.0f : in;
}
```

```
__kernel void findAllNodesMergedAabb(__global float* in, __global float* out,
                                      int num_elems)
```

```
{  
    int id = get_global_id(0);  
    if (id < num_elems)  
    {
```

```
        out[id] = myclamp(in[id]);  
    }
```

Does it compile?
Does it contain instructions?

~~Strip comments
Preprocess~~

```
float myclamp(float in) {  
    return in > 1.0f ? 1.0f : in < 0.0f ? 0.0f : in;  
}
```

```
__kernel void findAllNodesMergedAabb(__global float* in, __global float* out,  
                                     int num_elems)
```

```
{  
    int id = get_global_id(0);  
    if (id < num_elems)  
    {  
  
        out[id] = myclamp(in[id]);  
    }  
}
```

~~Does it compile?~~

~~Does it contain instructions?~~

(33% discard rate)

```
float myclamp(float in) {  
    return in > 1.0f ? 1.0f : in < 0.0f ? 0.0f : in;  
}
```

```
__kernel void findAllNodesMergedAabb(_global float* in, _global float* out,  
                                         int num_elems)
```

```
{  
    int id = get_global_id(0);  
    if (id < num_elems)  
    {
```

```
        out[id] = myclamp(in[id]);  
    }
```

~~Strip comments
Preprocess
Rewrite function names~~

~~Does it compile?~~

~~Does it contain instructions?~~

(33% discard rate)

```
float A(float in) {  
    return in > 1.0f ? 1.0f : in < 0.0f ? 0.0f : in;  
}  
  
__kernel void B(__global float* in, __global float* out,  
                int num_elems)  
{  
    int id = get_global_id(0);  
    if (id < num_elems)  
    {  
        out[id] = A(in[id]);  
    }  
}
```

Strip comments
Preprocess
Rewrite function names
Rewrite variable names

Does it compile?
Does it contain instructions?
(33% discard rate)

```
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)  
{  
↳ X c[e] = A(b[e]);  
}  
}
```

~~Strip comments~~
~~Preprocess~~
~~Rewrite function names~~
~~Rewrite variable names~~
~~Enforce code style~~

~~Does it compile?~~

~~Does it contain instructions?~~

(33% discard rate)

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

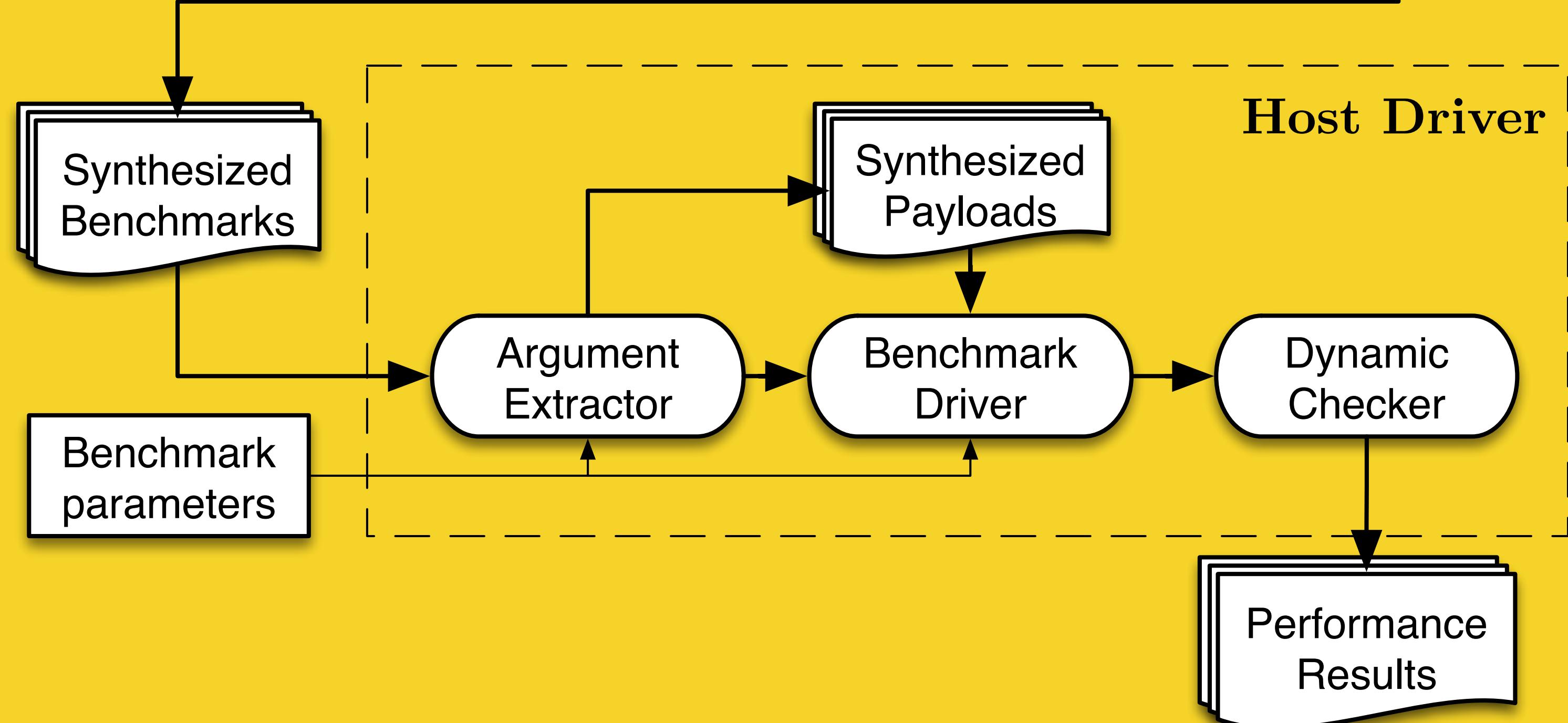
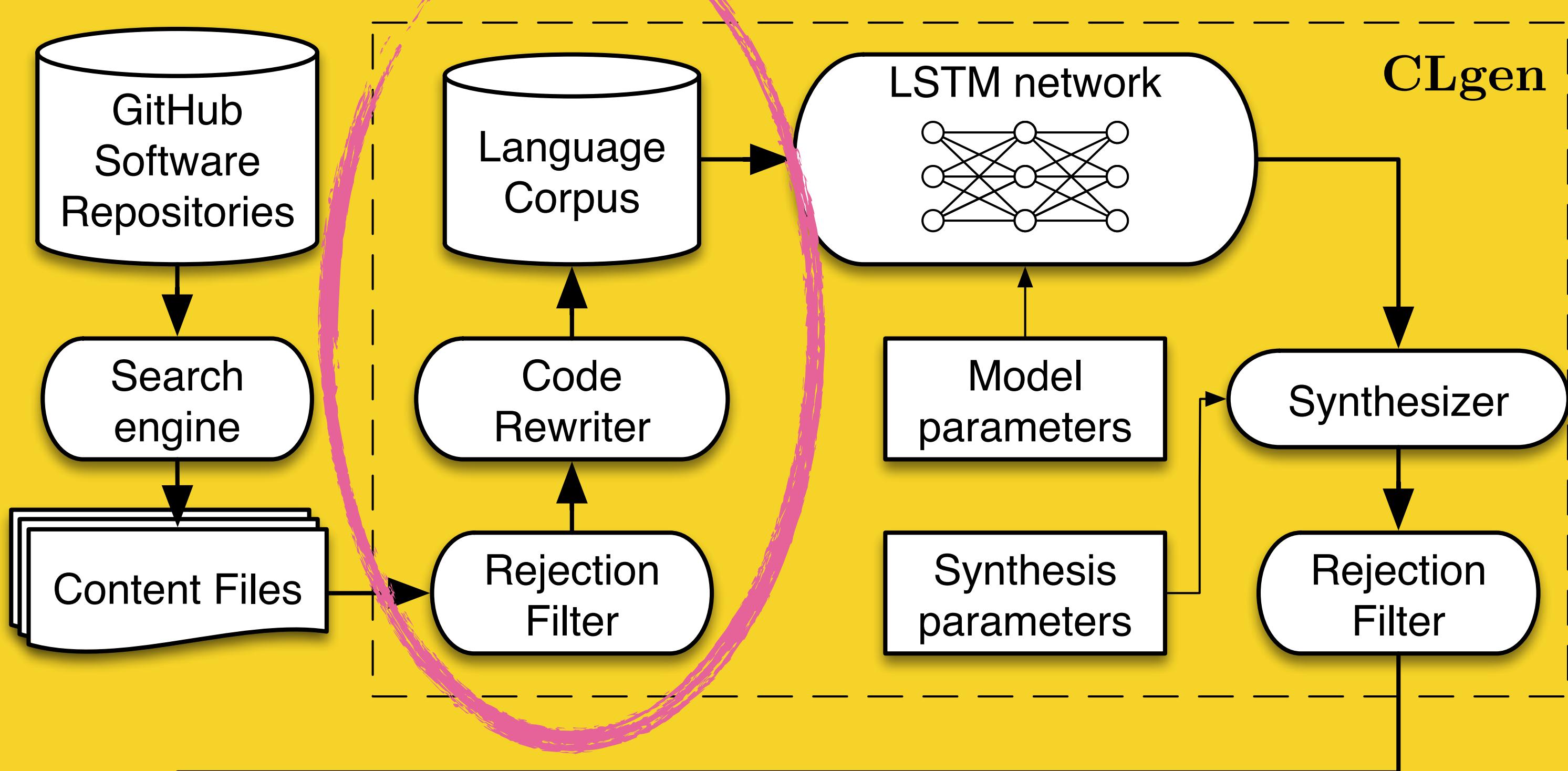
```
__kernel void B(__global float* b, __global float* c, int d) {  
    int e = get_global_id(0);  
    if (e < d) {  
        c[e] = A(b[e]);  
    }  
}
```

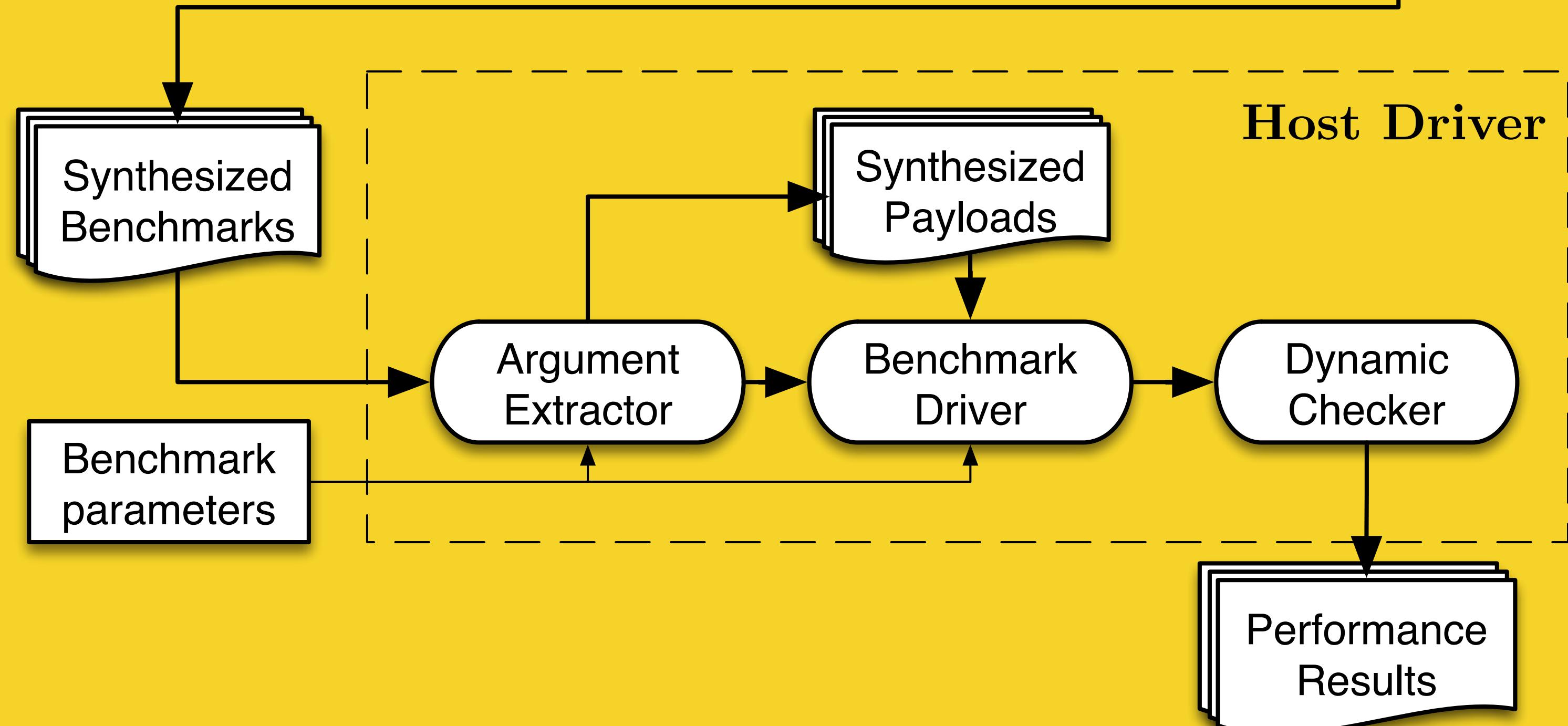
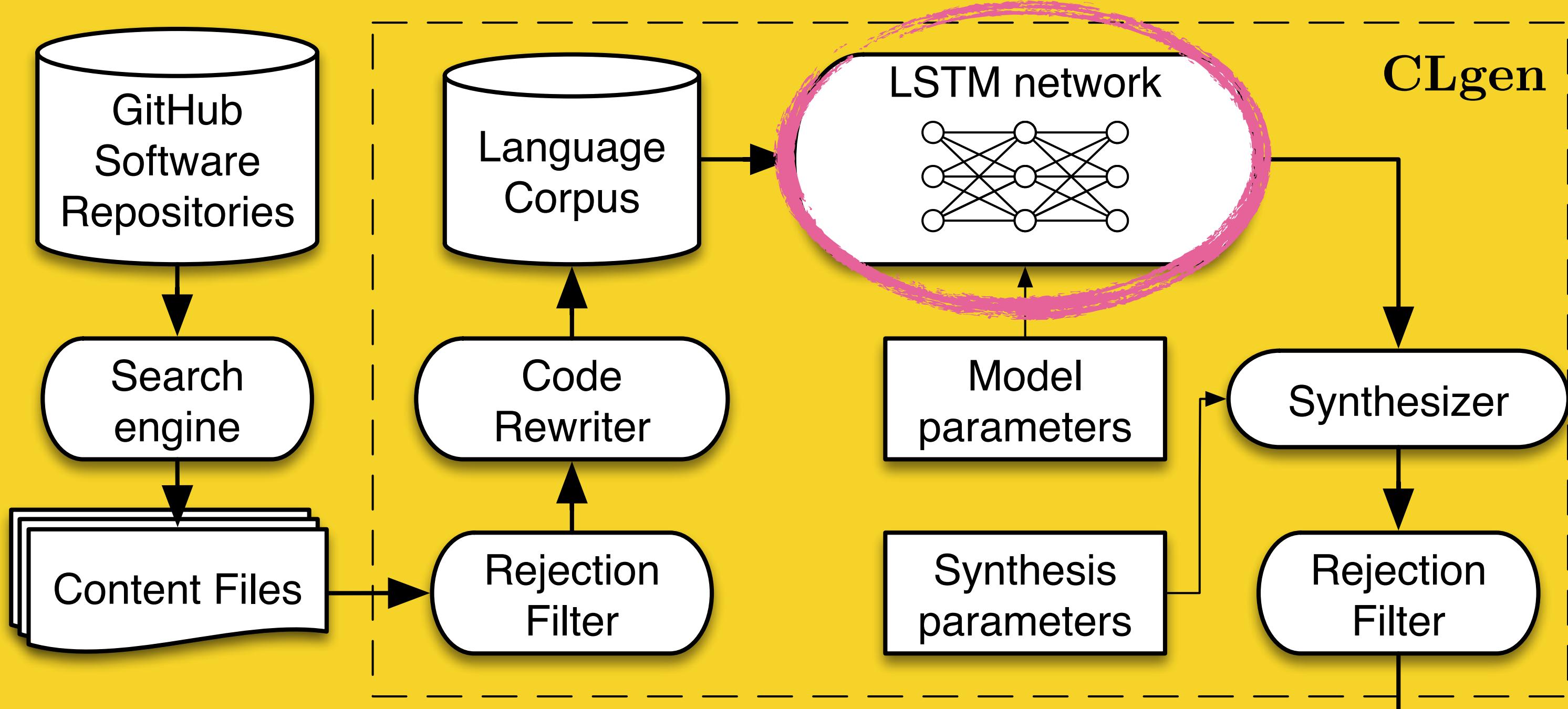
~~Strip comments~~
~~Preprocess~~
~~Rewrite function names~~
~~Rewrite variable names~~
~~Enforce code style~~

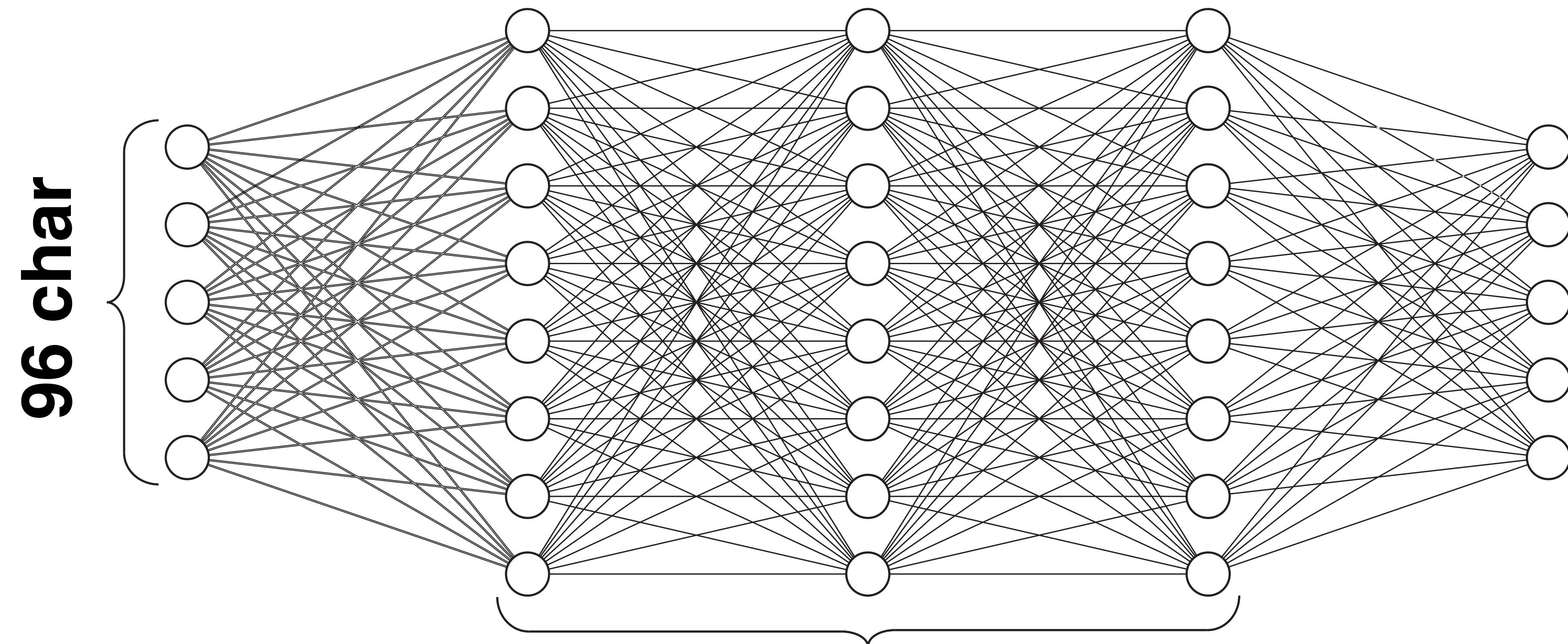
~~Does it compile?~~

~~Does it contain instructions?~~

(33% discard rate)





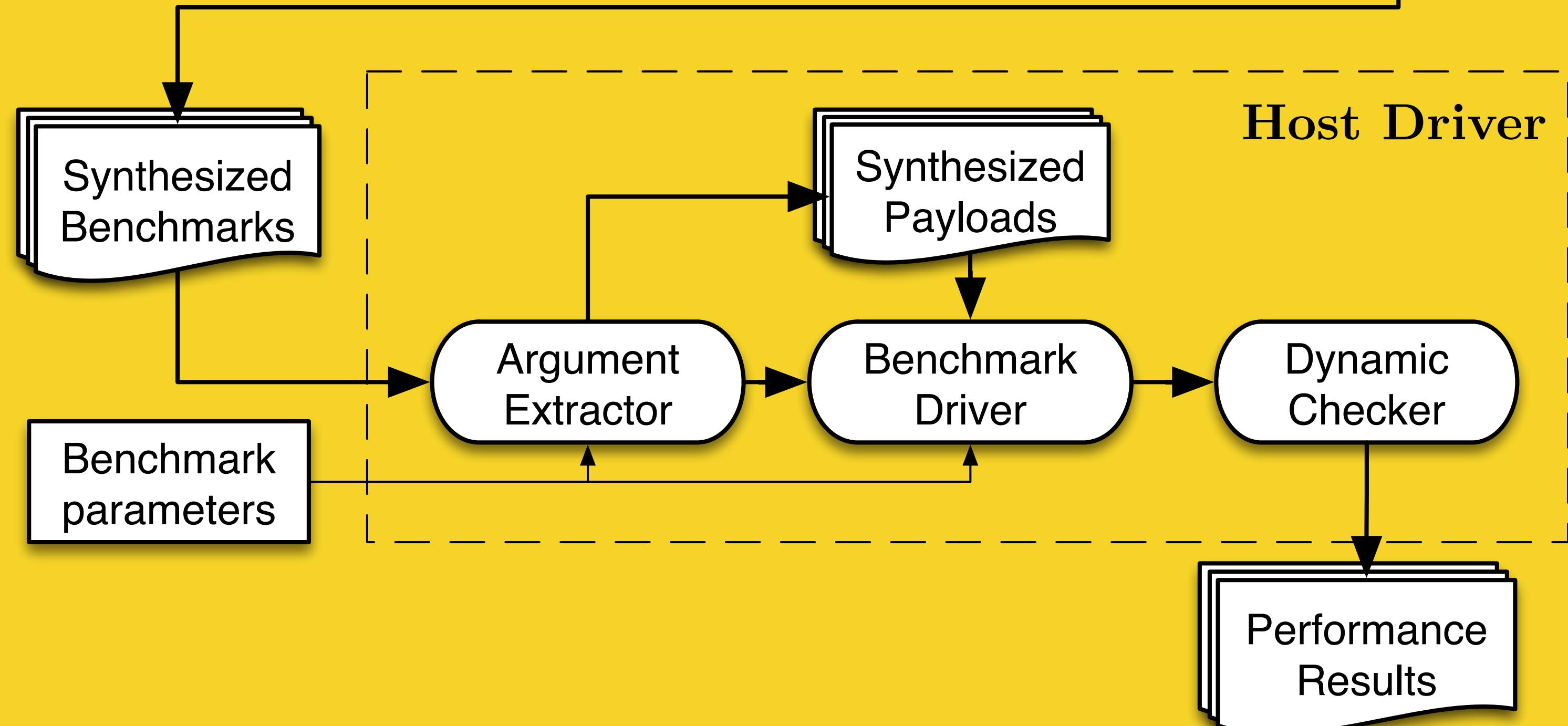
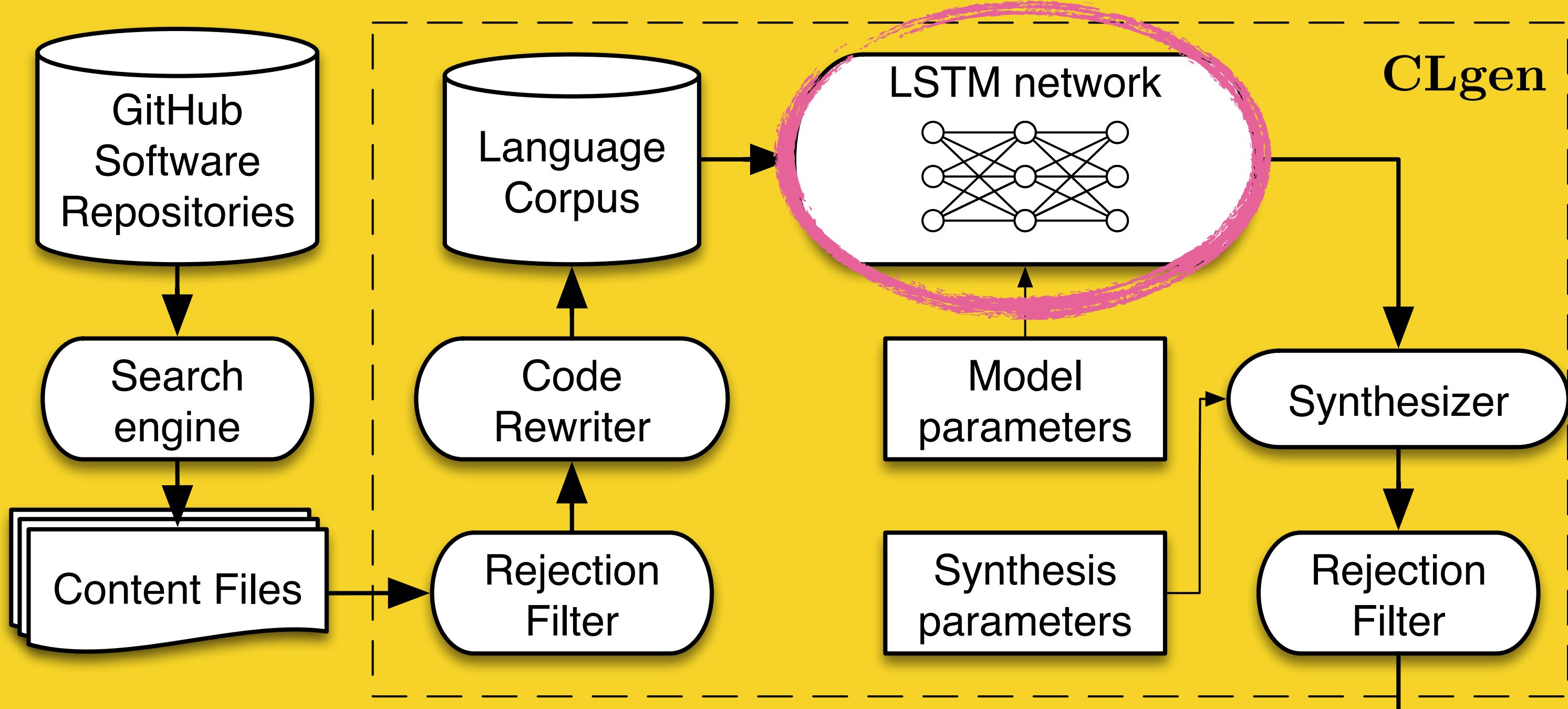


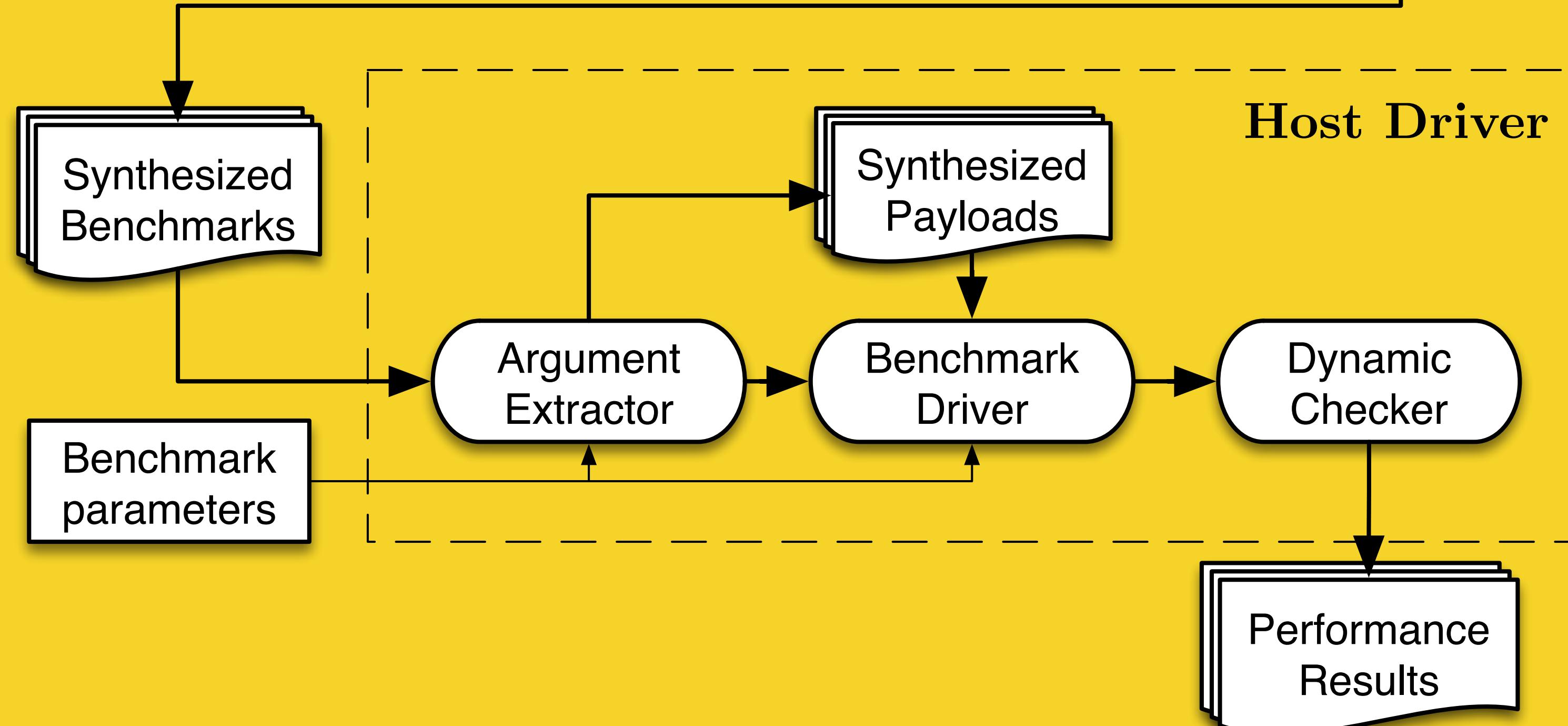
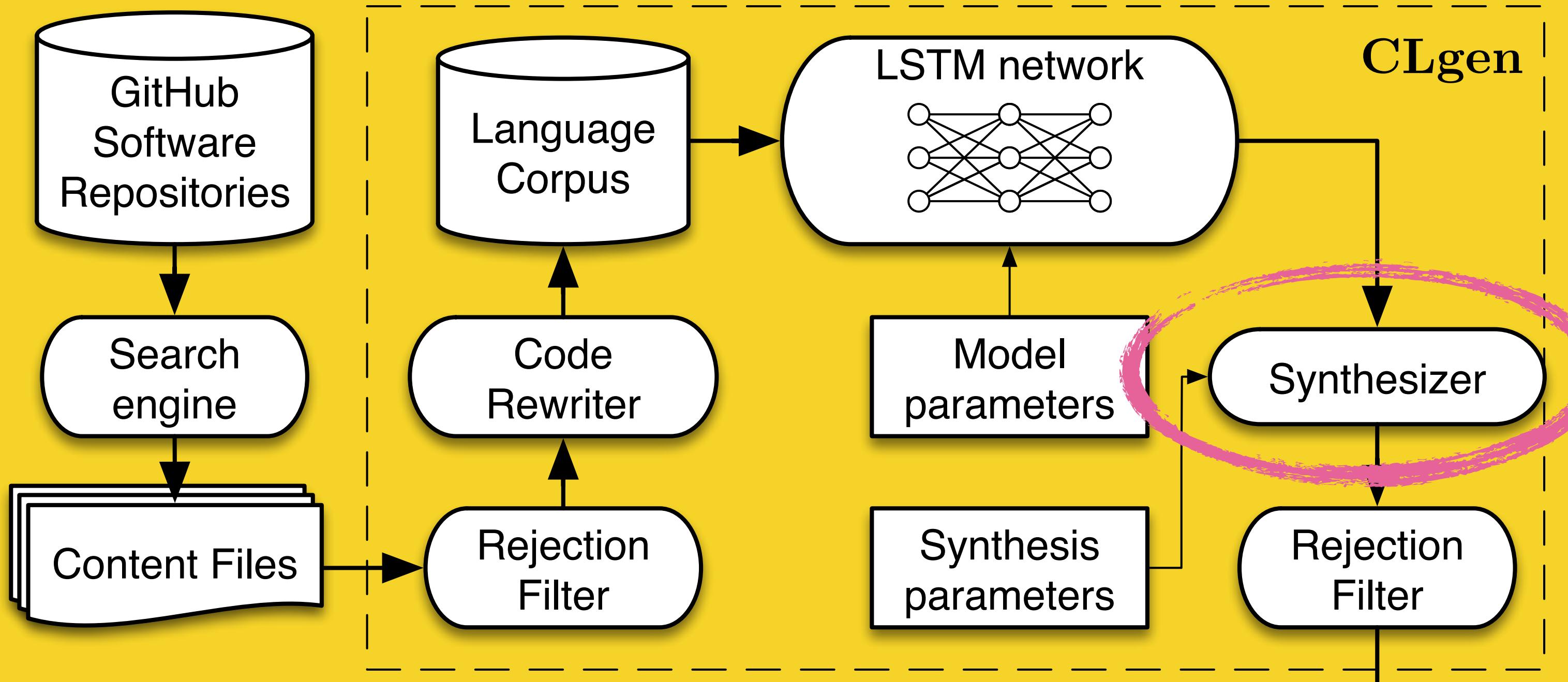
2048 node, 3 layer LSTM

character level language model

$$y = f(x)$$

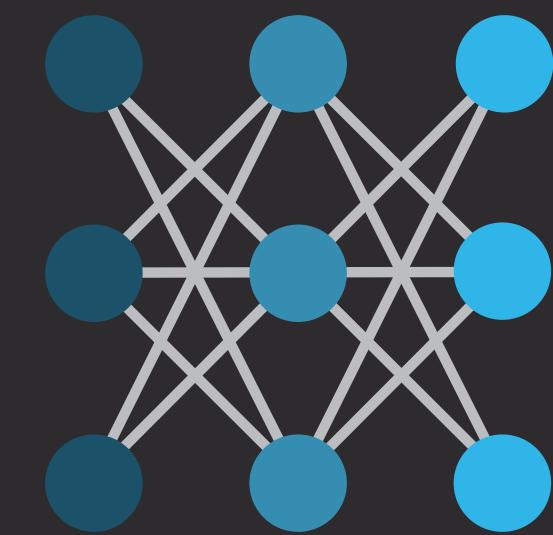
next char char array





kernel synthesis

```
string S = '__kernel void A(__global float* a) {'  
int depth = 1  
while depth > 0:  
    char c = predict_next_character(S)  
    if c == '{':  
        depth += 1  
    if c == '}':  
        depth -= 1  
    S += c  
  
return S
```



clgen

Deep Learning Program Generator.

```
$ clgen model.json sampler.json
```

```
{  
    "corpus": {  
        "path": "~/kernels"  
    },  
    "model_type": "lstm",  
    "rnn_size": 2048,  
    "num_layers": 3,  
    "max_epochs": 50  
}
```

```
{  
    "kernels": {  
        "args": [  
            "__global float*",  
            "__global float*",  
            "const int"  
        ]  
    },  
    "sampler": {  
        "max_kernels": 1000  
    }  
}
```

Fork me on GitHub

```
$ clgen model.json sampler.json 2>/dev/null
```



(you had to be there)

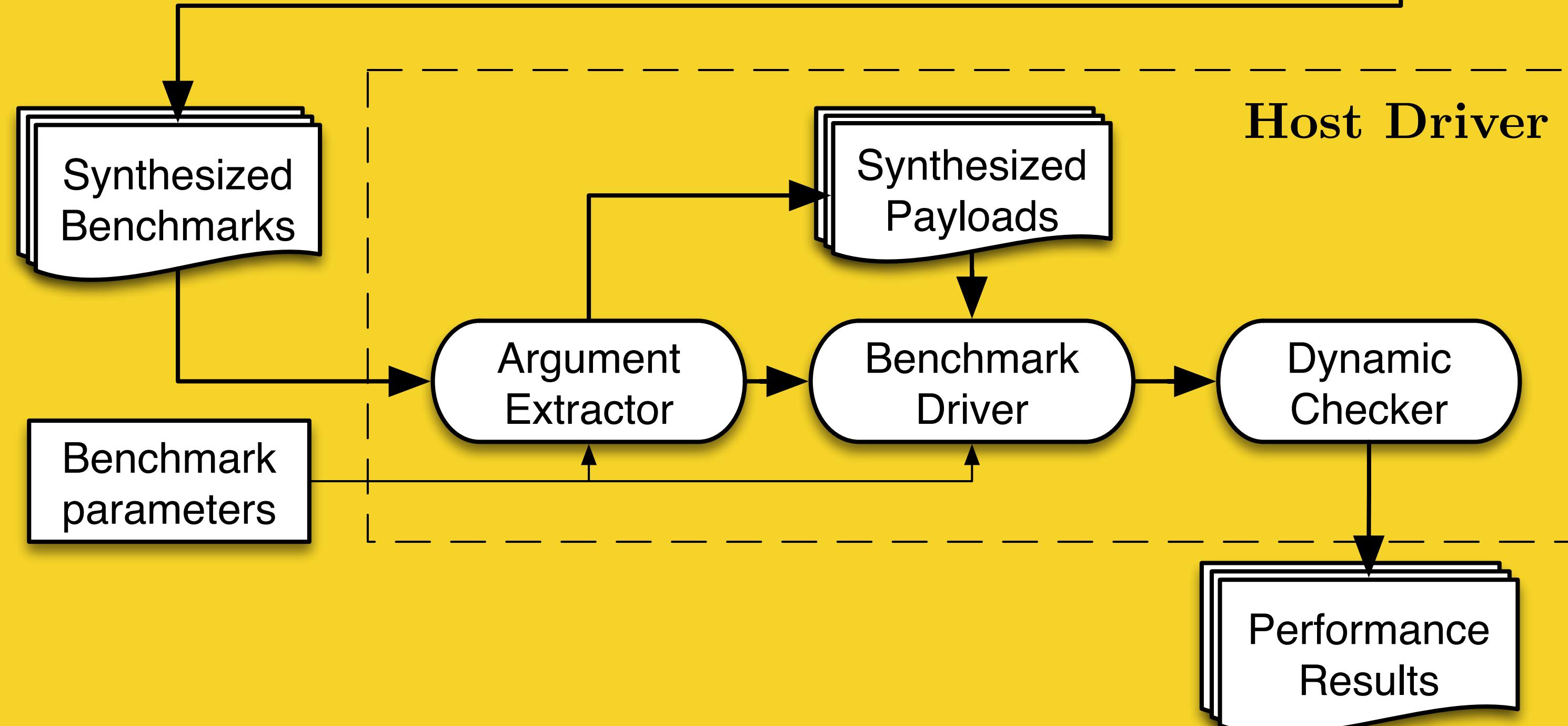
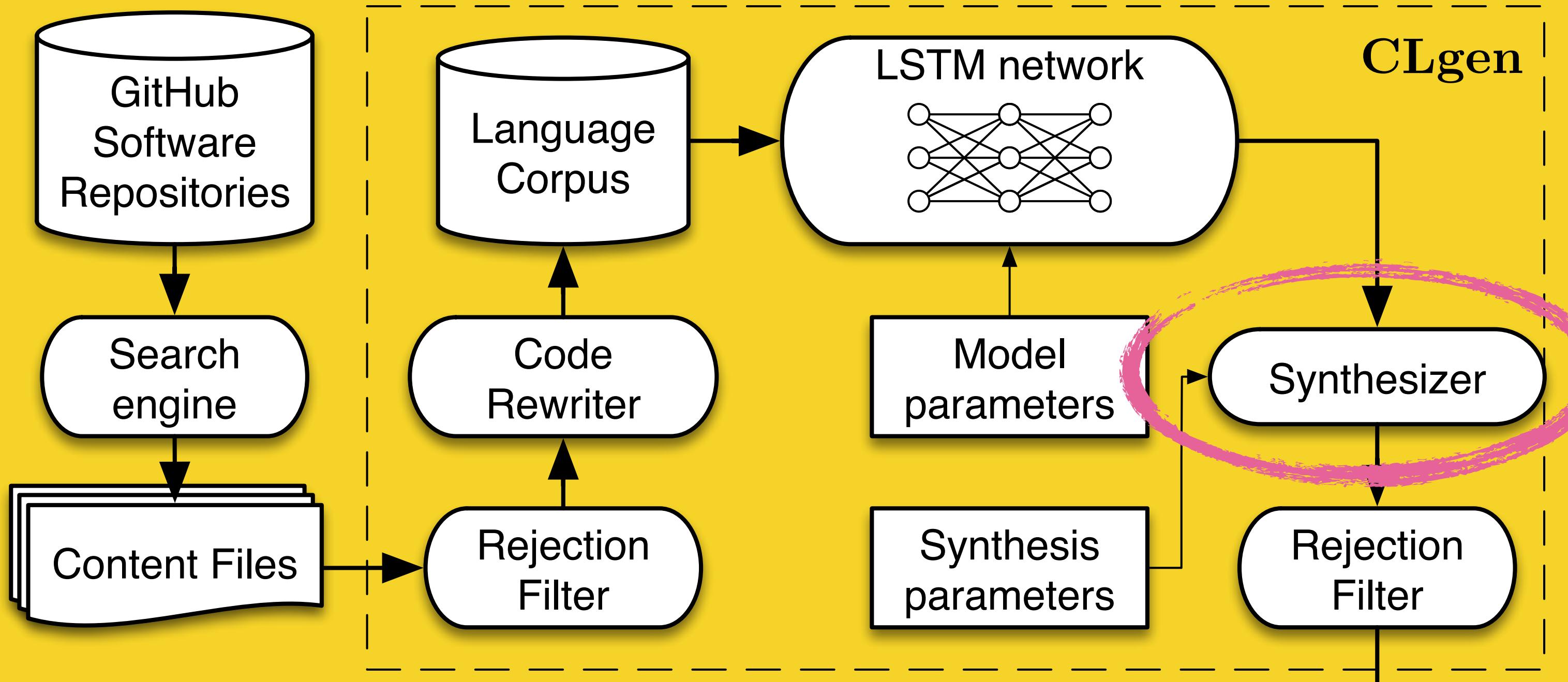
Fork me on GitHub

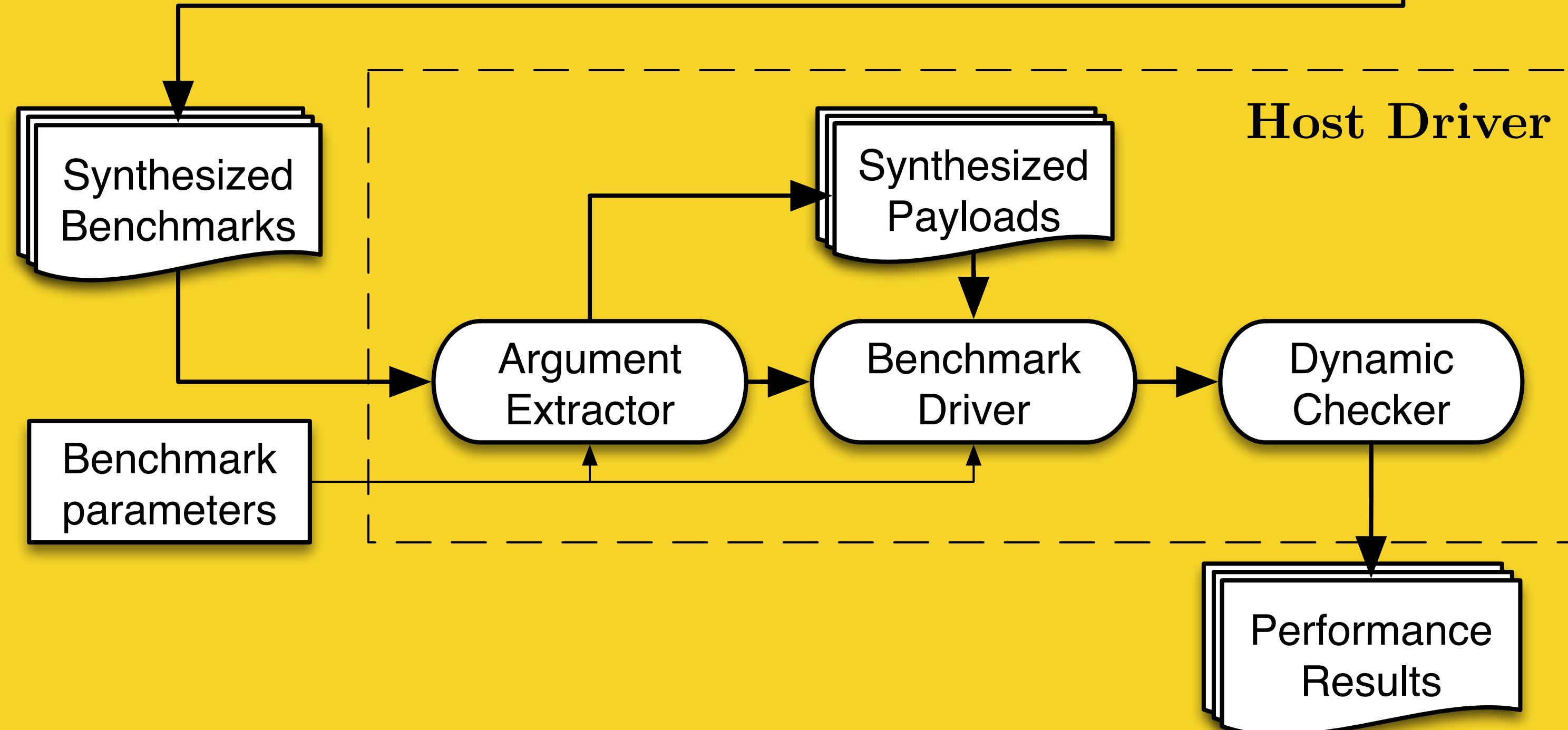
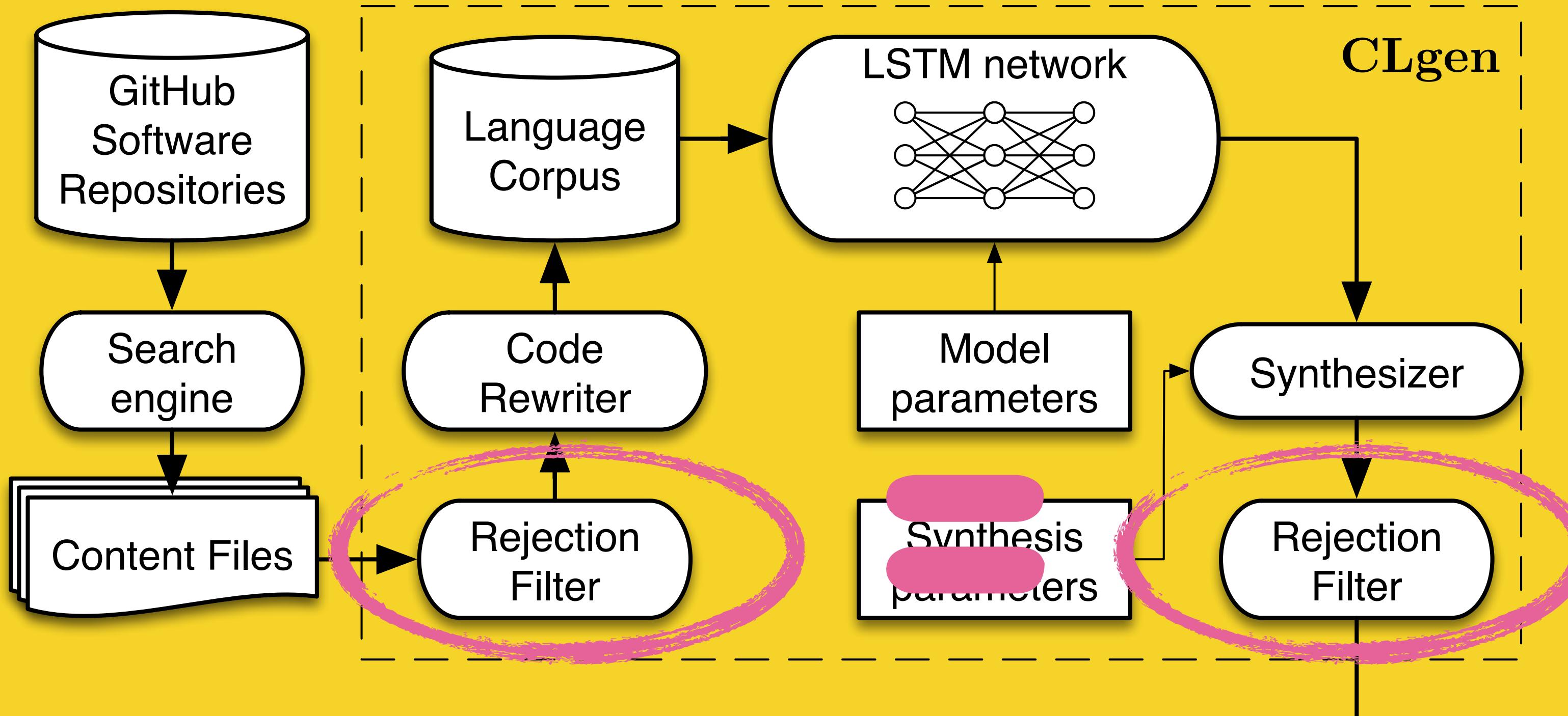
```
__kernel void A(__global float* a,
                __global float* b,
                __global float* c,
                const int d) {
    int e = get_global_id(0);
    float f = 0.0;
    for (int g = 0; g < d; g++) {
        c[g] = 0.0f;
    }
    barrier(1);

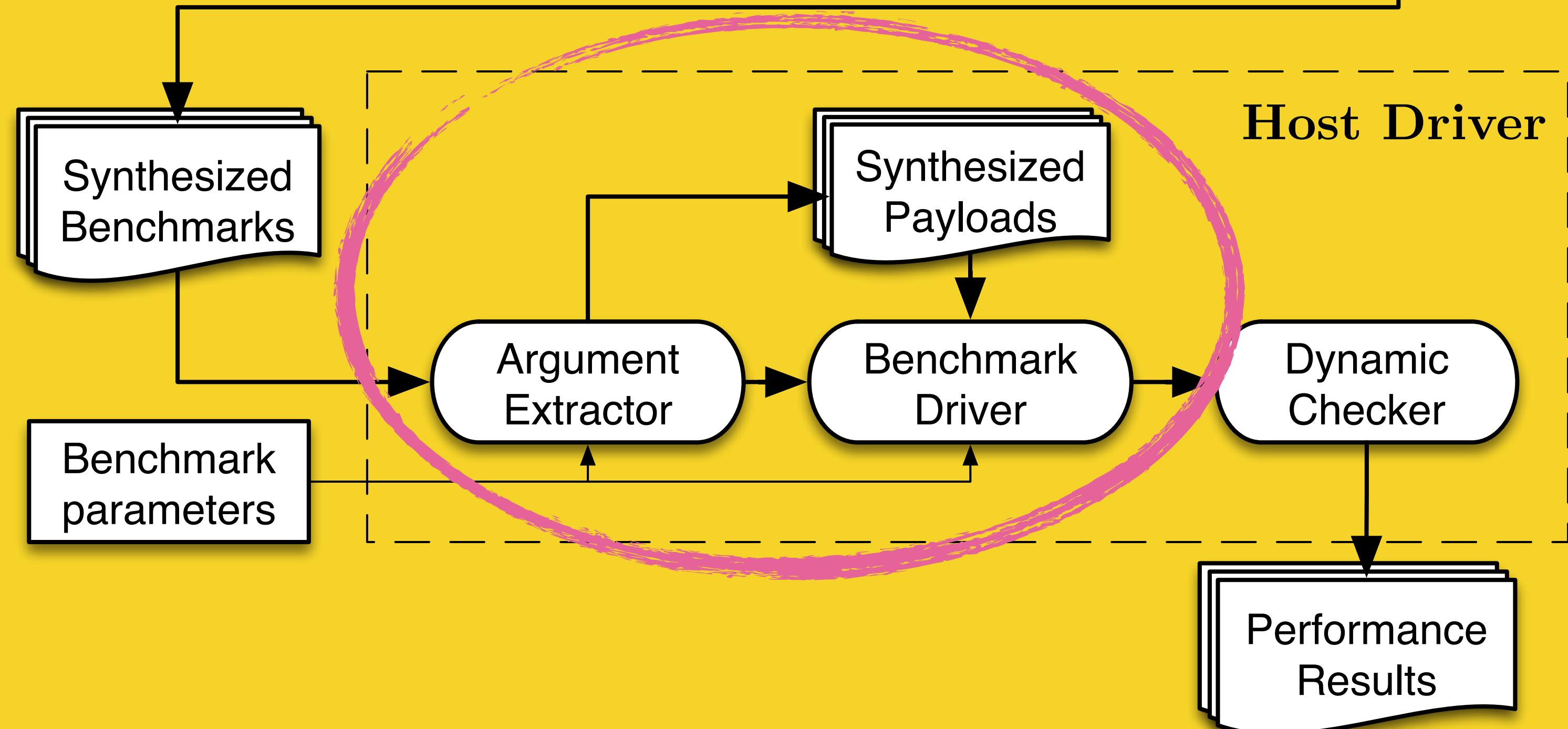
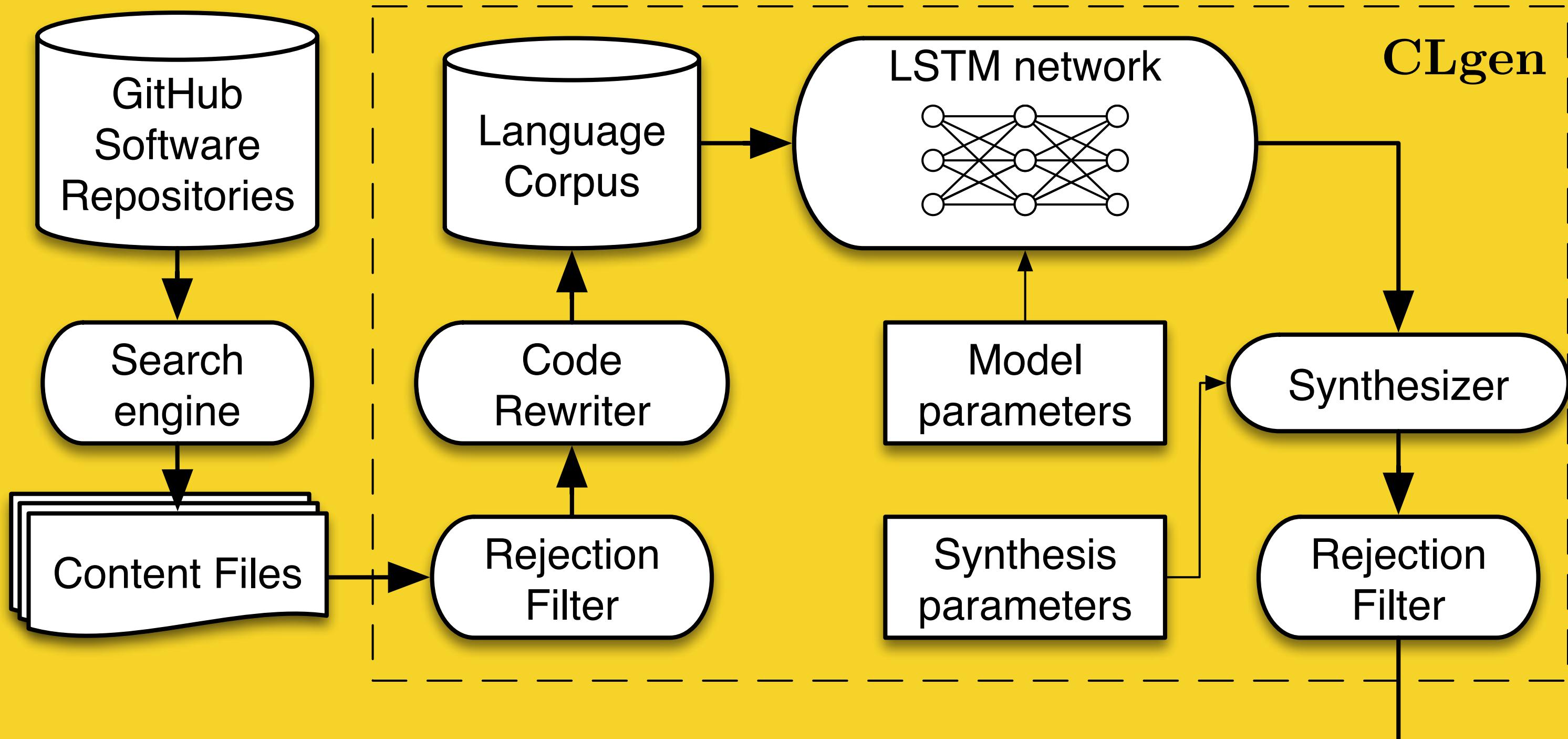
    a[get_global_id(0)] = 2 * b[get_global_id(0)];
}
```

```
__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;
}
```

```
__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;
}
```







```
__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:

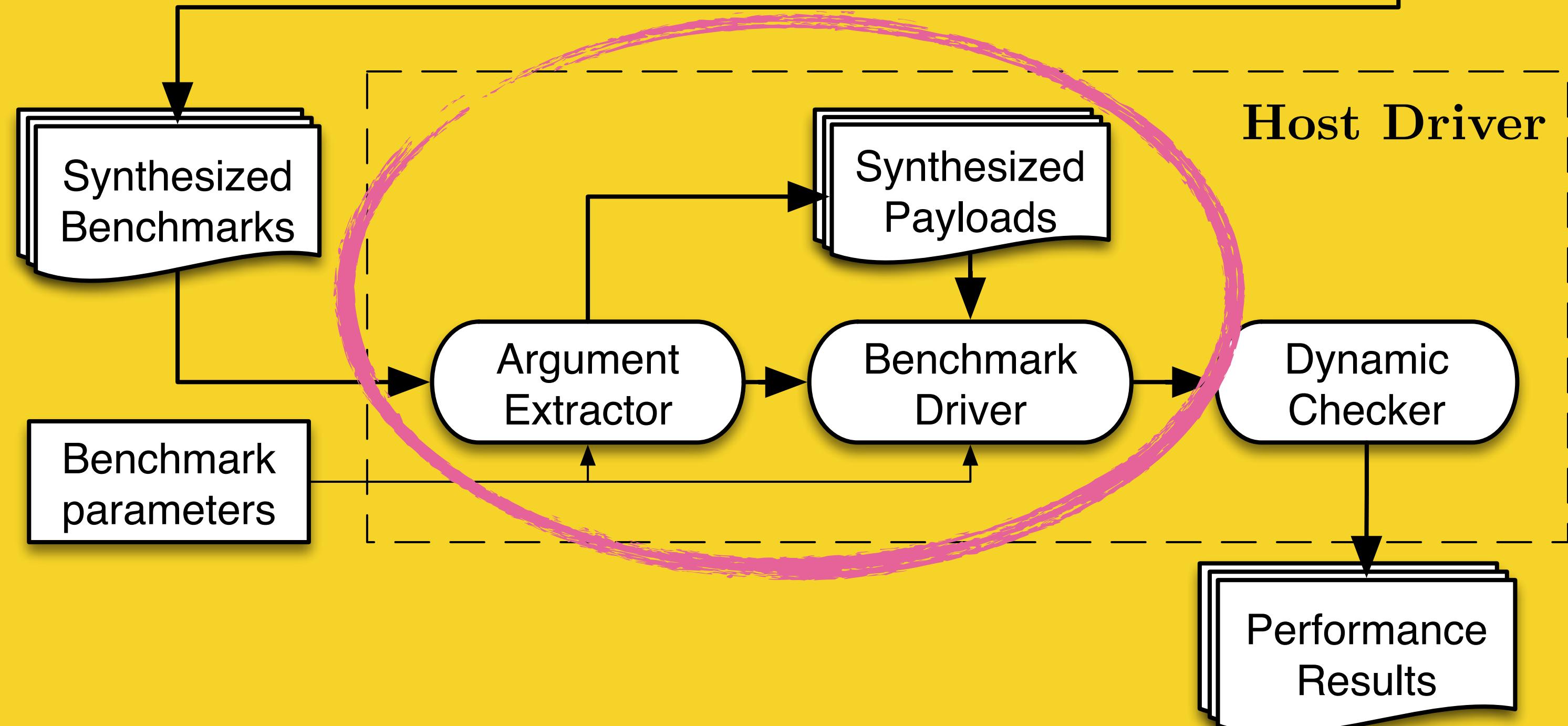
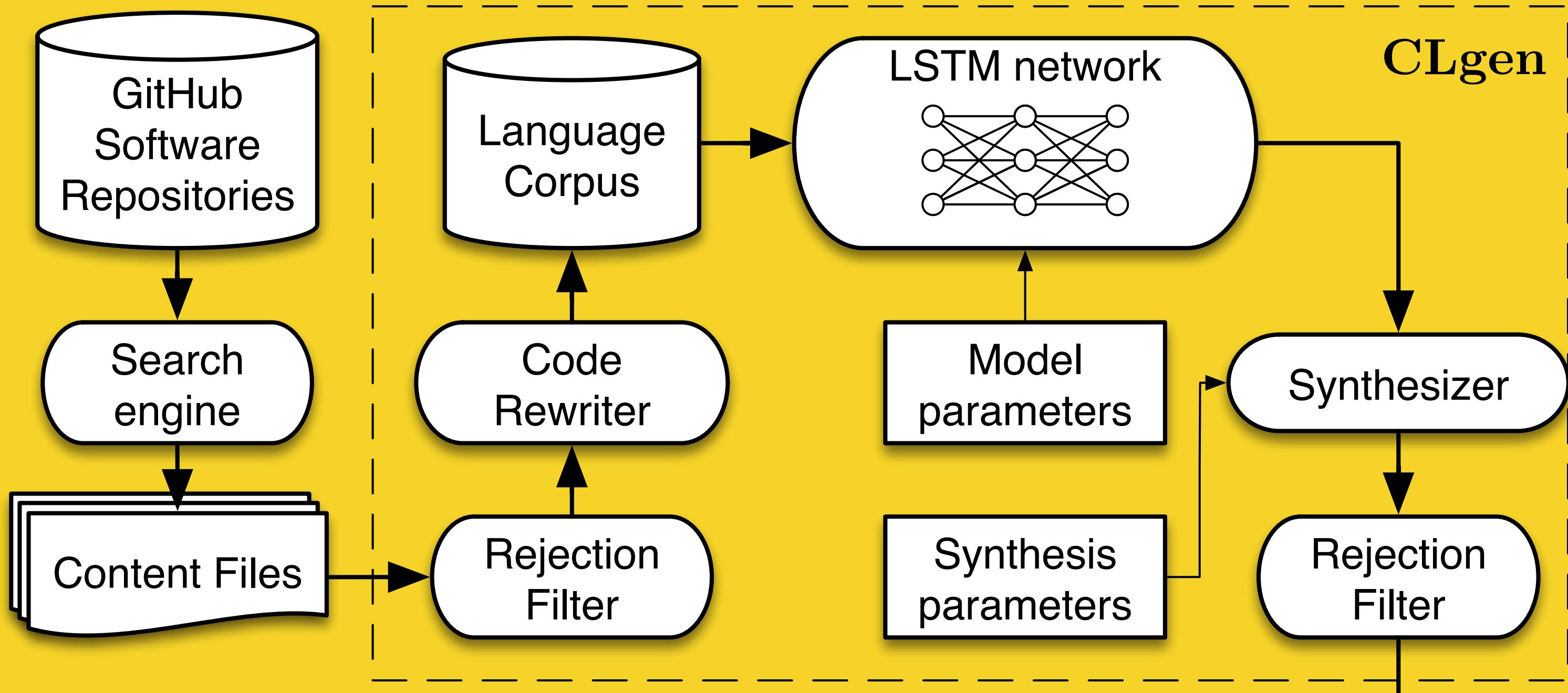
rand() * [S]

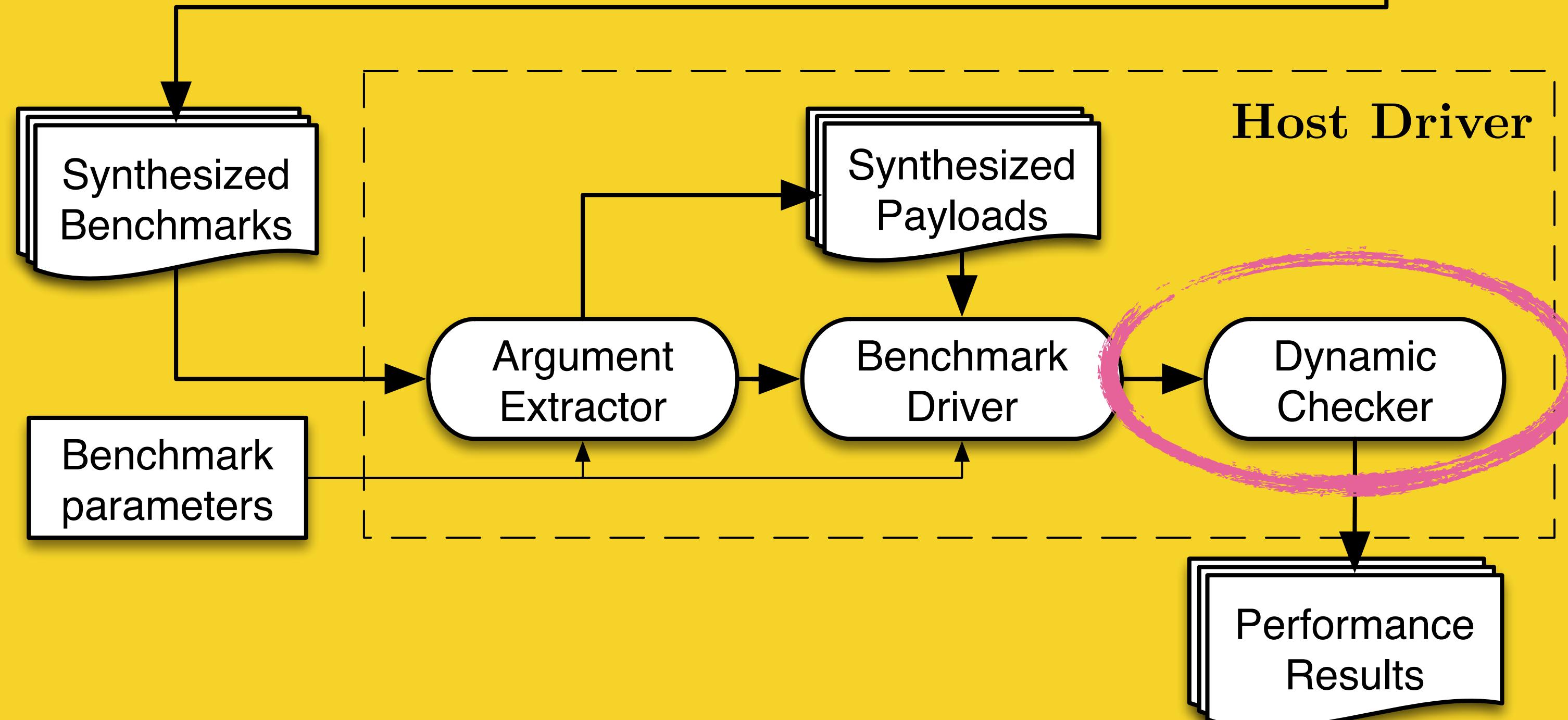
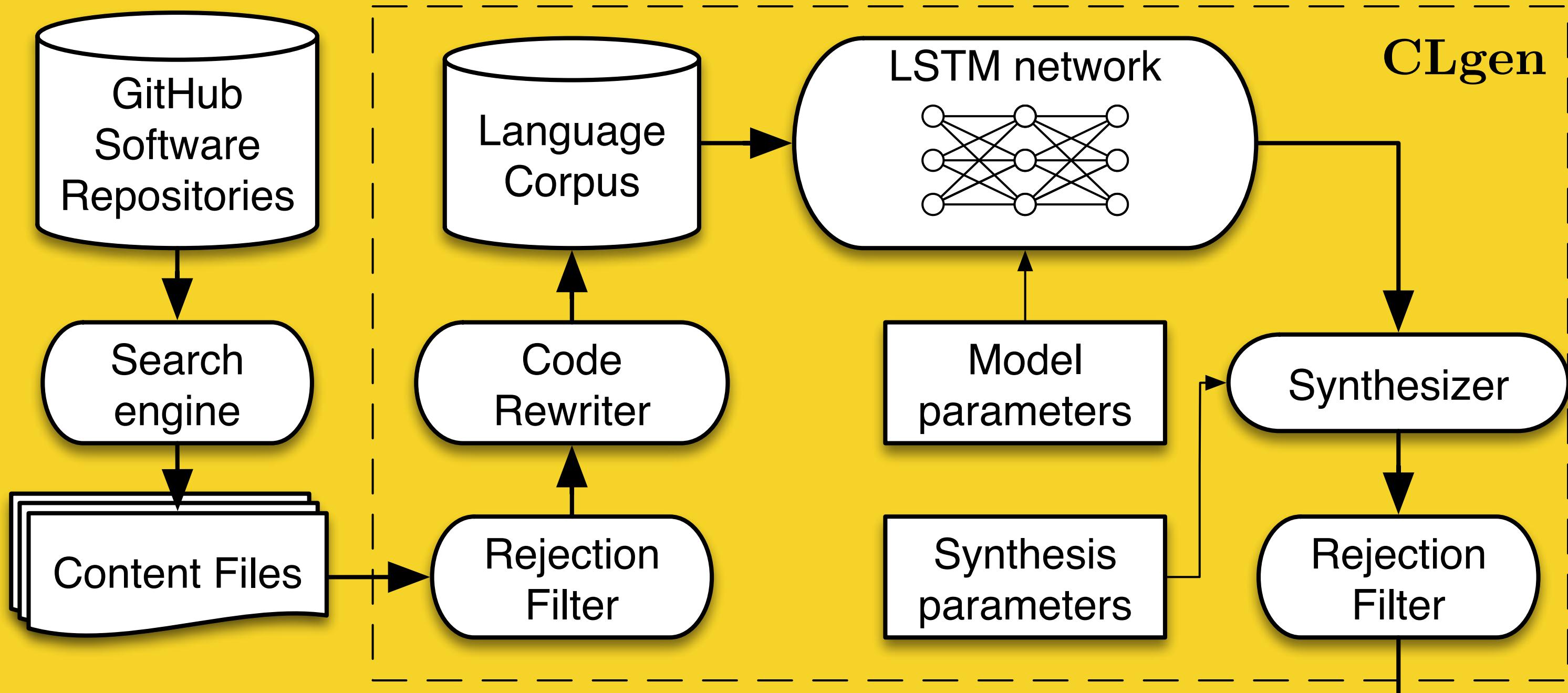
rand() * [S]

rand() * [S]

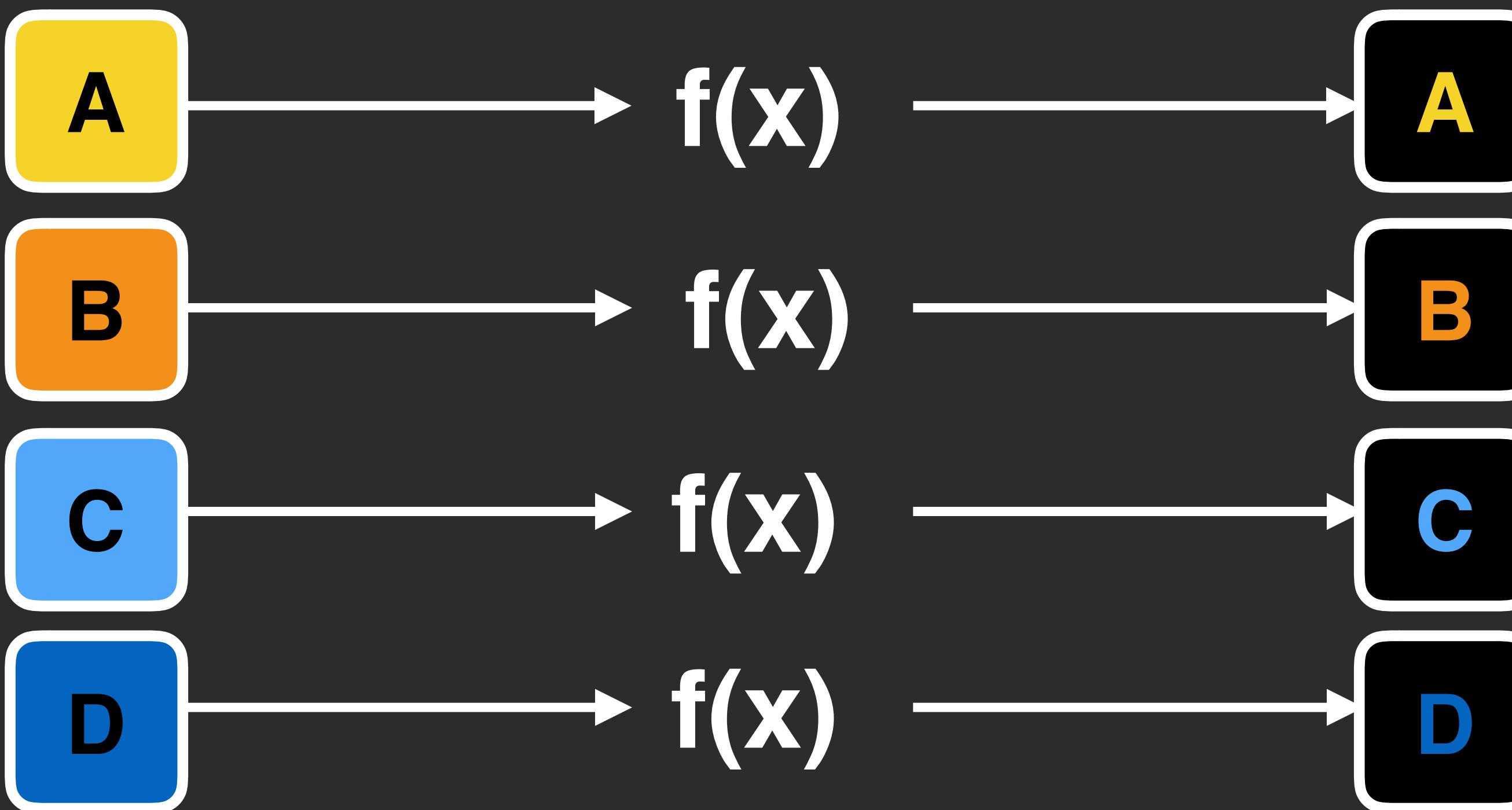
rand()

S



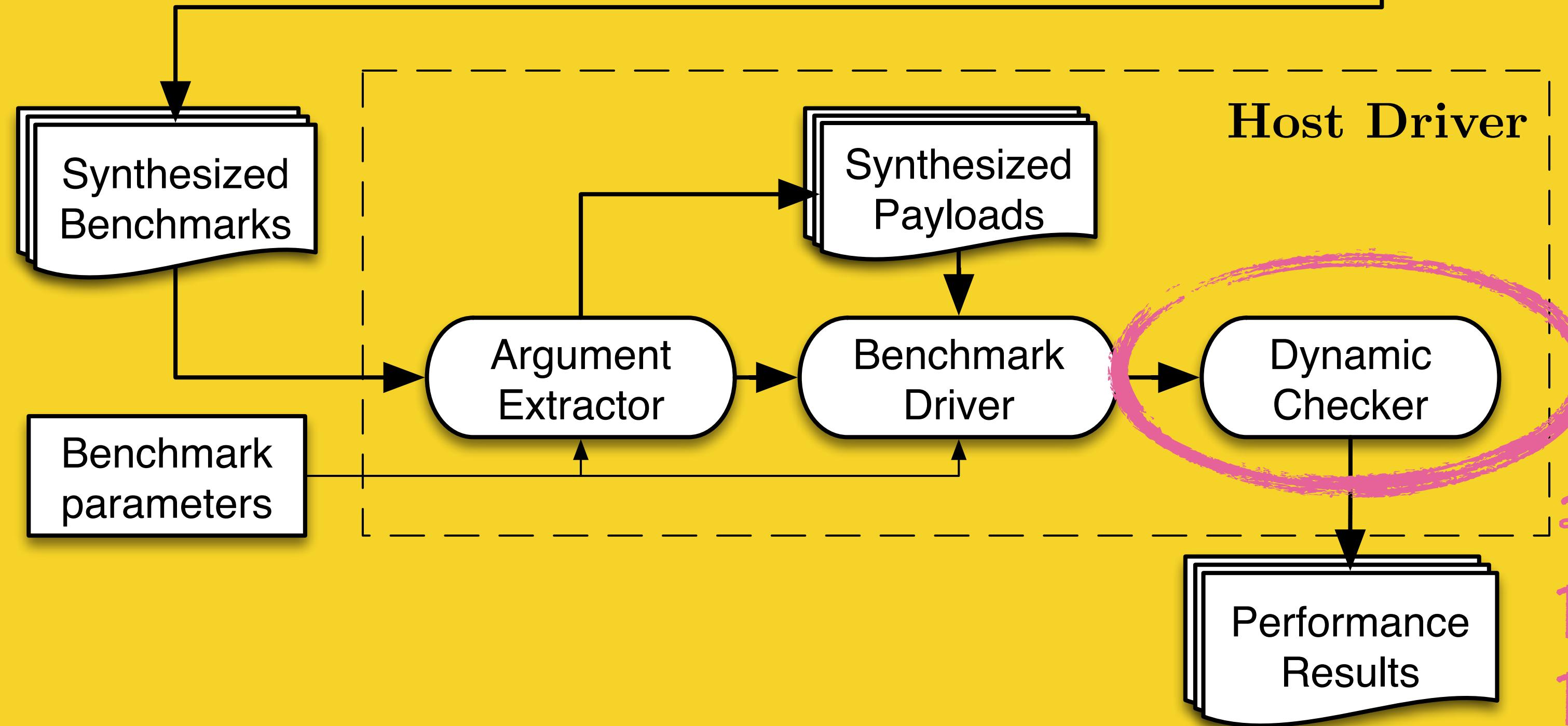
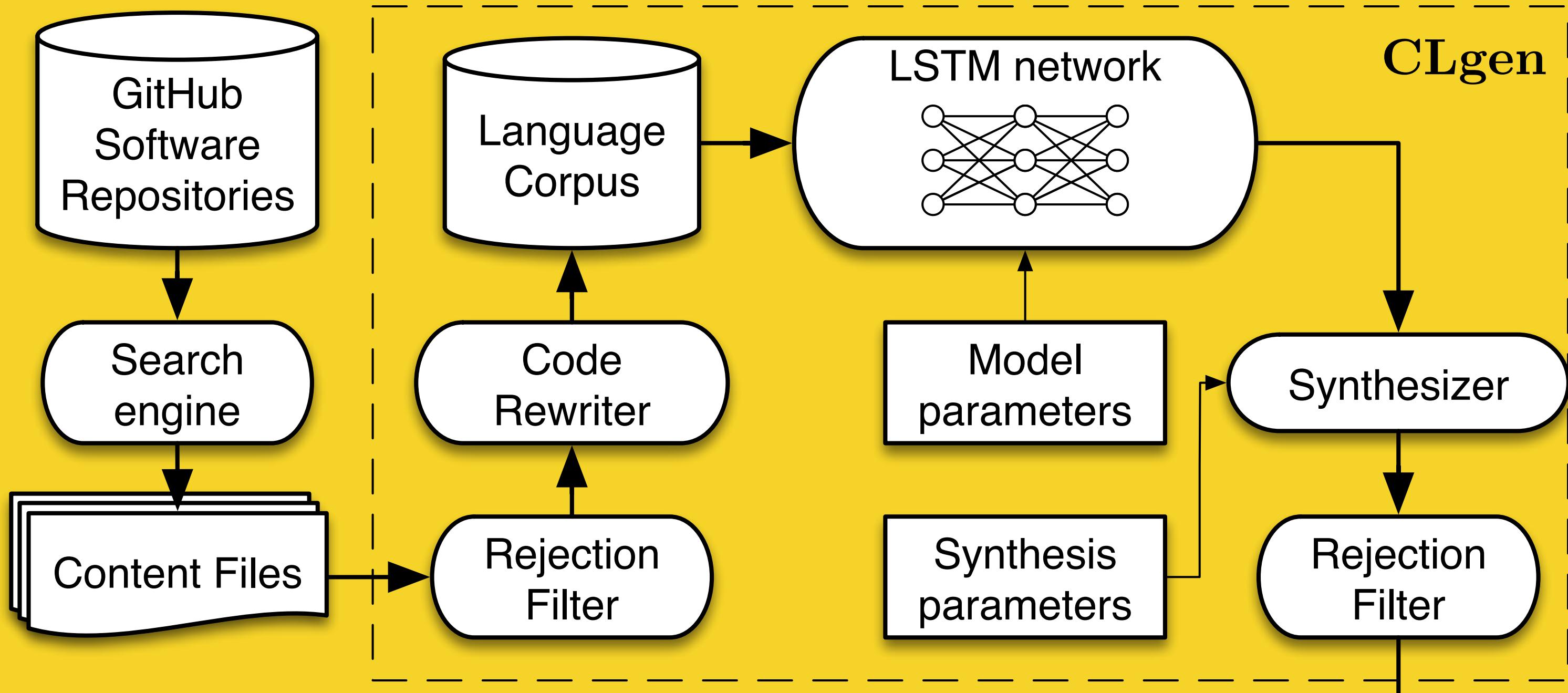


**Generate
Inputs**



Verify:

- $A \neq A$ else *no outputs*
- $A \neq C$ else *input insensitive*
- $A = B$ else *non deterministic*



results

Listing 3: Sample 3

```
1 __kernel void A(__global int* a, __global int* b, __global int* c,
2     ↪ __global int* d, const uint e) {
3     const uint f = get_global_id(0);
4
5     if (e == 0 && f == 0)
6         *d = 0;
7     else if (f < e) {
8         int g = b[f];
9         uint h = c[f];
10        if (g > 0) {
11            a[h] = f;
12            h++;
13        }
14        if (f == e - 1)
15            *d = h;
16    }
}
```

Listing 4: Sample 4

```
1 __kernel void A(__global float* a, __global float* b, __global float* c,
2     ↪ const int d) {
3     int e = get_global_id(0);
4
5     if (e < d) {
6         float f = b[e];
7         float g = a[e];
8         a[e] = f * 3.141592f / (f + 1.0f + e * 1024 - f) - (0.5f * f + g * 1.0f
9             ↪ / 18.0f + e / 2.0f);
10    }
11    for (c = 0; c < 30; c++) {
12        c[e] = 0;
13    }
}
```

Listing 4: Sample 4

```
1 __kernel void A(int a, __global float* b, __global int* c, __global int*
2     ↪ d, __local int* e, int f) {
3     int g = get_local_id(0);
4     e[get_local_id(0)] = 0;
5     barrier(1);
6     while (g < f) {
7         int h = c[g];
8
9         if (h != -1) {
10             __global float* i = b + g * a;
11             float j = 0;
```

Listing 7: Sample 7

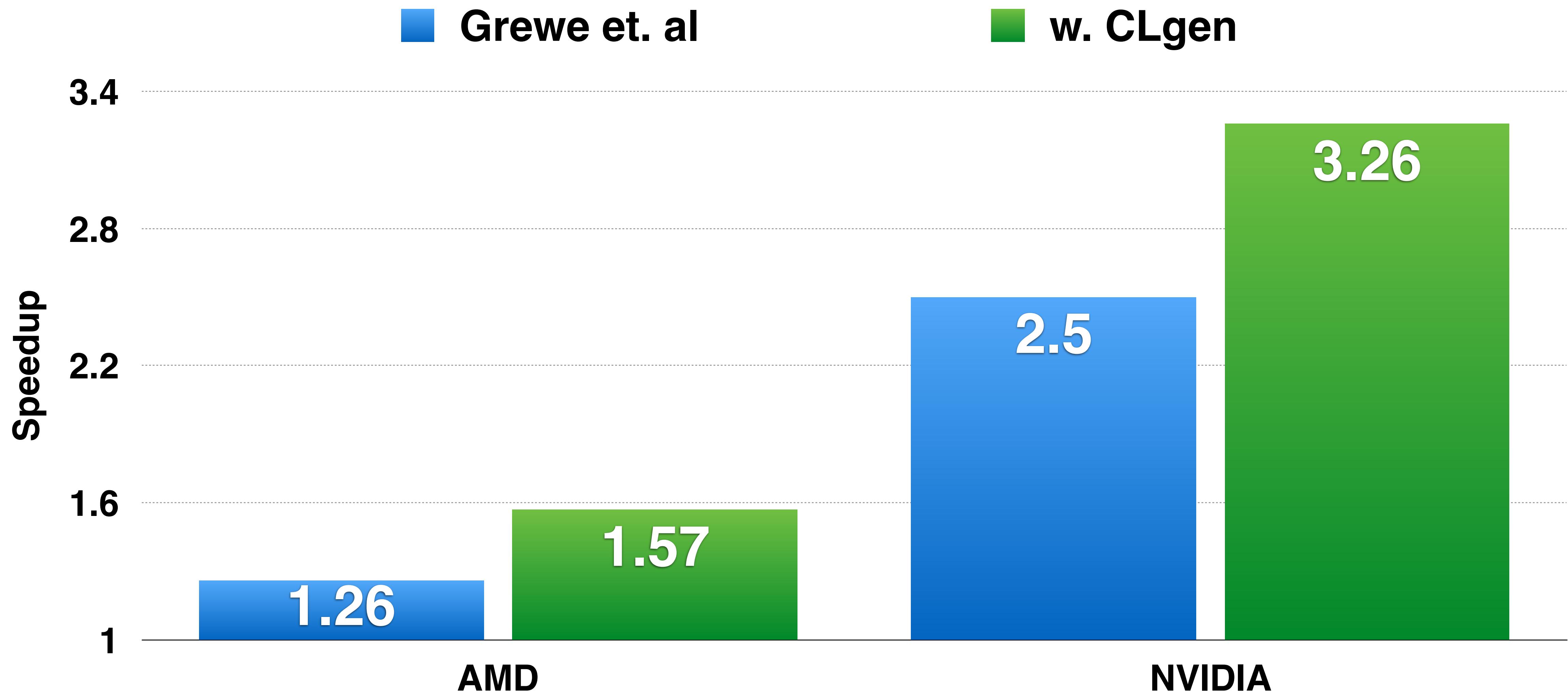
```
1 __kernel void A(int a, int b, int c, __global const float* d, __global
2     ↪ const float* e, __global float* f, float g) {
3     const int h = get_local_id(0);
4     const int i = get_group_id(0);
5
6     const int j = 4 * i + h;
7     const int k = 4 * i + h + a;
8     if (4 * i + h + a < c) {
9         float l = 0.0;
10        float m = 0.0;
11        float n = 0.0;
12        const float o = d[3 * (4 * i + h + a)];
13        const float p = d[3 * (4 * i + h + a) + 1];
14        const float q = d[3 * (4 * i + h + a) + 2];
15        for (int r = 0; r < c; r++) {
16            const float s = d[3 * r] - o;
17            const float t = d[3 * r + 1] - p;
18            const float u = d[3 * r + 2] - q;
19            const float v = (d[3 * r] - o) * (d[3 * r] - o) + (d[3 * r + 1] -
20                ↪ p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r +
21                ↪ 2] - q) + g;
22            const float w = e[r] / (((d[3 * r] - o) * (d[3 * r] - o) + (d[3 * r
23                ↪ + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r + 2]
24                ↪ - q) + g) * sqrt((d[3 * r] - o) * (d[3 * r] - o) + (d[3 * r + 1]
25                ↪ - p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r + 2] - q) + g));
26            l = l + (d[3 * r] - o) * w;
27            m = m + (d[3 * r + 1] - p) * w;
28            n = n + (d[3 * r + 2] - q) * w;
29        }
30        f[j] = l;
31        f[j + 1] = m;
32        f[j + 2] = n;
33    }
34 }
```

Listing 10: Sample 10

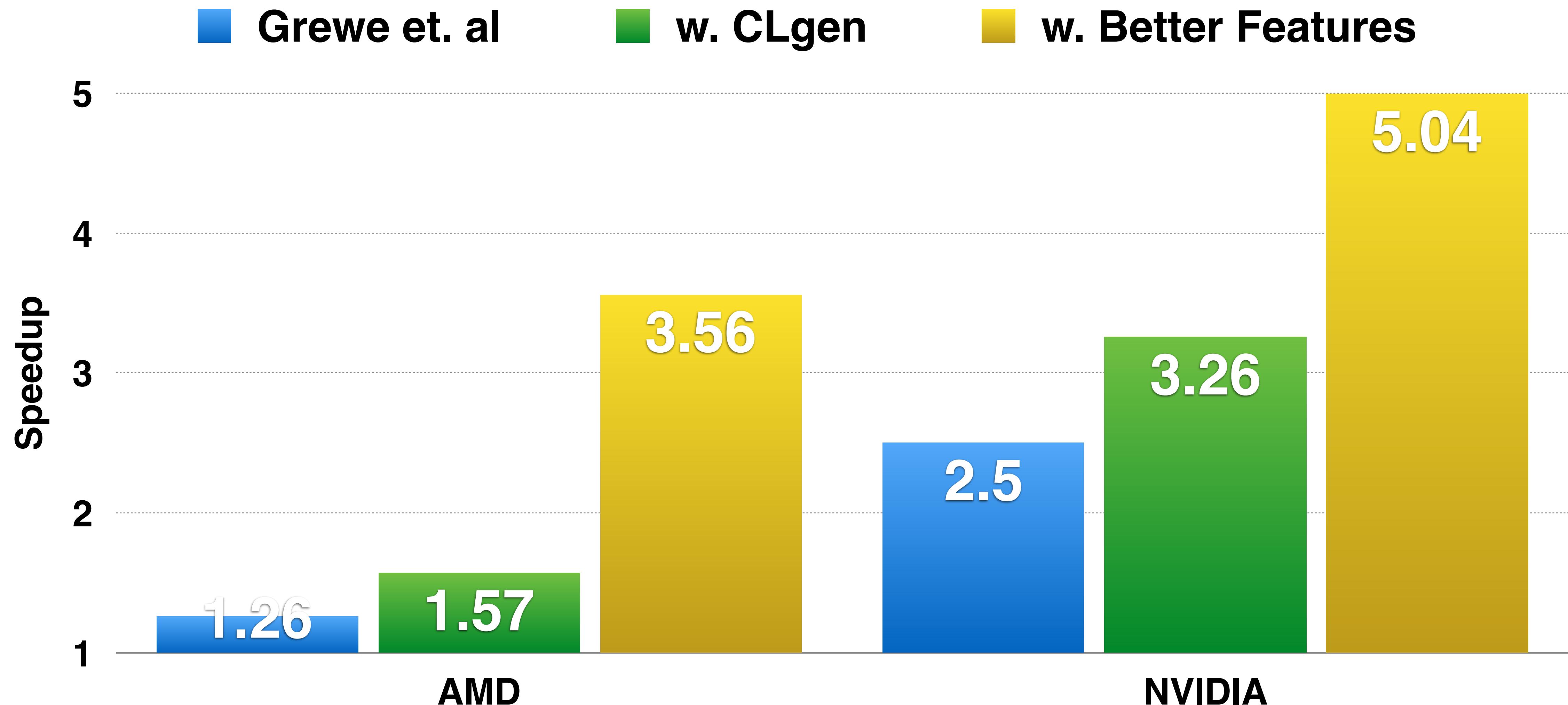
```
1 __kernel void A(__global ulong *a) {
2     int i, j;
3     struct S0 c_8;
4     struct S0* p_7 = &c_8;
5     struct S0 c_9 = {
6         {{0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
7          {0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
8          {0x43250E6DL,2UL},{0x43250E6DL,2UL}},
9         0x4BF90EDCAD2086BDL,
10     };
11     c_8 = c_9;
12     barrier(0 | 1);
13 }
```

52%
blind test
<http://humanorrobot.uk>

7 benchmarks, 1,000 synthetic benchmarks. 1.27x faster



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



concluding remarks

problem: insufficient benchmarks

use DL to learn PL semantics and usage

turing tested! ;-)

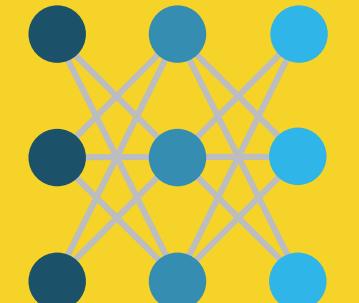
**improved model performance and
design**

Synthesizing Benchmarks for Predictive Modeling



For the paper, code and data:
<http://chriscummins.cc/cgo17>



 **CLgen**
Deep Learning Program Generator.

For the TensorFlow neural network:
<http://chriscummins.cc/clgen>