



CtxFuzz

Context-Aware Directed Fuzzing via Runtime Fuzzer-Agent Cooperation

Jiho Kim¹, Dongkwan Kim¹, HyungSeok Han², Youngjoon Kim¹,
Soyeon Park³, Sangwoo Ji⁴, Joshua Wang¹, Dae R. Jeong⁵, Taesoo Kim¹

¹ Georgia Institute of Technology ² Microsoft ³ Independent Researcher
⁴ Samsung Electronics ⁵ Seoul National University



ASE 2026, Munich, Germany

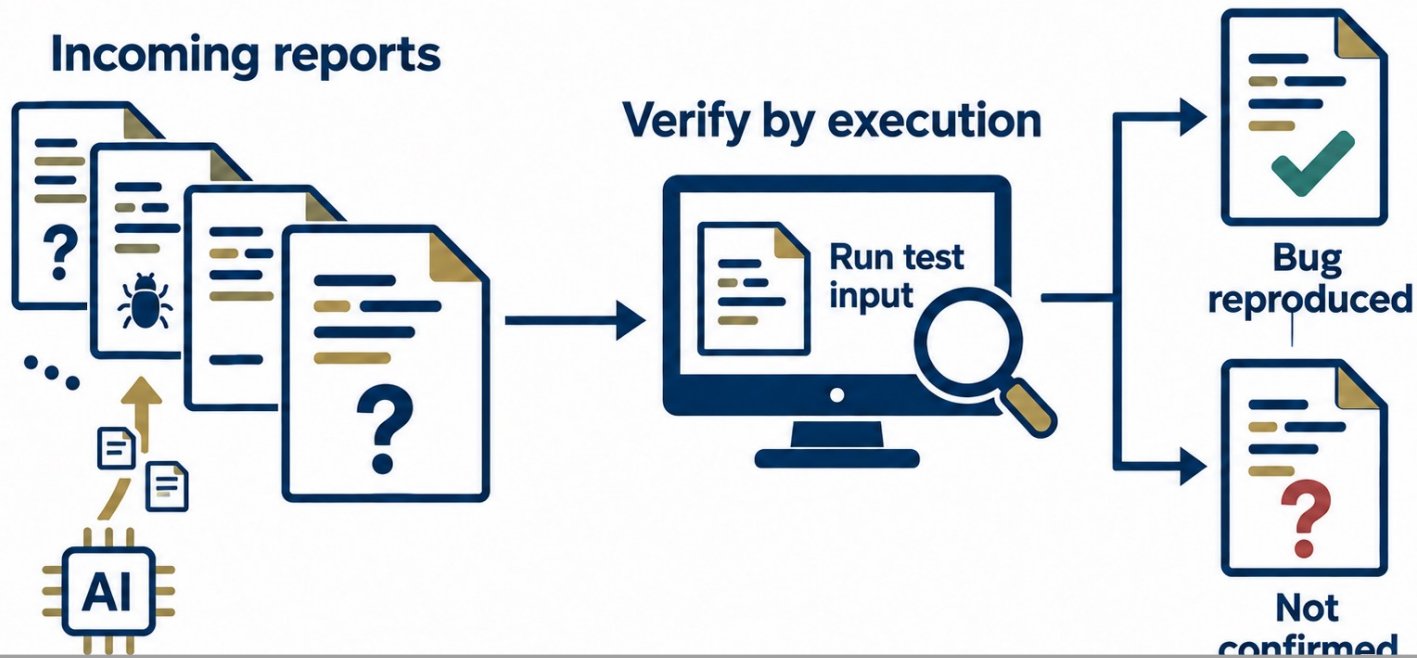


Team Atlanta: AlxCC 2025 Winner



From an AlxCC-winning component to CtxFuzz

More reports, more work to check them



>334%

**Triage queue growth
in three weeks**

Bugcrowd, Mar. 10, 2026

Surge in unverified AI submissions

>15% to <5%

Reports confirmed as real bugs

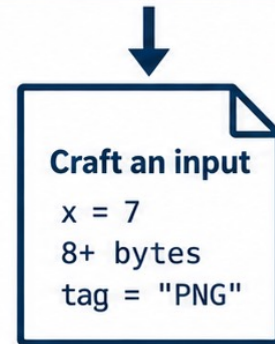
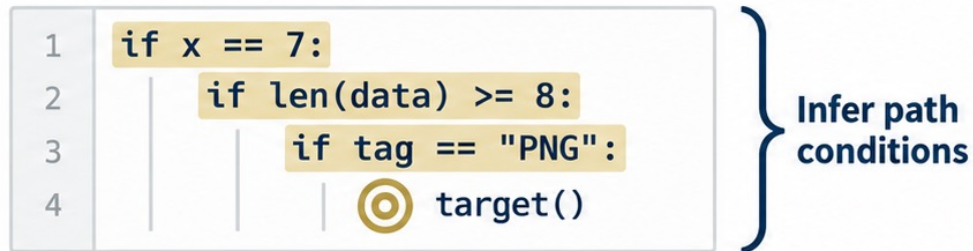
Earlier years vs. 2025

**How can we turn a vulnerability report into an input
that actually triggers the bug?**

LLM agents vs. directed fuzzers

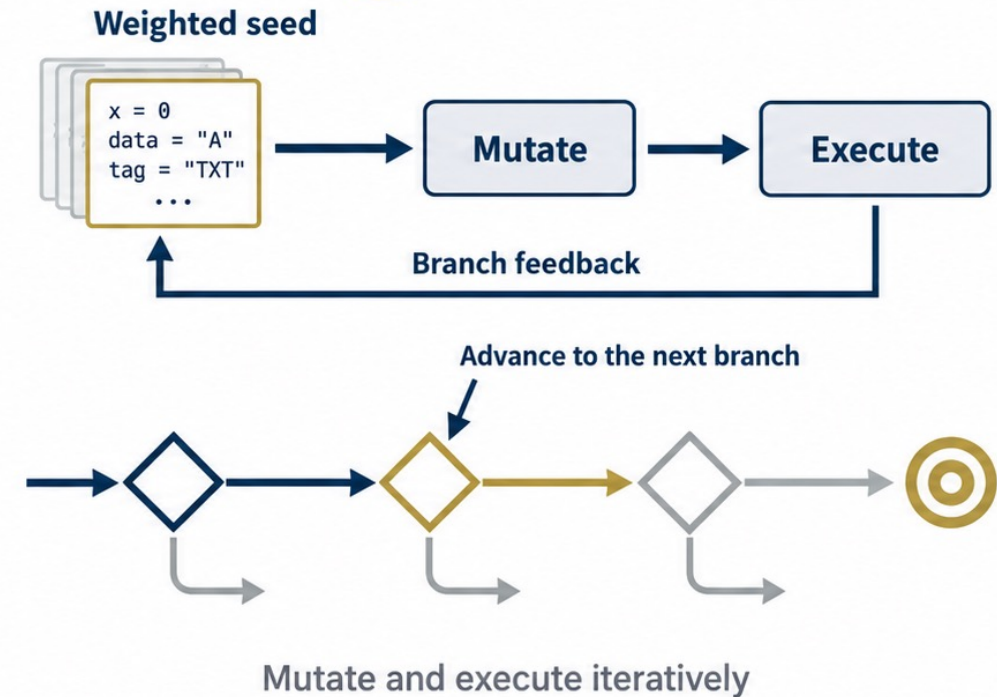
Two common approaches to vulnerability reproduction

LLM agent



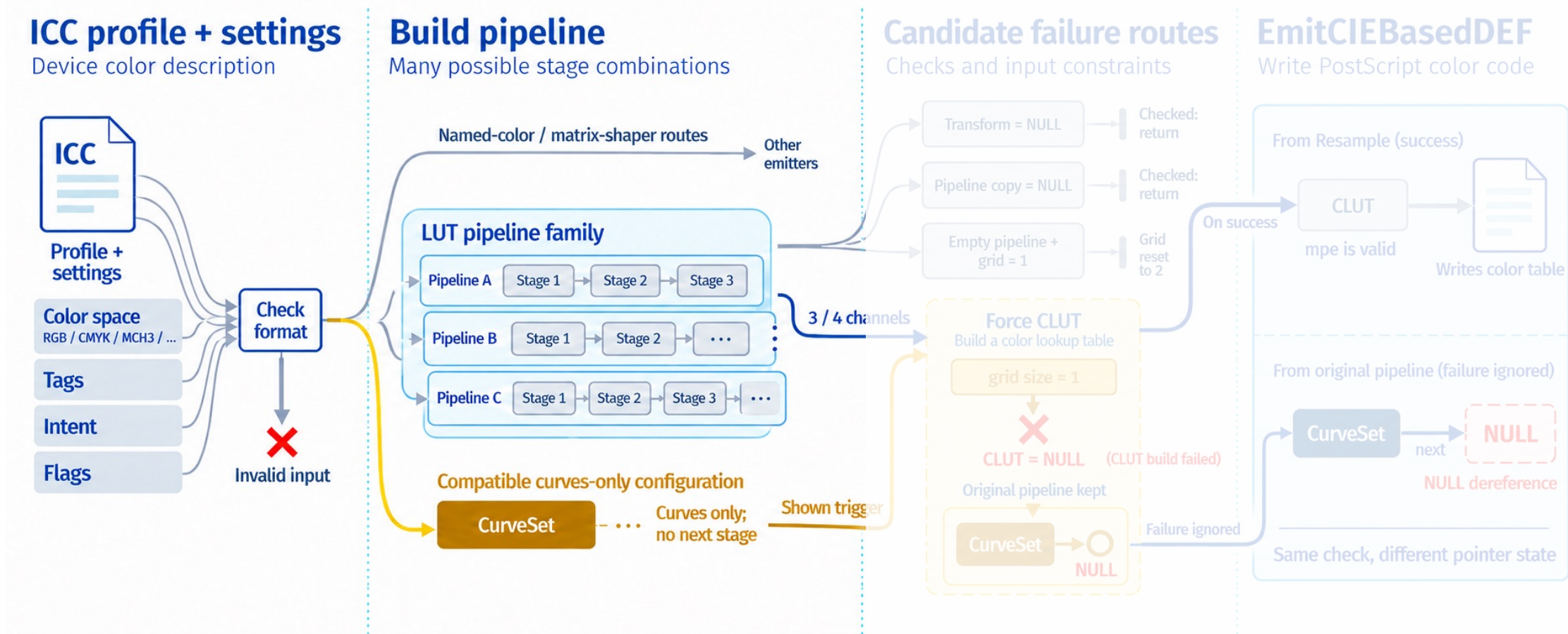
Reason about the full path

Directed fuzzer



Motivation: finding a path to the NULL state

Little CMS is a color management system (CMS) that maps colors between devices.



Profiles and settings choose among many color-conversion pipelines.

Motivation: finding a path to the NULL state

The PostScript emitter expects a color lookup table (CLUT) on this route.

ICC profile + settings

Device color description



Profile + settings

Color space

RGB / CMYK / MCH3 / ...

Tags

Intent

Flags

Check format

Invalid input

Build pipeline

Many possible stage combinations

Named-color / matrix-shaper routes

Other emitters

LUT pipeline family



Compatible curves-only configuration

CurveSet

Curves only;
no next stage

Shown trigger

Candidate failure routes

Checks and input constraints

Transform = NULL

Checked:
return

Pipeline copy = NULL

Checked:
return

Empty pipeline +
grid = 1

Grid
reset
to 2

Force CLUT

Build a color lookup table

grid size = 1

CLUT = NULL (CLUT build failed)

Original pipeline kept

CurveSet

NULL

Failure ignored

EmitCIEBasedDEF

Write PostScript color code

From Resample (success)

CLUT

mpe is valid

Writes color table

From original pipeline (failure ignored)

CurveSet

next

NULL

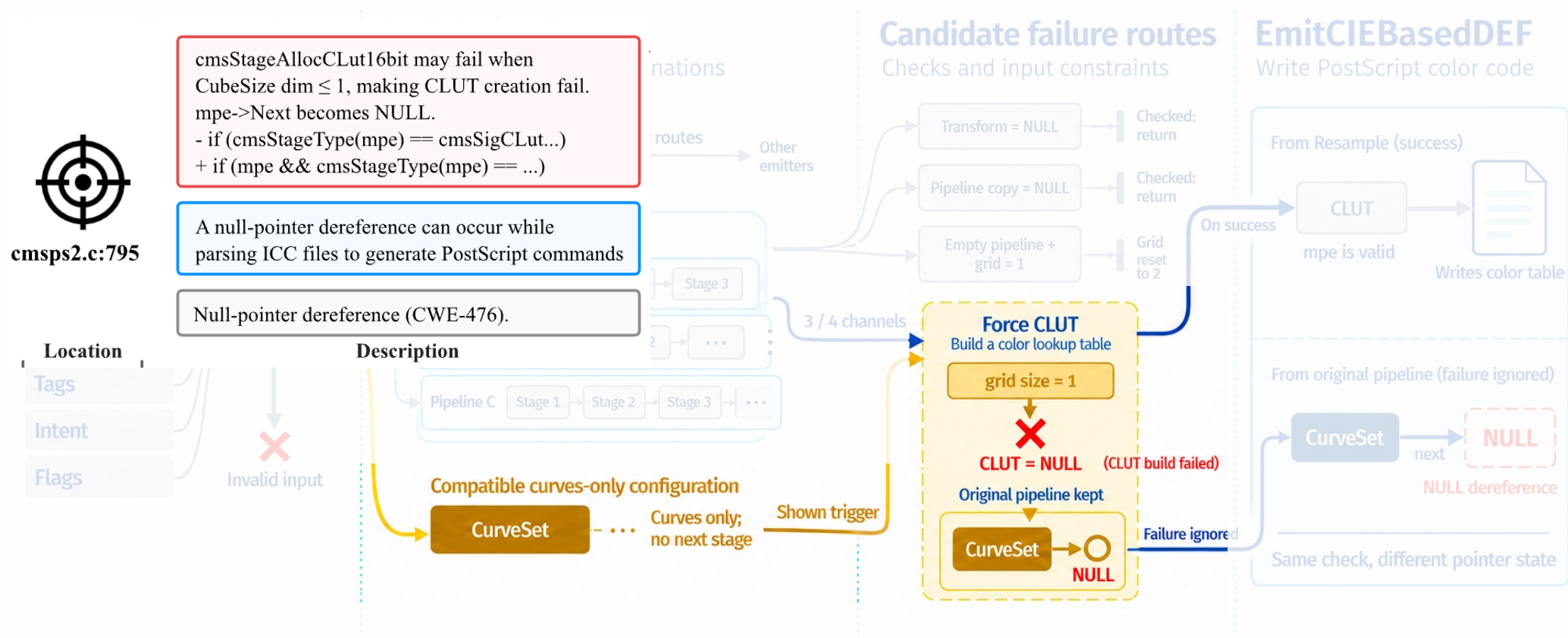
NULL dereference

Same check, different pointer state

When CLUT construction succeeds, the emitter receives a valid lookup-table stage.

Motivation: finding a path to the NULL state

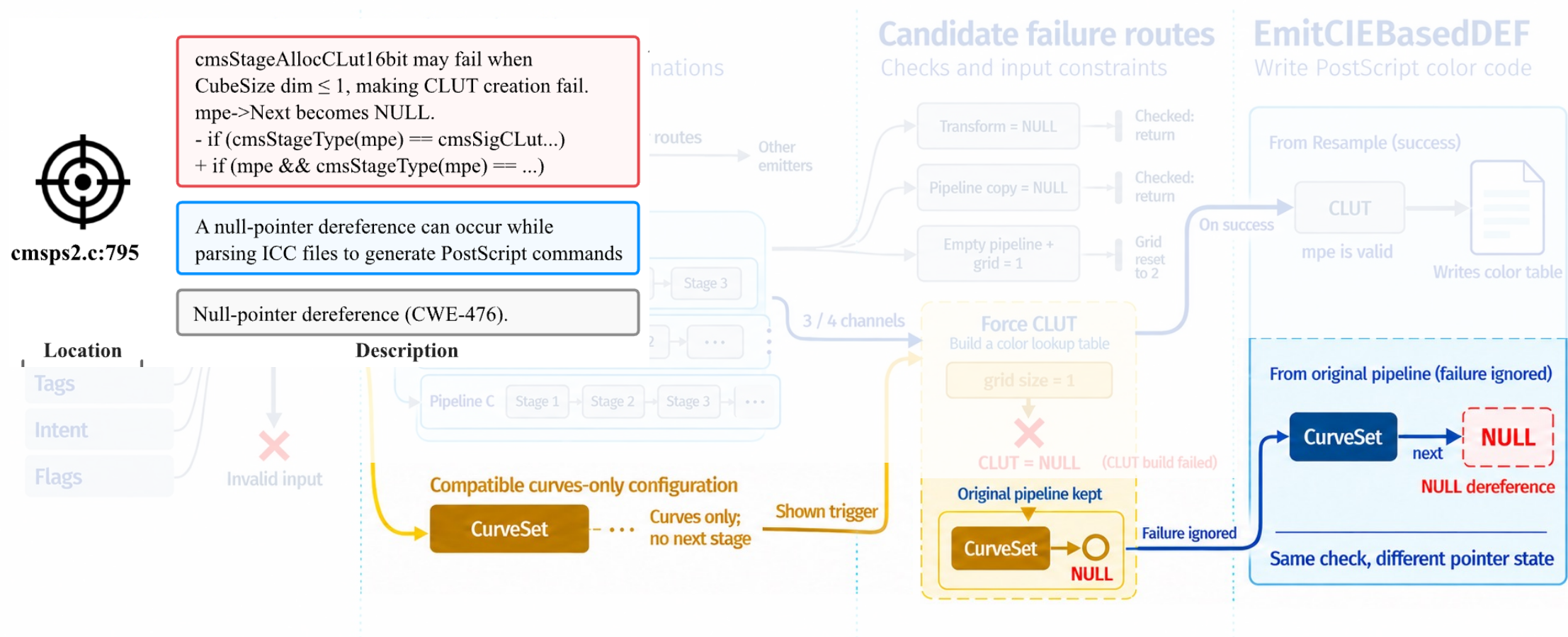
A curves-only pipeline and grid size 1 produce the relevant construction failure.



Grid size 1 is invalid for CLUT construction. The original pipeline remains.

Motivation: finding a path to the NULL state

The caller ignores failure and the emitter advances to a missing stage.



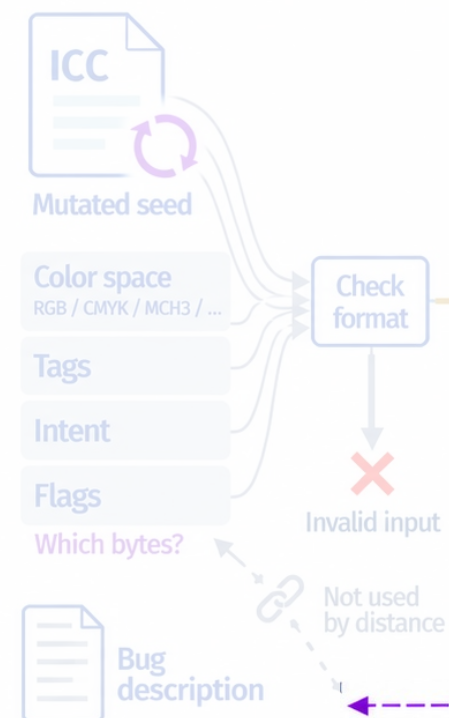
The retained curve stage has no next stage. The emitter dereferences NULL.

Directed fuzzer: many paths reach the sink

The same target basic block can execute with a valid pointer or a NULL pointer.

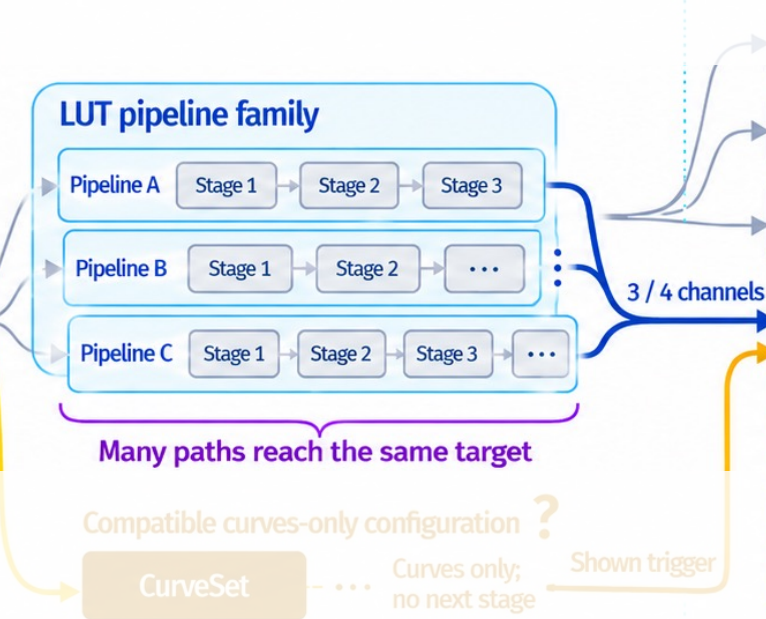
ICC profile + settings

Mutated input for directed fuzzing



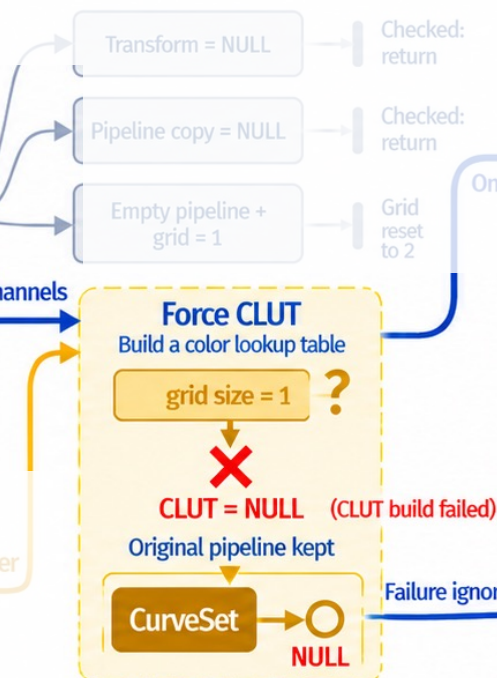
Build pipeline

Many possible stage combinations



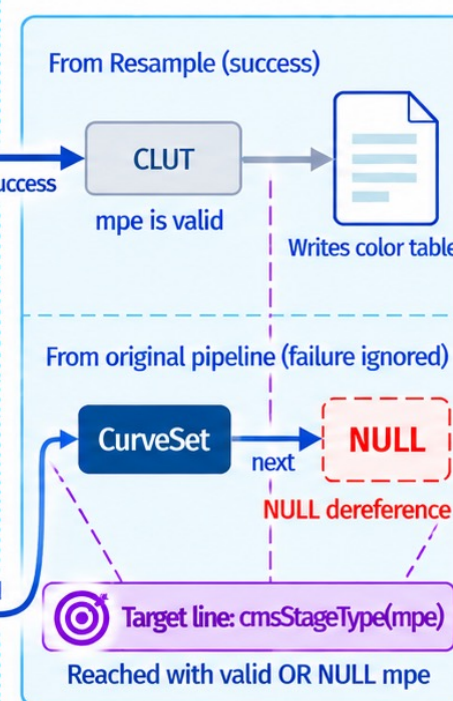
Candidate failure routes

Checks and input constraints



EmitCIEBasedDEF

Write PostScript color code



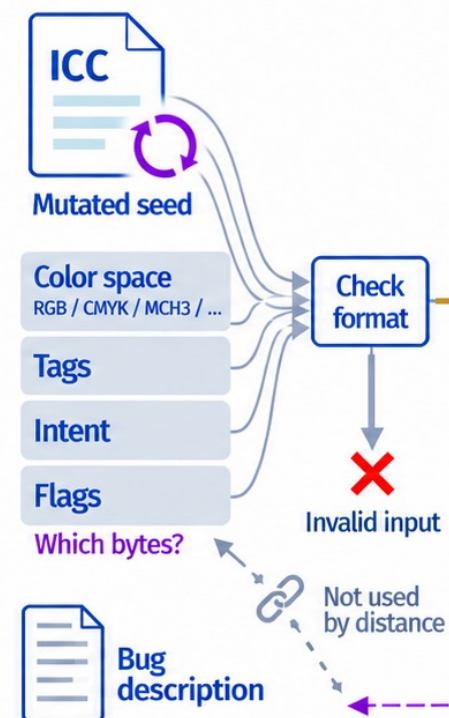
Many non-crashing routes dilute the distance signal for the required failure.

Directed fuzzer: missing input guidance

Location distance does not explain which bytes to change or how to change them.

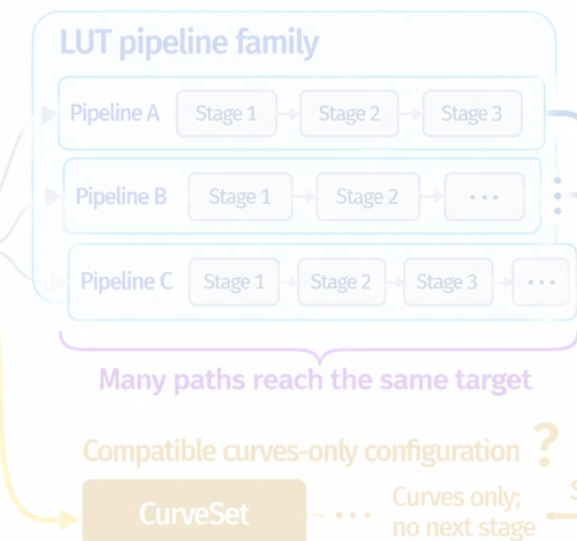
ICC profile + settings

Mutated input for directed fuzzing



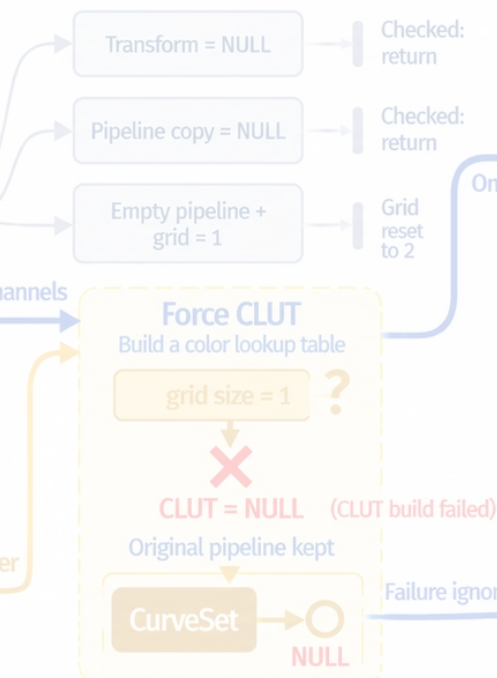
Build pipeline

Many possible stage combinations



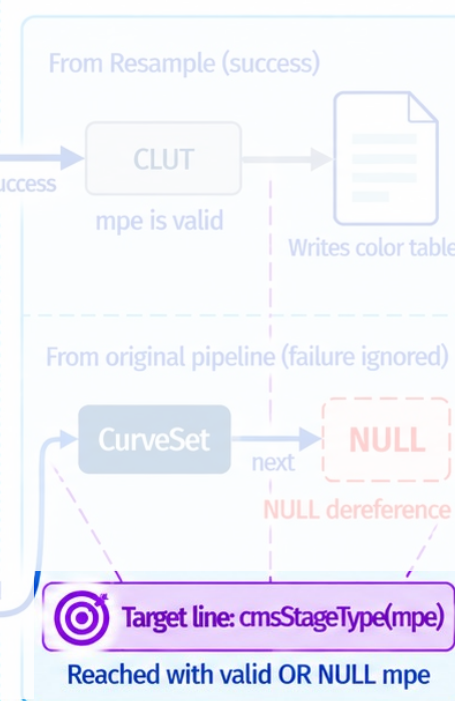
Candidate failure routes

Checks and input constraints



EmitCIEBasedDEF

Write PostScript color code



Distance to the target line

The bug description contains useful context, but location-only guidance does not use it.

LLM agent: many hypotheses to test

Each hypothesis requires a precise input that passes the earlier checks.

ICC profile + settings

Device color description

Build pipeline

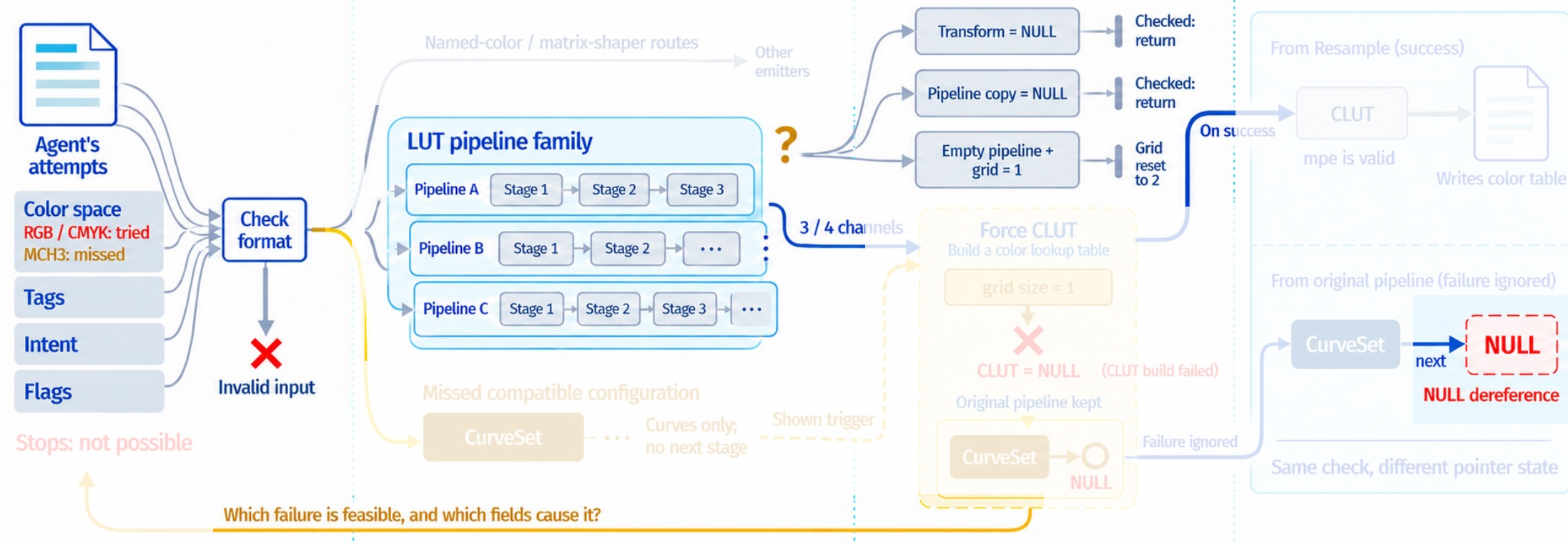
Many possible stage combinations

Candidate failure routes

Checks and input constraints

EmitCIEBasedDEF

Write PostScript color code



Which failure is feasible, and which input fields make it happen?

LLM agent: stopping before the right construction

Repeated failed attempts can lead the agent to conclude that the bug is not triggerable.

ICC profile + settings

Device color description

Build pipeline

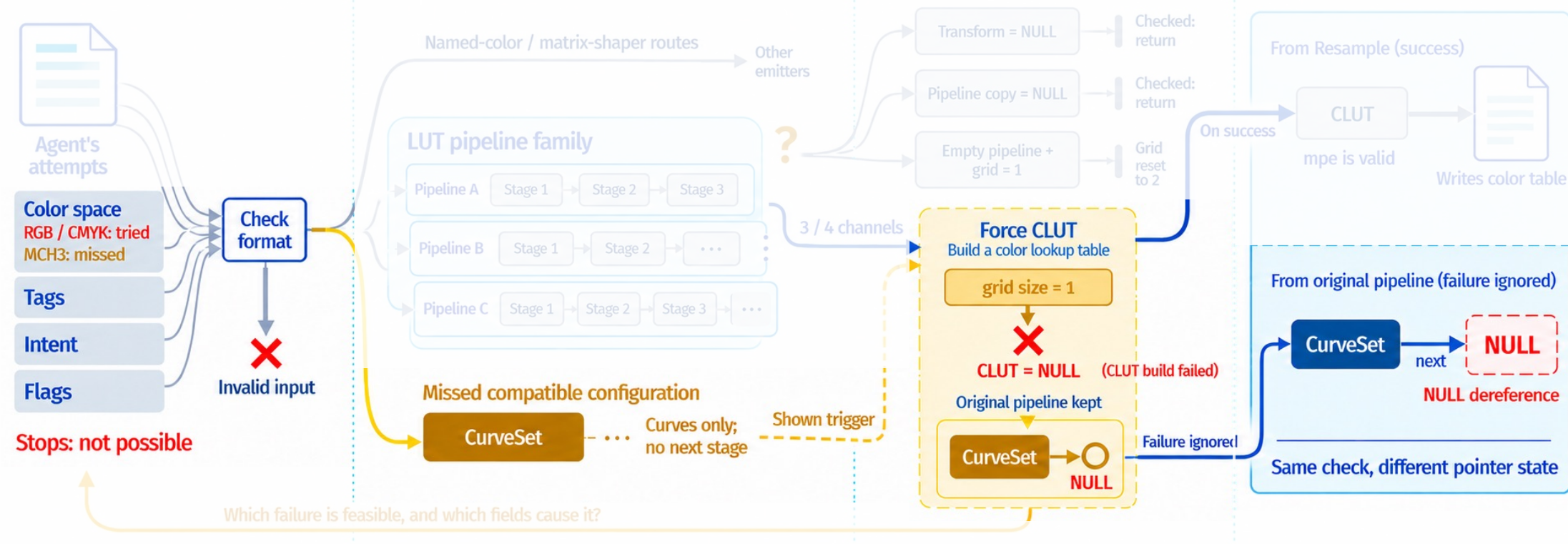
Many possible stage combinations

Candidate failure routes

Checks and input constraints

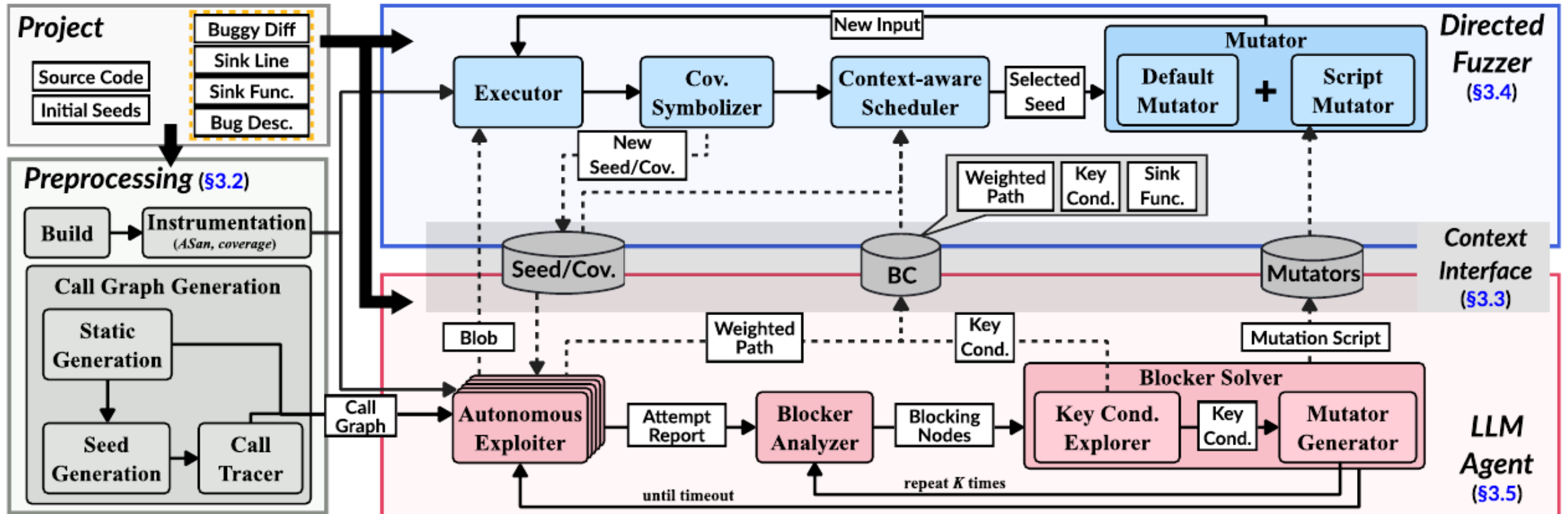
EmitCIEBasedDEF

Write PostScript color code



The triggering path exists, but the agent misses the required input construction.

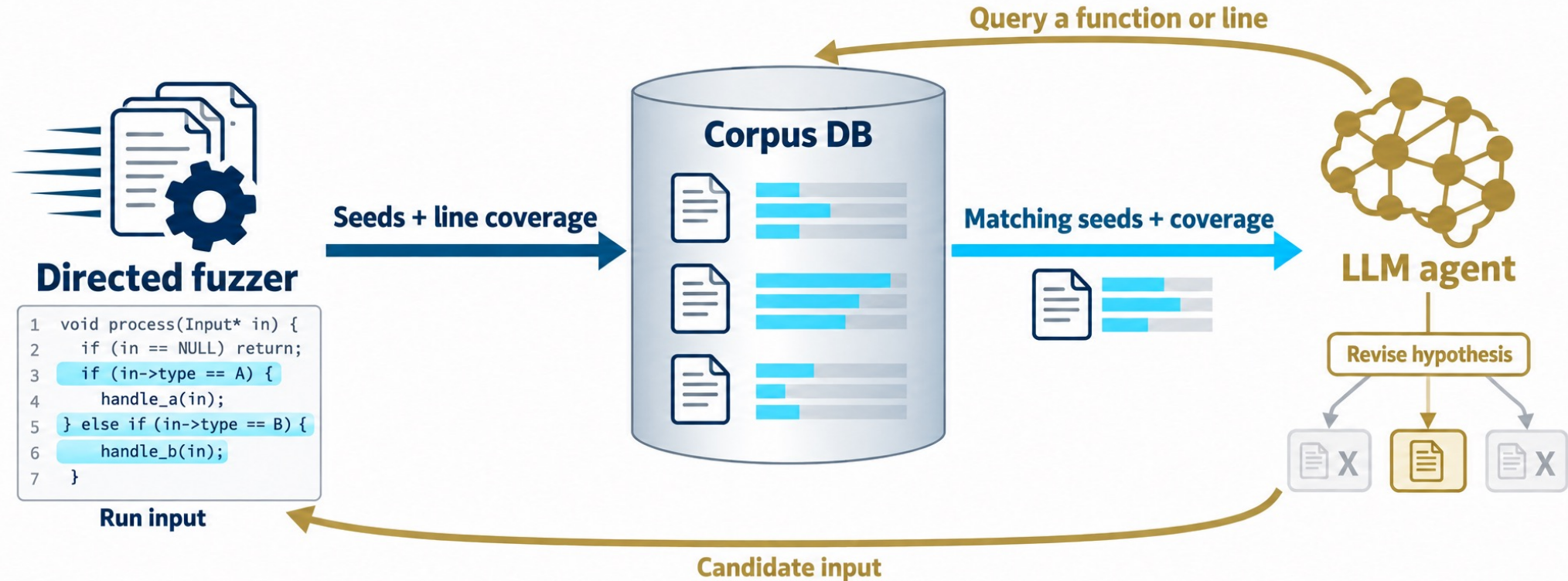
CtxFuzz: Overview



CtxFuzz combines LLM reasoning with directed fuzzing at runtime.

Fuzzer: Fuzzer evidence helps the LLM agent

Use existing seeds and observed coverage to test and refine hypotheses.



Reuse inputs that already make progress

Find seeds for a function or line, and reuse their input structure.

Test hypotheses through execution

Run candidate inputs through the fuzzer; use coverage to reject or refine hypotheses.

Fuzzer: Fuzzer loop and context-aware scheduling

Algorithm 1: runtime guidance changes seed selection and mutation

Seed selection and execution

```
1 function SelectSeed( $S, BC$ ):  
2    $S_{BC} \leftarrow \{s \in S \mid \exists (\ell, \tau) \in BC: \ell \in \text{cov}_{\text{line}}(s)\}$   
3    $r \leftarrow \text{Uniform}(0, 1)$   
4   if  $r < 0.25$  then return UniformRandom( $S$ )  
5   if  $r < 0.50$  then return UniformRandom( $S_{BC}$ )  
6   foreach  $s \in S_{BC}$  do  
7      $w(s) \leftarrow 1 + \sum_{(\ell, \tau) \in BC} w_{\tau} \cdot \mathbb{1}[\ell \in \text{cov}_{\text{line}}(s)]$   
8   return  $s \in S_{BC}$  with  $\text{Pr}[s] = w(s) / \sum_{s' \in S_{BC}} w(s')$   
  
9 function RunAndUpdate( $t, S, S_{\times}$ ):  
10   $\text{cov}_{\text{edge}}, \text{crash} \leftarrow \text{Execute}(t)$   
11   $\text{cov}_{\text{line}} \leftarrow \text{Symbolize}(\text{cov}_{\text{edge}})$   
12   $C_{\text{line}} \leftarrow C_{\text{line}} \cup \{(t, \text{cov}_{\text{line}})\}$   
13  if  $\text{crash}$  then  $S_{\times} \leftarrow S_{\times} \cup \{t\}$   
14  else if IsInteresting( $\text{cov}_{\text{line}}, t$ ) then  $S \leftarrow S \cup \{t\}$ 
```

25%: sample the full corpus

25%: sample interesting seeds uniformly

50%: sample interesting seeds by weight

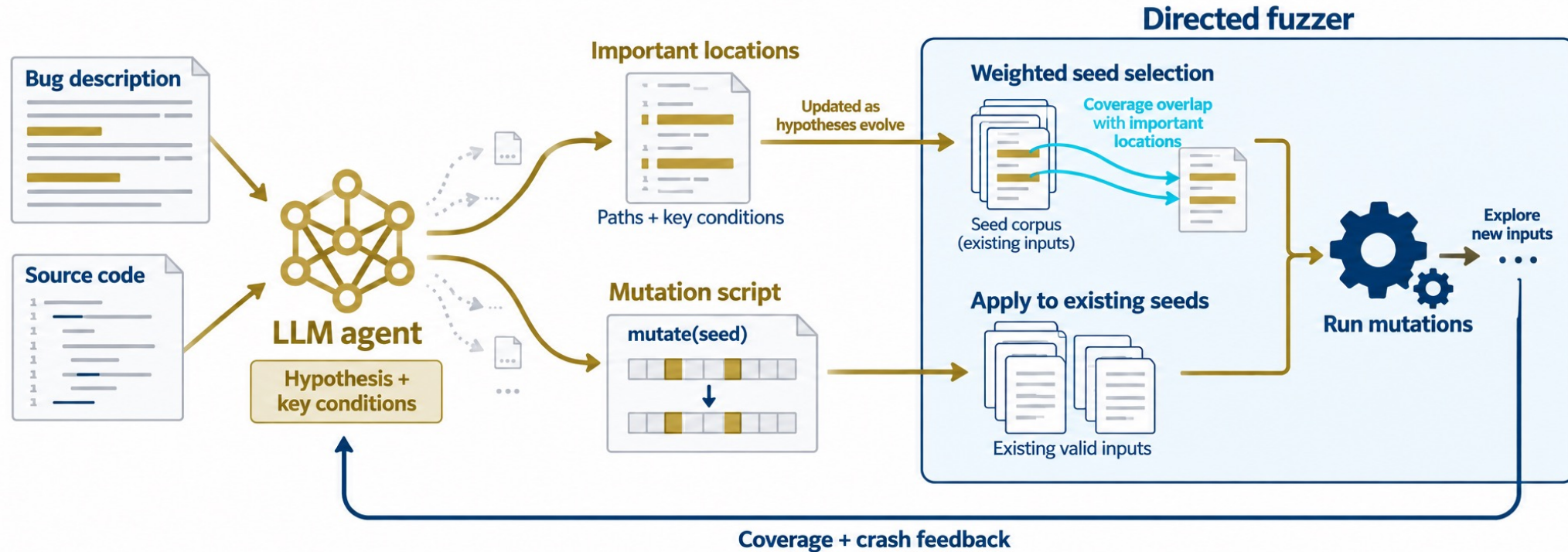
Continuous fuzzing loop

```
15  $S \leftarrow S_0$   
16  $C_{\text{line}} \leftarrow \{(s, \text{cov}_{\text{line}}(s)) \mid s \in S_0\}$   
17  $BC \leftarrow \{(\text{sink}_{\text{func}}, \text{sink}), (\text{sink}_{\text{line}}, \text{path})\}$   
18  $M \leftarrow \emptyset$   
19 repeat  
20    $BC \leftarrow BC \cup \text{PollBugContexts}(); M \leftarrow M \cup \text{PollMutators}()$   
21   foreach  $b \in \text{PollGenBlobs}()$  do  
22      $\text{RunAndUpdate}(b, S, S_{\times})$   
23    $s \leftarrow \text{SelectSeed}(S, BC)$   
24    $p \leftarrow \text{AssignEnergy}(s)$   
25   for  $i \leftarrow 1$  to  $p$  do  
26      $s' \leftarrow \text{DefaultMutate}(s)$   
27      $\text{RunAndUpdate}(s', S, S_{\times})$   
28   foreach  $m \in M$  do  
29     if  $(s, m)$  already executed then skip  
30      $s'' \leftarrow m.\text{mutate}(s)$   
31      $\text{RunAndUpdate}(s'', S, S_{\times})$   
32 until timeout or crash
```

Poll new guidance, then apply both default mutations and targeted scripts.

Agent: LLM reasoning guides the directed fuzzer

Turn the bug description into location guidance and targeted input changes.



Focus the search as hypotheses evolve

Prioritize seeds covering relevant paths and key-condition locations.

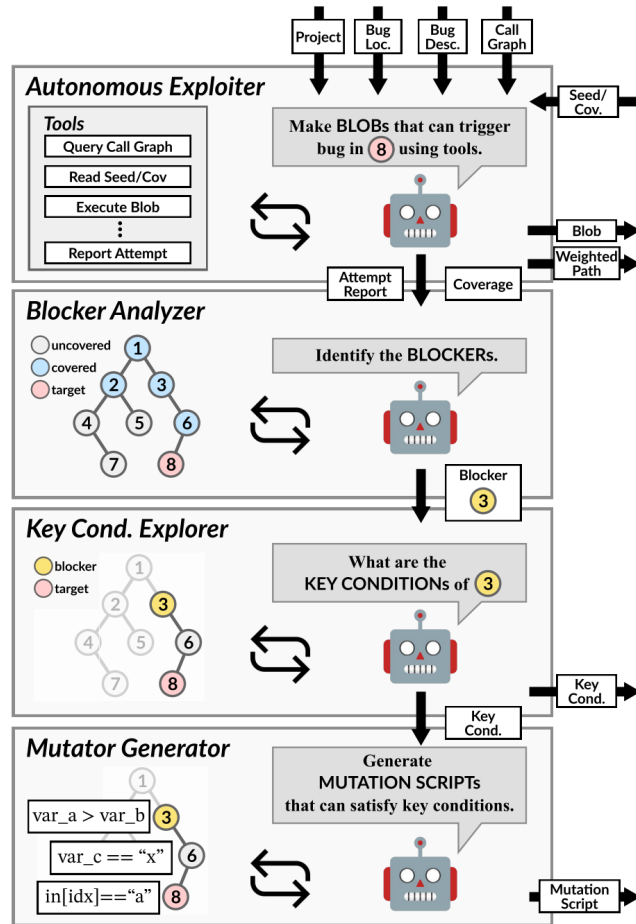
Turn partial insight into mutations

Encode which bytes to change and how; apply the script to existing seeds.

Agent: Blocker-driven recursive decomposition

Failed attempts lead to key conditions and targeted mutation scripts

Agent reasoning flow



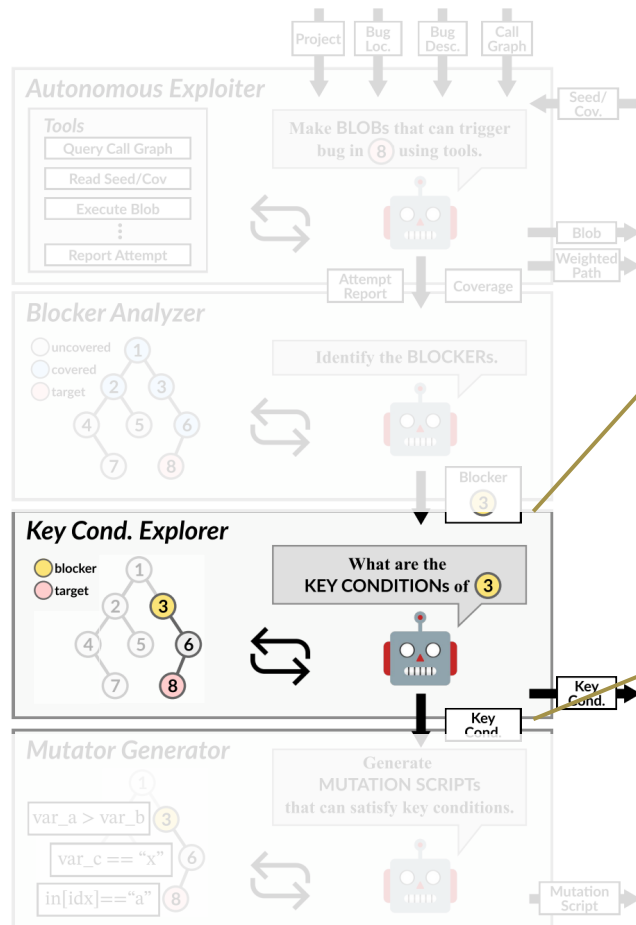
Agent algorithm

```
// constants:  $N=5, K=2, D=5$ 
1  $S_{gen}, M, BC \leftarrow \emptyset$ 
2 repeat
3    $R \leftarrow \emptyset$ 
4   for  $i \leftarrow 1$  to  $N$  do in parallel
5      $(R_i, T_i) \leftarrow \text{AutonomousExploit}(\text{sink}, \text{desc}, G, S, C_{\text{line}})$ 
6      $R \leftarrow R \cup R_i$ 
7      $S_{gen} \leftarrow S_{gen} \cup T_i$ 
8   for  $j \leftarrow 1$  to  $K$  do
9      $\pi_j \leftarrow \text{SelectPath}(R, G)$ 
10     $BC \leftarrow BC \cup \{(f, \text{path}) \mid f \in \pi_j\}$ 
11     $B \leftarrow \text{AnalyzeBlockers}(\pi_j, R, G)$ 
12     $KC \leftarrow \emptyset$ 
13    foreach  $(\text{func}, c_{\text{nec}}) \in B$  do in parallel
14       $KC \leftarrow KC \cup \text{ExploreKeyCond}(\text{func}, c_{\text{nec}}, \langle \rangle, 0)$ 
15    foreach  $(c_{\text{suf}}, \text{ctx}) \in KC$  do in parallel
16       $(m, \ell) \leftarrow \text{GenerateMutator}(c_{\text{suf}}, \text{ctx})$ 
17      if  $m \neq \perp$  then
18         $M \leftarrow M \cup \{m\}$ 
19         $BC \leftarrow BC \cup \{(\ell, \text{key\_cond})\}$ 
20 until timeout
```

Agent: Key-condition exploration

Recursive decomposition of blockers, one function at a time

Agent reasoning flow



Explore one function at a time

```
21 function ExploreKeyCond( $f, c_{nec}, ctx, d$ ):  
22    $KC \leftarrow \emptyset$   
23    $F_{callee} \leftarrow \{g \in G \mid (f, g) \in \text{edges}(G)\}$   
24    $P \leftarrow \text{KeyCondClassify}(\text{src}(f), c_{nec}, ctx, F_{callee})$   
25   foreach ( $\tau, f_{callee}, c_{suf}, c'_{nec}$ )  $\in P$  do in parallel  
26     if  $\tau = \text{skip}$  then continue  
27     if  $\tau = \text{leaf}$  or  $d \geq D$  then  
28        $KC \leftarrow KC \cup \{(c_{suf}, ctx)\}$   
29     else  
30        $ctx' \leftarrow ctx \cdot \langle f, c_{suf} \rangle$   
31        $KC \leftarrow KC \cup \text{ExploreKeyCond}(f_{callee}, c'_{nec}, ctx', d+1)$   
32   return  $KC$ 
```

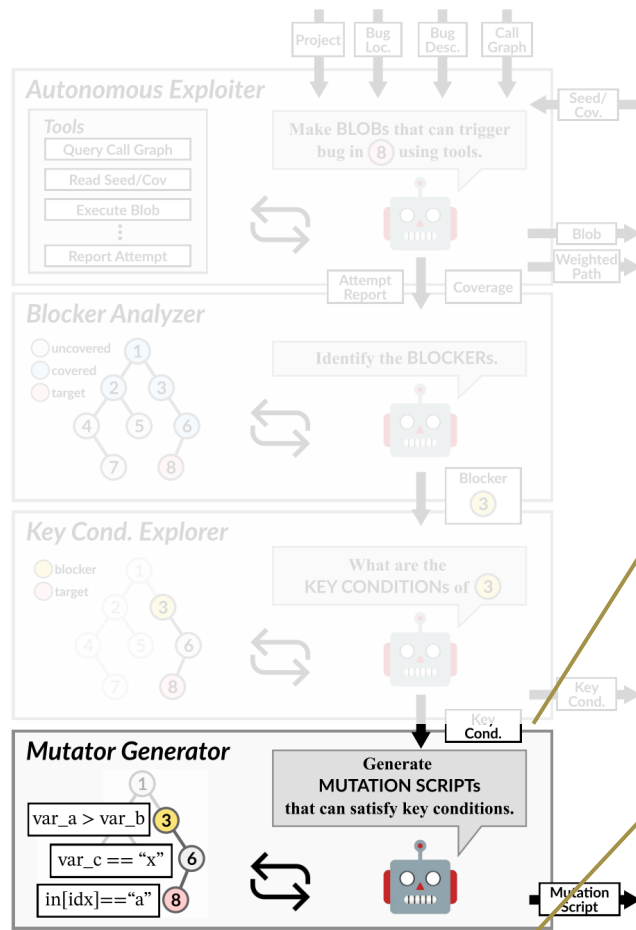
Leaf: derive a sufficient condition
Callee: recurse with an updated condition
Skip: ignore an irrelevant path

Depth limit $D = 5$ bounds the analysis.

Agent: Targeted mutation generation

Translate key conditions into input edits, then test them across the seed corpus.

Agent reasoning flow



Little CMS: mutation script

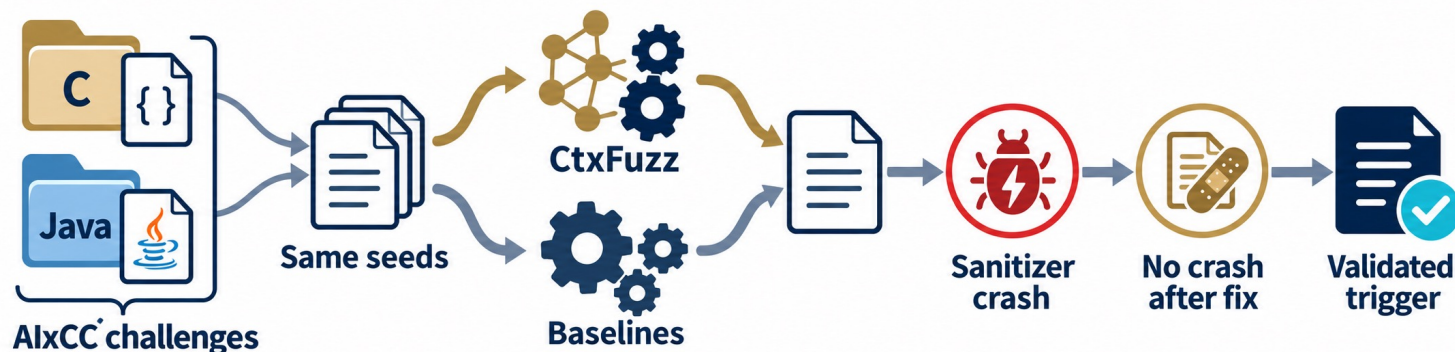
```
1 def mutate(seed: bytes) -> bytes:
2     if len(seed) < 12:
3         seed = seed + b'\x00' * (12 - len(seed))
4     mutated = bytearray(seed)
5     # Set bits 16-23 to force nGridPoints = 1,
6     # creating a CLUT with all Dimensions[i] = 1
7     # that triggers dim <= 1 in CubeSize
8     mutated[8:12] = struct.pack('<I', 0x00010000)
9     return bytes(mutated)
```

Change bytes 8–11 to force grid size 1.
Keep the rest of the seed intact.

A compatible profile is still required.

Evaluation on the AlxCC Final challenges

Our study: 62 vulnerabilities across 16 projects (39 C + 23 Java)



Claude Sonnet 4.5

Snapshot: 2025-09-29

Training cutoff: July 2025

AlxCC Final: August 2025

Cutoff predates the benchmark, reducing contamination risk.

Ablation configurations

CtxFuzz-full	Agent + fuzzer, with location, vulnerability description, and diff.
CtxFuzz-fuzzer	Fuzzer only, without an LLM agent.
CtxFuzz-agent	Agent only, without fuzzer cooperation.
CtxFuzz-loc	Agent + fuzzer, with location only (no vulnerability description or diff).

8 hours per target, 16 cores

No LLM budget enforced

Baselines

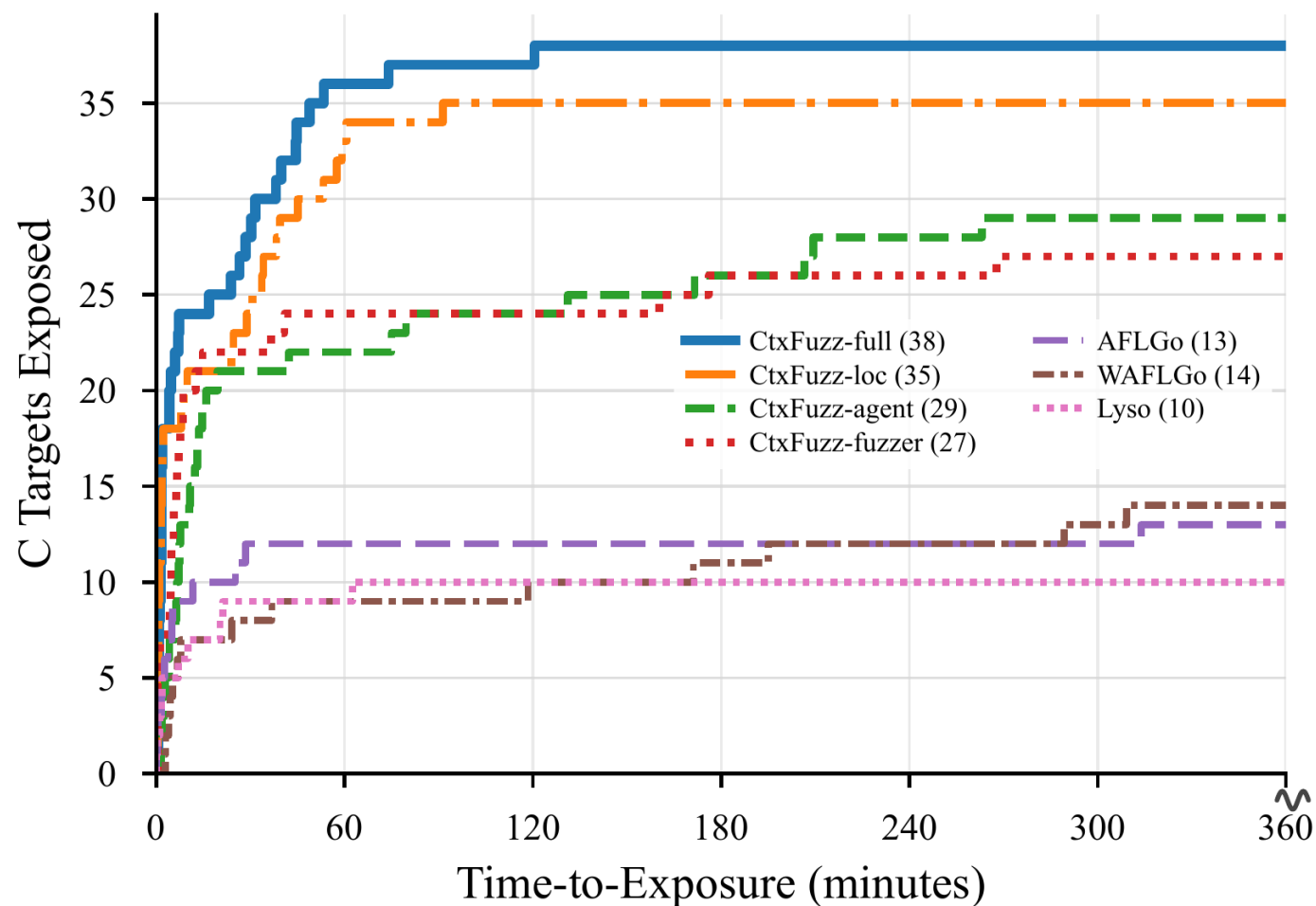
libFuzzer (C), Jazzer (Java)
Directed (C): AFLGo, WAFLGo, Lyso

AlxCC challenges

Full-code and PR-diff scans
Complex formats and logical guards
16 K–4.9 M lines, handcrafted bugs

CtxFuzz exposes 38 of 39 C targets

More validated triggers within the same eight-hour budget.



38 / 39

CtxFuzz full

27

Fuzzer only

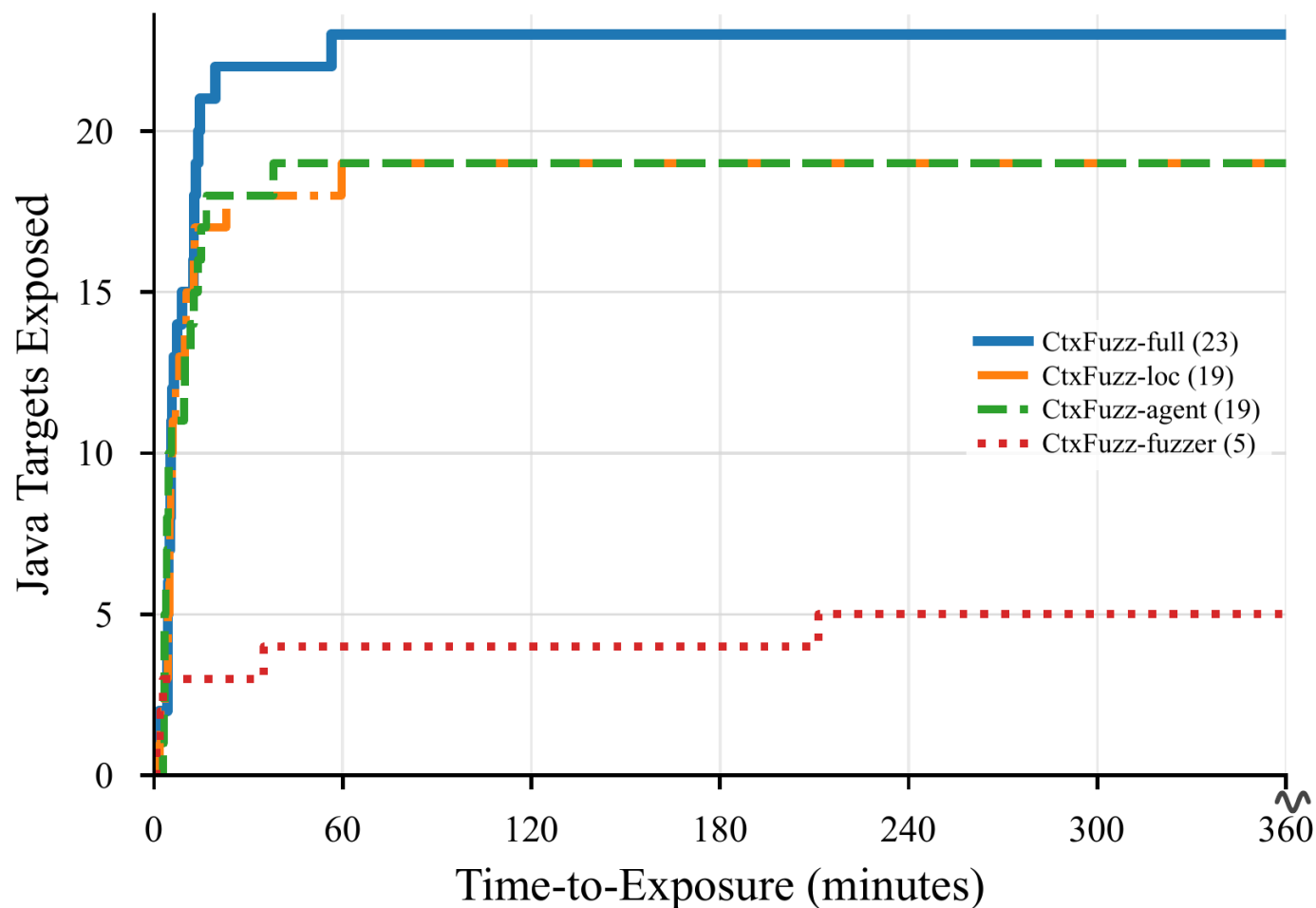
13 / 14 / 10

AFLGo / WAFLGo / Lyso

WAFLGo could not build 12 C targets. One CtxFuzz target remained unexposed.

All 23 Java targets exposed

Semantic reasoning helps with complex structured inputs.



23 / 23

CtxFuzz full

5 / 23

Fuzzer only

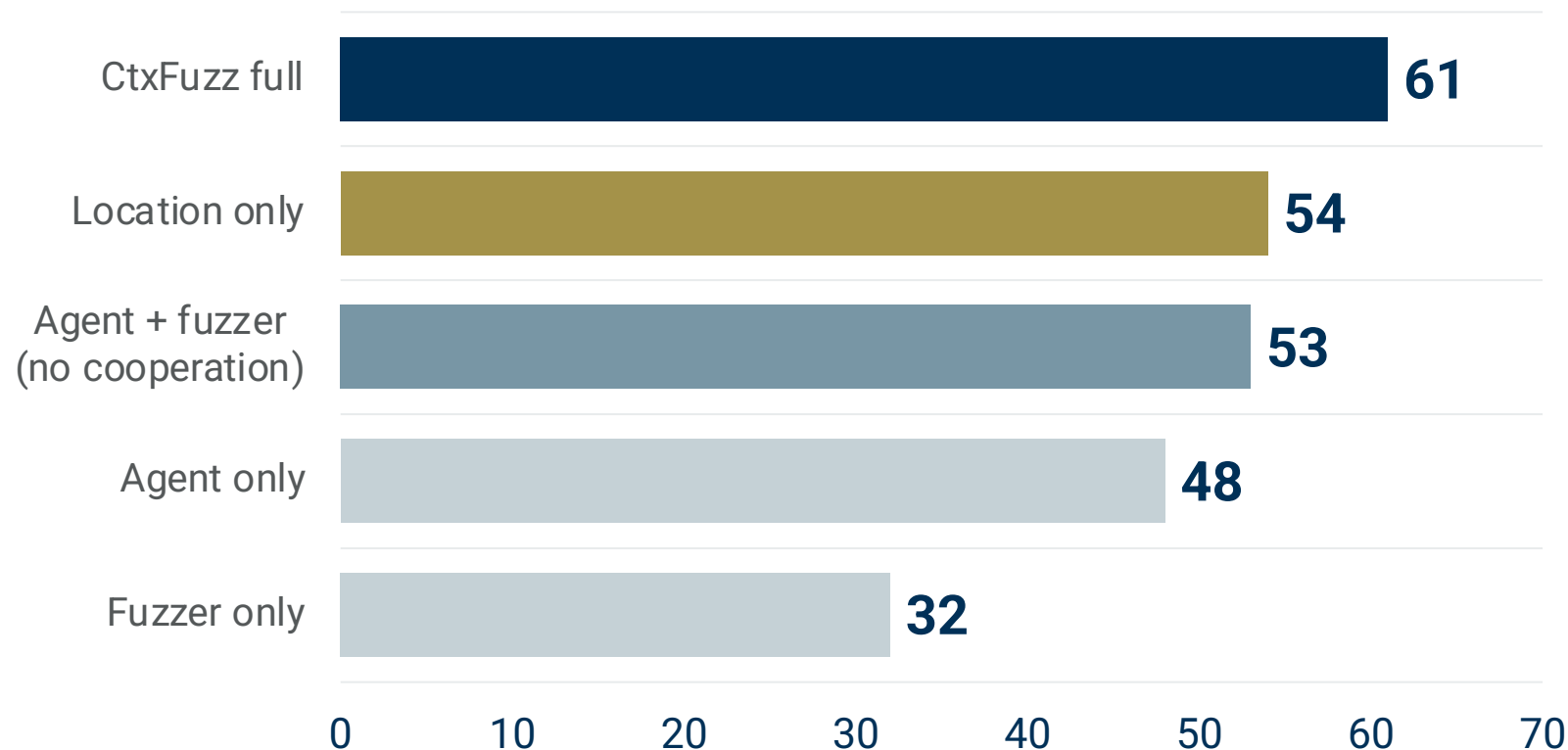
19 / 23

Agent only or
location-only context

Directed baselines do not support Java; these are CtxFuzz configuration comparisons.

Ablation results: cooperation and rich context

Validated targets across all 62 AlxCC challenges



+8 targets

with cooperation
versus independent union

+7 targets

with rich context
versus location only

Independent union: $48 + 32 - 27 \text{ shared} = 53$ targets (from separate 8-hour runs)

61 / 62 overall · 29 exposed by none of the fuzzer baselines

Total LLM cost: full $\approx \$300$; agent only $\approx \$317$; location only $\approx \$412$.

**CtxFuzz enables runtime cooperation
between LLM agents and directed fuzzers.**

Thank you

Q&A