Augmented Programming

Bloomberg

Engineering

Data Council New York City 2019
November 12, 2019

Gideon Mann
Head of Data Science
Office of the CTO

TechAtBloomberg.com

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

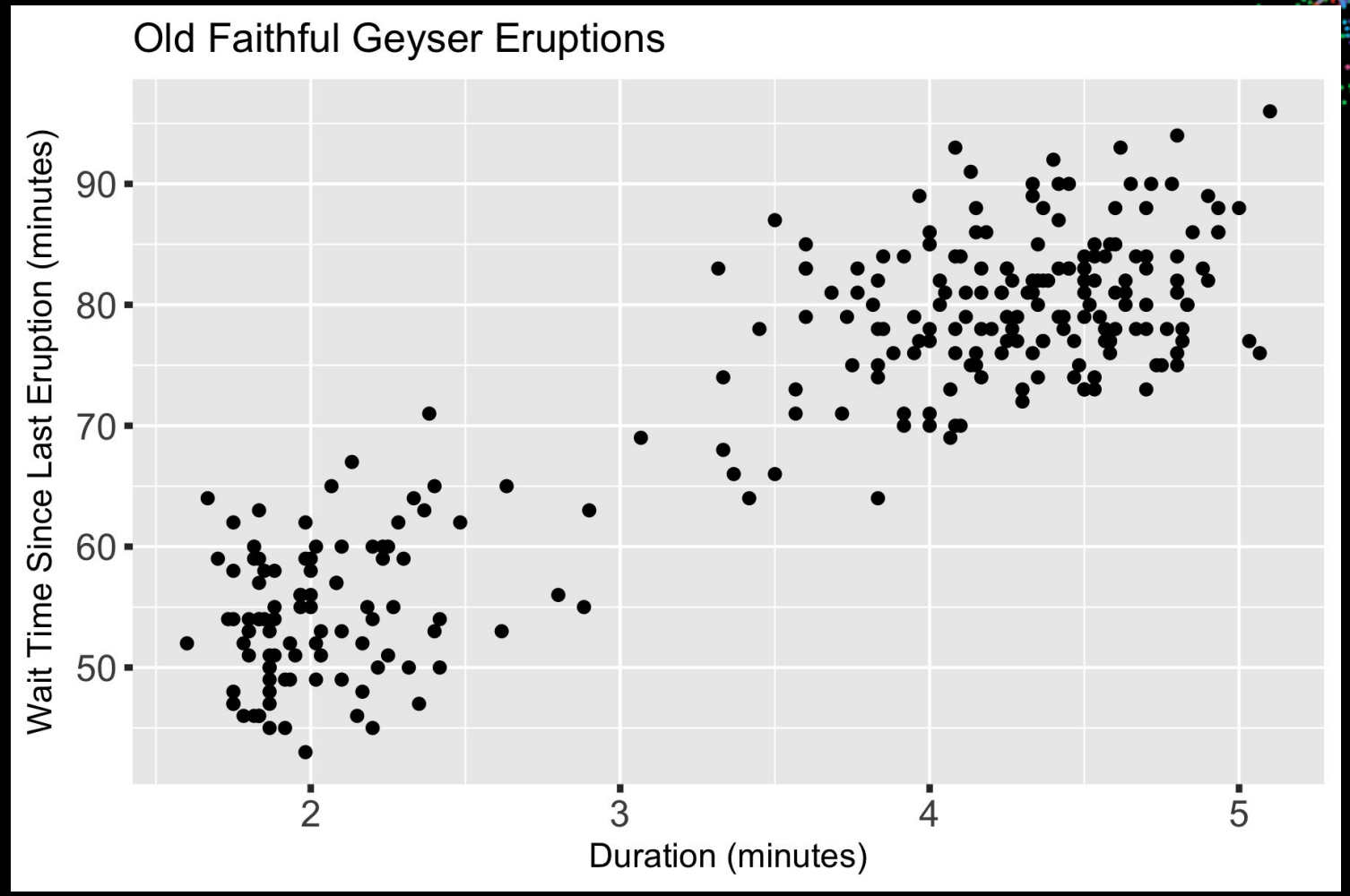
Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.



x	y
3.6	79
1.8	54
3.333	74
2.283	62
4.533	85
2.883	55
4.7	88
3.6	85
1.95	51
4.35	85
1.833	54
3.917	84
4.2	78
1.75	47
4.7	83
2.167	52
1.75	62
4.8	84
1.6	52
4.25	79
1.8	51
…	…

Sample	Limit (h^{-1} Mpc)	Range (h^{-1} Mpc)	γ_s	s_0	γ_r	r_0
ESOd	80	2.5 - 20	1.48	7.2	-	-
ESOm	80	2.5 - 20	1.65	7.5	-	-
UGCd	80	2.5 - 32	1.48	8.8	-	-
UGCm	80	1.6 - 32	1.64	6.5	-	-
ORSd	80	1 - 40	1.57	6.6	1.59	7.07
ORSm	80	1.3 - 25	1.51	6.8	1.56	7.26

```
\begin{tabular}
  {|l|c|c|c|c|c|c|} \hline
  Sample & Limit & Range & $\gamma_s$ & $s_0$ & $\gamma_r$ & $r_0$ \\ & ($h^{-1}$ Mpc) &
($h^{-1}$ Mpc) & & & & & \\
  ESOd & 80 & 2.5 - 20 & 1.48 & 7.2 & -    & - \\
  ESOm & 80 & 2.5 - 20 & 1.65 & 7.5 & -    & - \\
  UGCd & 80 & 2.5 - 32 & 1.48 & 8.8 & -    & - \\
  UGCm & 80 & 1.6 - 32 & 1.64 & 6.5 & -    & - \\
  ORSd & 80 &   1 - 40 & 1.57 & 6.6 & 1.59 & 7.07 \\
  ORSm & 80 & 1.3 - 25 & 1.51 & 6.8 & 1.56 & 7.26 \\
  \hline
\end{tabular}
```

Table Data Sets

Equation Data Set (im2latex-100k)

103,566 equations written in LaTeX extracted from arXiv articles

$$\hat{g} = -(dx^0)^2 + (dx^1)^2 + \left(dr^2 + r^2 d\Omega_8^2\right) ,$$

Tables Data Set (TABLE2LATEX-450K)

465,967 tables written in LaTeX extracted from arXiv articles

Class in E_6	Action on $L(G)'$	Action on $L(G)$
$2A_2$	$3^{23}, 1^8$	$3^{23}, 2, 1^7$
$2A_2 + A_1$	$3^{24}, 2^2, 1$	$3^{24}, 2^3$
A_5	$9^3, 8^2, 6^4, 3^2, 1^4$	$9^3, 8^2, 6^4, 3^2, 2, 1^3$
$E_6(a_3)$	$9^4, 7, 6^4, 3^3, 1$	$9^4, 7, 6^4, 3^3, 2$
$E_6(a_1)$	$9^8, 5$	$9^8, 6$
E_6	$19, 15^2, 9^3, 1$	$19, 15^2, 9^3, 2$

“High” level language

```
/*  Simple regular expression matching.  From:  The Practice of Programming,  Brian W. Kernighan, Rob Pike  */

static int matchhere(char*,char*);
static int matchstar(int c, char *re, char *text) {
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text != '\0' && (*text++ == c || c== '.'));
  return 0;
}

static int matchhere(char *re, char *text) {
  if (re[0] == '\0')
     return 0;
  if (re[1] == '*')
    return matchstar(re[0], re+2, text);
  if (re[0] == '$' && re[1]=='\0')
    return *text == '\0';
  if (*text!='\0' && (re[0]=='.' || re[0]==*text))
    return matchhere(re+1, text+1);
  return 0;
}

int match(char *re, char *text) {
  if (re[0] == '^')
    return matchhere(re+1, text);
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text++ != '\0');
  return 0;
}
```

~~Programming~~ by example: FlashFIll

Regular Expressions

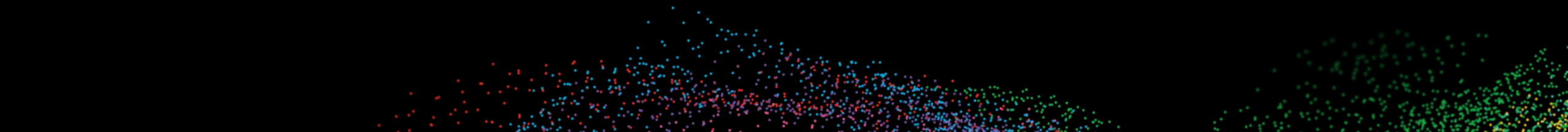
John DOE 3 Data [TS]**865-000-0000** - - 453442-00 06-23-2009
A FF MARILYN 30'S **865-000-0030** 4535871-00 07-07-2009
A GEDA-MARY 100MG **865-001-0020** - - 5941-00 06-23-2009

???-???-????

Program Synthesis

Neural Turing Machine: Attach memory network to LSTM (Graves et al 2014)

```
initialise: move head to start location
while input delimiter not seen do
    receive input vector
    write input to head location
    increment head location by 1
end while
return head to start location
while true do
    read output vector from head location
    emit output
    increment head location by 1
end while
```

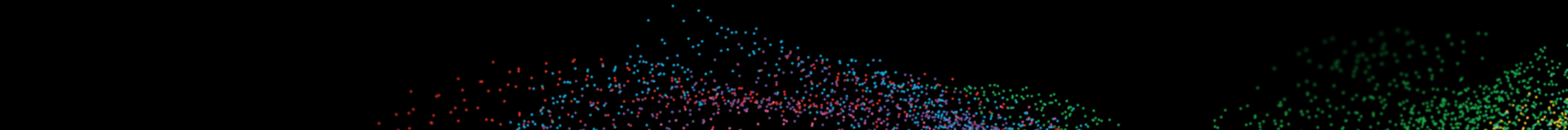


Program Synthesis

Neural Program Interpreters: Fully supervised execution traces (Reed & de Freitas 2016)

```
bubblesort(x):
  for(i=count-2;i>=0;i--){
      for(j=0;j<=i;j++){
        if(number[j]>number[j+1]){
           temp=number[j];
           number[j]=number[j+1];
           number[j+1]=temp;
        }
      }
   }
```

… Trained on length 20 examples, works up to length 60 examples



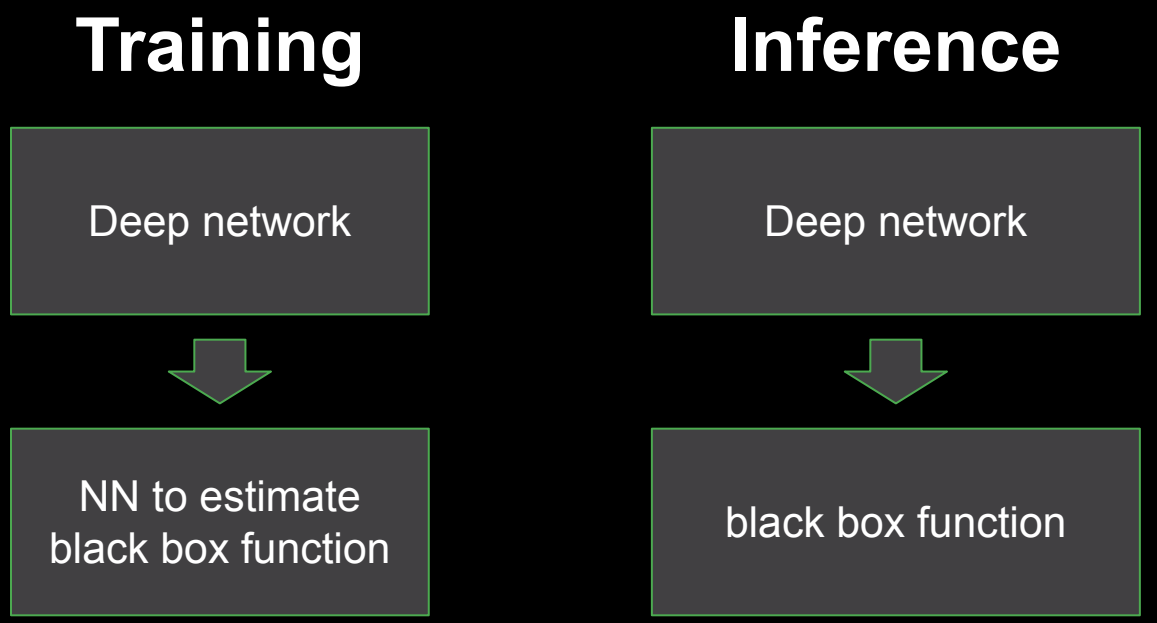
Learning to Infer and Execute (3D Shape) Programs (Tian et al 2019)

Infer shape program

Execute shape program

```
Draw("Top", "Circle", position, geometry)

for(i < 2, "translation", a)
   for(j < 2, "translation", b)
     Draw("Leg", "Cub", position + i*a + j*b, geometry)

for(i < 2, "translation", c)
   Draw("Layer", "Rec", position + i*c, geometry)
```

Learning NN with black-box functions inside of them (Jacovi et al 2019)

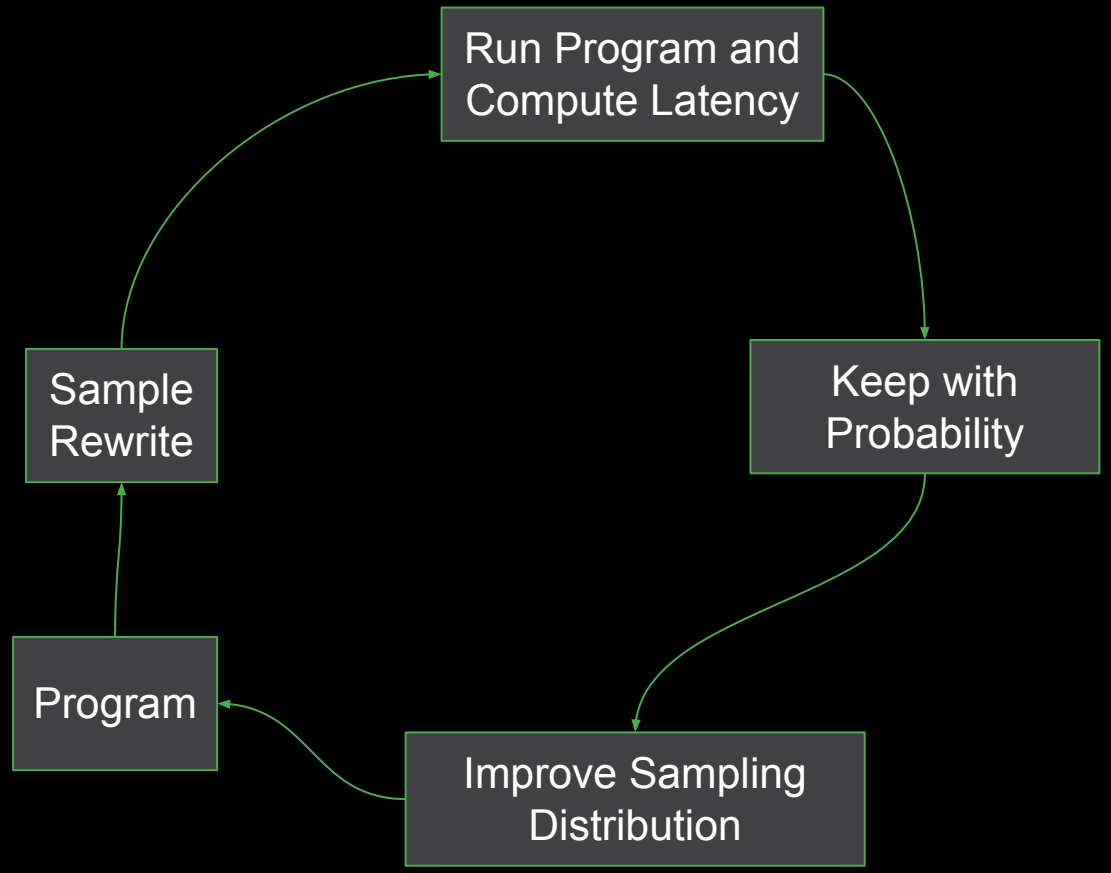


Assembly Code (LLVM)

```
/*  Simple regular expression matching.  From:  The Practice of Programming,
Brian W. Kernighan, Rob Pike  */

static int matchhere(char*,char*);
static int matchstar(int c, char *re, char *text) {
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text != '\0' && (*text++ == c || c== '.'));
  return 0;
}

static int matchhere(char *re, char *text) {
  if (re[0] == '\0')
     return 0;
  if (re[1] == '*')
    return matchstar(re[0], re+2, text);
  if (re[0] == '$' && re[1]=='\0')
    return *text == '\0';
  if (*text!='\0' && (re[0]=='.' || re[0]==*text))
    return matchhere(re+1, text+1);
  return 0;
}

int match(char *re, char *text) {
  if (re[0] == '^')
    return matchhere(re+1, text);
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text++ != '\0');
  return 0;
}
```

```
; Function Attrs: noinline nounwind optnone
define i32 @match(i8*, i8*) #0 {
 %3 = alloca i32, align 4
 %4 = alloca i8*, align 4
 %5 = alloca i8*, align 4
 store i8* %0, i8** %4, align 4
 store i8* %1, i8** %5, align 4
 %6 = load i8*, i8** %4, align 4
 %7 = getelementptr inbounds i8, i8* %6, i32 0
 %8 = load i8, i8* %7, align 1
 %9 = sext i8 %8 to i32
 %10 = icmp eq i32 %9, 94
 br i1 %10, label %11, label %16

; <label>:11:                                     ; preds = %2
 %12 = load i8*, i8** %4, align 4
 %13 = getelementptr inbounds i8, i8* %12, i32 1
 %14 = load i8*, i8** %5, align 4
 %15 = call i32 @matchhere(i8* %13, i8* %14)
 store i32 %15, i32* %3, align 4
 br label %31

; <label>:16:                                     ; preds = %2
 br label %17

; <label>:17:                                     ; preds = %24, %16
 %18 = load i8*, i8** %4, align 4
 %19 = load i8*, i8** %5, align 4
 %20 = call i32 @matchhere(i8* %18, i8* %19)
 %21 = icmp ne i32 %20, 0
 br i1 %21, label %22, label %23

; <label>:22:                                     ; preds = %17
 store i32 1, i32* %3, align 4
 br label %31

; <label>:23:                                     ; preds = %17
 br label %24

; <label>:24:                                     ; preds = %23
 %25 = load i8*, i8** %5, align 4
 %26 = getelementptr inbounds i8, i8* %25, i32 1
 store i8* %26, i8** %5, align 4
 %27 = load i8, i8* %25, align 1
 %28 = sext i8 %27 to i32
 %29 = icmp ne i32 %28, 0
 br i1 %29, label %17, label %30

; <label>:30:                                     ; preds = %24
 store i32 0, i32* %3, align 4
 br label %31

; <label>:31:                                     ; preds = %30, %22, %11
 %32 = load i32, i32* %3, align 4
 ret i32 %32
}
```

Learning to super-optimize programs (Bunel et al 2017)



Neural Decompilation (Katz et al 2019)

How to recover initial code?

- Variable Names?
- While loops vs. GOTOs?

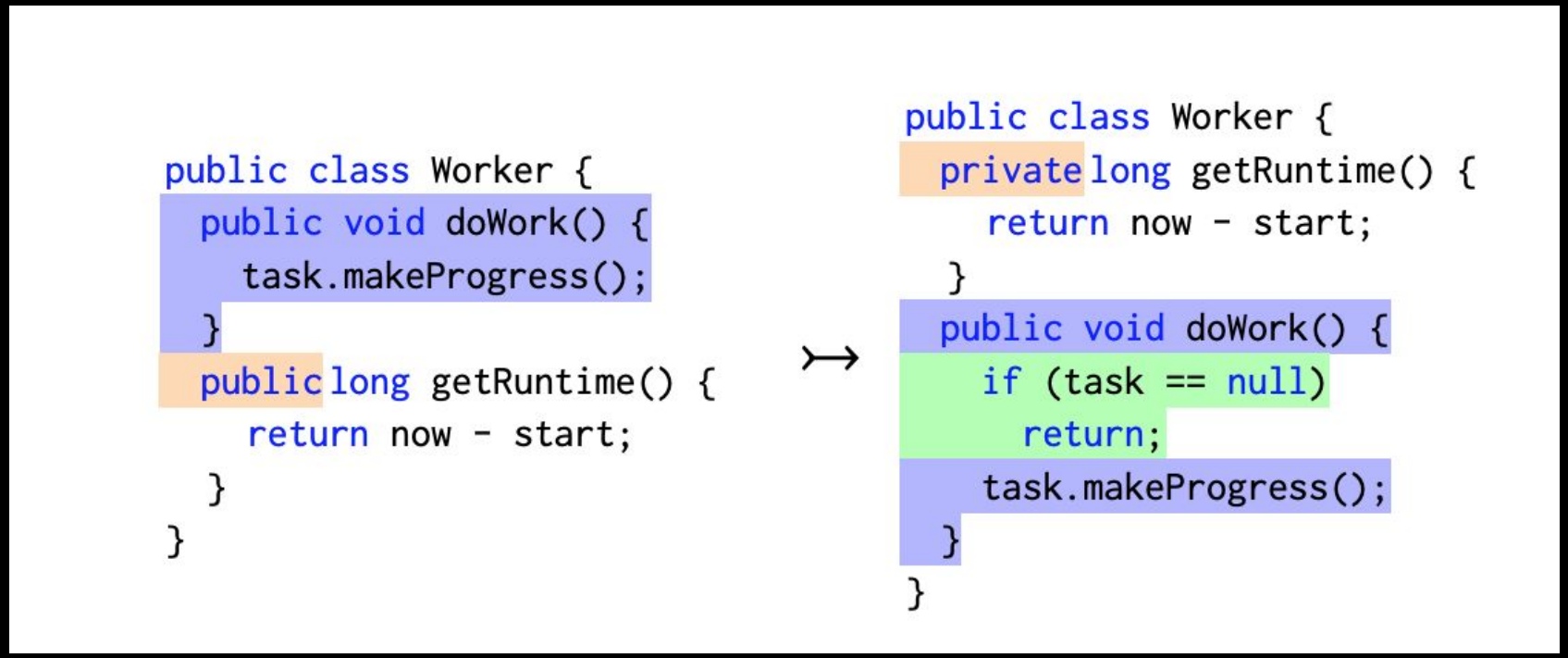
Approach A: use Neural Machine Translation

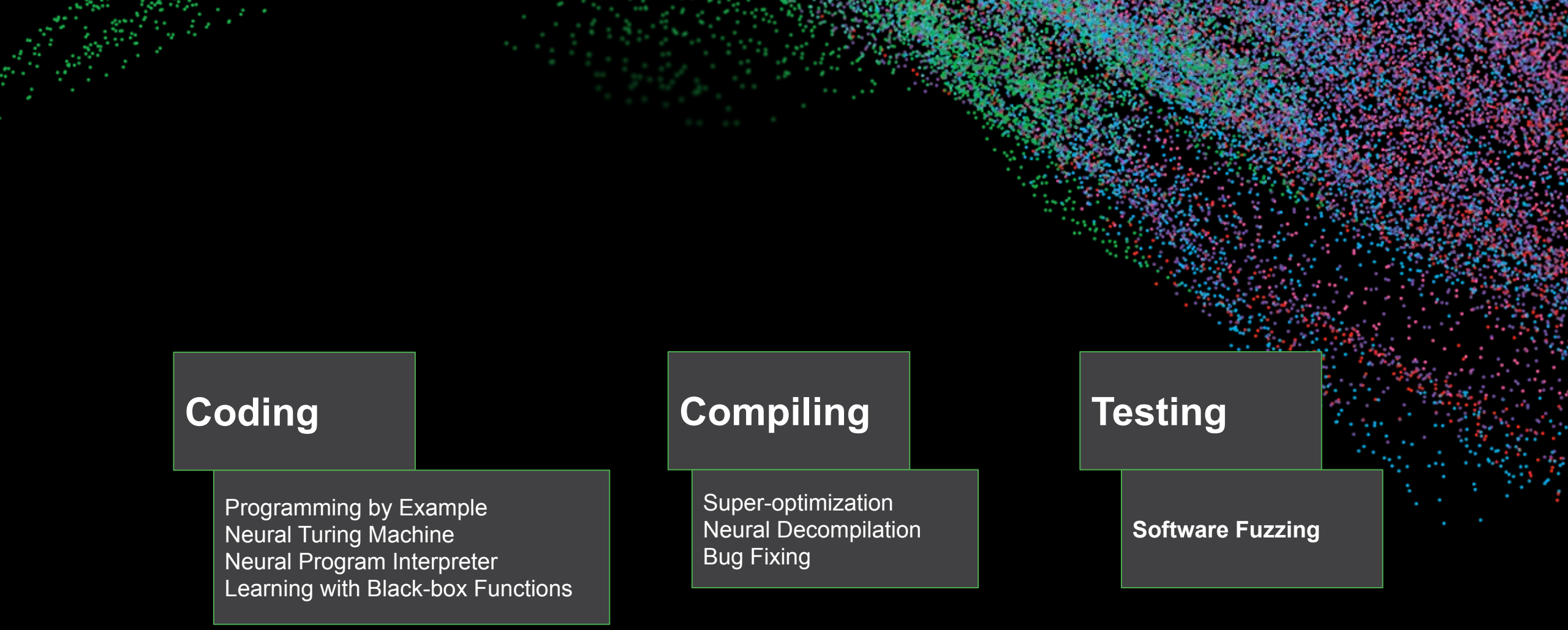
Approach B: use NMT to convert code into templates, and then use insert/compile/test to fill in these templates

```
; Function Attrs: noinline nounwind optnone
define i32 @match(i8*, i8*) #0 {
 %3 = alloca i32, align 4
 %4 = alloca i8*, align 4
 %5 = alloca i8*, align 4
 store i8* %0, i8** %4, align 4
 store i8* %1, i8** %5, align 4
 %6 = load i8*, i8** %4, align 4
 %7 = getelementptr inbounds i8, i8* %6, i32 0
 %8 = load i8, i8* %7, align 1
 %9 = sext i8 %8 to i32
 %10 = icmp eq i32 %9, 94
 br i1 %10, label %11, label %16

; <label>:11:                                     ; preds = %2
 %12 = load i8*, i8** %4, align 4
 %13 = getelementptr inbounds i8, i8* %12, i32 1
 %14 = load i8*, i8** %5, align 4
 %15 = call i32 @matchhere(i8* %13, i8* %14)
 store i32 %15, i32* %3, align 4
 br label %31

; <label>:16:                                     ; preds = %2
 br label %17

; <label>:17:                                     ; preds = %24, %16
 %18 = load i8*, i8** %4, align 4
 %19 = load i8*, i8** %5, align 4
 %20 = call i32 @matchhere(i8* %18, i8* %19)
 %21 = icmp ne i32 %20, 0
 br i1 %21, label %22, label %23

; <label>:22:                                     ; preds = %17
 store i32 1, i32* %3, align 4
 br label %31

; <label>:23:                                     ; preds = %17
 br label %24

; <label>:24:                                     ; preds = %23
 %25 = load i8*, i8** %5, align 4
 %26 = getelementptr inbounds i8, i8* %25, i32 1
 store i8* %26, i8** %5, align 4
 %27 = load i8, i8* %25, align 1
 %28 = sext i8 %27 to i32
 %29 = icmp ne i32 %28, 0
 br i1 %29, label %17, label %30

; <label>:30:                                     ; preds = %24
 store i32 0, i32* %3, align 4
 br label %31

; <label>:31:                                     ; preds = %30, %22, %11
 %32 = load i32, i32* %3, align 4
 ret i32 %32
}
```

Bug Fixing

DeepFix (Gupta et al 2017)
Getafix (Bader et al 2019)




Coding
Programming by Example
Neural Turing Machine
Neural Program Interpreter
Learning with Black-box Functions
Compiling
Super-optimization
Neural Decompilation
Bug Fixing
Testing
Software Fuzzing

Fuzzing

Given an arbitrary binary, discover the inputs that cause program crashes

Discover the inputs that exercise as much program behavior as possible

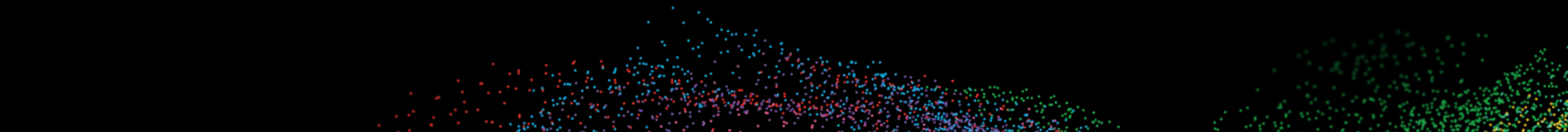
```
; Function Attrs: noinline nounwind optnone
define i32 @match(i8*, i8*) #0 {
 %3 = alloca i32, align 4
 %4 = alloca i8*, align 4
 %5 = alloca i8*, align 4
 store i8* %0, i8** %4, align 4
 store i8* %1, i8** %5, align 4
 %6 = load i8*, i8** %4, align 4
 %7 = getelementptr inbounds i8, i8* %6, i32 0
 %8 = load i8, i8* %7, align 1
 %9 = sext i8 %8 to i32
 %10 = icmp eq i32 %9, 94
 br i1 %10, label %11, label %16

; <label>:11:                                     ; preds = %2
 %12 = load i8*, i8** %4, align 4
 %13 = getelementptr inbounds i8, i8* %12, i32 1
 %14 = load i8*, i8** %5, align 4
 %15 = call i32 @matchhere(i8* %13, i8* %14)
 store i32 %15, i32* %3, align 4
 br label %31

; <label>:16:                                     ; preds = %2
 br label %17

; <label>:17:                                     ; preds = %24, %16
 %18 = load i8*, i8** %4, align 4
 %19 = load i8*, i8** %5, align 4
 %20 = call i32 @matchhere(i8* %18, i8* %19)
 %21 = icmp ne i32 %20, 0
 br i1 %21, label %22, label %23

; <label>:22:                                     ; preds = %17
 store i32 1, i32* %3, align 4
 br label %31

; <label>:23:                                     ; preds = %17
 br label %24

; <label>:24:                                     ; preds = %23
 %25 = load i8*, i8** %5, align 4
 %26 = getelementptr inbounds i8, i8* %25, i32 1
 store i8* %26, i8** %5, align 4
 %27 = load i8, i8* %25, align 1
 %28 = sext i8 %27 to i32
 %29 = icmp ne i32 %28, 0
 br i1 %29, label %17, label %30

; <label>:30:                                     ; preds = %24
 store i32 0, i32* %3, align 4
 br label %31

; <label>:31:                                     ; preds = %30, %22,
%11
 %32 = load i32, i32* %3, align 4
 ret i32 %32
}
```

White-box vs. Black-box vs. Grey-box Fuzzing

- **White-box:** Assume lots of transparency (e.g., KLEE)
  - Suffers from path explosion as programs get big

- **Black-box:** Assume no transparency into program
  - Random generators (good for testing parsers)

- **Grey-box**: Minimal transparency into the program



Grey-Box Mutational Fuzzer: AFL

- Mutate existing seed(s) to generate new test inputs
  - Light instrumentation to check code paths
  - Execute test input, add to set of seeds if new behavior
- AFL is best of the bunch!
  - Uses heuristics to pick inputs
  - Randomness for mutation

```
1:  // Core Algorithm for American Fuzzy Lop (AFL)
2:  // time: Fixed time window to fuzz (e.g. 24 hours)
3:  // queue: Queue of inputs that exercise new code paths.
4:  while time has not elapsed do
5:      parent, energy ← pick_input(queue)
6:      for i ∈ range(energy) do
7:          child ← parent
8:          for j ∈ 1 to sample_num_mutations() do
9:              mutation ← sample_mutation()
10:             site ← sample_mutation_site()
11:             child ← apply_mutation(mutation, child, site)
12:         end for
13:         path ← execute_path(child, code)
14:         if (path is new) then queue ← child
15:     end for
16: end while
```

AFL: Directly Instrument Code Binary

Parent process runs an instrumented child thread with a shared memory block. At every basic block:

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

After each run, the parent inspects the shared memory to see what code paths the child has traversed

```
/*  Simple regular expression matching.  From:  The Practice of Programming,
Brian W. Kernighan, Rob Pike  */

static int matchhere(char*,char*);
static int matchstar(int c, char *re, char *text) {
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text != '\0' && (*text++ == c || c== '.'));
  return 0;
}

static int matchhere(char *re, char *text) {
  if (re[0] == '\0')
     return 0;
  if (re[1] == '*')
    return matchstar(re[0], re+2, text);
  if (re[0] == '$' && re[1]=='\0')
    return *text == '\0';
  if (*text!='\0' && (re[0]=='.' || re[0]==*text))
    return matchhere(re+1, text+1);
  return 0;
}

int match(char *re, char *text) {
  if (re[0] == '^')
    return matchhere(re+1, text);
  do {
    if (matchhere(re, text))
      return 1;
  } while (*text++ != '\0');
  return 0;
}
```

AFL: Serious Bugs Found

IJG jpeg, libjpeg-turbo, libpng, libtiff, mozjpeg, PHP, Mozilla, Firefox, Internet Explorer, Apple Safari, Adobe Flash / PCRE, sqlite, OpenSSL, LibreOffice, poppler, freetype, GnuTLS, GnuPG, OpenSSH, PuTTY, ntpd, nginx, bash (post-Shellshock), tcpdump, JavaScriptCore, pdfium, ffmpeg, libmatroska, libarchive, wireshark, ImageMagick, BIND, QEMU, lcms, Oracle BerkeleyDB, Android / libstagefright, iOS / ImageIO, FLAC audio library, libsndfile, less / lesspipe, strings (+ related tools), file, dpkg, rcs, systemd-resolved, libyaml, Info-Zip unzip, libtasn1, OpenBSD pfctl, NetBSD bpf, man & mandoc, IDA Pro [reported by authors] clamav, libxml2, glibc, clang / llvm, nasm, ctags, mutt, procmail, fontconfig, pdksh, Qt, wavpack, redis / lua-cmsgpack, taglib, privoxy, perl, libxmp, radare2, SleuthKit, fwknop [reported by author], X.Org, exifprobe, jhead [?], capnproto, Xerces-C, metacam, djvulibre, exiv, Linux btrfs, Knot DNS, curl, wpa_supplicant, libde265 [reported by author], dnsmasq, libbpg, lame, libwmf, uudecode, MuPDF, imlib2, libraw, libbson, libsass, yara, W3C tidy-html5, VLC, FreeBSD syscons, John the Ripper, screen, tmux, mosh, UPX, indent, openjpeg, MMIX, OpenMPT, rxvt, dhcpcd, Mozilla NSS, Nettle, mbed TLS, Linux netlink, Linux ext4, Linux xfs, botan, expat, Adobe Reader, libav, libical, OpenBSD kernel, collectd, libidn, MatrixSSL, jasper, MaraDNS, w3m, Xen, OpenH232, irssi, cmark, OpenCV, Malheur, gstreamer, Tor, gdk-pixbuf, audiofile, zstd, lz4, stb, cJSON, libpcre, MySQL, gnulib, openexr, libmad, ettercap, lrzip, freetds, Asterisk, ytnef, raptor, mpg123, Apache, httpd, exempi, libgmime, pev, Linux mem mgmt, sleuthkit, Mongoose OS, iOS kernel

. . .

Approach 1: Treat AFL as a stochastic process

```
1:  // Core Algorithm for American Fuzzy Lop (AFL)
2:  // time: Fixed time window to fuzz (e.g. 24 hours)
3:  // queue: Queue of inputs that exercise new code paths.
4:  while time has not elapsed do
5:      parent, energy ← pick_input(queue)
6:      for i ∈ range(energy) do
7:          child ← parent
8:          for j ∈ 1 to sample_num_mutations() do
9:              mutation ← sample_mutation()
10:             site ← sample_mutation_site()
11:             child ← apply_mutation(mutation, child, site)
12:         end for
13:         path ← execute_path(child, code)
14:         if (path is new) then queue ← child
15:     end for
16: end while
```

AFL samples uniformly at random

Why is this the right distribution?

Thompson Sampling (aka Posterior Sampling)

Assume:

1. Likelihood function: $P(r|\theta, a, x)$
2. Posterior: $P(\theta|D) \propto P(D|\theta)P(\theta)$
3. Take a sample: $\theta^\star \sim P(\theta|D)$
4. Choose $a^\star = \arg\max_a E[r|\theta^\star, a, x]$

Thompson Sampling (aka Posterior Sampling)

a = mutation operator

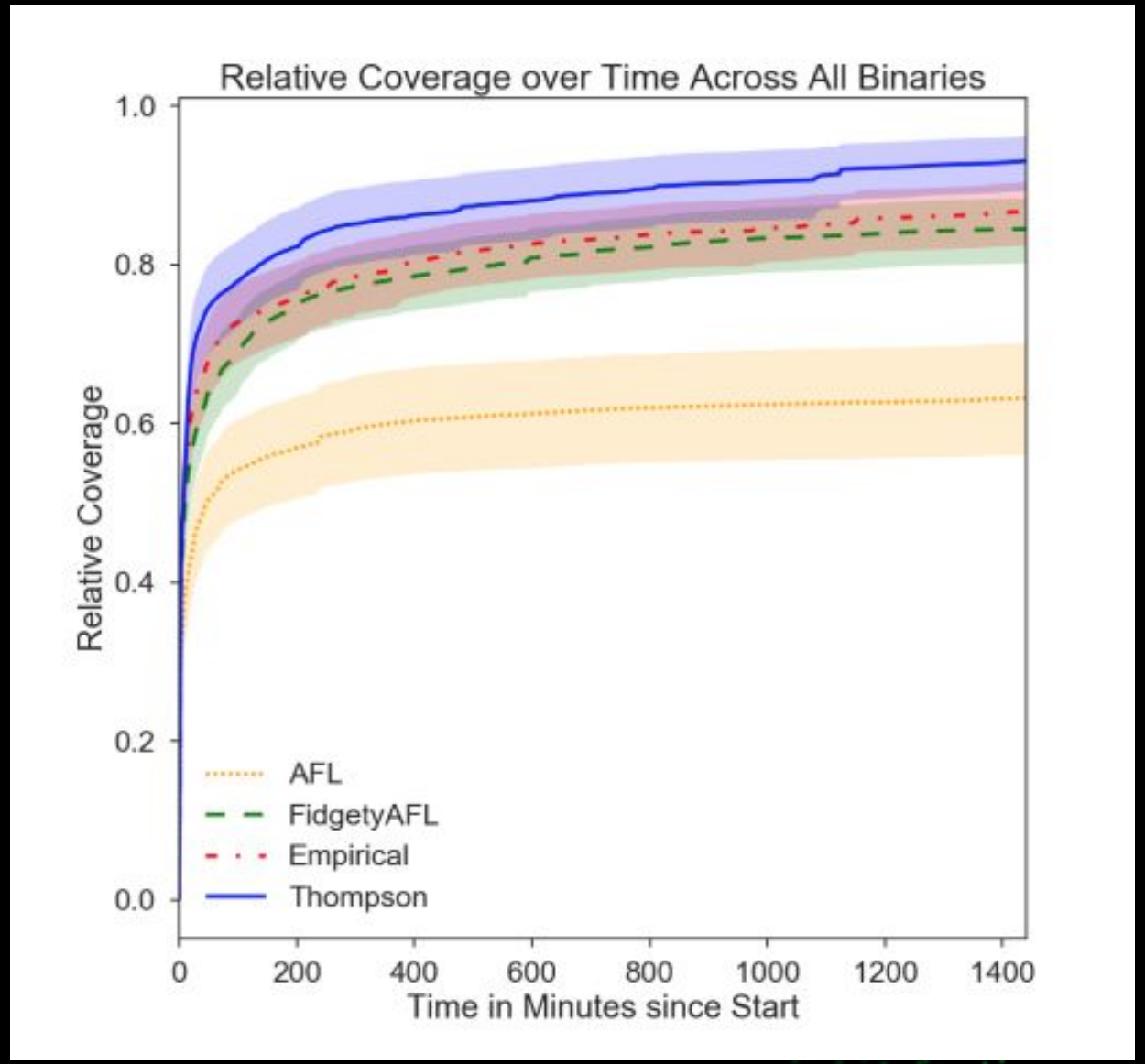
Assume:

1. Likelihood function: $P(r|\theta, a, x)$
2. Posterior: $P(\theta|D) \propto P(D|\theta)P(\theta)$
3. Take a sample: $\theta^\star \sim P(\theta|D)$
4. Choose $a^\star = \arg\max_a E[r|\theta^\star, a, x]$

r = does this find a new queue element?

Thompson Sampling: Results

- **We use the DARPA Cyber Grand Challenge Binaries**
  - Set of 200 binaries released for testing bug discovery + patching



CITATION

Approach 2: Treat AFL as function learning

```
1:  // Core Algorithm for American Fuzzy Lop (AFL)
2:  // time: Fixed time window to fuzz (e.g. 24 hours)
3:  // queue: Queue of inputs that exercise new code paths.
4:  while time has not elapsed do
5:      parent, energy ← pick_input(queue)
6:      for i ∈ range(energy) do
7:          child ← parent
8:          for j ∈ 1 to sample_num_mutations() do
9:              mutation ← sample_mutation()
10:             site ← sample_mutation_site()
11:             child ← apply_mutation(mutation, child, site)
12:         end for
13:         path ← execute_path(child, code)
14:         if (path is new) then queue ← child
15:     end for
16: end while
```

Previously only optimized the new child creation process. Is it possible to build a model that assesses the completed child itself?

Function: Input strings -> Program paths

- For every input AFL executes, it stores the set of control flow graph edges traversed
  - Treat string input as **x**, set of edges as **y**
  - Fit a model using a classifier of your choice —> just needs to output probabilities!
  - Can update model online! After executing new inputs, train on resulting data!

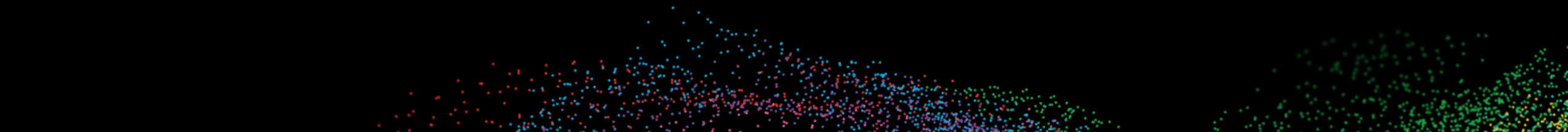
- In our work, we *keep it simple:*
  - *Featurizer:* Bag of Bytes (0 - 255) for encoding input strings
  - *Classifier:* Logistic Regression

Program Testing as Active Learning

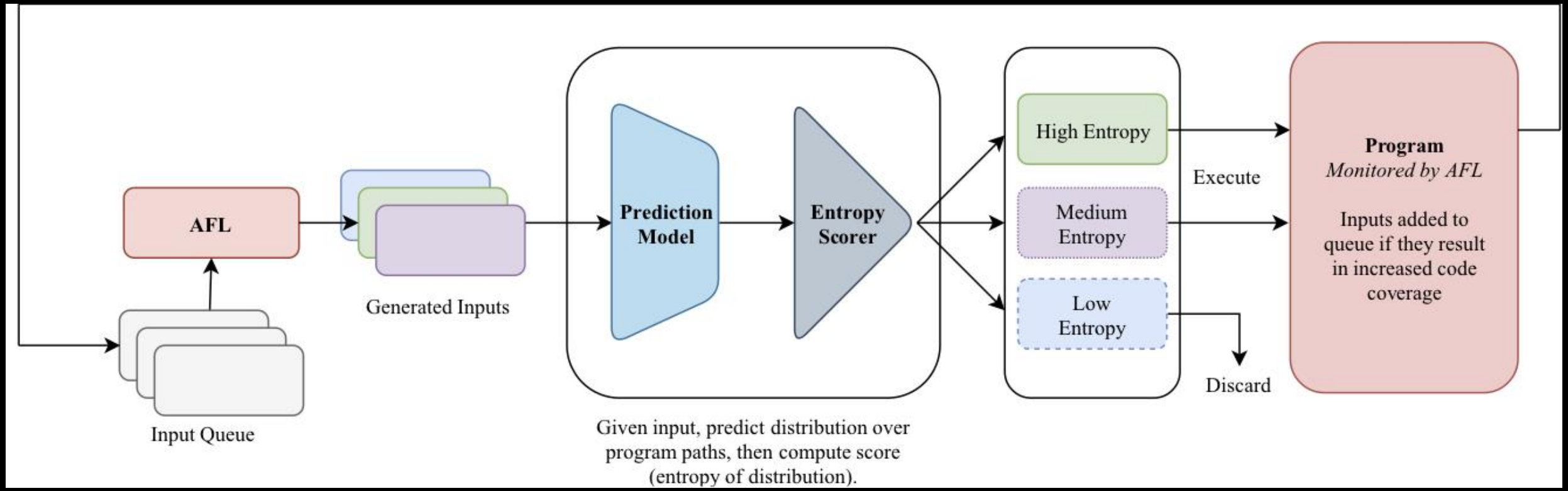
- **We have a (incomplete) model that maps inputs to a distribution over control flow paths**

- Use Entropy to rank candidates:

$$\sum P(y|x) log P(y|x)$$

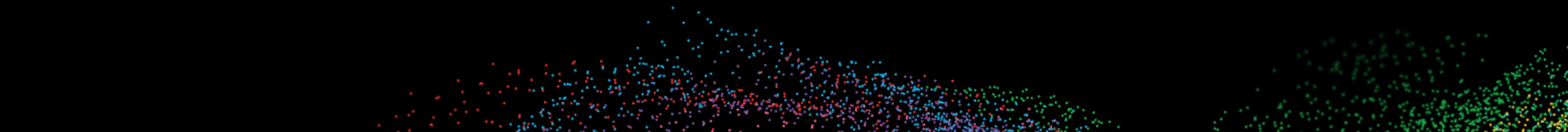


Fuzzing by Modeling Program Behavior

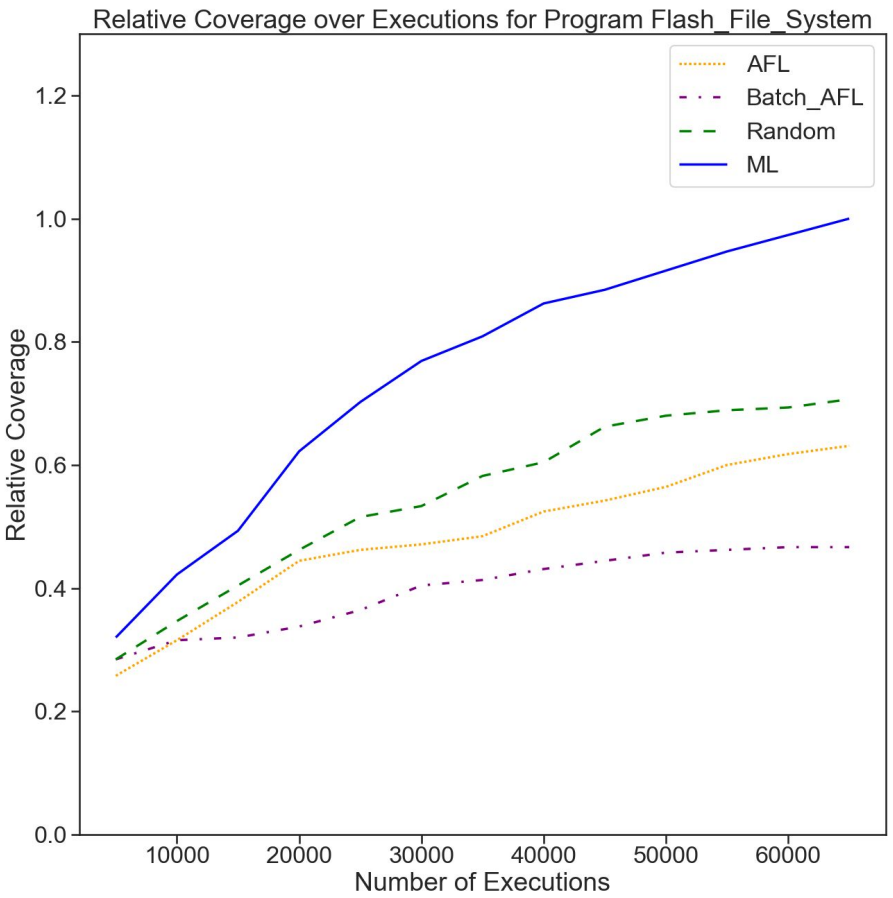


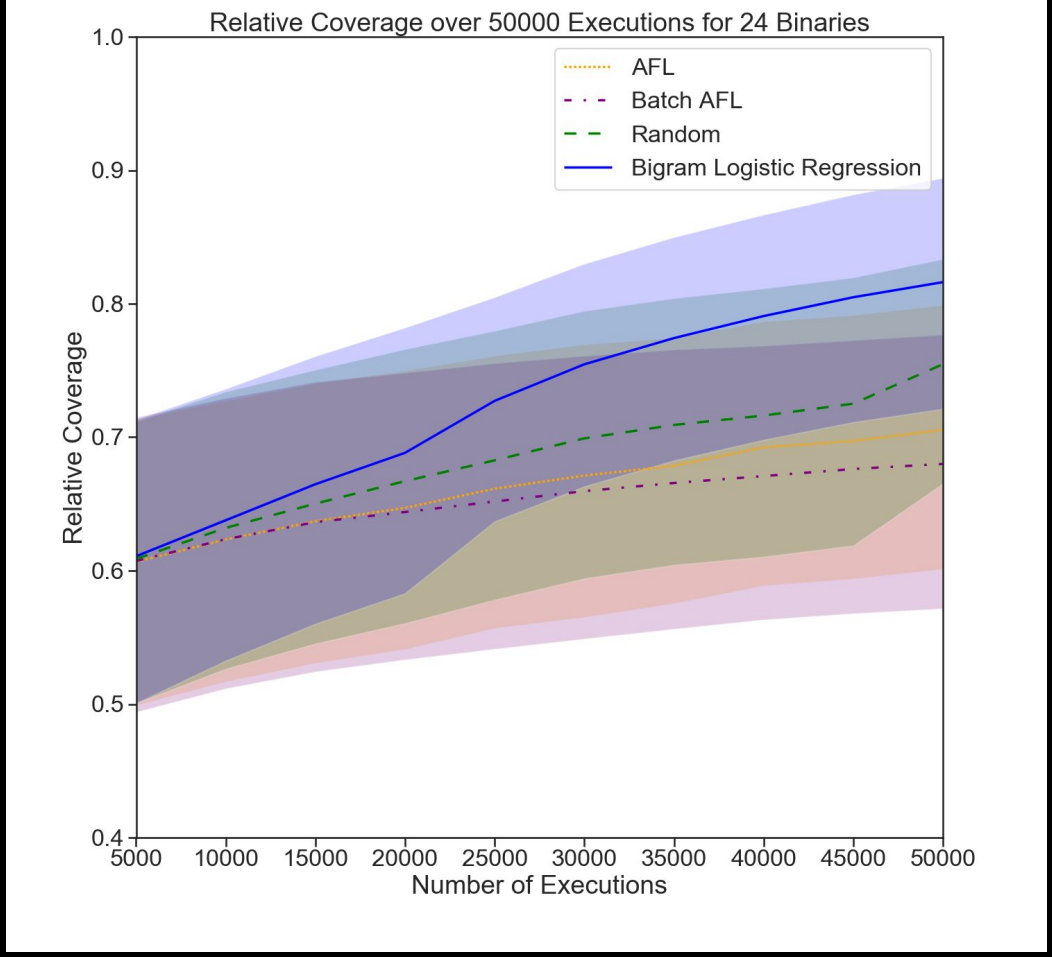
Preliminary Experiments - Baselines

- Run AFL for 3 minutes to warm-start/populate queue
- Start the following 4 Strategies using the resulting queue:
  1. **AFL:** Generates an input then immediately executes
  2. **Batch-AFL:** Meant to mimic our program modeling procedure.
  — Generate 50,000 inputs, execute the first 5,000 (slightly different than standard AFL due to the heuristic sampling seed inputs)
  3. **Random Batch-AFL:** Generate 50,000 inputs, select random 5,000
  4. **ML/Logistic Regression:** Generate 50,000, use model to pick 5,000

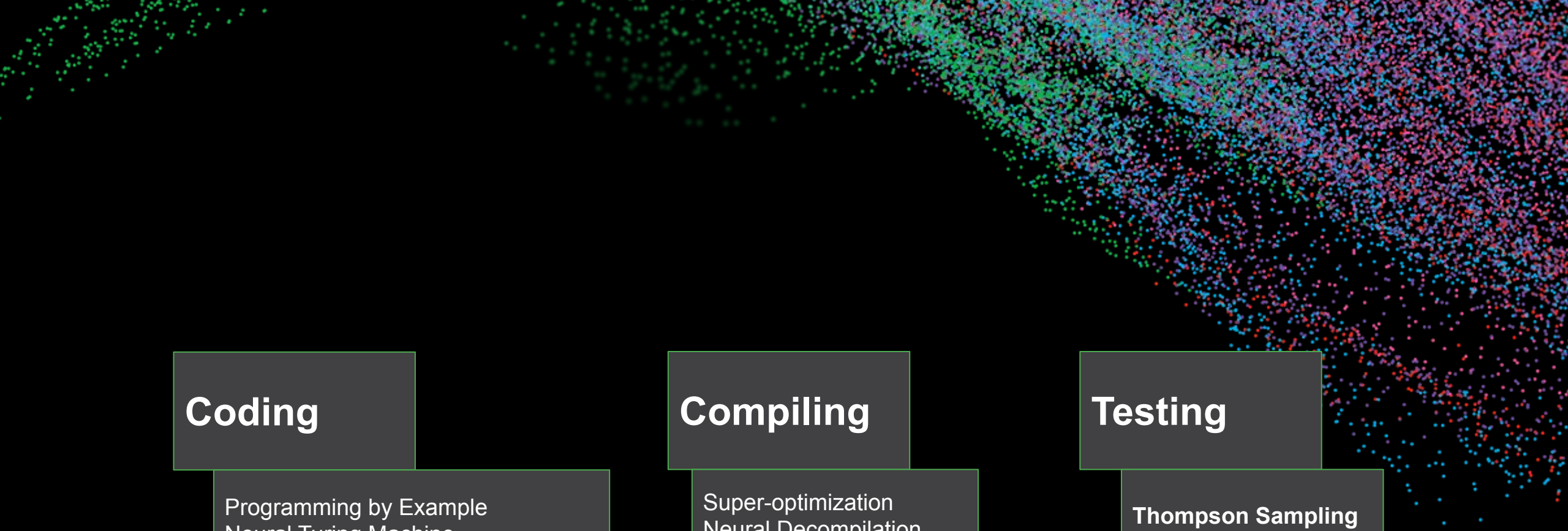


Results on 24 CGC Binaries



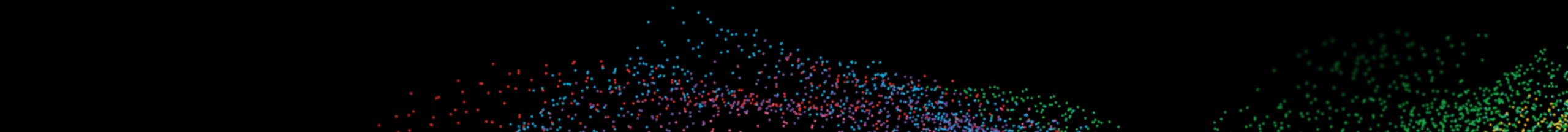


CITATION


Coding
Programming by Example
Neural Turing Machine
Neural Program Interpreter
Learning with Black-box Functions
Compiling
Super-optimization
Neural Decompilation
Bug Fixing
Testing
Thompson Sampling
Program Modeling

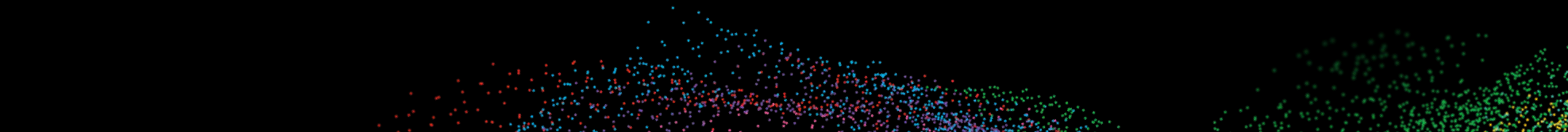
Conclusion

- Reinforcement Learning is a powerful way to replace heuristic decisions in deployed infrastructure
- Black-box functions approximation is a key technique for understanding and modeling programs



Larger Conclusion

Machine Learning on Programs is in its infancy
compared to progress in other domains



Thank you!

https://www.bloomberg.com/careers

Questions?

Bloomberg

Engineering

TechAtBloomberg.com