Extreme Fuzzing Machine

20th June 2022

Joshua Williamson

Road Map

NEUZZ (D.She *et al* .2019)

↓

KELMFUZZ

↓

KELMFUZZ++

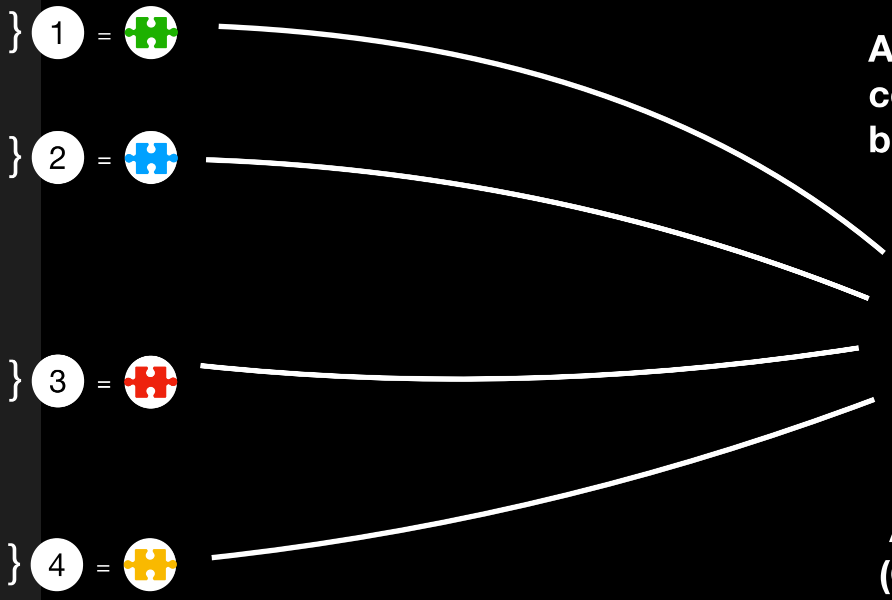
↓

Extreme Fuzzing Machine

Coverage guided fuzzing

```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c



Amount of code covered is corresponds to the amount of bugs found.

Fuzzers implement instrumentation 'bits' during compilation

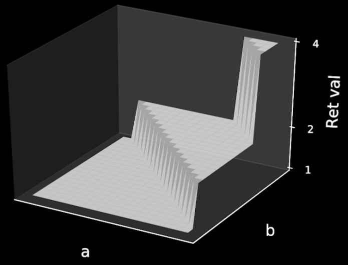
AFL- American Fuzzy Lop (Google).

NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c



(a) Original

3.

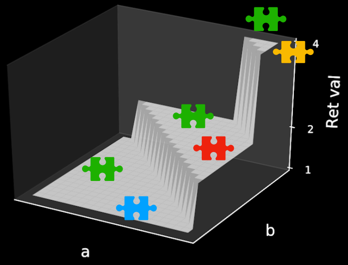
NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

UCL CDT DIS

```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c



(a) Original

nccgroup

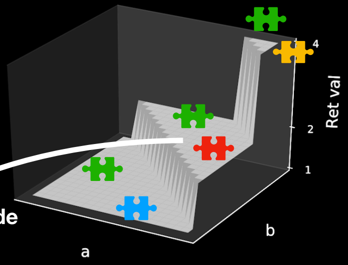
NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c

Difficult to reach vulnerable code



(a) Original

UCL CDT DIS

nccgroup

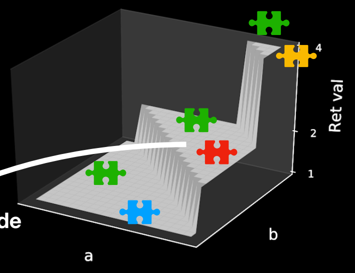
NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

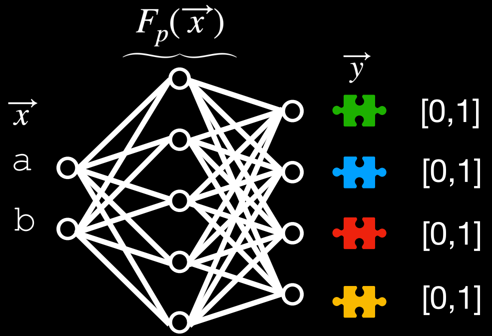
```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c

Difficult to reach vulnerable code



(a) Original



3.

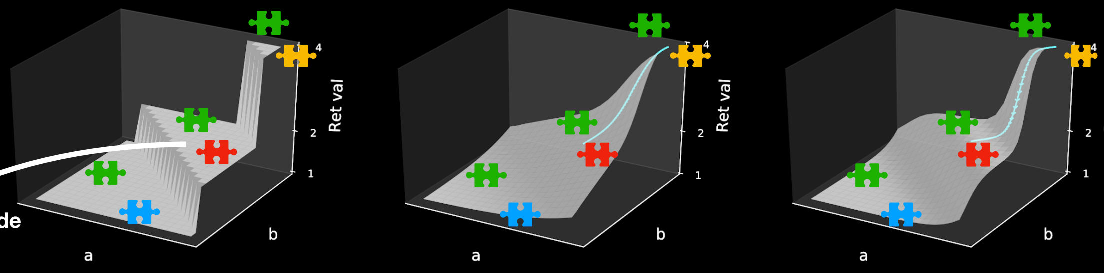
NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

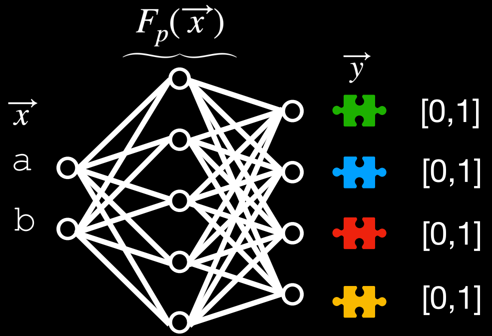
```c
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c

Difficult to reach vulnerable code



(a) Original (b) NN smoothing (c) NN smoothing + refining



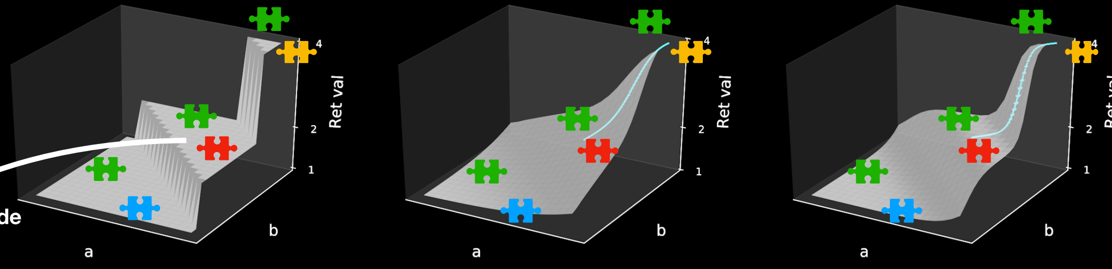
NEUZZ

A novel neural smoothing technique (D.She *et al* .2019)

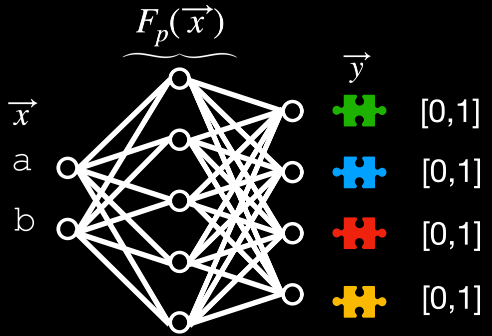
```
int main(int a, int b){
    z=pow(3,a+b)

    if (z < 1){
        //Blue bit
        return 1;
    }

    else if (z < 2){
        //Buffer overflow
        char buff[10];
        buff[10] = 'a';
        //Red bit
        return 2;
    }

    else if (z < 4){
        //Yellow
        return 4;
    }
}
```

program.c

Difficult to reach vulnerable code



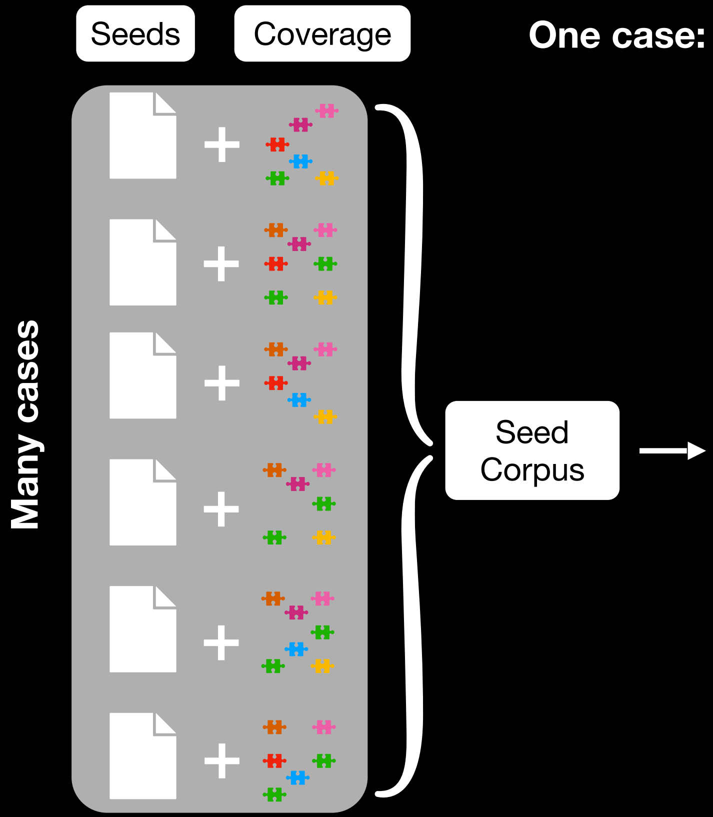
(a) Original (b) NN smoothing (c) NN smoothing + refining



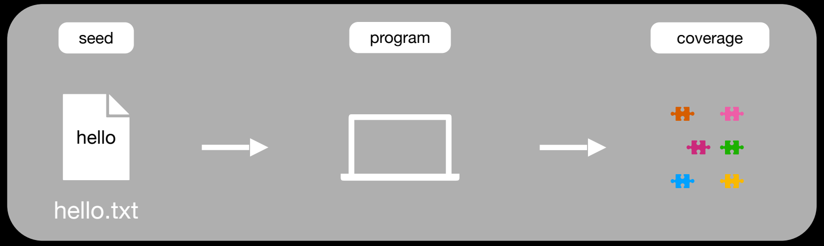
We can take gradients of the smooth surrogate function w.r.t each byte of input

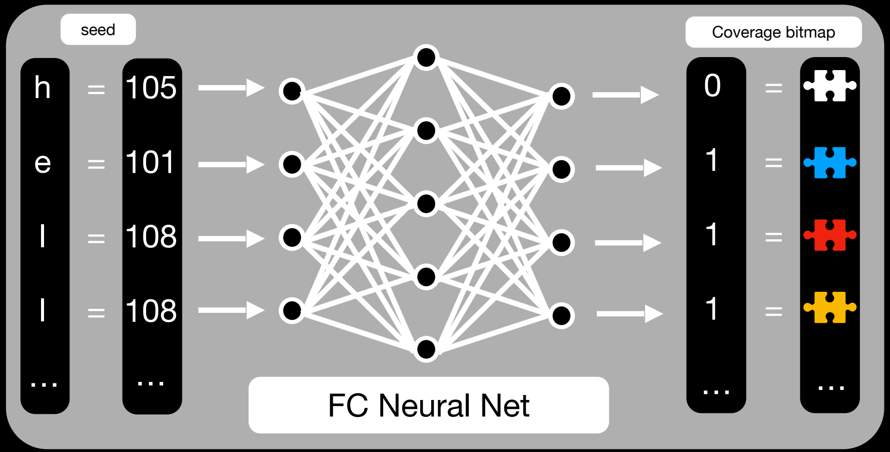
$$m_{ij} = \frac{\partial F_p(\vec{x})_j}{\partial x_i}$$

NEUZZ



One case:

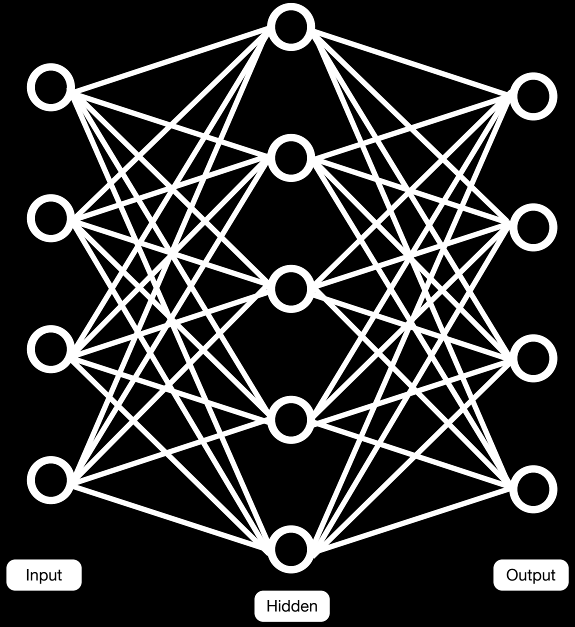




Fully connected NN: works well but back propagation is slow!

KELMFUZZ

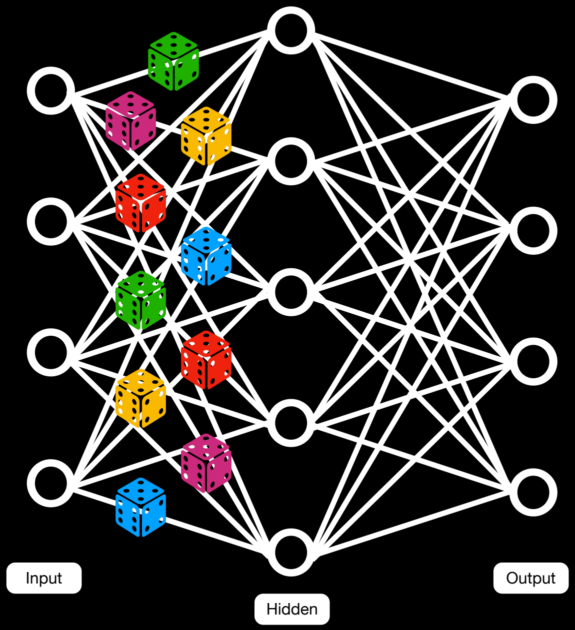
Extreme Learning Machines (ELM's)



Fully connected NN: works well but back propagation is slow!

KELMFUZZ

Extreme Learning Machines



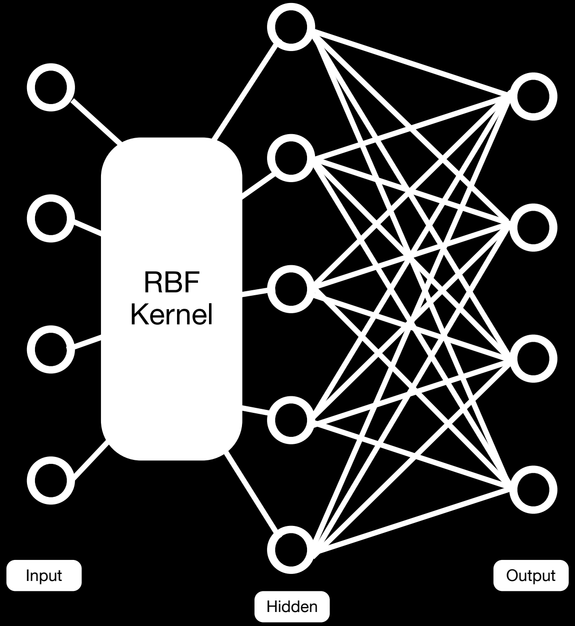
1. Parameters between input and hidden layer randomised
2. Parameters between hidden and output layer are solved by directly inverting the minimisation problem.

No backpropagation = Fast

Universal approximation capable

KELMFUZZ

Kernelized Extreme Learning Machine (KELM) (A.Losifidis *et al* .2015)



1. Parameters between input and hidden layer mapped with kernel

2. Parameters between hidden and output layer are solved by directly inverting the training set minimisation problem.

No backpropagation = Fast

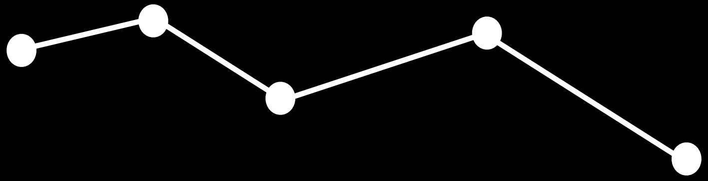
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2} \right)$$

Only one hyperparameter: σ of the RBF kernel

KELMFUZZ

Improvements- KELMFUZZ++

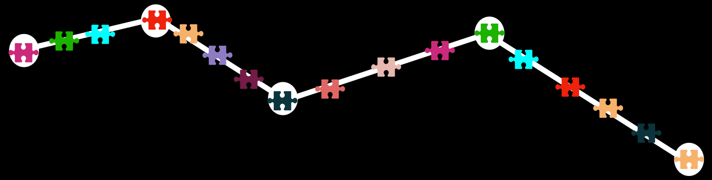
- Use entire bitmap, no data reduction



KELMFUZZ

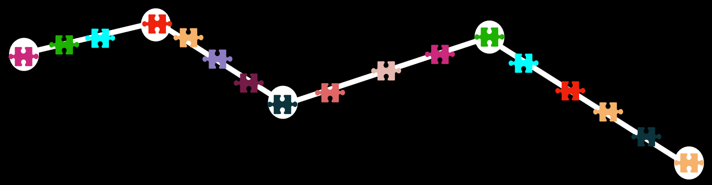
Improvements- KELMFUZZ++

- Use entire bitmap, no data reduction
  ~10,000X more mutation opportunities



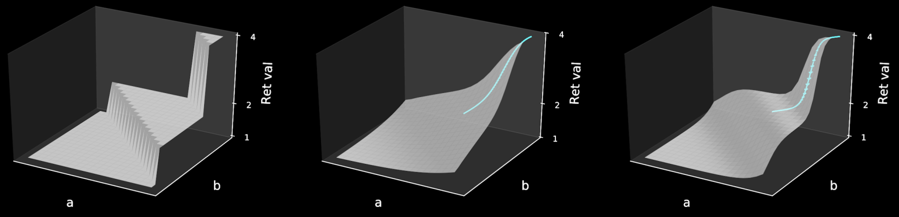
KELMFUZZ

Improvements- KELMFUZZ++



- Use entire bitmap, no data reduction
  ~10,000X more mutation opportunities

- Train on larger seed corpus



(a) Original (b) NN smoothing (c) NN smoothing + refining

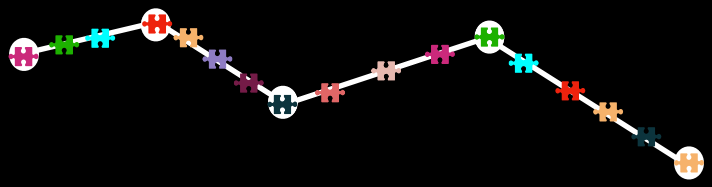
Faster refining = better quality mutations

ᐞUCL CDT DIS

nccgroup

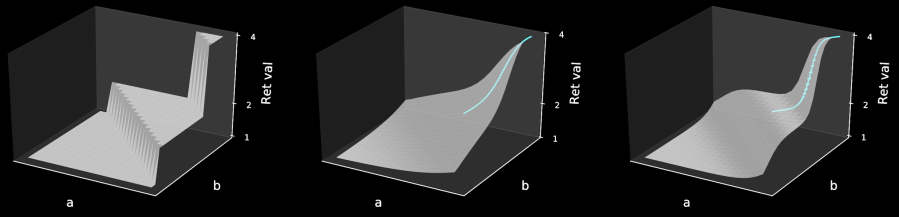
KELMFUZZ

Improvements- KELMFUZZ++



- Use entire bitmap, no data reduction
  ~10,000X more mutation opportunities

- Train on larger seed corpus



(a) Original (b) NN smoothing (c) NN smoothing + refining

Faster refining = better quality mutations

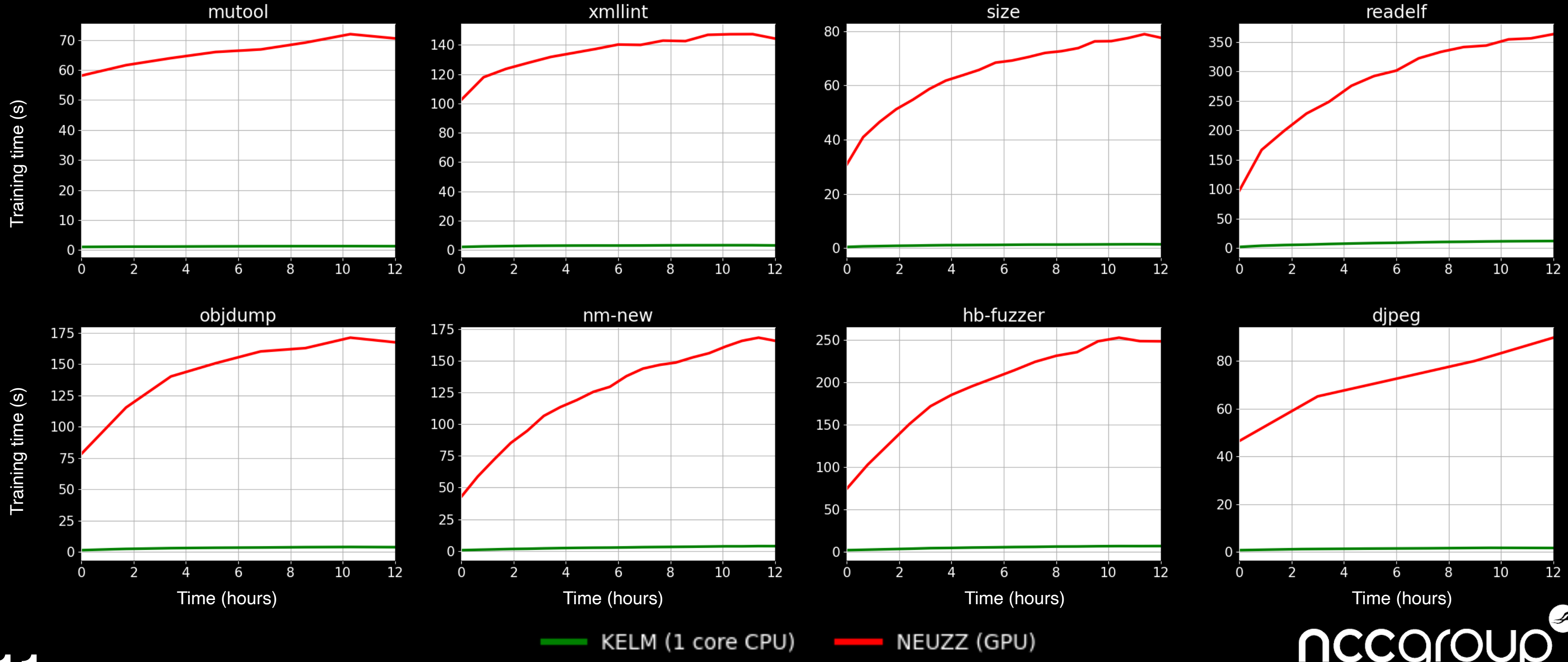
- Implemented “Modified Havoc” mutation algorithm **(M.Wu *et al* .2021)**

UCL CDT DIS

nccgroup

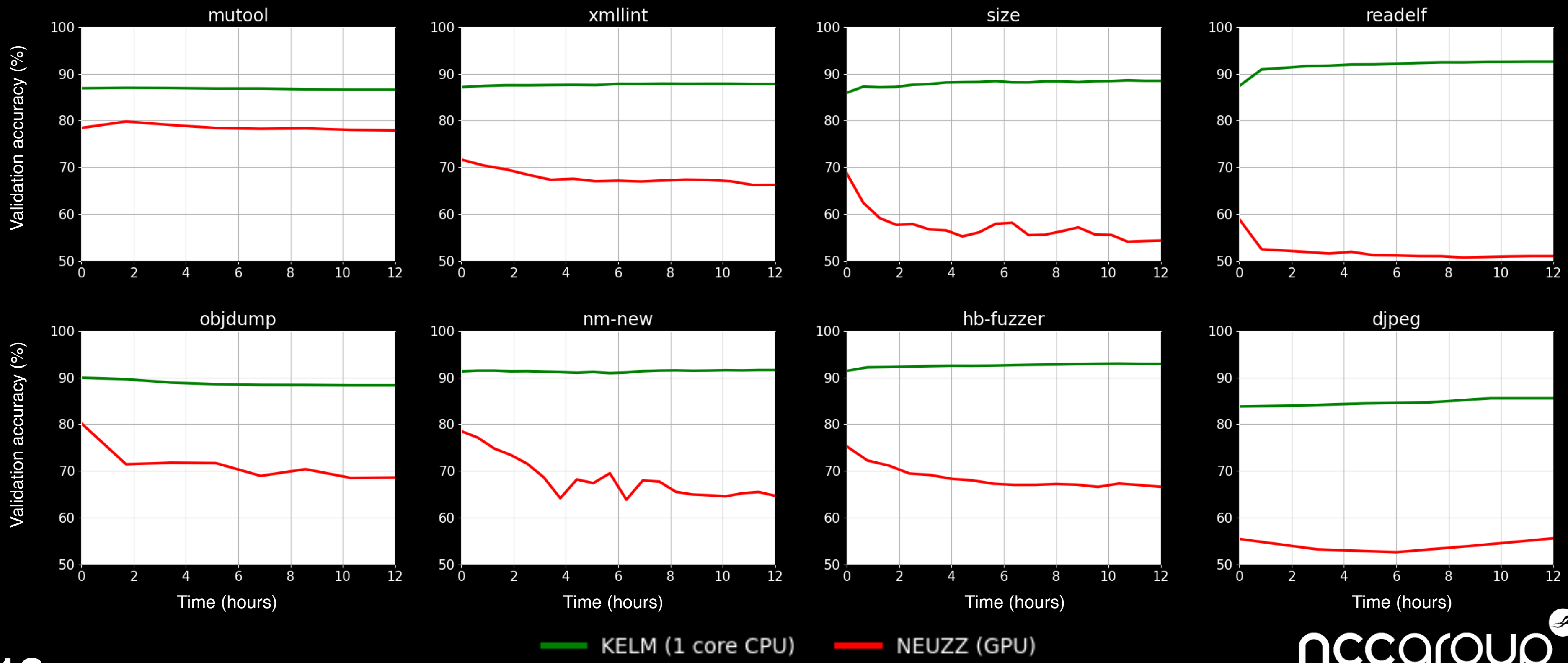
KELMFUZZ

Results: training speed



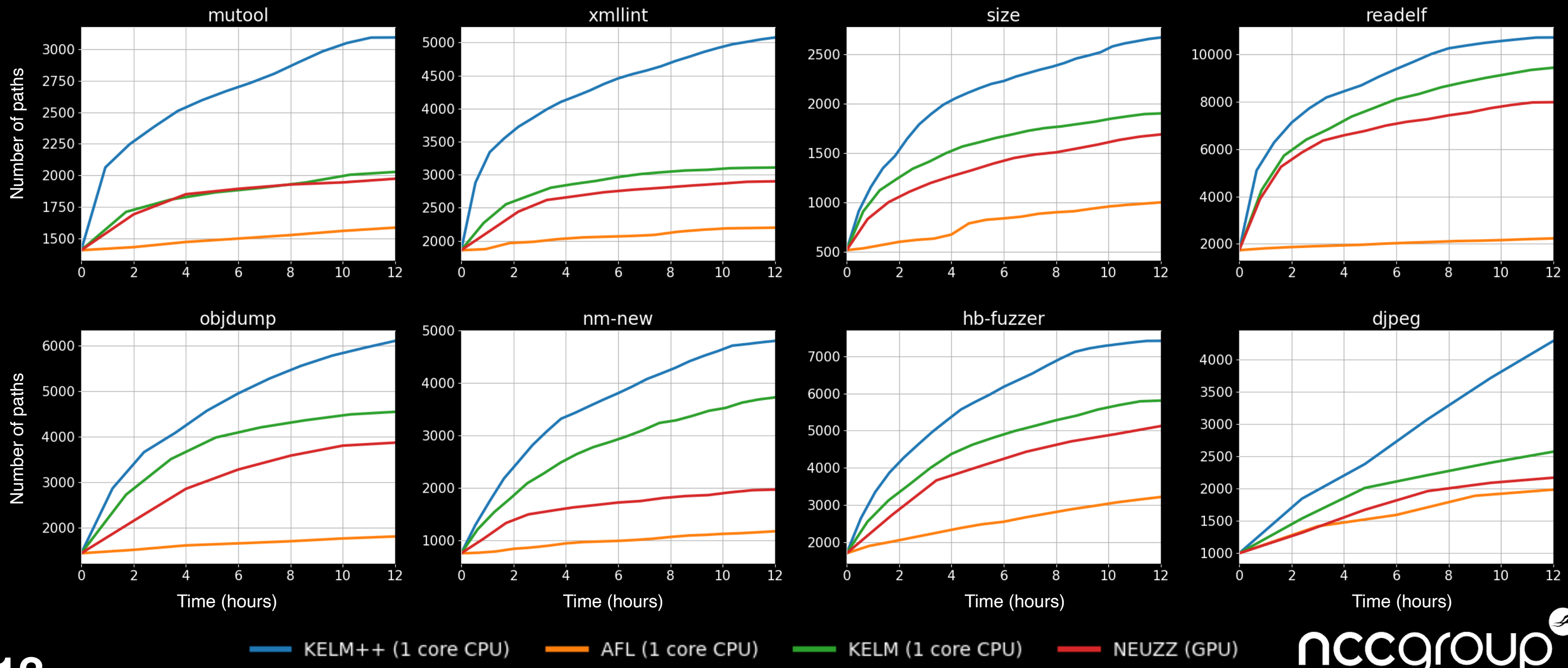
KELMFUZZ

Results: training accuracy

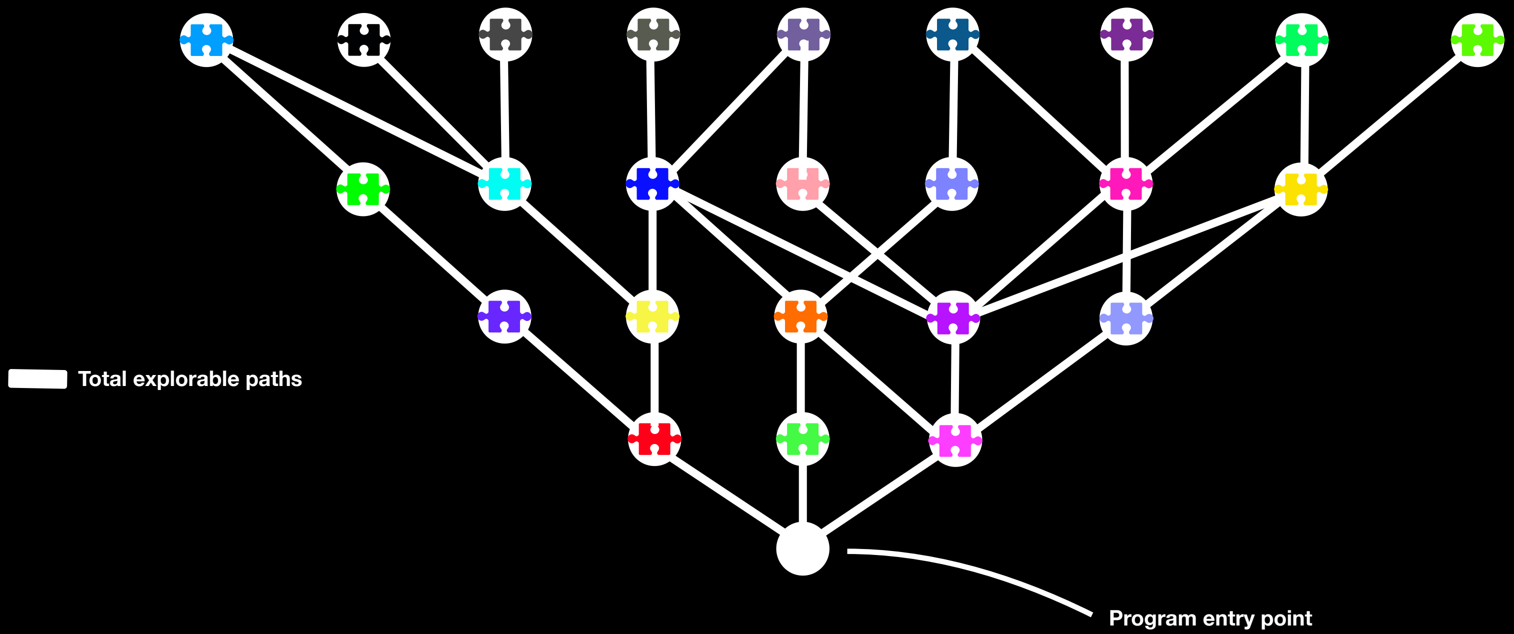


KELMFUZZ

Results: unique code paths explored



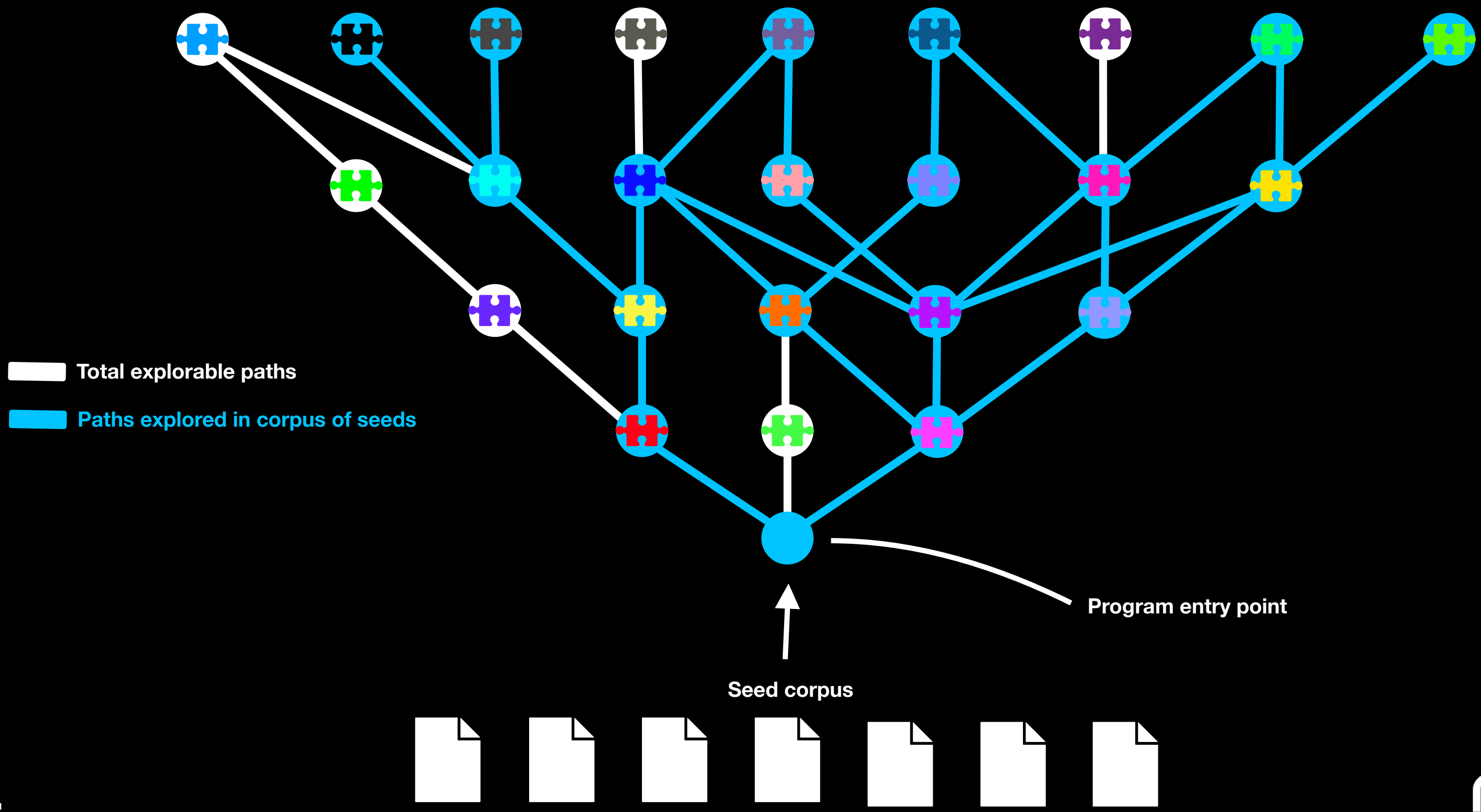
EFM: EDGE CONTEXT



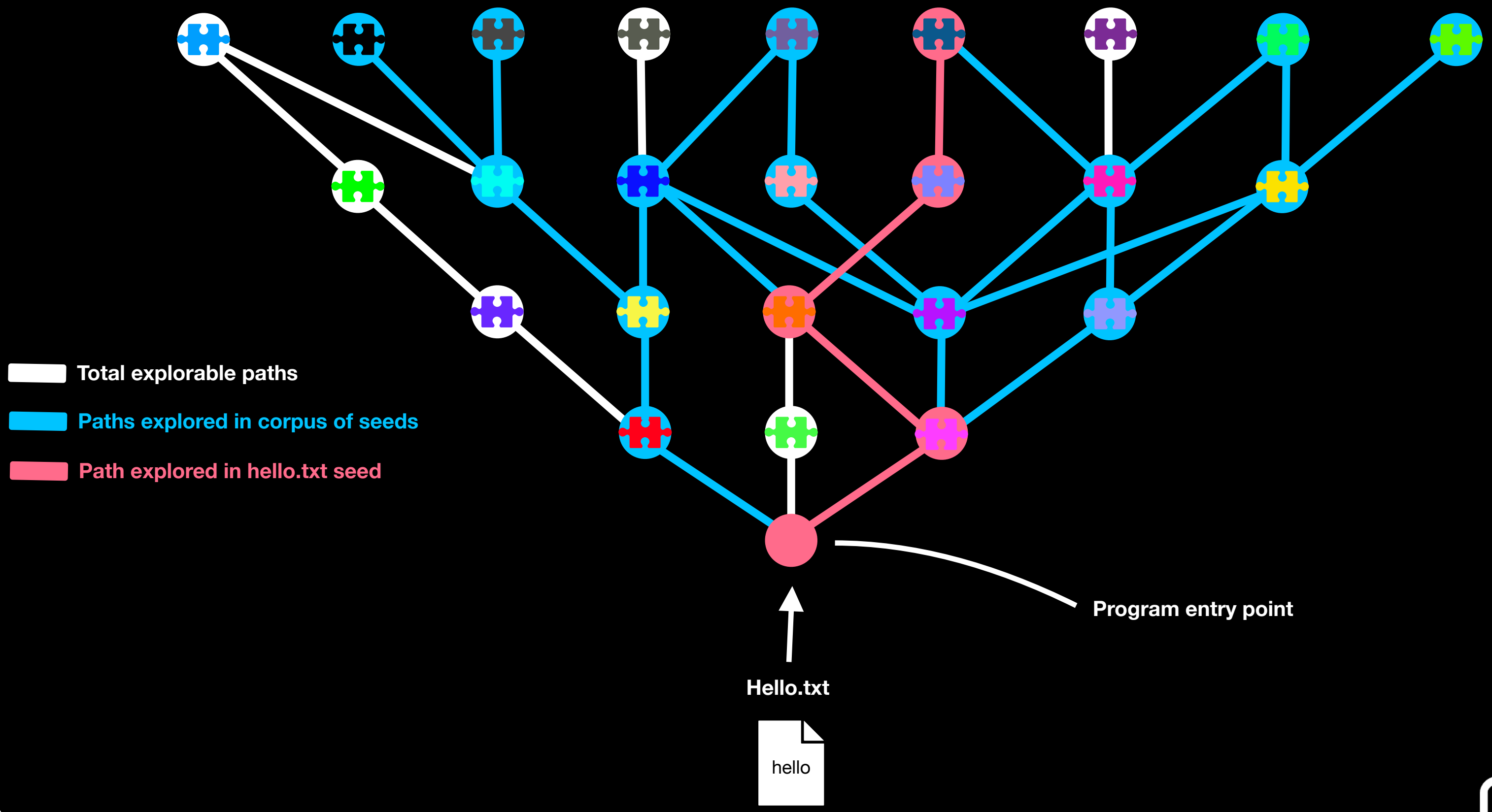
UCL CDT DIS

nccgroup

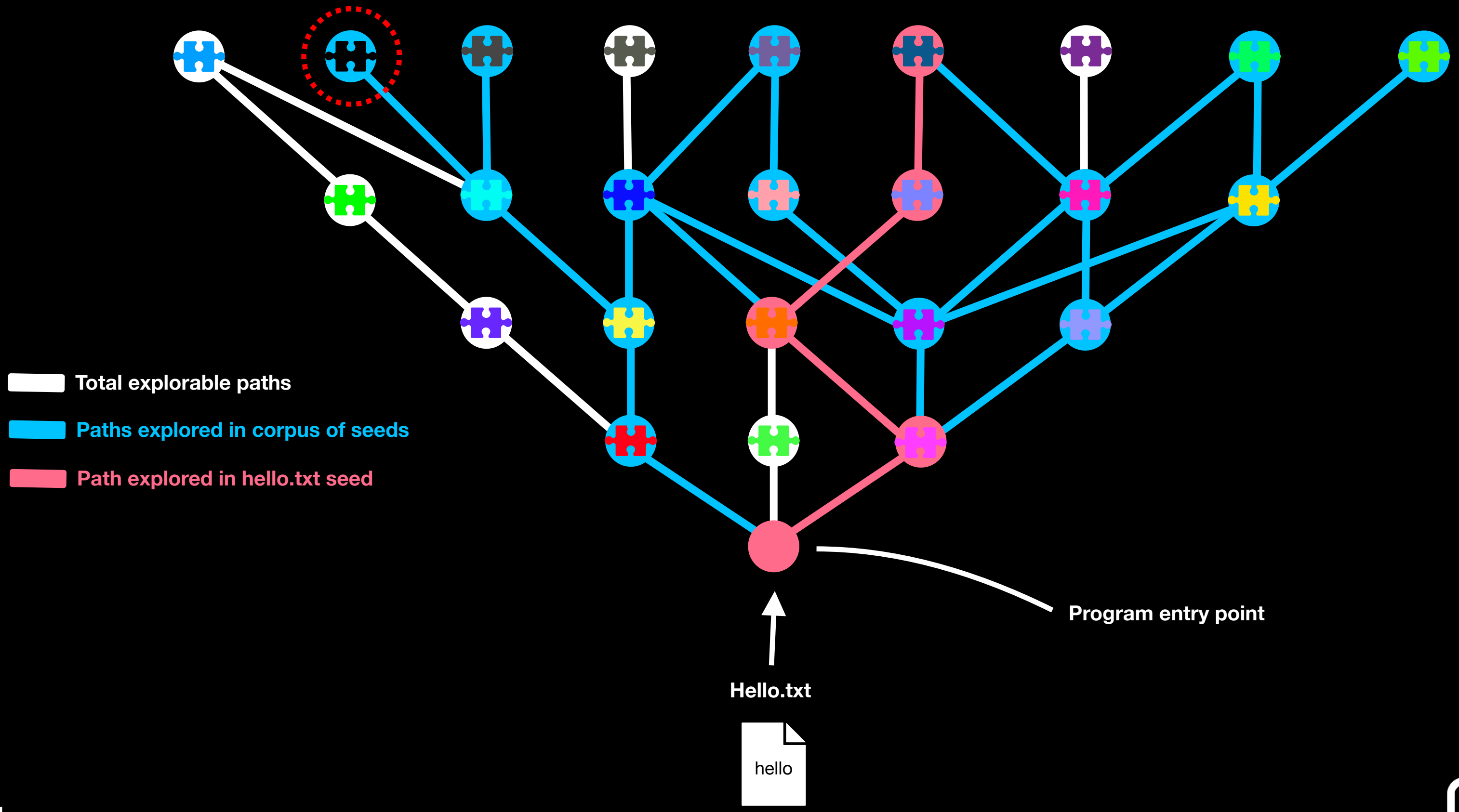
EFM: EDGE CONTEXT



EFM: EDGE CONTEXT



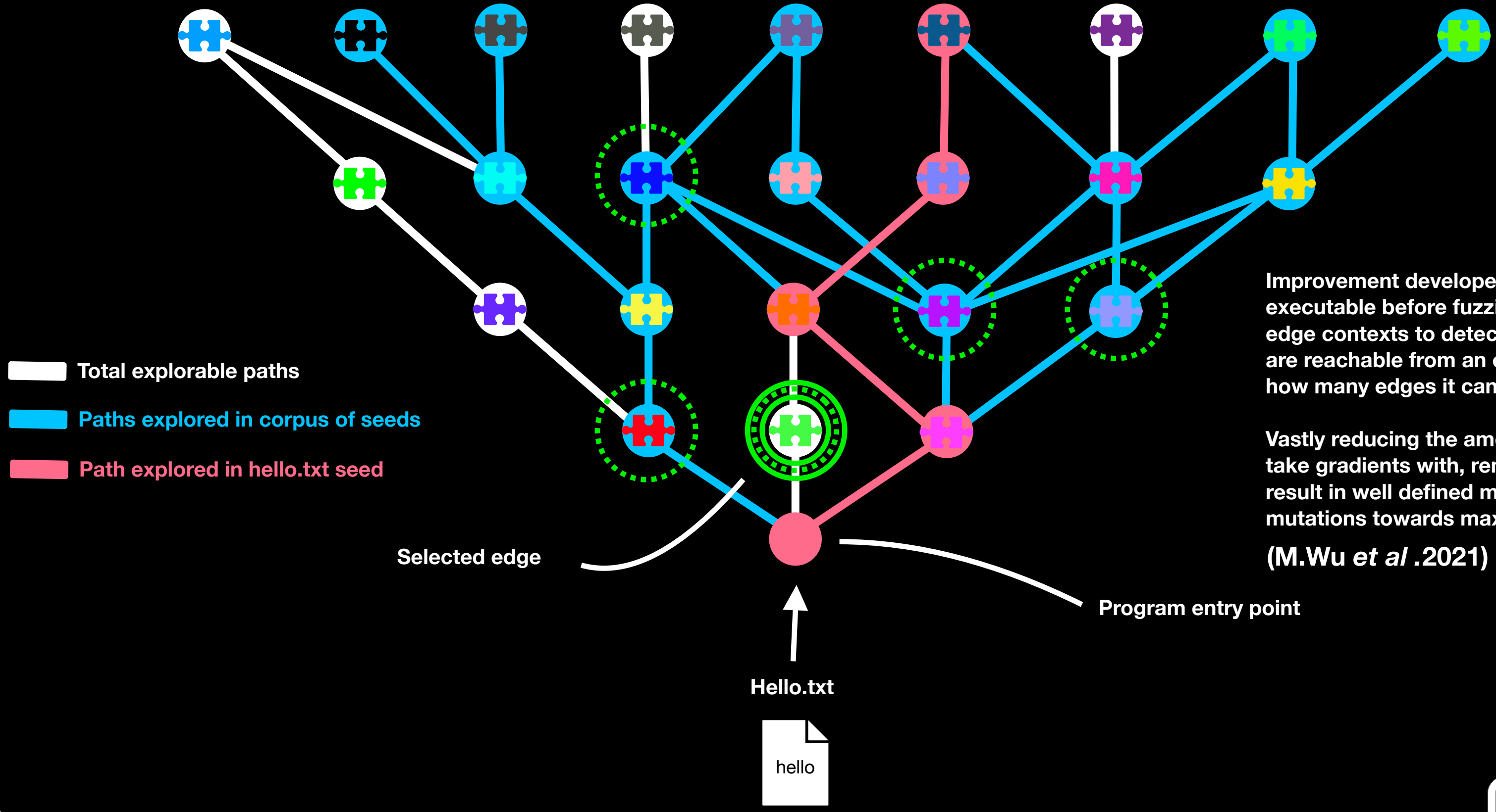
EFM: EDGE CONTEXT



UCL CDT DIS

nccgroup

EFM: EDGE CONTEXT



Improvement developed upon Wu. *et al,* parses the executable before fuzzing to build a dictionary of edge contexts to detect the neighbouring edges that are reachable from an edges of a given execution and how many edges it can access.

Vastly reducing the amount of node candidates to take gradients with, removing those that would not result in well defined mutations. And directing mutations towards maximising code coverage

(M.Wu *et al* .2021)

EFM: SEED REDUCING

UCL CDT DIS

- Stochastic processes such as random insertion and deletion and havoc are good at producing seeds that gain code coverage but often lead to long execution times
- When the python module is not fuzzing, it parses through seeds and reduces their size as much as possible within a time budget without changing their edge coverage
- This speeds up fuzzing
- Reduces very large seeds to a threshold that means they can be used in training
- Made from a modified version of afl-tmin

nccgroup

EFM: USABILITY

Major code refactoring and good documentation

Reduces amount of user dependent input and any knowledge of AI / ML / python

Aims at an AFL level of usability

Monitoring screen for better user understanding/engagement

Crash and hang counters

Can now be used with ASAN

A big to do list !

```
     __________  ___
    / __/ __/  |/  /
   / _// _// /|_/ /
  /___/_/ /_/  /_/  Extreme Fuzzing Machine (2022)

[+] Setting up mutation templates, max file size: 5836
[*] You have 4 CPU cores and 2 runnable tasks (utilization: 50%).
[*] Checking CPU core loadout...
[*] Found a free CPU core, binding to #0.
[*] Setting up shared memory buffers
[*] Setting up output directories...
[*] Checking core_pattern...
[*] Spinning up neural network server
[+] Neural network server up and running
[+] Spinning up the fork server...
[+] All right - fork server is up.
[+] Ok, all set up and ready to go:

      Memory limit : 1024 Mb
     Timeout limit : 1000 ms
```

Start up screen

```
                Extreme Fuzzing Machine  (djpeg)

          Time
            run time : 0 days, 0 hrs, 14 min, 57 sec
          last grads : 0 days, 0 hrs, 14 min, 9 sec
 Neural Net Engine                    Fuzzer
       Status : Sleeping               Rounds done : 0
 training acc : 90.08%                  Exec speed : 978.6/sec
 data                   state
 Bitmap size : 3131        Stage : gradient (mutation)
 Corpus size : 994      Progress : 48/150 (32%)
  Nocov size : 0            Seed : ./seeds/id:000507,src:000000 ...
 module load                          findings
  Mapping time : 0 min, 15 sec           Crashes : 0, 0 unique
 T-mining time : 1 min, 27 sec         Time outs : 6346, 294 unique
 Training time : 1.02 sec             Edge count : 3168
                                      log messages
                                      [!] 0         [-] 0
```

Monitoring screen