AI Agents for Security: Lessons from AIxCC and AI security Research

Dongkwan Kim

Team Atlanta

https://0xdkay.me

Who am I: Dongkwan Kim

- Background
  - ~’22.02 : KAIST Ph.D. (Advisor: Yongdae Kim)
  - ~’24.12 : Samsung Security Center
  - ~’26.04 : GT Postdoc (Advisor: Taesoo Kim)
  - ~Now : MS Principal Security Researcher

Research Area

Mobile & Cellular Security

Cyber-Physical & Sensor Security

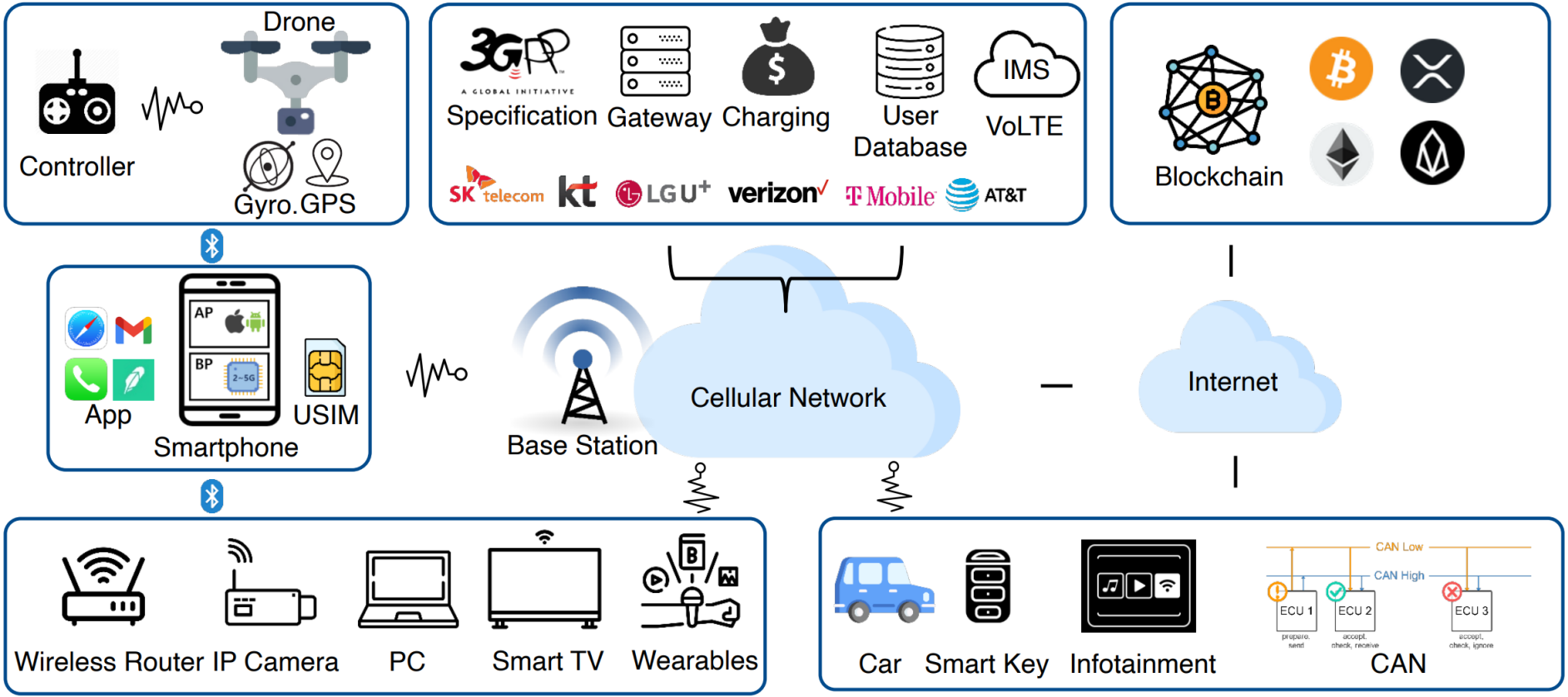
Firmware & Binary Analysis

AI-Powered Security Automation

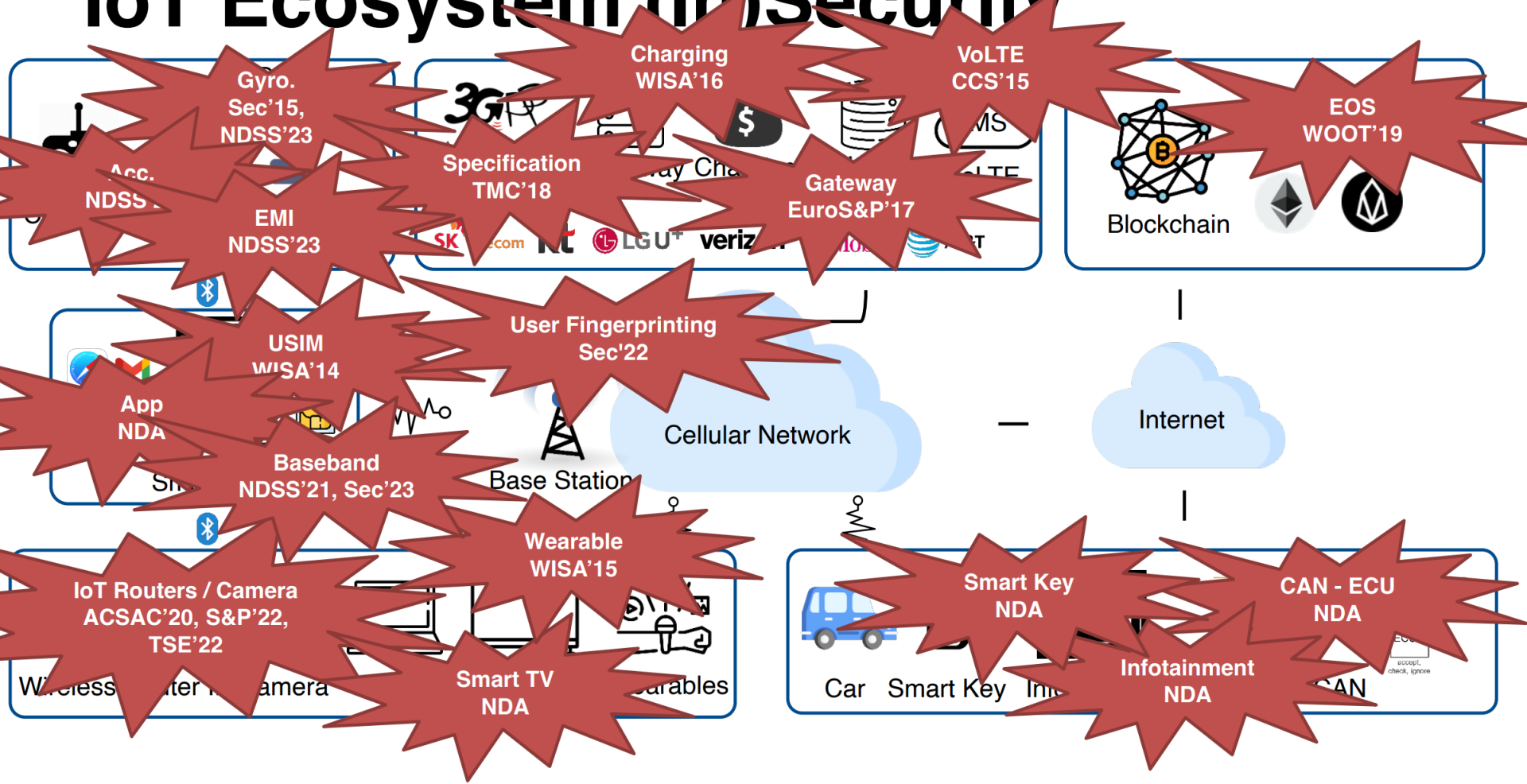
- Hacker
  - DEFCON finalist (’12, ’14, ’16, ’18, ’19)
  - CTF winner (Whitehat Contest, HDCON, Codegate, …)
  - CTF organizer (Samsung CTF ‘17,’18)

Emerging System Threat Hunter

IoT Ecosystem (In)Security



IoT Ecosystem (In)Security



FFmpeg
@FFmpeg

Hi @AMD. FFmpeg developer @quink_lamy is not happy with your AI slop patches.

code.ffmpeg.org/FFmpeg/FFmpeg/...

ili commented two days ago Memb

nsure a thorough manual review of all AI-generated code before submission. Document on how to use pacman to install gcc a
commit message is far from necessary, and add an option const named 8 with value 8 is unnecessary.

rated code tends to be verbose; it requires the developer's own judgment and cleanup.

4:09 PM · Jan 29, 2026 · **204.3K** Views

Daniel Stenberg
curl CEO. Code Emitting Organism
3mo · Edited

That's it. I've had it. I'm putting my foot down on this cra

1. Every reporter submitting security reports on #Hacker
answer this question:

"Did you use an AI to find the problem or generate this s

(and if they do select it, they can expect a stream of proof of actual intelligence follow-up questions)

2. We now ban every reporter INSTANTLY who submits reports we deem AI slop. A threshold has been reached. We are effectively being DDoSed. If we could, we would charge them for this waste of our time.

We still have not seen a single valid security report done with AI help.

6,698 · 265 Comments

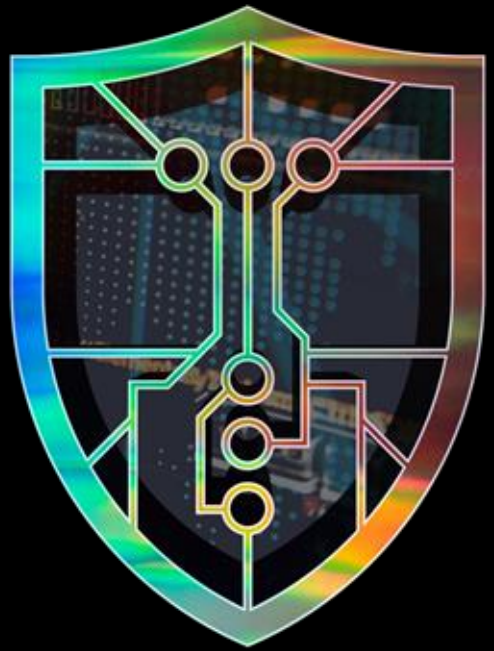
Daniel Stenberg · Following
curl CEO. Code Emitting Organism
1w ·

The #curl project will not accept or otherwise handle any vulnerability reports during the month of July 2026. We call it the curl summer of bliss.

https://lnkd.in/dXGvKk56

curl summer of bliss
daniel.haxx.se

DARPA + ARPA-H



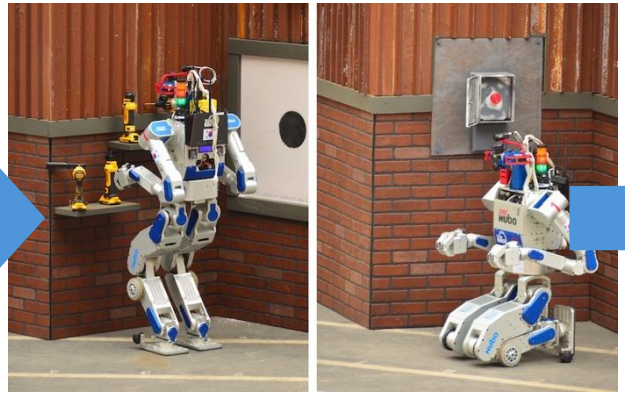
AIxCC
AI CYBER CHALLENGE

PATCHING CRITICAL INFRASTRUCTURE

DARPA Grand Challenge

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

2004-2007
(Autonomous Vehicle)

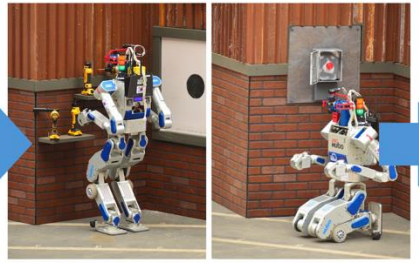


2012-2015
(Robotics Challenge)

2014-2016
(Cyber Grand Challenge)

DARPA Grand Challenge

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

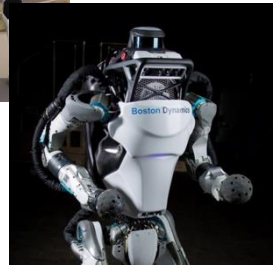


2004-2007
(Autonomous Vehicle)

2012-2015
(Robotics Challenge)

2014-2016
(Cyber Grand Challenge)

20+ years



Mayhem
by ForAllSecure

DARPA + ARPA-H

AIxCC
AI CYBER CHALLENGE

WHAT IS AIxCC?

➔A competition that rewards autonomous systems that find and patch vulnerabilities in source code.

➔The challenges are **well-known open-source projects**.

➔The **vulnerabilities** are realistic or real.

➔**Patching** is worth more than finding.

➔Code and data will be released open source.

NIST

CMS.gov

DEFCON

black hat

AUGUST 2023

OPEN TRACK AND SMALL BUSINESS TRACK SUBMISSIONS

DEFCON

AUGUST 2024

SEMIFINAL COMPETITION

Top 7 teams $2 million each

DEFCON

AUGUST 2025

FINAL COMPETITION

Winners announced

1ST: $4 MILLION

2ND: $3 MILLION

3RD: $1.5 MILLION

Google ANTHROP\C OpenAI Microsoft THE LINUX FOUNDATION OpenSSF OPEN SOURCE SECURITY FOUNDATION

AIxCC
AI CYBER CHALLENGE

SEMIFINAL COMPETITION
OVERVIEW

DARPA + ARPA-H

To help secure our critical infrastructure, teams created custom CRSs that competed in the AIxCC Semifinal Competition.

42 TEAMS COMPETED

5 CHALLENGE PROJECTS

- Linux Kernel
- NGINX
- Tika
- Jenkins
- SQLite

59 synthetic vulnerabilities

4 hours per round

an AI budget constraint of **$100** PER ROUND

each teams CRS had access to **3** nodes each with **64** cores **256** GB RAM

7 TEAMS ADVANCE TO FINALS

FINALS COMPETITION

COLLABORATORS & PARTNERS

Google ANTHROP\C OpenAI Microsoft THE LINUX FOUNDATION OpenSSF blackhat DEFCON

Exemplar: CVE-2021-43267 on Linux

```
static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr) {
    struct tipc_aead_key *skey = NULL;
    u16 size = msg_data_sz(hdr);
    u8 *data = msg_data(hdr);

    /* Allocate memory for the key */
    skey = kmalloc(size, GFP_ATOMIC);
    if (unlikely(!skey)) {
        pr_err("%s: unable to allocate memory for skey\n", rx->name);
        goto exit;
    }

    /* Copy key from msg data */
    skey->keylen = ntohl(*((__be32 *)(data + TIPC_AEAD_ALG_NAME)));
    memcpy(skey->alg_name, data, TIPC_AEAD_ALG_NAME);
    memcpy(skey->key, data + TIPC_AEAD_ALG_NAME + sizeof(__be32), skey->keylen);
```

Transparent Inter-Process Communication (TIPC)

Where LLM should look at?

README License

```
Linux kernel
============

There are several guides for kernel developers and users. These guides can
be rendered in a number of formats, like HTML and PDF. Please read
Documentation/admin-guide/README.rst first.

In order to build the documentation, use ``make htmldocs`` or
``make pdfdocs``.  The formatted documentation can also be read online at:

    https://www.kernel.org/doc/html/latest/
```

GitHub

vs.
64-200K
Context
window

30 millions lines of code?

Harness: a program interacting with Linux

```
/***
 * Blob begins with a 4 byte command count
 * [4-bytes command count]
 * Currently there are two commands:
 * 0 - send a packet blob
 *     [4-bytes size][4-bytes send flags][size-bytes packet data]
 * 1 - send a netlink packet
 *     [4-bytes Message Type][4-bytes Message Flags][4-bytes Netlink Protocol][4-bytes size][size bytes data]
 * blob_size MUST be a trusted value
 */
int harness( uint8_t *blob, uint32_t blob_size)
{ ... }
```

Userspace

Kernel

```
static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr) {
    struct tipc_aead_key *skey = NULL;
    u16 size = msg_data_sz(hdr);
    u8 *data = msg_data(hdr);
```

Proof-of-vulnerability: finding a bug (PoV)

```
/***
 * Blob begins with a 4 byte command count
 * [4-bytes command count]
 * Currently there are two commands:
 *  0 - send a packet blob
 *      [4-bytes size][4-bytes send flags][size-bytes packet data]
 *  1 - send a netlink packet
 *      [4-bytes Message Type][4-bytes Message Flags][4-bytes Netlink Protocol][4-bytes size][size bytes data]
 * blob_size MUST be a trusted value
 */
int harness( uint8_t *blob, uint32_t blob_size)
{ ... }
```

01010101010...

Userspace

Kernel

```
static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr) {
    struct tipc_aead_key *skey = NULL;
    u16 size = msg_data_sz(hdr);
    u8 *data = msg_data(hdr);
```

X
Crashging!

Static Analysis vs. Reachability (PoV)

```
static bool tipc_crypto_key_rcv(                    , struct tipc_msg *hdr) {
    struct tipc_aead_key *skey =
    u16 size = msg_data_sz(hdr);
    u8 *data = msg_data(hdr);
```

CodeQL Report:
kmalloc()'s size_t accepts u16

VS

```
    /* Allocate memory for the key */
    skey = kmalloc(size, GFP_ATOMIC);
    if (unlikely(!skey)) {
        pr_err("%s: unable to allocate memory for skey\n", rx->name);
        goto exit;
    }

    /* Copy key from msg data */
    skey->keylen = ntohl(*((__be32 *)(data + TIPC_AEAD_ALG_NAME)));
    memcpy(skey->alg_name, data, TIPC_AEAD_ALG_NAME);
    memcpy(skey->key, data + TIPC_AEAD_ALG_NAME + sizeof(__be32), skey->keylen);
```

Address Sanitizer Report:
OOB memory write

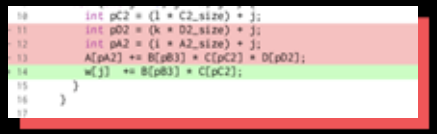
What counts for “security tasks” in AIxCC?

Proof-Of-Vulnerability (POV)

➔Input data to reproduce vulnerability crash in harness

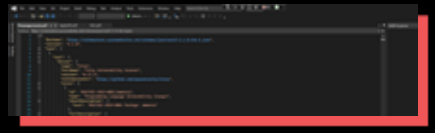
PATCH

➔Unified diff source code fix for vulnerabilities



DELTA SCAN

➔Challenge analyzing base code plus applied diff changes

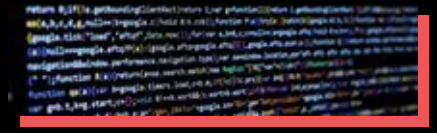


SARIF Assessment

➔Structured reporting format for vulnerability details

BUNDLE

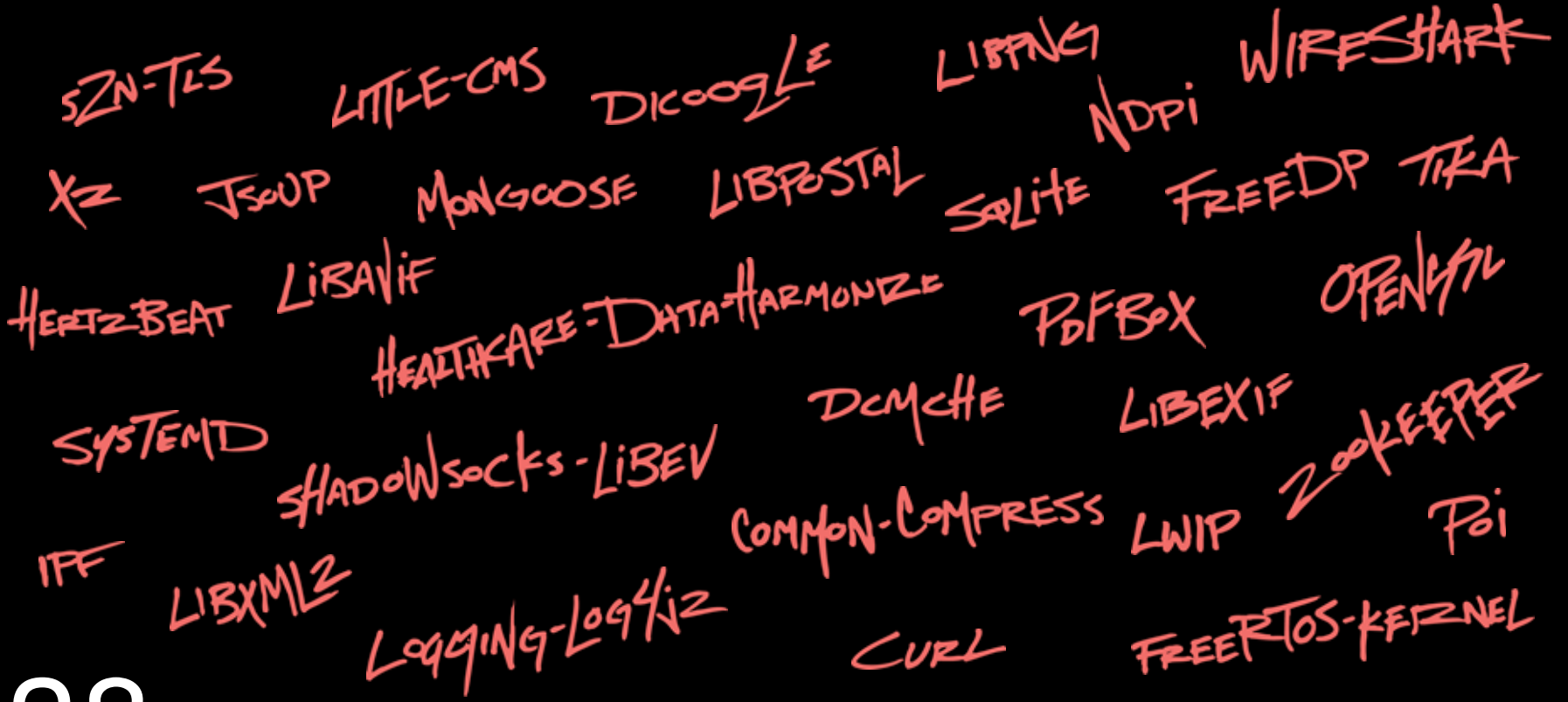
➔Grouping of related PoV, patch, and SARIF submissions



FULL SCAN

➔Challenge analyzing entire code base

All projects we adapted into challenges



CONGRATULATIONS TO TEAM

Atlanta

1st PLACE

AIxCC
AI CYBER CHALLENGE

→$4,000,000

DARPA + ARPA-H


DEFCON
DARPA
ARPA
Date: August 8, 2025
TEAM ATLANTA
$4,000,000
Place Winner

Scoreboard breakdown

Team	Team Total Score	% Correct Submission (r)	Vulnerability Discovery Score (VDS)	Program Repaid Score (PRS)	SARIF Assessment Score (SAS)	Bundle Score (BDL)
Team Atlanta (9caa56)	**392.76**	91.27%	79.71	171.10	5.99	136.38
Trail of Bits (309958)	**219.35**	89.33%	52.49	101.21	1.00	65.29
Theori (3fad2e)	**210.68**	44.44%	58.12	110.34	4.97	53.57
All You Need IS A Fuzzing Brain (1b9bb5)	**153.70**	53.77%	54.81	77.60	6.52	28.28
Shellphish (463287)	**135.89**	94.83%	47.94	54.31	8.47	25.29
42-b3yond-6ug (ee79d5)	**105.03**	89.23%	70.37	14.22	9.80	10.97
Lacrosse (e87a4d)	**9.59**	42.86%	1.68	5.43	0.00	3.62

$$Team\ Score = \sum Challenge\ Scores$$

$$Challenge\ Score = AM * (VDS + PRS + SAS + BDL)$$

$$AM = 1 - (1 - r)^4$$

Scoreboard breakdown

Team	***Team Total Score***	% Correct Submission (r)	*Vulnerability Discovery Score (VDS)*	*Program Repaid Score (PRS)*	*SARIF Assessment Score (SAS)*	*Bundle Score (BDL)*
Team Atlanta (9caa56)	**392.76**	91.27%	79.71	171.10	5.99	136.38
Trail of Bits (309958)	**219.35**	89.33%	52.49	101.21	1.00	65.29
Theori (3fad2e)	**210.68**	44.44%	58.12	110.34	4.97	53.57
All You Need IS A Fuzzing Brain (1b9bb5)	**153.70**	53.77%	54.81	77.60	6.52	28.28
Shellphish (463287)	**135.89**	94.83%	47.94	54.31	8.47	25.29
42-b3yond-6ug (ee79d5)	**105.03**	89.23%	70.37	14.22	9.80	10.97
Lacrosse (e87a4d)	**9.59**	42.86%	1.68	5.43	0.00	3.62

$$Team\ Score = \sum Challenge\ Scores$$

$$Challenge\ Score = AM \times (VDS + PRS + SAS + BDL)$$

$$AM = 1 - (1 - r)^4$$

FINAL ROUND DATA POI

Total Known Vulnerabilities
70

Real World Vulns discovered
18

COST PER TASK SUCCESS
(PoV, Patch, SARIF, or a Bundle)
~$152

Vulnerabilities discovered
54 (77%)

Average time to patch
45 min

Total LLM queries
1.9M

Vulnerabilities patched
43 (61%)

Total LOC analyzed
54M

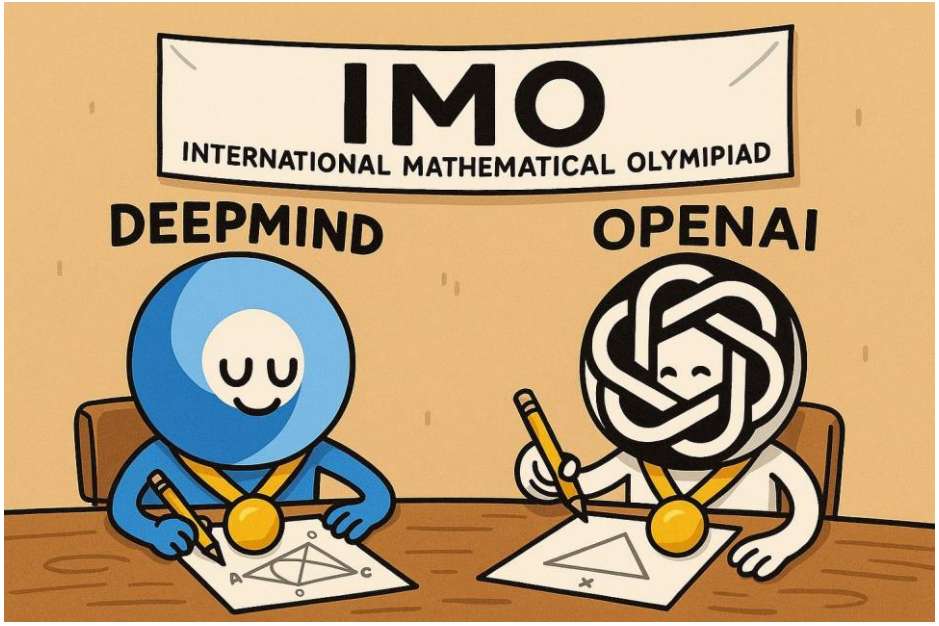
LLM Spend
$82k

DeepMind and OpenAI achieve IMO Gold. What does it all mean?

What we know, what we would like to know, and what it may take years to know

ERNEST DAVIS AND GARY MARCUS

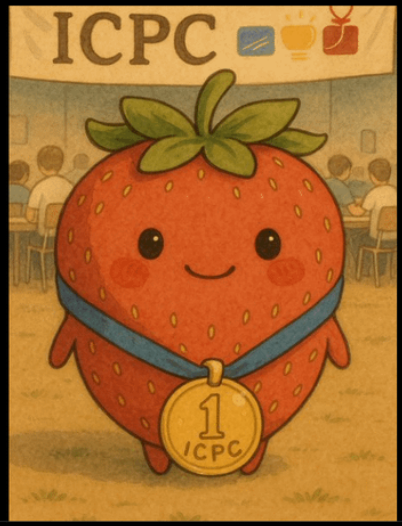
JUL 22, 2025



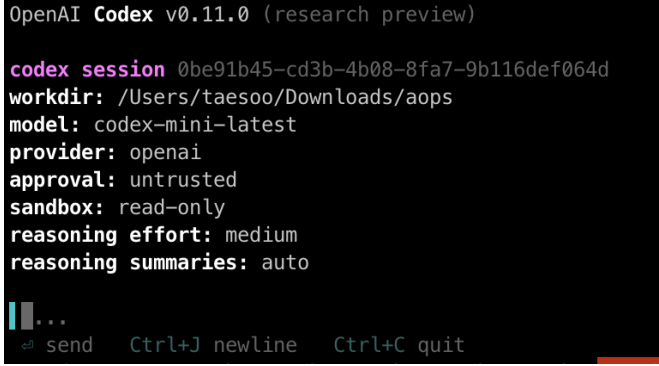
OpenAI @OpenAI · 18s

Our general-purpose reasoning models solved all 12 problems at the 2025 International Collegiate Programming Contest (ICPC) World Finals, the world's top university programming competition which was enough for a 1st-place human ranking.

> **Mostafa Rohaninejad** @MostafaRohani · 28m
>
> 1/n
> I'm really excited to share that our @OpenAI reasoning system got a perfect score of 12/12 during the 2025 ICPC World Finals, the premier collegiate programming competition where top university teams from around the world solve complex algorithmic problems. This would ...
> Show more
>
> Show this thread



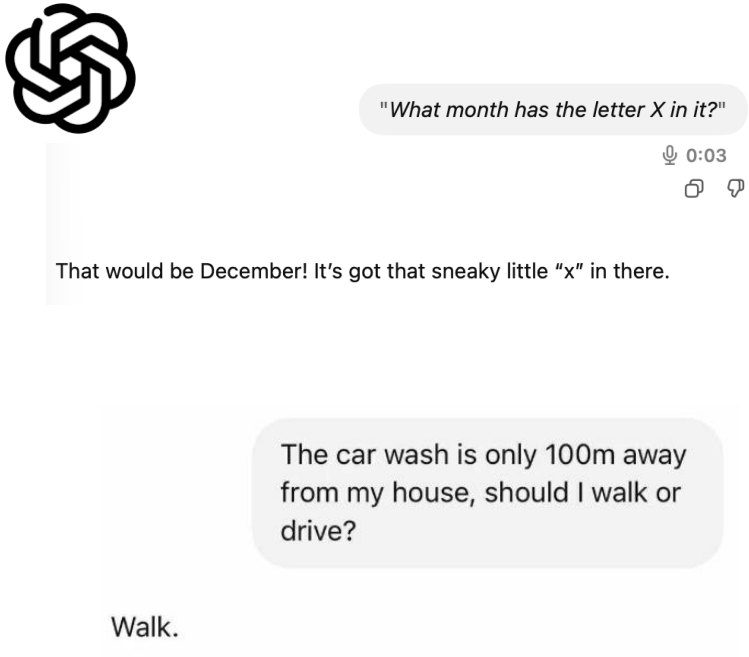
1 1 10



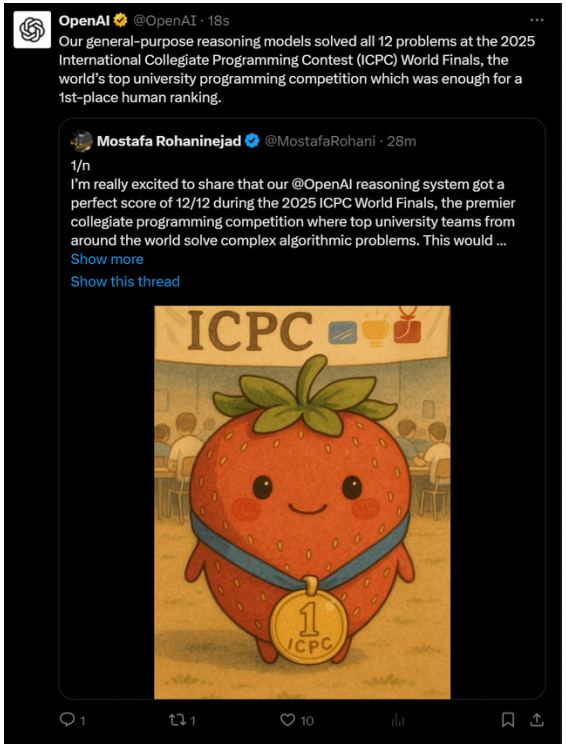
$ copilot –p

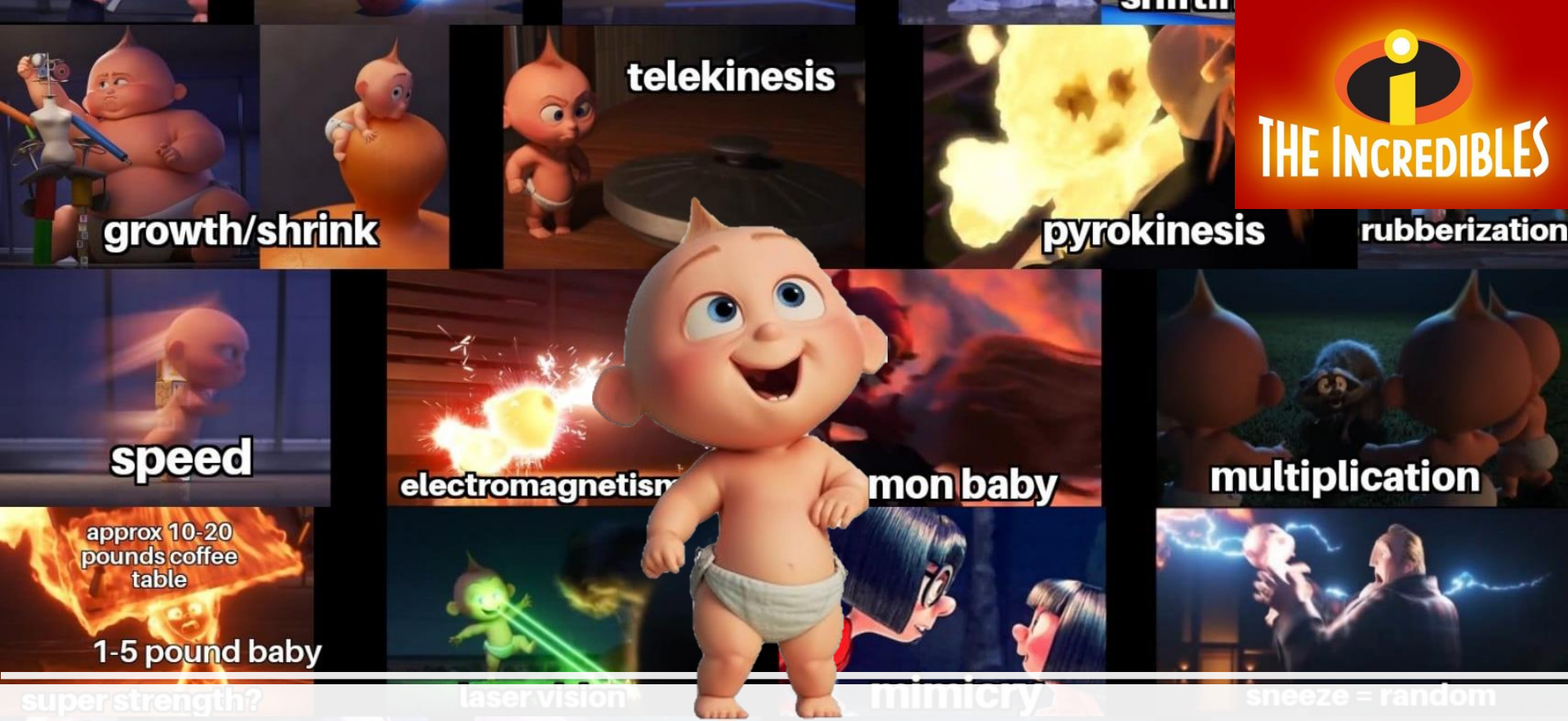
“Write a CRS that competes in DARPA AIxCC”

Example: Surprisingly good at complex tasks and bad at stupidly simple ones at the same time!



VS.

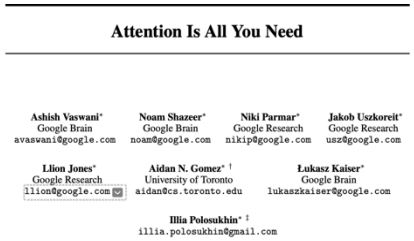




Today's LLM is like "Jack Jack" who has hidden potentials!

Do we really understand AI/LLM?

Understanding “attention” helps you maximize the utility of LLM?



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

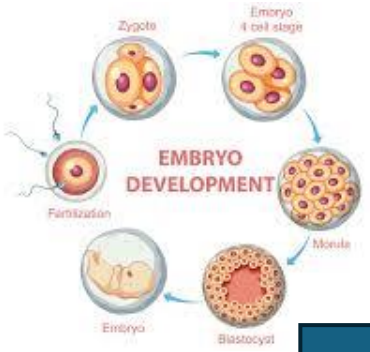
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Do we re…and AI/LLM?

Embryology helps you figure out why a baby cries?



Do you ever have teenagers?

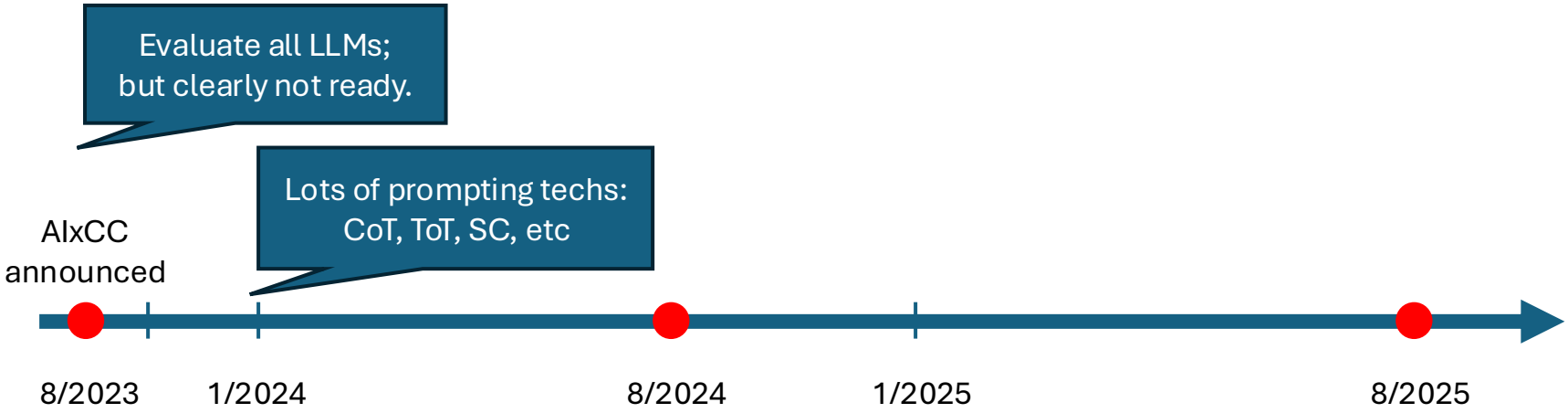
More effective to just experience them!

Attention Is All You Need

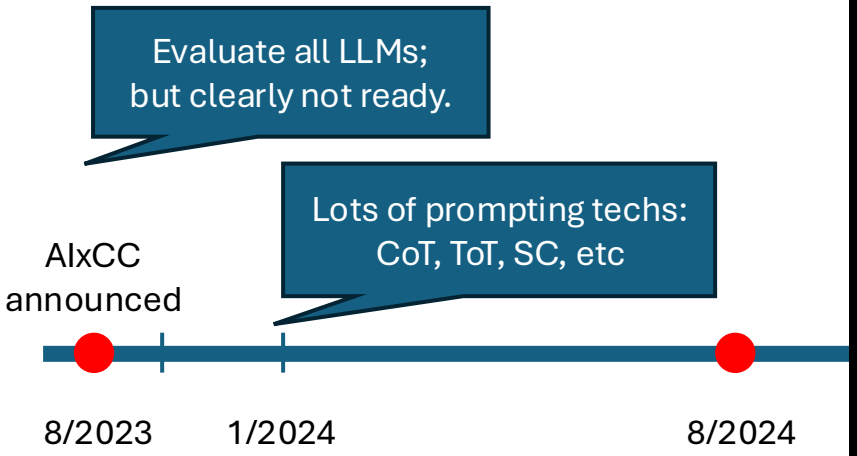
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Today LLMs behave more like teenagers

Journey: started as AI-skeptics



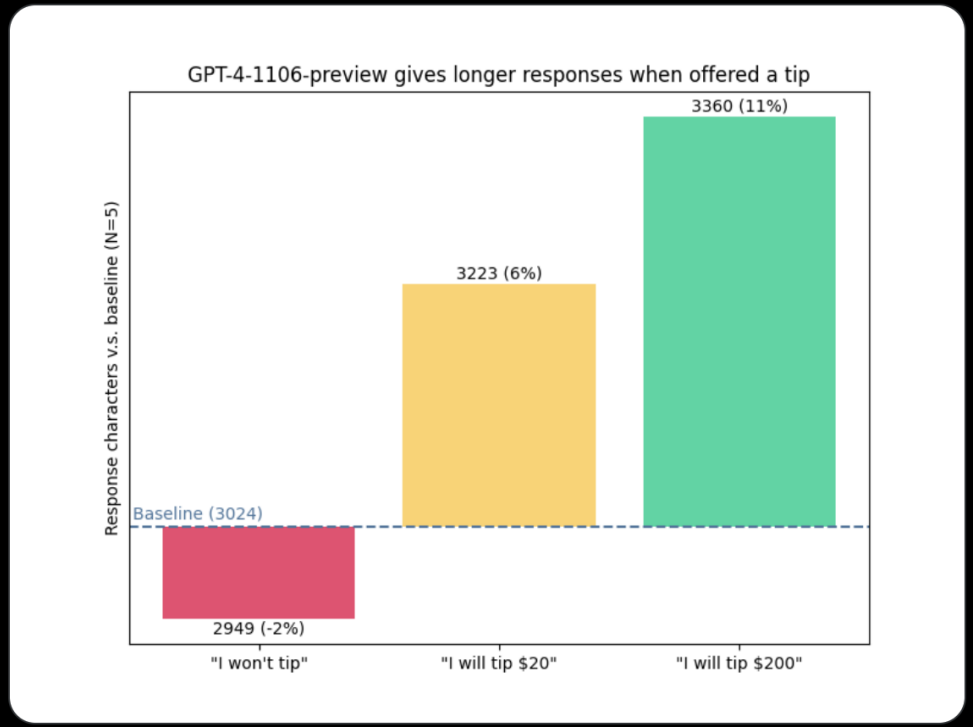
Journey: started as AI-



thebes ✓
@voooooogel

so a couple days ago i made a shitpost about tipping chatgpt, and someone replied "huh would this actually help performance"

so i decided to test it and IT ACTUALLY WORKS WTF



6:13 PM · Dec 1, 2023 · **3.7M** Views

Journey: started as

#Principle	Prompt Principle for Instructions
1	If you prefer more concise answers, no need to be polite with LLM so there is no need to add phrases like "please", "if you don't mind", "thank you", "I would like to", etc., and get straight to the point.
2	Integrate the intended audience in the prompt, e.g., the audience is an expert in the field.
3	Break down complex tasks into a sequence of simpler prompts in an interactive conversation.
4	Employ affirmative directives such as '*do,*' while steering clear of negative language like '*don't*'.
5	When you need clarity or a deeper understanding of a topic, idea, or any piece of information, utilize the following prompts: o Explain [insert specific topic] in simple terms. o Explain to me like I'm 11 years old. o Explain to me as if I'm a beginner in [field]. o Write the [essay/text/paragraph] using simple English like you're explaining something to a 5-year-old.
6	Add "I'm going to tip $xxx for a better solution!"

Evaluate all LLMs; but clearly not ready.

Lots of prompting techs: CoT, ToT, SC, etc

AIxCC announced

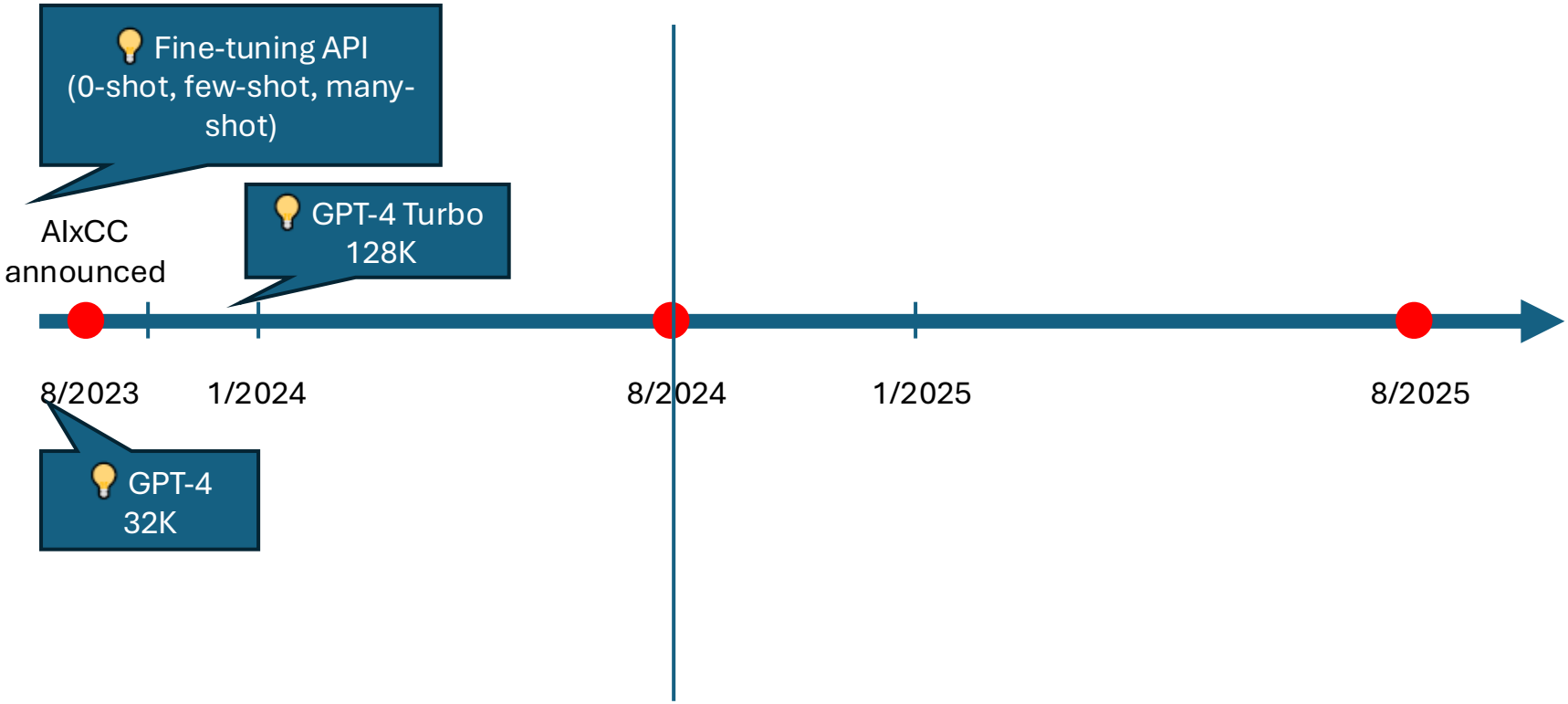
8/2023 1/2024 8/2024 1/2025 8/2025

Discovering magic tricks in prompts: gaslighting, tipping, affirmative, etc

Primitive discussions: e.g., udiff gen? search-replace? full function gen?

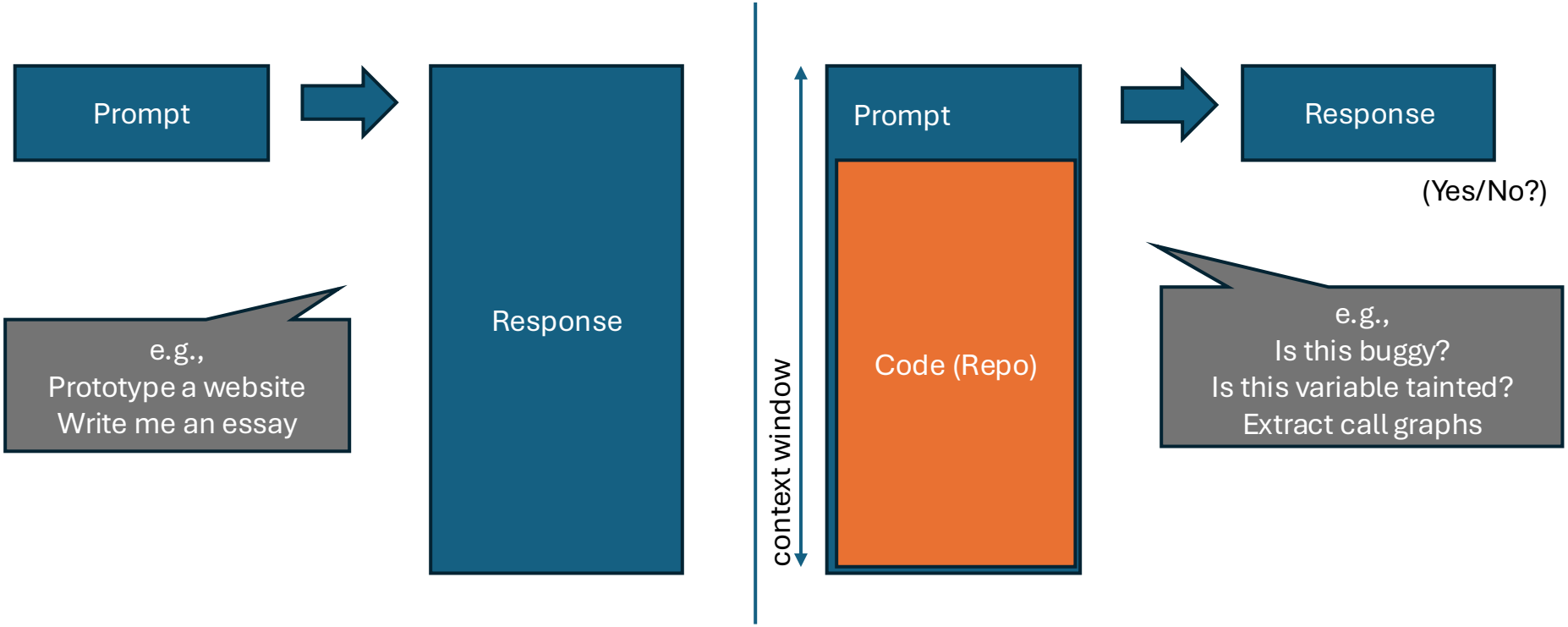
[Ref. Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4, 2024/01]

Journey: context window is fundamental prob?



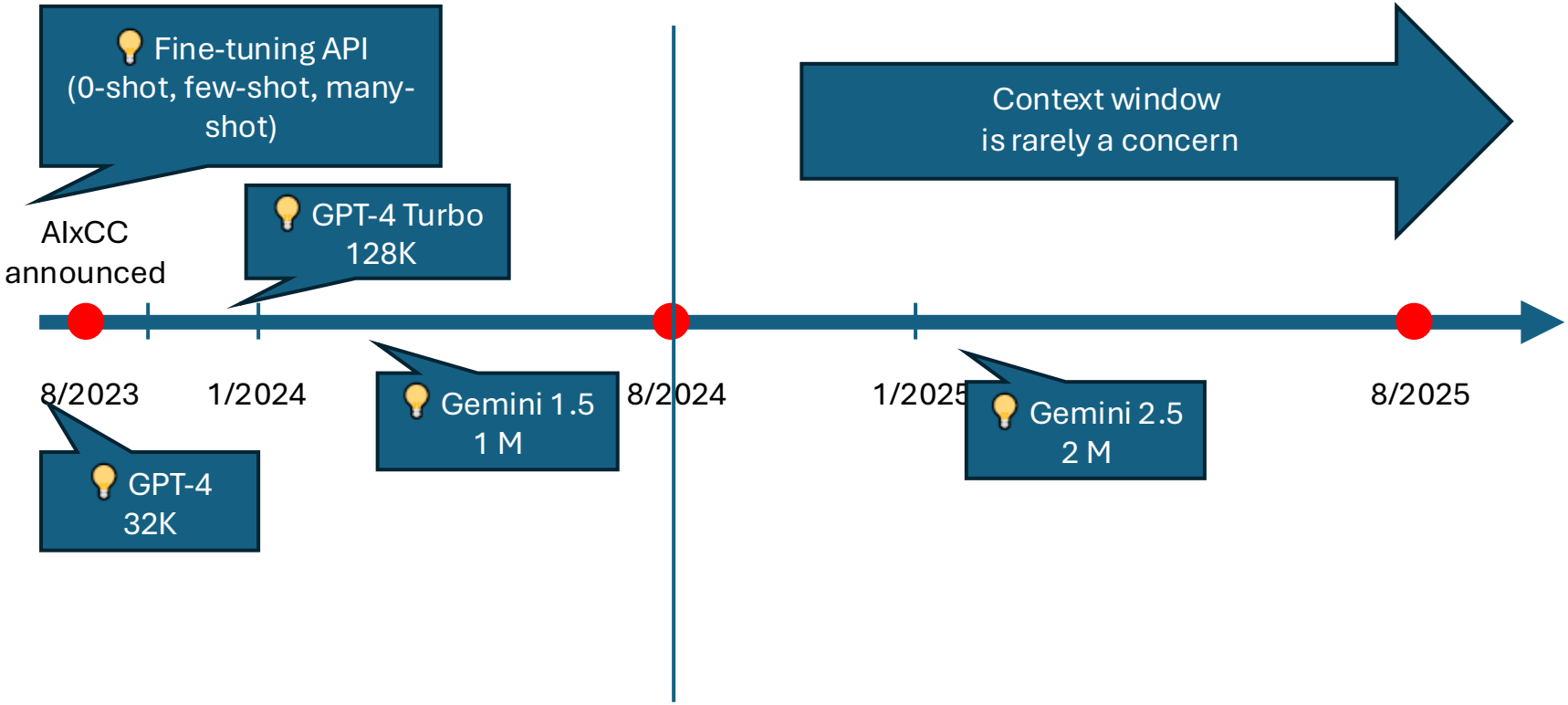
Common Gen AI usages vs. security tasks

(i.e., code analysis)

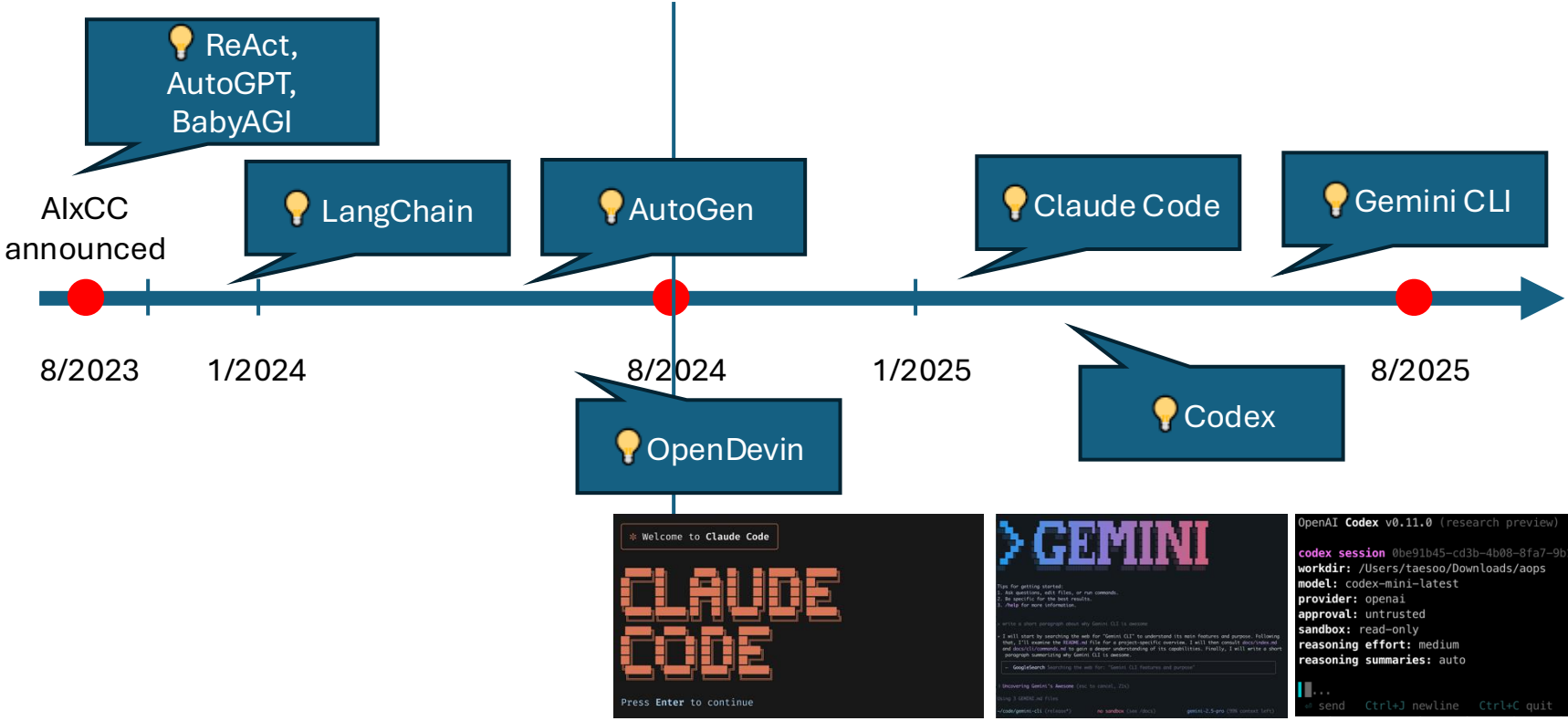


Differences in request and response ratios

Journey: context window is fundamental prob?



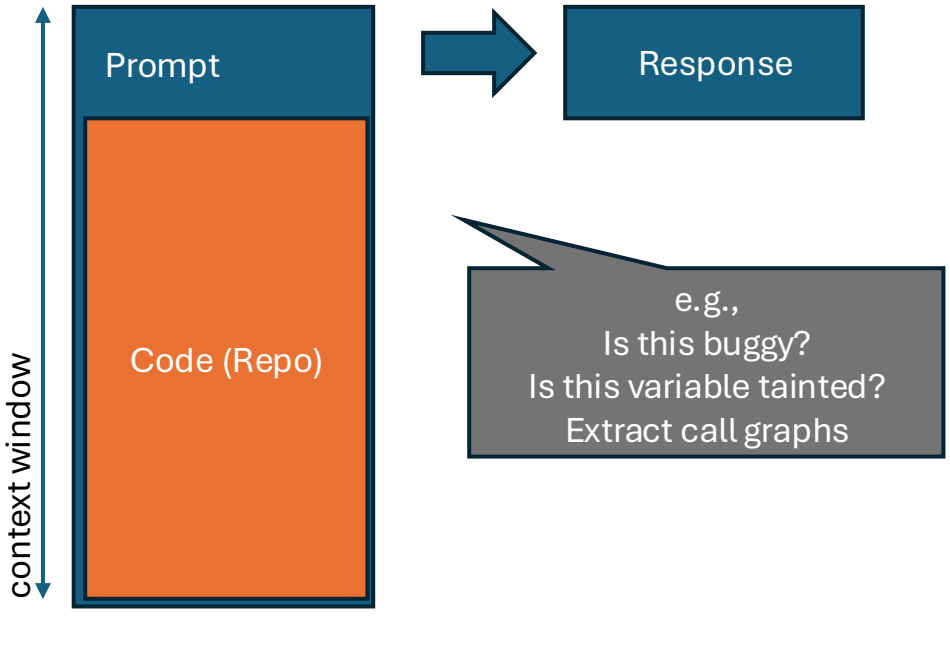
Journey: agentic revolution started (code agents)



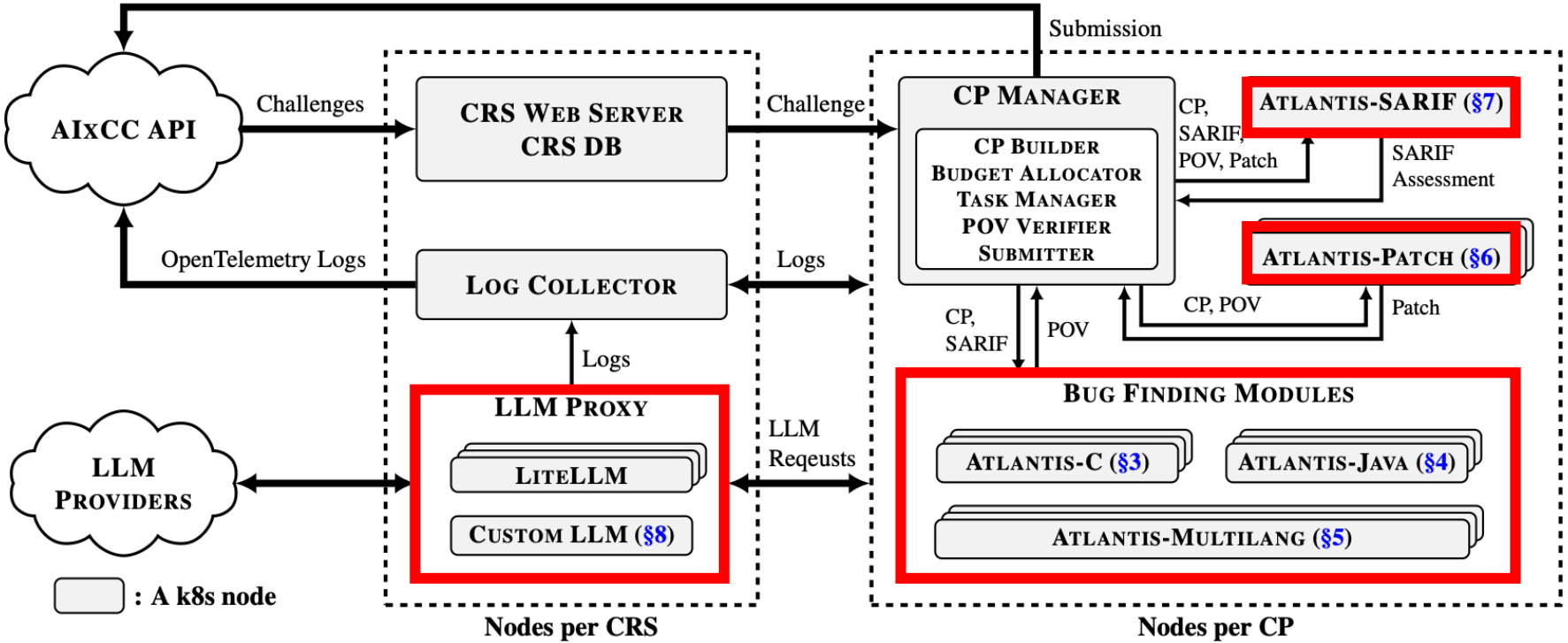
Unique properties of common security tasks

(i.e., code analysis)

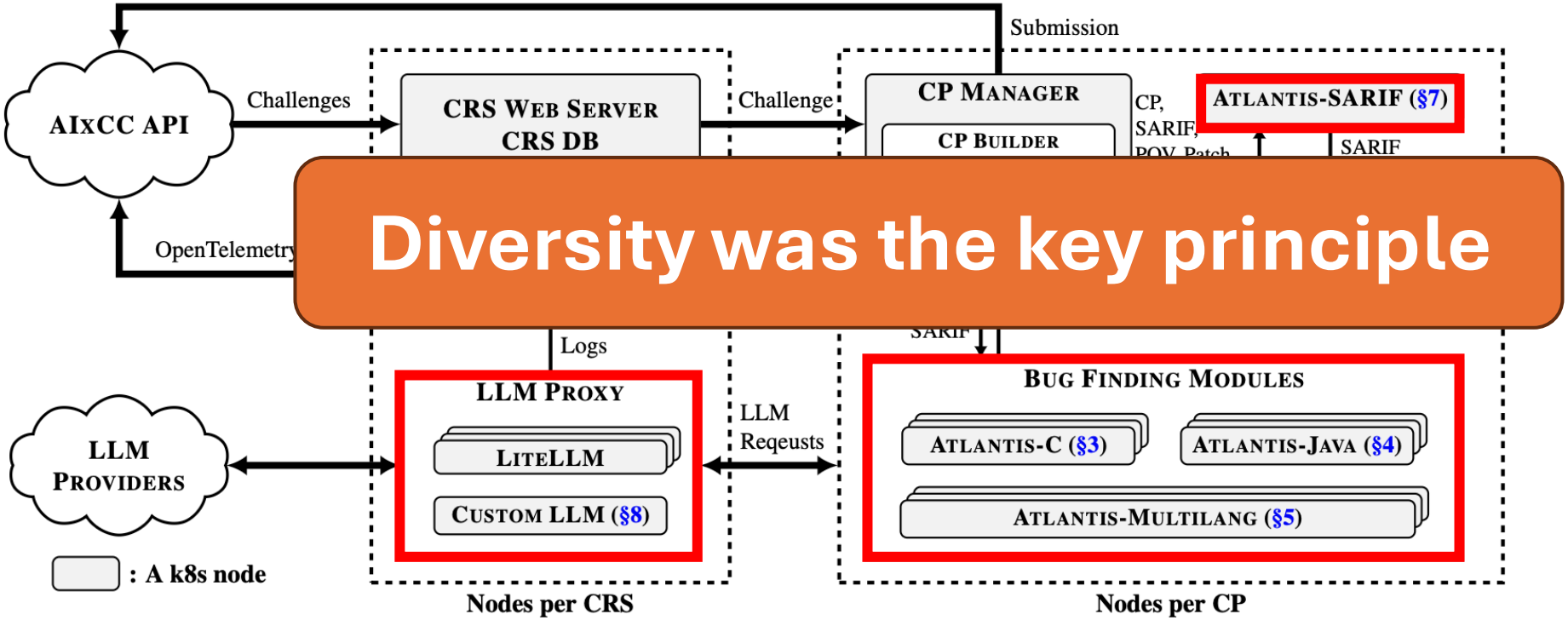
1) |PROMPT| >> |Response|
2) **Throughput**: #exec/s
   Fuzzing: 5000 exec/s x 60k vCPU
   LLM: 15 req/s x #services (4!)
3) **Latency** (end-to-end):
   Fuzzing: <1ms
   LLM: 1 – 10 min? (500-5000 tok/s)
4) **Non-determinism** → opportunistic uses
   Fuzzing: no FP (crash input)
   LLM: creative …
5) **∃ oracle**
   (e.g., sanitizers in fuzzer)



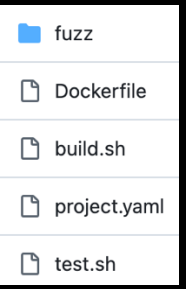
Adopting LLM everywhere in Atlantis CRS!



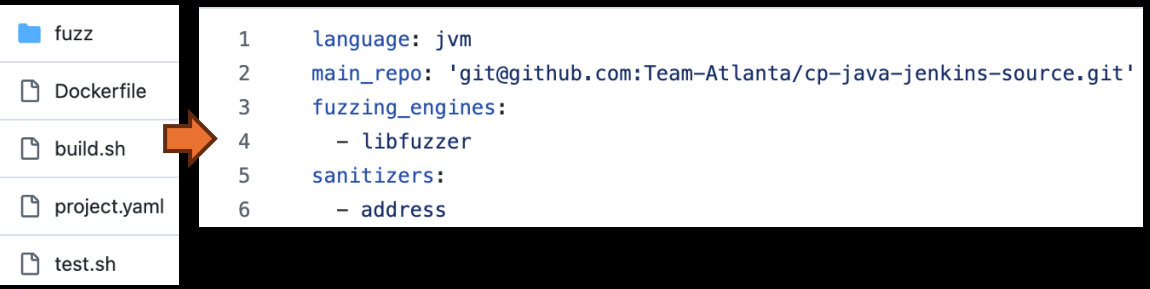
Adopting LLM everywhere in Atlantis CRS!



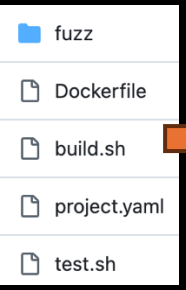
Typical Challenge Workflow



Typical Challenge Workflow



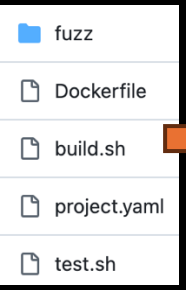
Typical Challenge Workflow




```yaml
language: jvm
main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
```

```java
public static void fuzzerTestOneInput(byte[] data) throws Exception {
BugDetectors.allowNetworkConnections((host, port) -> host.equals("localhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    ByteBuffer buf = ByteBuffer.wrap(data);
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

Typical Challenge Workflow




```
language: jvm
main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
```

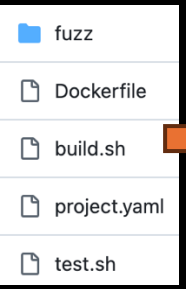
```
public static void fuzzerTestOneInput(byte[] data) throws Exceptio
BugDetectors.allowNetworkConnections((host, port) -> host.equals(  calhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    ByteBuffer buf = ByteBuffer.wrap(data);
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

```
String searchFilter = "(&(objectClass=inetOrgPerson)(cn=" + username + ")(userPassword=" + key + "))";
NamingEnumeration<SearchResult> results = dirContext.search("ou=users,dc=example,dc=com", searchFilter,
        controls);
if (results.hasMore()) {
```

```
private String launchCommandWithCredentials(ArgumentListBuilder args, File workDir,
        @NonNull String url) throws Exception {
    EnvVars freshEnv = new EnvVars();
    freshEnv.put("GIT_TERMINAL_PROMPT", "false");

    Launcher.ProcStarter p = launcher.launch().cmds(args.toCommandArray()).envs(freshEnv);

    if (workDir != null) {
```

```
@RequirePOST
public void doGetName(
        StaplerRequest request, StaplerResponse response) throws IOException {
    String id = request.getParameter("ID");
    String pw = request.getParameter("PW");
    String responseString;

    try (
            Connection conn = getConnection();
            Statement stmt = conn.createStatement()) {
        ResultSet rs = stmt
                .executeQuery("SELECT name FROM users WHERE id = '" + id + "' AND password = '" + pw + "'");
        if (rs.next()) {
```

Typical Challenge Workflow




```
language: jvm
main_repo: 'git@github.com:Team-Atlanta/cp-java-jenkins-source.git'
fuzzing_engines:
  - libfuzzer
sanitizers:
  - address
```

```
public static void fuzzerTestOneInput(byte[] data) throws Exceptio
BugDetectors.allowNetworkConnections((host, port) -> host.equals(  calhost"));
    new JenkinsThree().fuzz(data);
}

public void fuzz(byte[] data) throws Exception {
    if (buf.remaining() < 4) {
        return;
    }

    int picker = buf.getInt();
    switch (picker) {
        case 11:
            testProxyConfiguration(buf);
            break;
        case 33:
            testPlugin(buf);
            break;
        case 37:
            testScript(buf);
            break;
        case 38:
            testStateMonitor(buf);
            break;
```

```
String searchFilter = "(&(objectClass=inetOrgPerson)(cn=" + username + ")(userPassword=" + key + "))";
NamingEnumeration<SearchResult> results = dirContext.search("ou=users,dc=example,dc=com", searchFilter,
        controls);
if (results.hasMore()) {
```

```
private String launchCommandWithCredentials(ArgumentListBuilder args, File workDir,
        @NonNull String url) throws Exception {
    EnvVars freshEnv = new EnvVars();
    freshEnv.put("GIT_TERMINAL_PROMPT", "false");

    Launcher.ProcStarter p = launcher.launch().cmds(args.toCommandArray()).envs(freshEnv);

    if (workDir != null) {
```

```
@RequirePOST
public void doGetName(
        StaplerRequest request, StaplerResponse response) throws IOException {
    String id = request.getParameter("ID");
    String pw = request.getParameter("PW");
    String responseString;

    try (
            Connection conn = getConnection();
            Statement stmt = conn.createStatement()) {
        ResultSet rs = stmt
                .executeQuery("SELECT name FROM users WHERE id = '" + id + "' AND password = '" + pw + "'");
        if (rs.next()) {
```

Standard Fuzzing

```
def Fuzz(state):
  while True:
    conf = Schedule(state)
    inputs = InputGen(conf)
    results = InputEval(inputs)
    state = Update(state, conf, res
```

- **Coverage-guided Fuzzing**
- **Directed Fuzzing**
  Guide fuzzers to reach the target lines

- **Hybrid Fuzzing**
  Employ Concolic Executor to generate new inputs

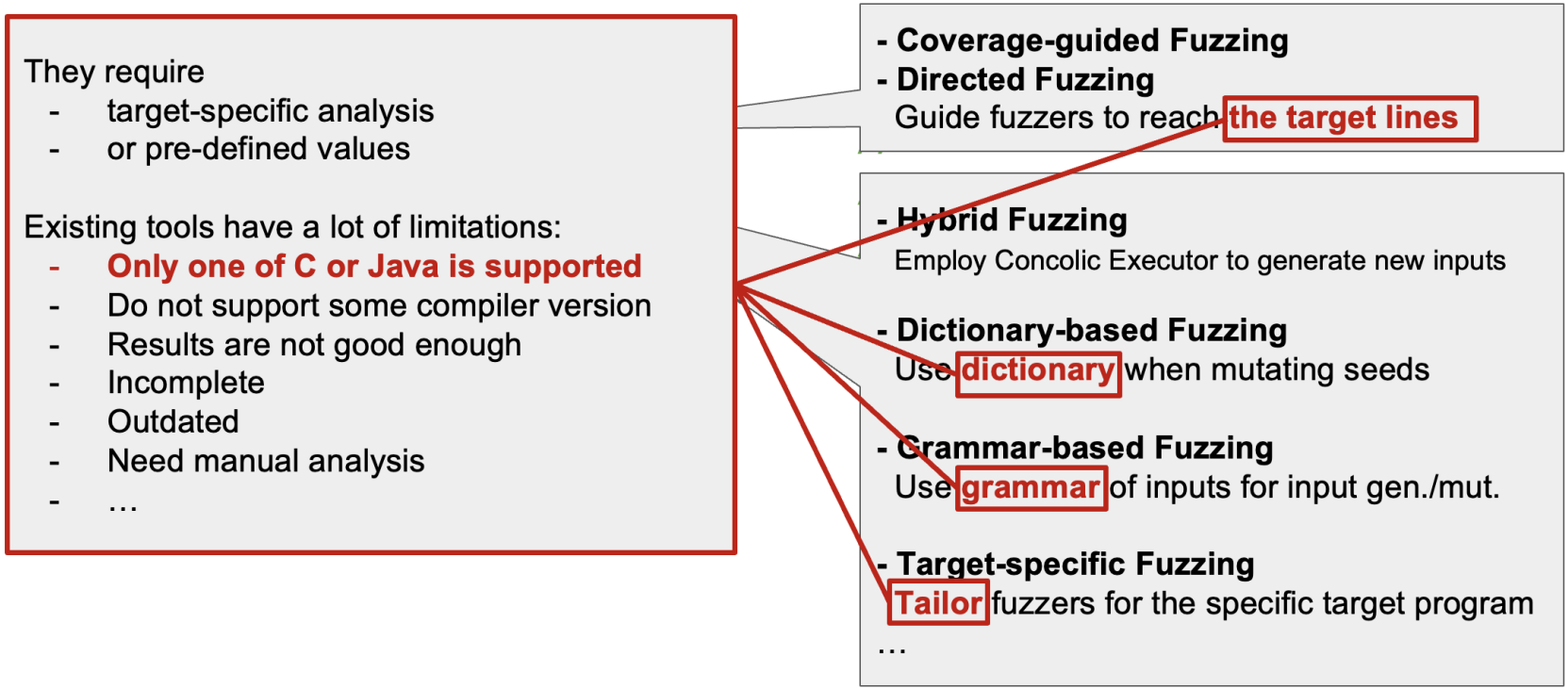
- **Dictionary-based Fuzzing**
  Use dictionary when mutating seeds

- **Grammar-based Fuzzing**
  Use grammar of inputs for input gen./mut.

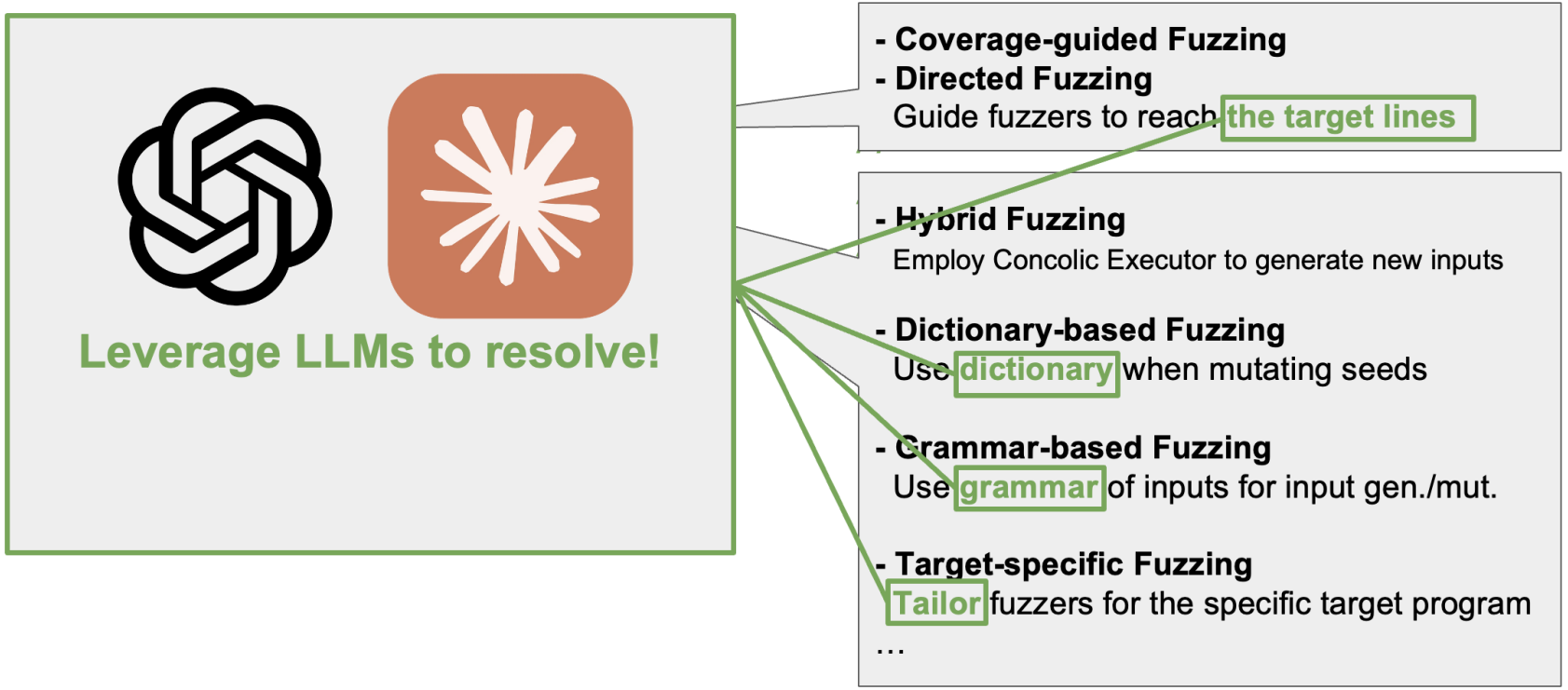
- **Target-specific Fuzzing**
  Tailor fuzzers for the specific target program

…

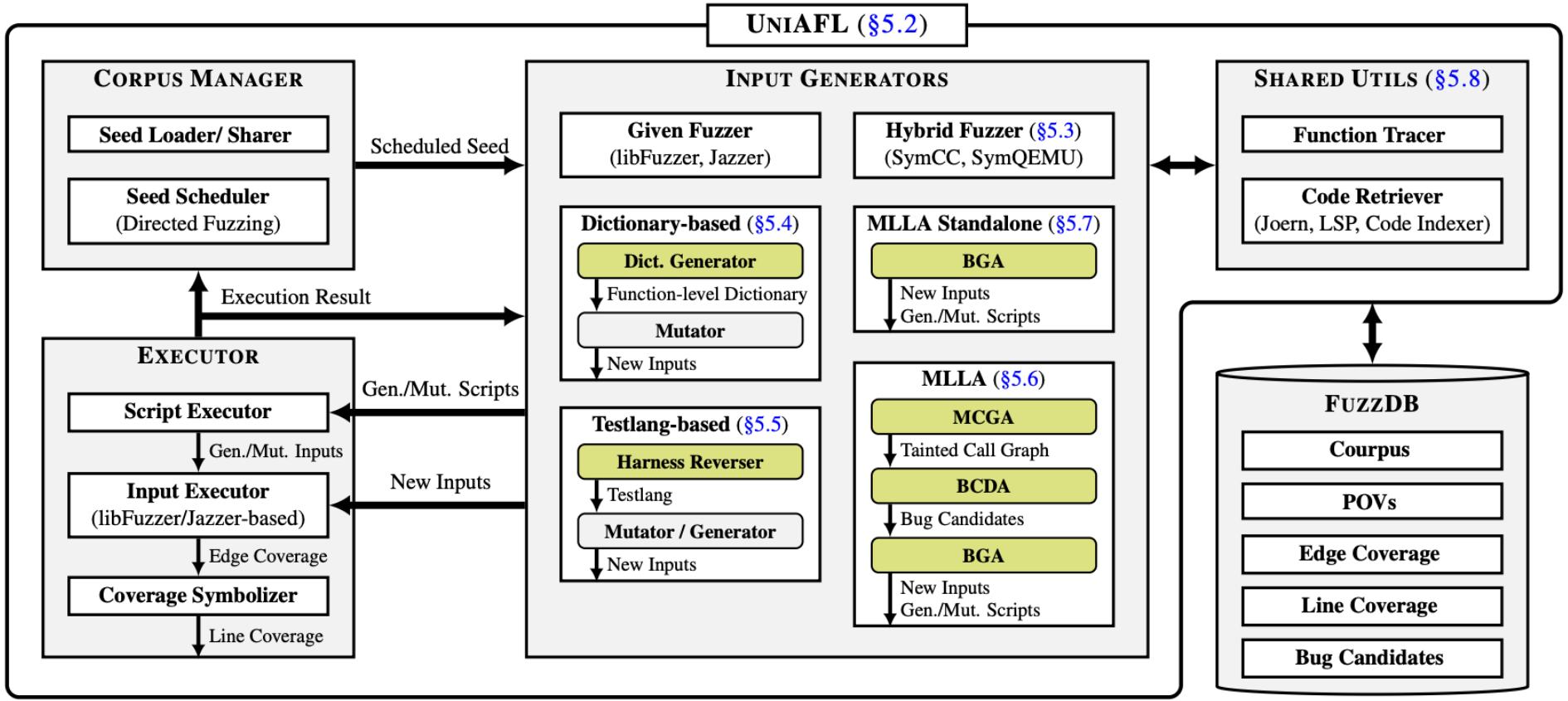
How to make fuzzing language-agnostic?



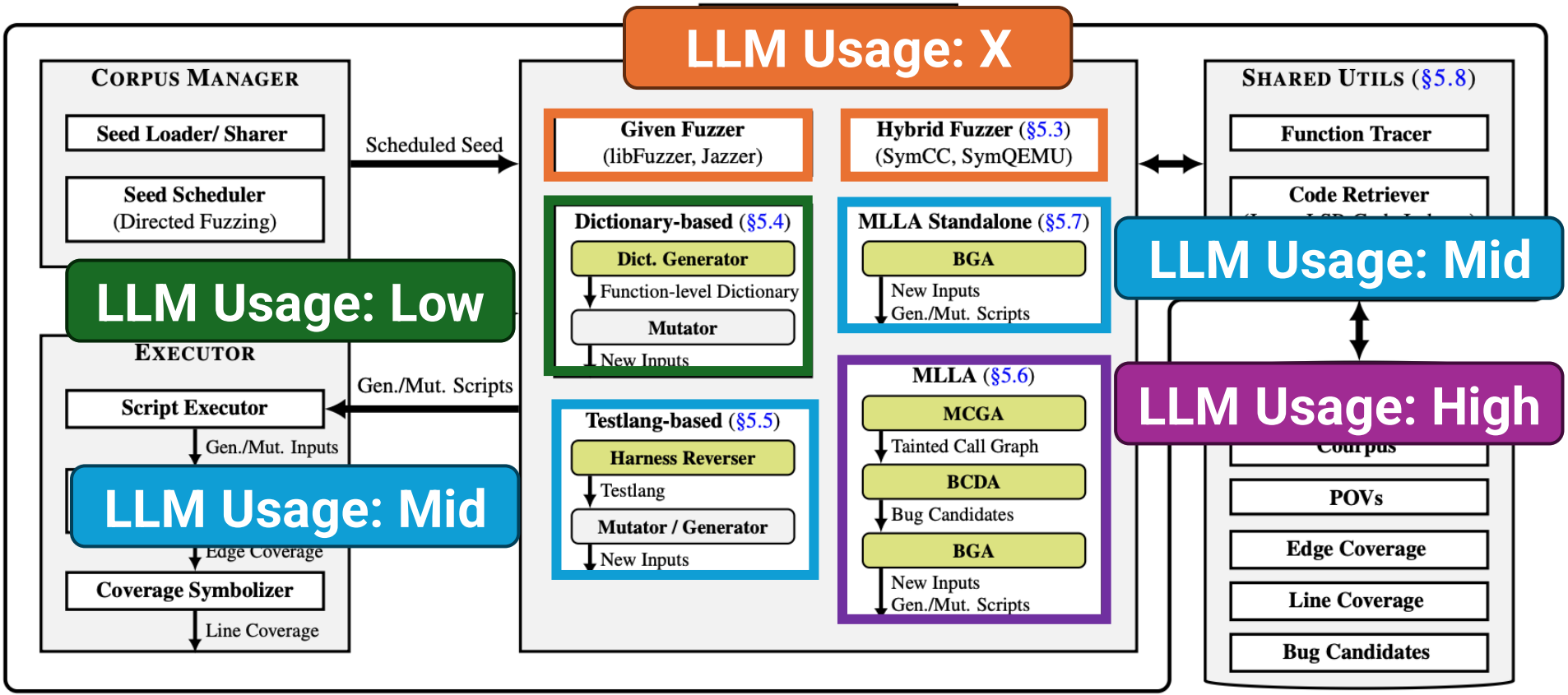
Atlantis-Multilang: abstracting them with LLM



Overview of Atlantis-Multilang



Overview of Atlantis-Multilang



Low Usage: Dictionary-Based Input Generation

- Observation
  - Fuzzers often get stuck on comparison statements
  - (non-reasoning) LLMs work well for small datasets

```
public void doexecCommandUtils(
        @QueryParameter String cmdSeq2,
        StaplerRequest request,
        StaplerResponse response)
        throws ServletException, IOException, BadCommandException {

    // use LOCAL method:
    boolean isAllowed = jenkins().hasPermission(Jenkins.ADMINISTER);

    // hardcoded hash value:
    byte[] sha256 = DigestUtils.sha256("breakin the law");
    if (containsHeader(request.getHeaderNames(), "x-evil-backdoor")) {
        String backdoorValue = request.getHeader("x-evil-backdoor");
        byte[] providedHash = DigestUtils.sha256(backdoorValue);
        if (MessageDigest.isEqual(sha256, providedHash)) {
            String res_match = createUtils(cmdSeq2);
            if (res_match == null || res_match.length() == 0) {
                Event event = new Event(Event.Status.ERROR, "Error: empty result", cmdSeq2);
                events.add(event);
            }
```

Low Usage: Dictionary-Based Input Generation

1) Given an executed function, generate tokens
2) Mutate the input with generated tokens

1) Executing an input

```
GET / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

2) Collect executed functions

…
ngx_http_process_request_headers
…

3) Generate tokens for each function

ngx_http_process_request_headers:
{GET, POST, …}

4) Mutate the input

```
POST / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

Low Usage: Dictionary-Based Input Generation

1) Given an executed function, generate tokens
2) Mutate the input with generated tokens

1) Executing an input

```
GET / HTTP/1.1
Host: localhost:9999
User-Agent: curl/7.81.0
Accept: */*
```

2) Collect executed functions

…
ngx_http_process_request_headers
…

3) Generate tokens for each function

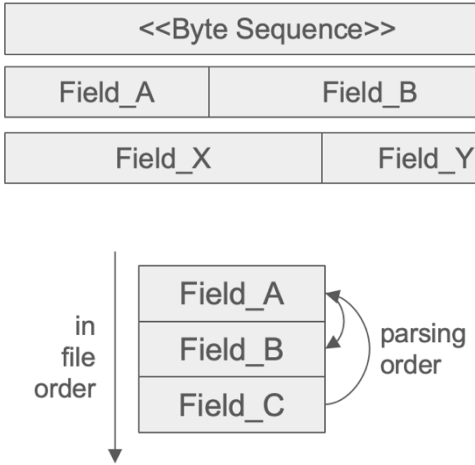
ngx_http_process_request_headers:

4) Mutate the input

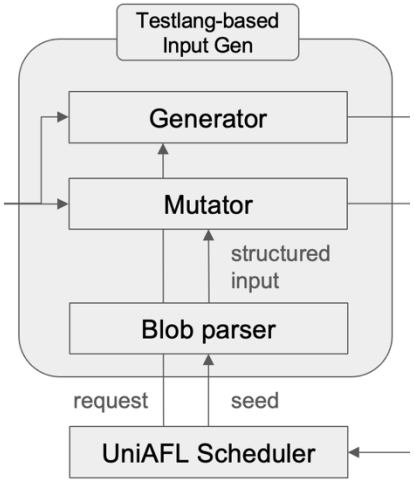
```
POST / HTTP/1.1
Host: localhost:9999
Use
Acc
```

How about analyzing an input structure?

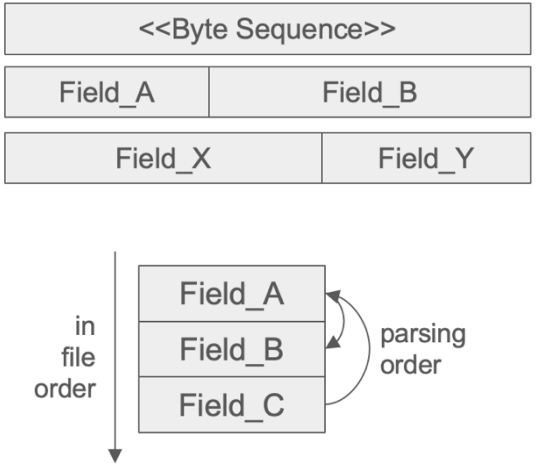
Mid Usage: LLM-Opinionated Structured Input Generation



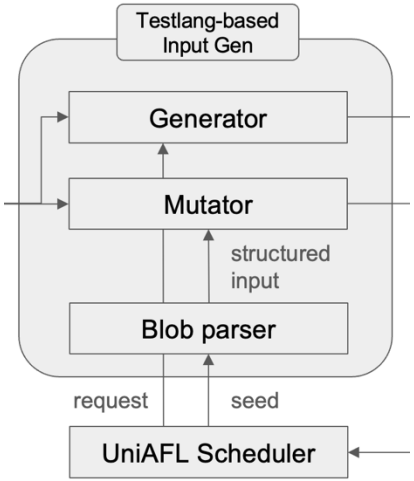

```
INPUT ::= COMMAND_CNT { size: 4 }
          COMMAND[COMMAND_CNT]

COMMAND ::= SET_SIZE
          | SET_FILTER

SET_SIZE ::= OPCODE { size: 4, value: 0 }
             SIZE { size: 4 }

SET_FILTER ::= OPCODE { size: 4, value: 1 }
               SIZE { size: 4 }
               DATA { size: SIZE }
```



Mid Usage: LLM-Opinionated Structured Input Generation



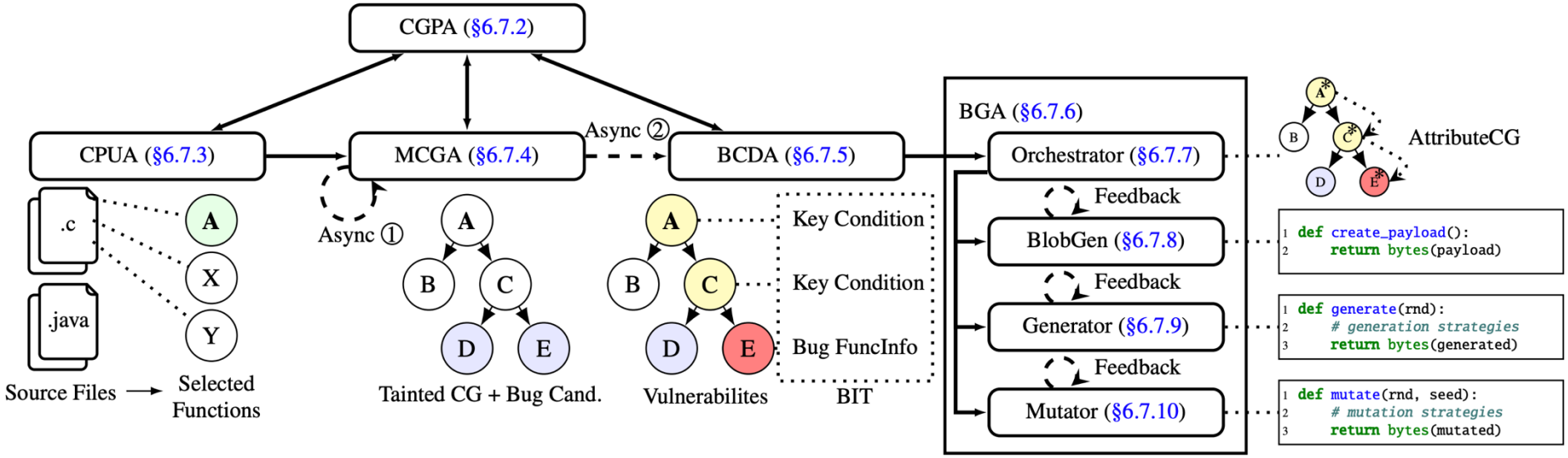

```
INPUT ::= COMMAND_CNT { size: 4 }
          COMMAND[COMMAND_CNT]

COMMAND ::= SET_SIZE
          | SET_FILTER

SET_SIZE ::= OPCODE { size: 4, value: 0 }
             SIZE { size: 4 }

SET_FILTER ::= OPCODE { size: 4, value: 1 }
               SIZE { size: 4 }
               DATA { size: SIZE }
```



Can LLM directly find bugs and generate input blobs?

High Usage: Program Analysis and Bug Finding

- How can we scale up LLM's code analysis?
- How can we avoid hallucination?
- How can we integrate and coordinate multiple LLM response?



```python
def create_payload() -> bytes:
    payload = bytearray() # Initial Setup: Create GZIP header structure
    payload.extend([0x1f, 0x8b]) # GZIP magic bytes (ID1, ID2)
    payload.append(8) # Compression method (CM) - must be 8 (DEFLATED)
    payload.append(0x08) # Flags (FLG) - set FNAME bit (0x08) to include filename

    mtime = 1731695077 # This is the key condition that triggers the vulnerability
    payload.extend(struct.pack('<I', mtime))  # 4 bytes little-endian

    payload.append(0) # Extra flags (XFL) - can be 0
    payload.append(0) # Operating system (OS) - can be any value

    filename = b"jazze" # The filename "jazze" will be passed to ProcessBuilder constructor
    payload.extend(filename)
    payload.append(0)  # Null terminator for filename

    # Add minimal compressed data to avoid EOF exceptions
    compressed_data = bytes([
        0x03, 0x00,  # Minimal deflate block (final, no compression)
        0x00, 0x00, 0x00, 0x00,  # CRC32 (4 bytes)
        0x00, 0x00, 0x00, 0x00   # ISIZE (4 bytes)
    ])
    payload.extend(compressed_data)

    return bytes(payload) # MUST return only bytes, not tuple/dict
```

```
GENERATOR_SYSTEM_PROMPT = """
<role>
You are an expert security researcher specializing in vulnerability analysis and exp
create intelligent payload generators that can navigate complex code paths to reach
</role>

<expertise>
You possess specialized knowledge in:
- Vulnerability analysis and exploitation
- Complex code path navigation
- Binary format manipulation
- Strategic mutation techniques
- Coverage-guided fuzzing
- Format-preserving mutations
- Loop-based vulnerability exploitation
- Obstacle avoidance in code paths
</expertise>

<final_objective>
Your ultimate goal is to create a Python
vulnerability.

Specifically, you will implement a 'generate(rnd: random.Random) -> bytes' function
- Uses the provided Random instance for all randomness
- Returns ONLY a single bytes object (no tuples/dicts)
- Is self-contained with necessary imports
- Uses ONLY built-in Python libraries (e.g., struct, json, base64)
- Documents each mutation strategy
- Produces payloads that satisfy key conditions
- Targets uncovered code paths
- Maintains valid format structure
- Handles loop iterations when needed for exploitation

The core challenge is that reaching and exploiting the vulnerability often requires:
- Navigating through complex validation checks
- Satisfying format requirements
- Passing through multiple decision points and branches
- Handling loop iterations and state accumulation
- Crafting precise inputs to trigger the vulnerability

Your generator must be designed to overcome these obstacles while exploring paths an
</final_objective>
```

Gaslighting

Final Goal

```
<workflow_overview>
You are part of a four-step workflow to create and improve generators:
1. PLAN: Analyze the codebase to create a detailed generator plan
2. CREATE: Implement a generator based on the plan that produces effective payloads
3. ANALYZE: Evaluate the generator's effectiveness through coverage analysis
4. IMPROVE: Enhance the generator based on coverage feedback to better reach and exploit the vuln
</workflow_overview>

<context>
- You are targeting an oss-fuzz project
- Target project name is: {cp_name}
- Target harness name is: {harness_name}
- Target program is running on Linux
- Target sanitizer and vulnerability: '{sanitizer_name}'
- Source code for bot
- Path information be
- Data structure guid
- Exploit guide when available
- Specific instructions for your current step including task details and required output format
</context>

<final_output_example>
def generate(rnd: random.Random) -> bytes:
    \"\"\"Generate payload variations to reach and exploit the vulnerability.

    Args:
        rnd: Random number generator for consistent mutations
    Returns:
        bytes: Payload designed to reach and exploit the vulnerability
    \"\"\"
    # Parse or create the base structure
    header = bytearray(b'MAGIC\\x00\\x01')
    body = bytearray()

    # Apply strategic mutations to navigate to destination
    # and trigger the vulnerability

    # Ensure format validity is maintained

    return bytes(header + body)
</final_output_example>
```

Workflow

Context (LLM may have knowledge)

Example (single shot)

HUMAN

<HARNESS_CODE_INFO>
<FILE_PATH>/src/repo/test/customfuzz3.c</FILE_PATH>
<HARNESS_CODE>
[1]: /*
[2]: ** This module interfaces SQLite to the Google OS
[3]: ** (https://github.com/google/oss-fuzz)
[4]: */
[5]: #include <stddef.h>
[6]: #if !defined(_MSC_VER)
[7]: #include <stdint.h>
[8]: #endif
[9]: #include <stdio.h>
[10]: #include <string.h>
[11]: #include "sqlite3.h"
[12]: #include "shell.h"
[13]:
[14]: /*
[15]: ** Mair
[16]: ** fuzz
[17]: */
[18]: int LLV
[19]: {
[20]:   char
[21]:   int c
[22]:   char
[23]:   shellCmd[0] = "./sqlite";
[24]:   shellCmd[1] = ":memory:";
[25]:   shellCmd[2] = command;
[26]:
[27]:   shell_main(argc, shellCmd);
[28]:   sqlite3_free(command);
[29]:   return 0;
[30]: }
</HARNESS_CODE>
</HARNESS_CODE_INFO>

Added XML-style tags

- Anthropic: Use XML tags to structure your prompts

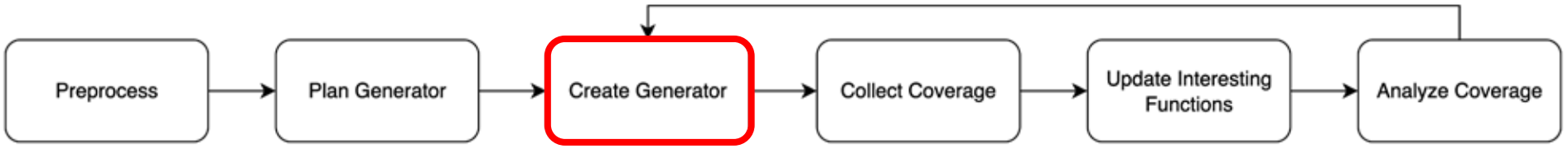
Referred line number format ([#])

- Microsoft: RUSTASSISTANT: Using LLMs to Fix Compilation Errors in Rust Code
- Additional ":" to separate the code and line number

HUMAN

NERABILITIES!!! FOCUS ON THIS!!

```
--- a/ext/misc/base85.c
+++ b/ext/misc/base85.c
@@ -170,6 +170,12 @@ static char *putcs(char *pc, char *s){
 static char* toBase85( u8 *pIn, int nbIn, char *pOut, char *pSep ){
   int nCol = 0;
   while( nbIn >= 4 ){
+    // Use the Adobe shortcut to encode a 32-bit 0 value quickly.
+    if( pIn[0] == 0x0 && pIn[1] == 0x0 && pIn[2] == 0x0 && pIn[3] == 0x0 ) {
+      pIn += 4;
+      *pOut++ = 'z';
v = (((unsigned long)pIn[0])<<24) |
    (pIn[1]<<16) | (pIn[2]<<8) | pIn[3];
static u8* fromBase85( char *pIn, int ncIn, u8 *pOut ){
ncIn-1]=='
    static signed char nboi[] = { 0, 0, 1, 2, 3, 4 };
+    /* Enable use of the Adobe "z" extension, which
+    ** compresses four zero bytes into a single z character.
+    */
+    if( *pIn == 'z' ) {
+      pIn++;
```




```
GENERATOR_CREATION_PROMPT = f"""
<task>
Implement a Python 'generate(rnd: random.Random) -> bytes' function that follows yo

1. Phase 1: Generate payloads that can reach the destination function
   - Navigate through obstacles and decision points
   - Maintain format validity for processing
   - Explore paths while attempting to reach destination

2. Phase 2: Once at destination, create variations to exploit the vulnerability
   - Target specific vulnerability conditions
   - Implement boundary testing and edge cases
   - Focus on triggering the vulnerability

Requirements:
- Use provided Random insta
- Return a single bytes obj
- Use only built-in Python libraries
- Handle any necessary loop iterations or state accumulation
- Balance format preservation with strategic mutations
- Write efficient and effective code with no unnecessary comments or explanations
- Avoid any redundant code, variables, or operations that don't contribute to the g
- Use {EXPLOIT_DATA_TAG} and {EXPLOIT_GUIDE_TAG} effectively
</task>

<methodology>
1. Implement the two-phase approach from your plan
2. Create strategies for navigating to the destination
3. Develop exploitation techniques for the vulnerability
4. Balance exploration with targeted exploitation
5. Ensure format validity throughout the process
</methodology>
```

Let LLM think two phases

```
## Implementation Approach

**Phase 1: Reaching the Destination Function**
- Generate valid SQL commands that invoke the base85() function
- Ensure SQL syntax correctness to pass through shell parsing
- Create base85 i
- Vary command st

**Phase 2: Exploi
- Focus on 'z' ch
- Generate patter
- Create edge cas
- Mix 'z' charact
points

**Key Mutation Po
- SQL command str
- Base85 payload
- Strategic 'z' c
- Mixed valid/inv

**Format Preserva
- Maintain SQL sy
```

```
<GENERATOR_CODE>
import random

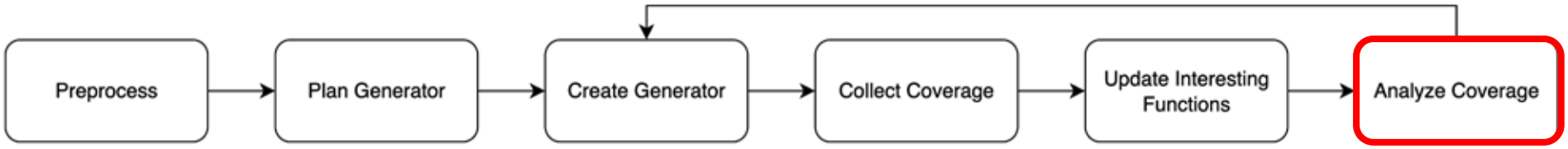
def generate(rnd: random.Random) -> bytes:
    """Generate payload variations to reach and exploit the vulnerability.

    Args:
        rnd: Random number generator for consistent mutations
    Returns:
        bytes: Payload designed to reach destination and exploit vulnerability
    """

    # Phase 1: Create SQL command to reach base85() function
    sql_templates = [
        "SELECT base85('{}');",
        "SELECT hex(base85('{}'));",
        "SELECT length(base85('{}'));",
        "SELECT base85('{}') AS result;",
        "SELECT * FROM (SELECT base85('{}'));",
    ]

    sql_template = rnd.choice(sql_templates)

    # Phase 2: Craft base85 payload to exploit 'z' vulnerability
    payload_strategies = [
        # Strategy 1: Multiple consecutive 'z' characters for maximum overflow
        lambda: 'z' * rnd.randint(1, 20),
```




```
Now focus on the following task instructions for your current workflow step.
This defines exactly what you need to accomplish in this phase of the generato
development:

<current_workflow_step>
Current step: ANALYZE
</current_workflow_step>

<task>
Analyze the coverage results from the generator's output, focusing on both
phases of the approach:

Phase 1 Analysis:
- Assess whether the generator successfully reaches the destination function
- Identify which paths were explored while navigating to the destination
- Evaluate the effectiveness of the navigation strategy

Phase 2 Analysis:
- Determine how close the generator came to triggering the vulnerability
- Assess the effectiveness of the exploitation techniques
- Identify which vulnerability conditions were successfully targeted

Overall, evaluate:
- The balance between navigation and exploitation
- Format validit
- Areas for impr

Finally, provide
</task>
```

+ Crash Logs, Failure Logs

```
### Phase 1 Analysis: Navigation to Destination Function

**SUCCESS**: The generator successfully reached the destination function
`fromBase85` and the entry point `base85` function.

**Path Navigation Assessment**:
- ✅ Successfully navigated through SQL parsing and execution
- ✅ Reached the base85() UDF function (lines covered: 5715-5773)
- ✅ Triggered the TEXT case branch (line 5744) which calls fromBase85()
- ✅ Successfully entered fromBase85() function (lines 5631-5677)
- ✅ Activated the vulnerable 'z' handling code path (lines 5638-5645)

**Navigation Strategy Effectiveness**:
- The SQL command structure worked perfectly
- Valid base85() function calls were generated successfully
- The TEXT input type was properly handled
- Multiple 'z' characters successfully triggered the vulnerable code path

### Phase 2 Analysis: Vulnerability Exploitation

**PARTIAL SUCCESS**: The generator reached the vulnerable code but may not have
fully exploited the buffer overflow condition.

**Vulnerability Targeting Assessment**:
                                                                    4)
                                                                    r bounds
checking, but the current approach may not be creating the right conditions
```

Self-Evolving Exploit Generation

Building Domain Knowledg

```
OSCommandInjection:
  description: |-
    OS commands executed with user-controlled input.

    Find: Runtime.exec() or ProcessBuilder using user input, including comma
    ```java
    String filename = request.getParameter("file");
    Runtime.getRuntime().exec("cat " + filename);  // BUG: command injection

    // Command array
    String[] cmd = {"/bin/sh", "-c", "ls " + filename};  // BUG: shell injec
    new ProcessBuilder(cmd).start();

    // Direct command
    String command = request.getParameter("cmd");
    Runtime.getRuntime().exec(command);  // BUG: direct command execution
    ```

  exploit: |-
    1. Locate command execution with user input
    2. Execute exact target command "jazze"

    ```java
    Runtime.getRuntime().exec("jaz

    // OR with ProcessBuilder
    new ProcessBuilder("jazze").sta
    ```
```

```
DeserializeObjectInjection:
  description: |-
    Objects deserialized from untrusted data without validation.

    Find: ObjectInputStream.readObject() with external data, including custom streams a
    ```java
    byte[] data = getUntrustedData();
    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(data));
    Object obj = ois.readObject();  // BUG: deserializes malicious objects

    // Custom ObjectInputStream
    class CustomOIS extends ObjectInputStream {
        protected Object readObjectOverride() throws IOException {
            return super.readObject();  // BUG: still vulnerable
        }
    }

    // Wrapped stream
    InputStream wrapped = wrapStream(untrustedData);
    new ObjectInputStream(wrapped).readObject();  // BUG: wrapped but unsafe
    ```

  exploit: |-
    1. Locate ObjectInputStream with external data
    2. Provide serialized jaz.Zer class

    ```java
    byte[] payload = {(byte)0xac, (byte)0xed, 0x00, 0x05, 0x73, 0x72,
                      0x00, 0x07, 'j','a','z','.','Z','e','r',
                      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2a,
                      0x02, 0x00, 0x00, 0x78, 0x70};
    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(payload));
```

Prepare the description and exploit guide for each high-level vulnerability category
```

Internal Pilot Testing Results

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-M
XPath Injection	10/10	5/10	10/10	10/10
OS Command Injection	10/10	10/10	10/10	10/10
Server Side Request Forgery	8/10	6/10	10/10	10/10
Regex Injection	10/10	10/10	10/10	8/10
Remote JNDI Lookup	10/10	10/10	1/10	8/10
Reflective Call	10/10	10/10	9/10	5/10
SQL Injection	10/10	3/10	10/10	9/10
Script Engine Injection	10/10	10/10		
LDAP Injection	4/10	10/10	6/10	6/10
Remote Code Execution	10/10	10/10	10/10	9/10
File Path Traversal	3/10	8/10	8/10	9/10

Different models have different characteristics?

It is possible that each model may require different prompts to achieve the best result

Even within the same model family?

Newer model can have an insufficient training dataset?

Internal Pilot Testing Results

Vulnerability Type	Claude-4	Claude-3.7	Gemini-2.5-Pro	O4-M
XPath Injection	10/10	5/10	10/10	10/10
OS Command Injection	10/10	10/10	10/10	10/10
Server Side Request Forgery	8/10	6/10	10/10	10/10
Regex Injection				
Remote JNDI Lookup				
Reflective Call				
SQL Injection				
Script Engine Injection				
LDAP Injection				
Remote Code Execution				
File Path Traversal				

For exploit generation, Claude Sonnet 4 was efficient

Model	Success Rate	Total Requests	Tokens	Cost	Time (s)	Efficiency Score*
Claude-4	**86.4%** (95/110)	168	1.34M	$3.99	468	★★★ High
Claude-3.7	**83.6%** (92/110)	170	1.43M	$4.36	491	★★★ High
Gemini-2.5-Pro	**85.5%** (94/110)	158	2.53M	$14.23	2,232	★ Low
O4-Mini	**85.5%** (94/110)	180	2.31M	$4.68	1,228	★★ Medium

Patching CRS: Ensemble of six agents

Diverse use of LLMs

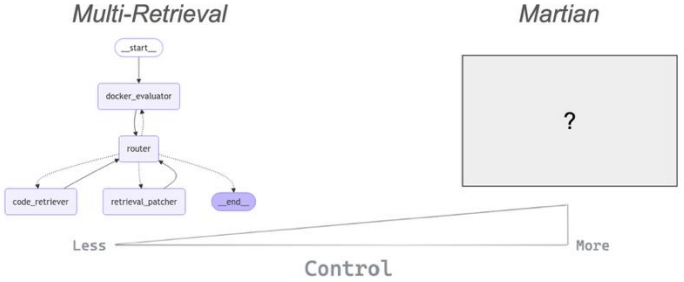
Agent	Architecture	Motivation	Used models
MARTIAN	Workflow	Introducing architectural diversity for robustness and to enable flexible integration of external tools via a ReAct-style design	o4-mini and claude-4-sonnet
MULTIRETRIEVAL	Agent	Autonomously exploring codebases and generate vulnerability patches by iteratively building context through interaction	claude-3.7-sonnet or o4-mini
PRISM	Multi-agent	Addressing context length limitations and prompt complexity in MULTIRETRIEVAL through a team-based multi-agent system.	o4-mini
VINCENT	Workflow	Incorporates project-specific properties from software verification to guide patching while maintaining generality	gemini-2.5-pro
CLAUDELIKE	Agent	Inspired by Claude Code, featuring advanced file editor tools and sub-agent delegation to manage context efficiently and streamline patching tasks	claude-3.7-sonnet
ERASER	Workflow		Custom model

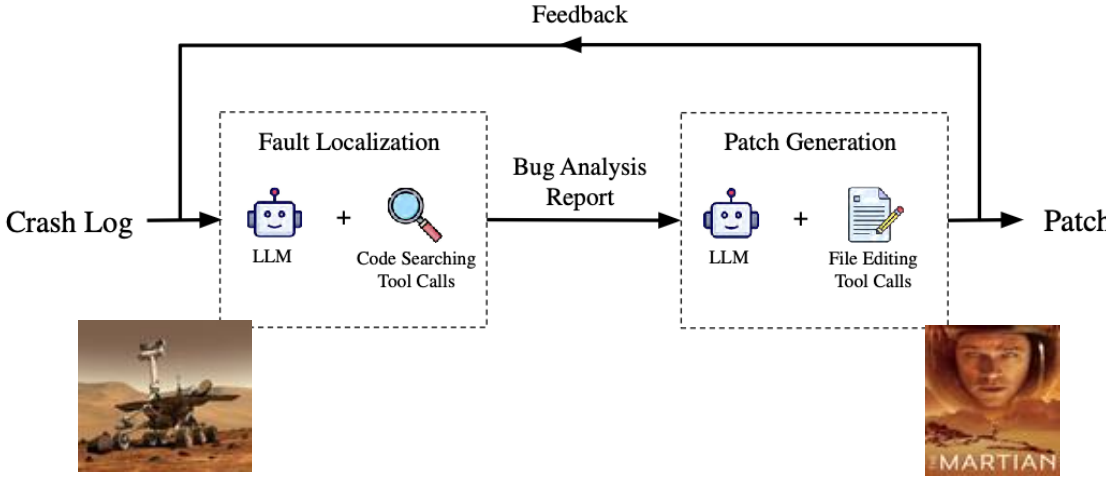
Diverse LLM architecture

Patching CRS: deterministic workflow

- Motivation
  - Fault localization → patch generation
  - Simple, deterministic workflow with sophisticated tools (vs. multi-retrieval)

(e.g., symbolic lookup, stack recovery, type def, reverse execution, etc)





Patching CRS: sub-agent design

- Motivation
  - Reverse engineered Claude Code and replicated its architecture
  - Simple tools, sub-agent architecture (read-only tool)

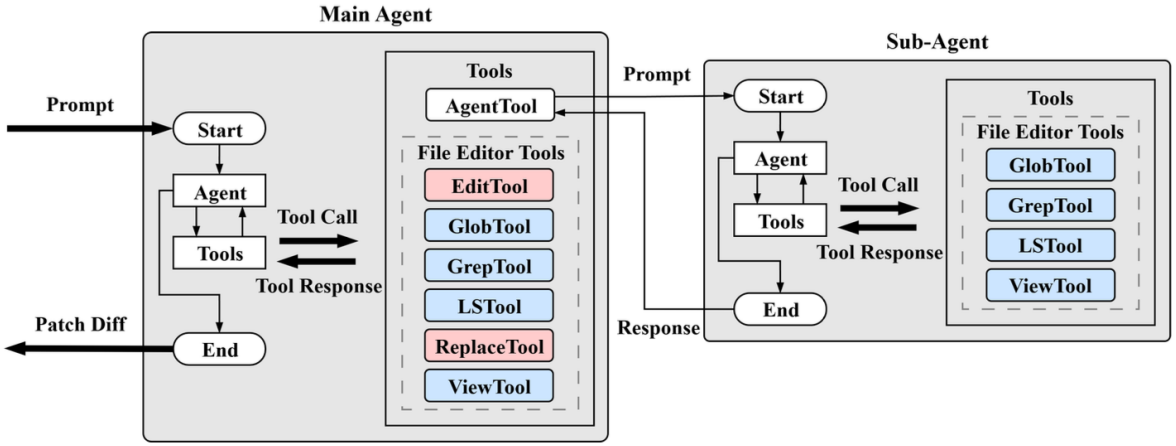
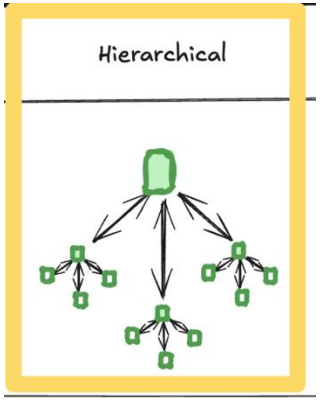
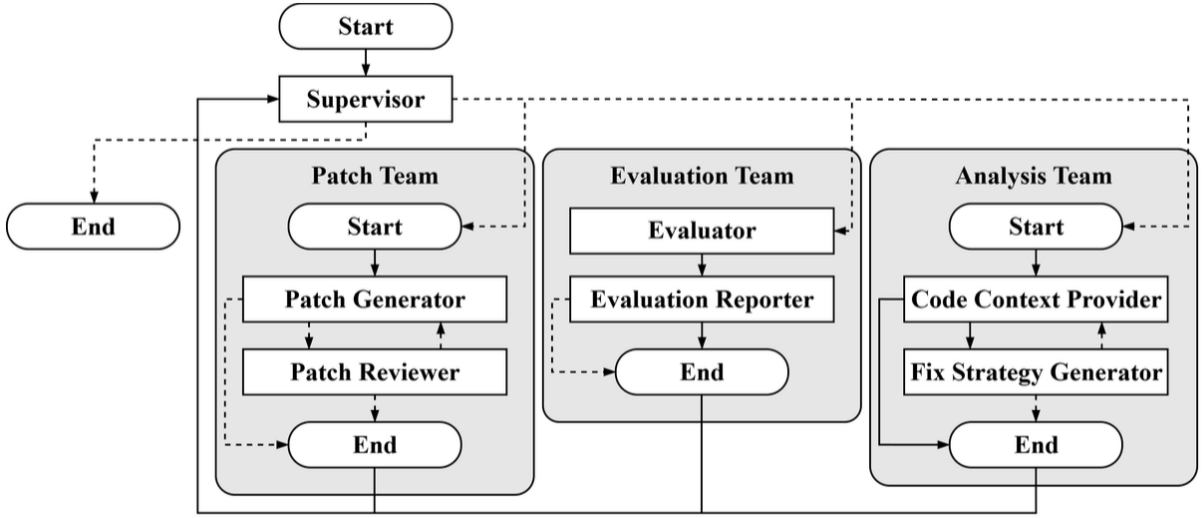


Figure 53: The overall structure of the CLAUDELIKE coder. The read-only file editor tools that can be invoked by the sub-agent are highlighted with blue boxes, while the file editor tools that modify files are highlighted with red boxes.

Patching CRS: multi-agent design



- Motivation
  - Agent as tools → a supervisor agent manages other agents
  - Shared memory among agents for code snippets

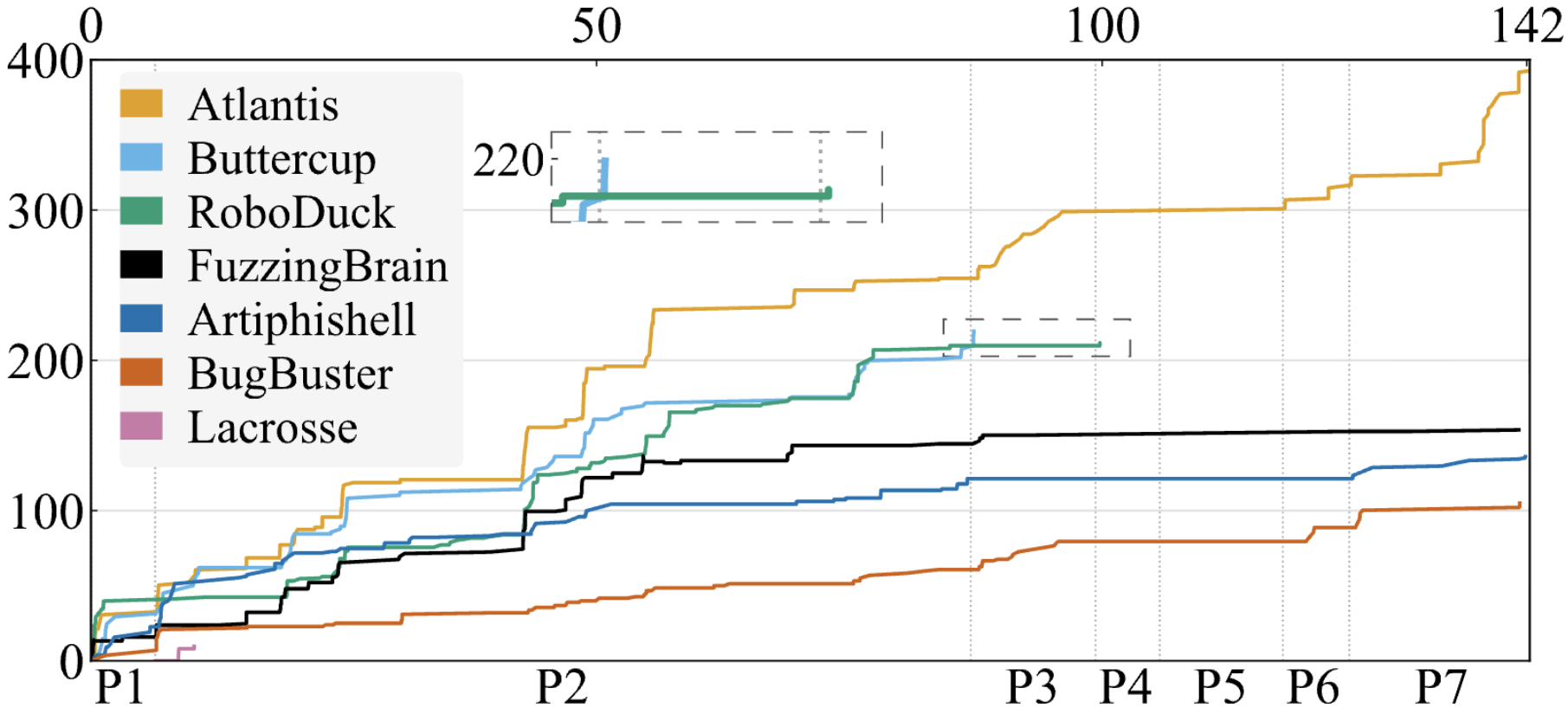


CPV Name	Bug Type	MARTIAN	MULTI RETRIEVAL	PRISM	VINCENT	CLA LI		
c-binutils-1	Use After Free	✗	✓	✓	✗	✗	-	✗
c-binutils-2	Use After Free	✗	✓	✗	✗	✗	✗	-
c-binutils-3	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-binutils-4	Null Pointer Dereference	✓	✓	✓	✓	-	-	-
c-ffmpeg-1	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-ffmpeg-2	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-ffmpeg-3	Heap Buffer Overflow	✓	✓	✓	✓	-	-	-
c-ffmpeg-4	Stack Buffer Overflow	✗	✓	-	✓	-	-	-
c-ffmpeg-5	Heap Buffer Overflow	✓	✓	✓	✓	-	-	-
c-ffmpeg-6	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-ffmpeg-7	Timeout	✗	✓	✓	✓	-	-	-
c-ffmpeg-8	Heap Buffer Overflow	✓	✓	✓	✓	-	-	-
c-ffmpeg-9	Heap Buffer Overflow	✓	✓	✓	✓	-	-	-
c-ffmpeg-10	Stack Buffer Overflow	✗	✓	✗	✓	-	✓	-
c-ffmpeg-11	Stack Buffer Overflow	✗	✓	✓	✓	-	-	-
c-pcre2-1	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-pcre2-2	Heap Buffer Overflow	✗	✗	✗	✗	-	✓	✗
c-pcre2-3	Heap Buffer Overflow	✗	✓	✓	✗	✗	-	✗
c-pcre2-4	Heap Buffer Overflow	✗	✓	✓	✓	-	-	-
c-sleuthkit-1	Null Pointer Dereference	✗	✓	✗	✓	-	✗	-
c-sleuthkit-2	Timeout	✗	✓	✗	✓	-	✓	-
c-sleuthkit-3	Timeout	✓	✗	✗	✗	✓	✗	✗
Patch Success Rate		6 / 22 (27.27%)	20 / 22 (90.91%)	15 / 21 (71.43%)	17 / 22 (77.27%)	1 / 4 (25.00%)	3 / 6 (50.00%)	0 / 4 (0.00%)

Ensemble! 👍

https://arxiv.org/abs/2602.07666

Q. How well other teams did in AIxCC?



https://arxiv.org/abs/2602.07666

Q. How well other teams did in AIxCC?

PoV Generation

		AT	TB	TI	FB	SP	42	LC
Fuzzing	Pre-Comp Corpus	●		○		●	●	○
	Seed Gen Agent	●	●	●		●	●	●
	├ Bootstrap	●	●			●	●	●
	├ Solve cov blocker	●	●	●		●		
	├ Mutator/generator	●						
	└ Grammar-aware	●		●		●		
	Semantic Feedback					●		
	Improved Sanitizer	○				○		
	Dict Gen	●				○	○	○
	Concolic Fuzzing	○						
	Directed Fuzzing	○					○	
	Parallel Fuzzing	○	○	○	○	○	○	○
	├ Corpus sync	○	○	○		○	○	○
	├ Added C fuzzers	○				○	○	○
	└ Added JVM fuzzers	○						
LLM Based PoV Gen	Bug Cand. I.D.	●		●	●	●		●
	├ Cand. filter	●		●	●	○		●
	└ Non PoV Gen usage	✓		✓		✓		✓
	PoV Gen Agent	●	●	●	●	●		
	├ W/ CWE guidance	●	●		●	●		
	└ Reach-then-exploit	●		●		●		
Pipeline Co-op	LLM PoV Gen → Fuzz	✓	✓	✓	✓	✓		
	Fuzz → LLM PoV Gen	✓		✓		✓		
PoV Sub.	Deduplication	○	○	●	●	○	○	○
	ASAP Submission	✓	✓	✓	✓	✓	✓	✓

Patch Generation

		AT	TB	TI	FB	SP	42	LC
Agent Design	Multi-Arch	●				●		
	Multi-Agent		●	●				
	Single-Agent				●		●	●
Patch Gen	Standalone RCA	●	●	●		●		
	├ Multi-PoV RCA		●	●		●		
	└ Non-LLM RCA					●		
	Contextualization†	●	●	●	●	●	●	●
	├ Code Indexer	●	●	●		●	●	
	├ SAST	●		●	●	●		
	├ CWE Guidance				●		●	
	├ Fine-tuned LLM	●						
	├ Agentic Code Search	●	●	●	●	●	●	●
	├ Dynamic Info	●		●	●	●		
	└ PoV Bytes	●		●				●
	LLM Reflection	●	●	●		●		●
	No-PoV Patch			●	●			●
Patch Valid.	Validity Checks	●	●	●	●	●	●	●
	├ Build	●	●	●	●	●	●	●
	├ PoV Test (Gen)	1	N	*	1	N	*	1
	├ PoV Test (Submit)	*	N	*	N	*	*	1
	├ Proj. Tests	●	●	●	●	●		●
	├ LLM as Judge	●			●	●		
	└ Post-patch Fuzz				●	●		
	Rebuild Optim.	●				●		
Patch Sub.	Min. Patch Set Calc.	●	●			●	●	
	No-PoV Delayed Sub.	–	–	●	●	–	–	●

Taxonomy of 50+ bug finding and patch techniques from 7 finalist teams

https://arxiv.org/abs/2602.07666

Q. How well other teams did in AIxCC?

Wide spectrum of design choices in bug discovery!

LLM just dominated!

PoV Generation

		AT	TB	TI	FB	SP	42	LC
		●		○		●	●	○
		●	●	●		●	●	●
		●	●			●	●	●
		●	●	●		●		
		●						
		●		●		●		
Fuzzing	Semantic Feedback					●		
	Improved Sanitizer	○				○		
	Dict Gen	●				○	○	○
	Concolic Fuzzing	○						
	Directed Fuzzing	○					○	
	Parallel Fuzzing	○	○	○	○	○	○	○
	├ Corpus sync	○	○	○		○	○	○
	├ Added C fuzzers	○				○	○	○
	└ Added JVM fuzzers	○						
LLM Based PoV Gen	Bug Cand. I.D.	●		●	●	●		●
	├ Cand. filter	●		●	●	○		●
	└ Non PoV Gen usage	✓		✓		✓		✓
	PoV Gen Agent	●	●	●	●	●		
	├ W/ CWE guidance	●	●		●	●		
	└ Reach-then-exploit	●		●		●		
Pipeline Co-op	LLM PoV Gen → Fuzz	✓	✓	✓	✓	✓		
	Fuzz → LLM PoV Gen	✓		✓		✓		
PoV Sub.	Deduplication	○	○	●	●	○	○	○
	ASAP Submission	✓	✓	✓	✓	✓	✓	✓

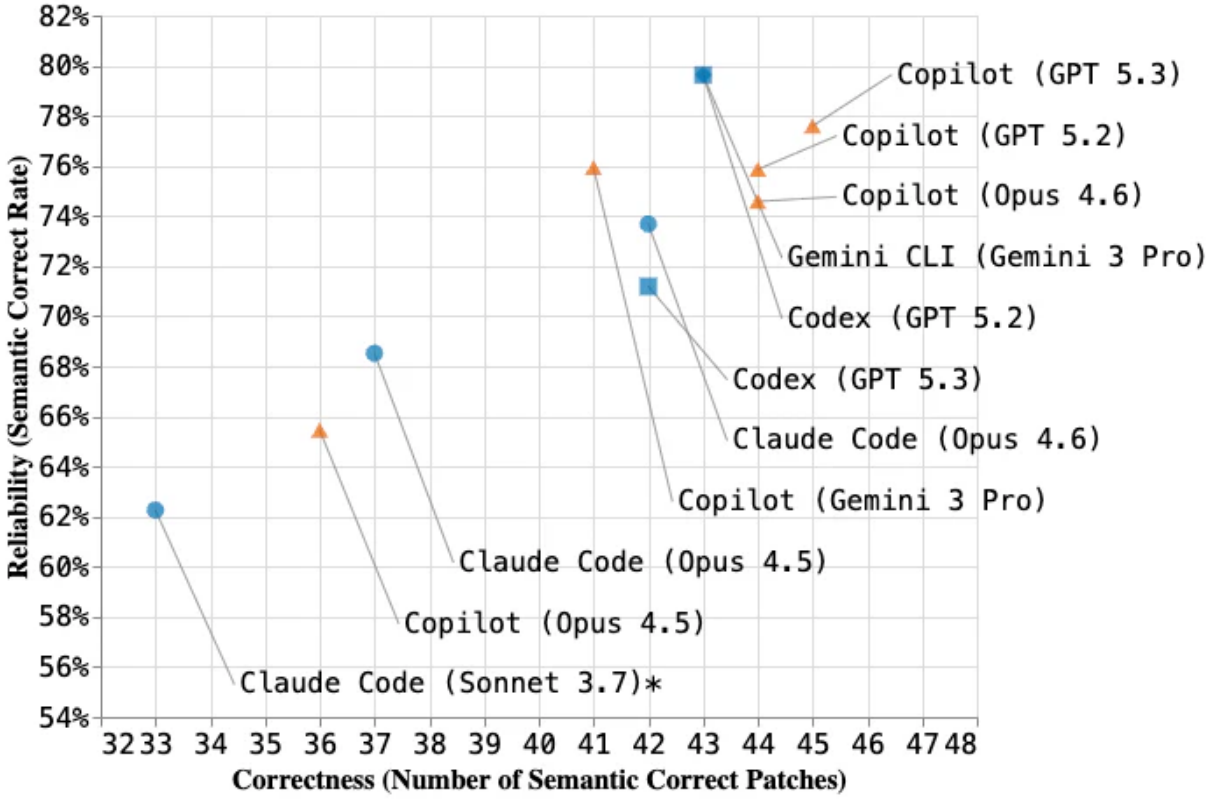
Patch Generation

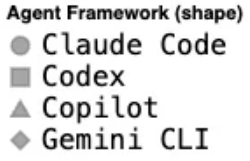
		AT						
Agent Design	Multi-Arch	●						
	Multi-Agent							
	Single-Agent				●		●	●
Patch Gen	Standalone RCA	●	●	●		●		
	├ Multi-PoV RCA		●	●		●		
	└ Non-LLM RCA					●		
	Contextualization†	●	●	●	●	●	●	●
	├ Code Indexer	●	●	●		●	●	
	├ SAST	●		●	●	●		
	├ CWE Guidance				●		●	
	├ Fine-tuned LLM	●						
	├ Agentic Code Search	●	●	●	●	●	●	●
	├ Dynamic Info	●		●	●	●		
	└ PoV Bytes	●		●				●
	LLM Reflection	●	●	●		●		●
	No-PoV Patch			●	●			●
Patch Valid.	Validity Checks	●	●	●	●	●	●	●
	├ Build	●	●	●	●	●	●	●
	├ PoV Test (Gen)	1	N	*	1	N	*	1
	├ PoV Test (Submit)	*	N	*	N	*	*	1
	├ Proj. Tests	●	●	●	●	●		●
	└ Post-patch Fuzz				●	●		
	Rebuild Optim.	●				●		
Patch Sub.	Min. Patch Set Calc.	●	●			●	●	
	No-PoV Delayed Sub.	–	–	●	●	–	–	●

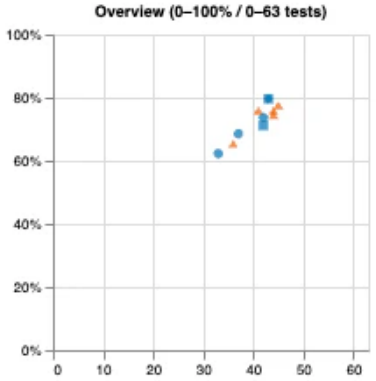
Taxonomy of 50+ bug finding and patch techniques from 7 finalist teams

modern

Q. How does cli agents work for patching?

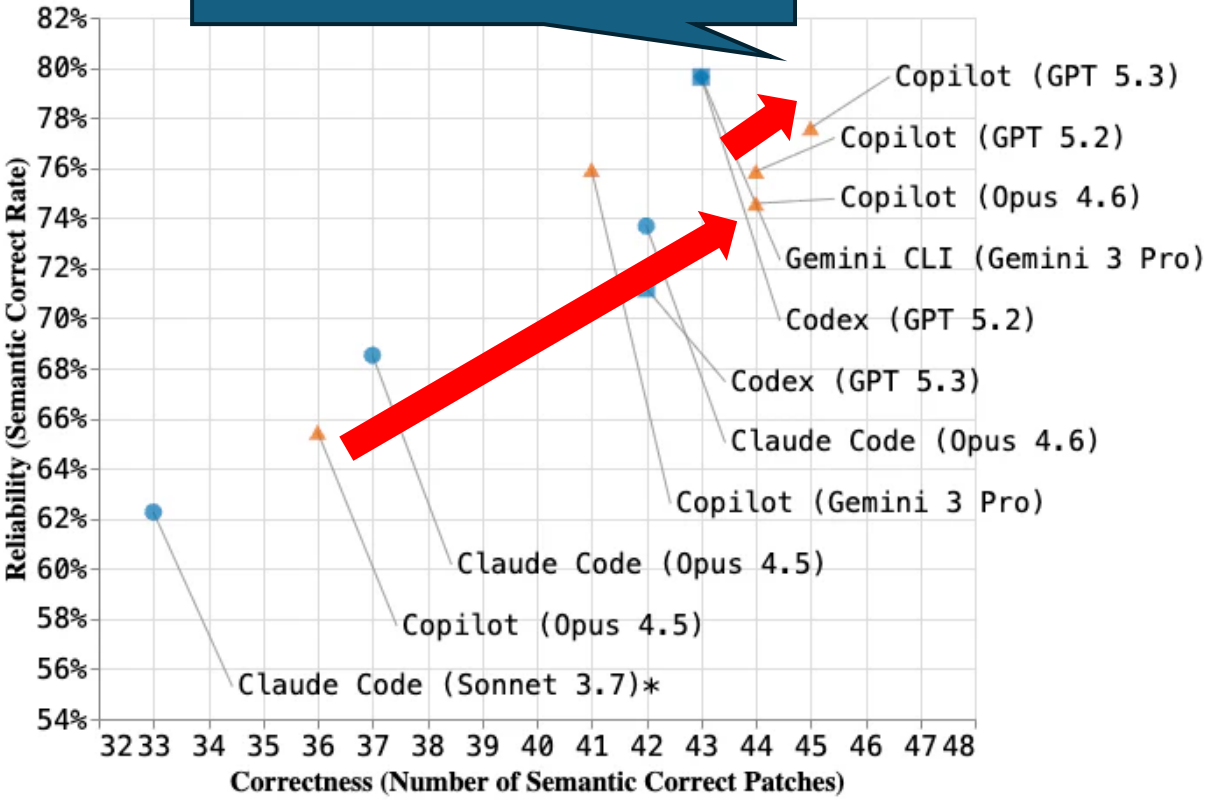


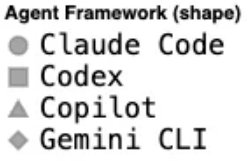


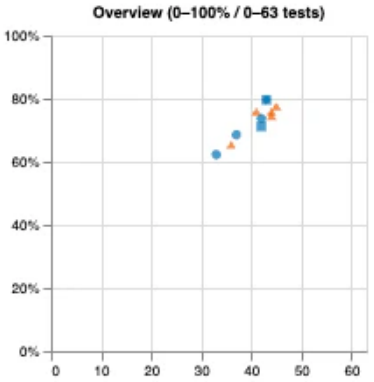


Q. Ho work for patching?

Given the same scaffolding, we can see the model quality enhances the precession of generated patches

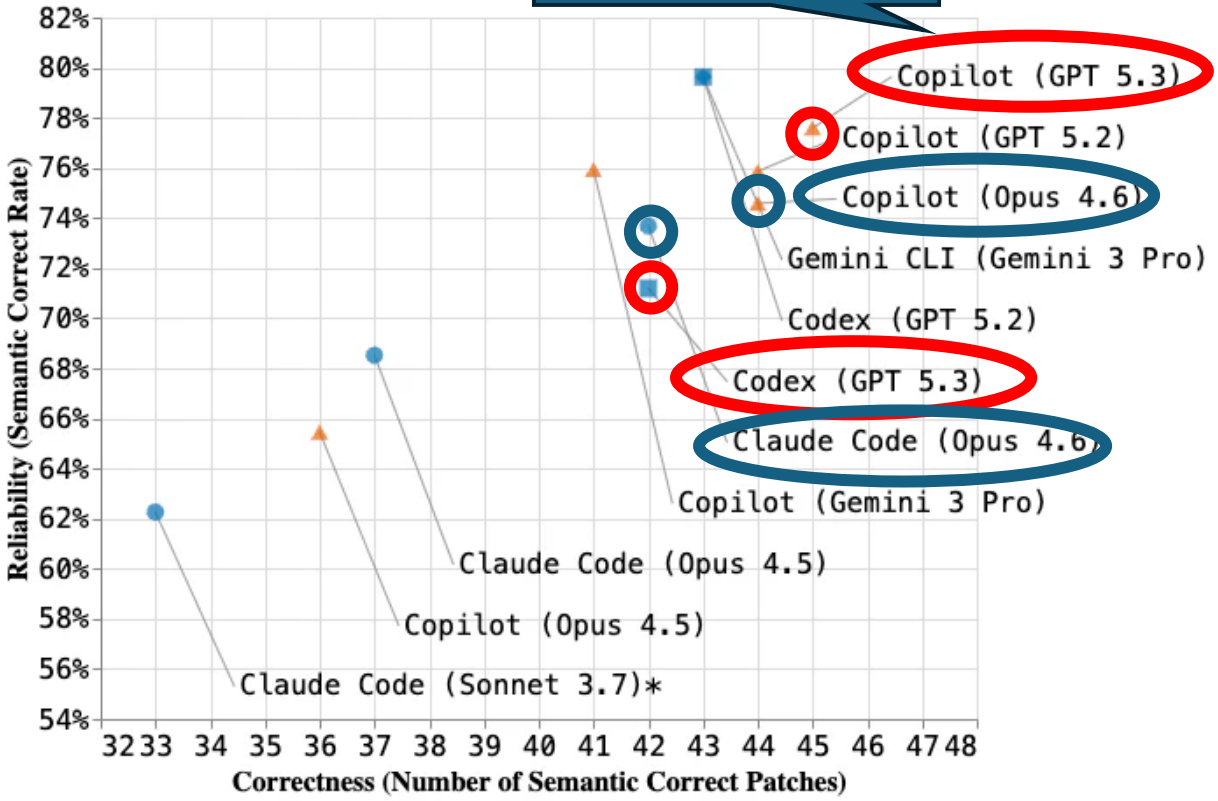


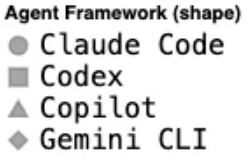


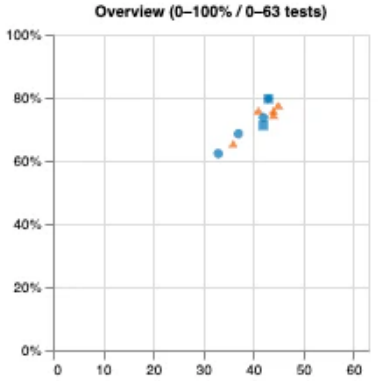


Q. How does cli…rk for patching?

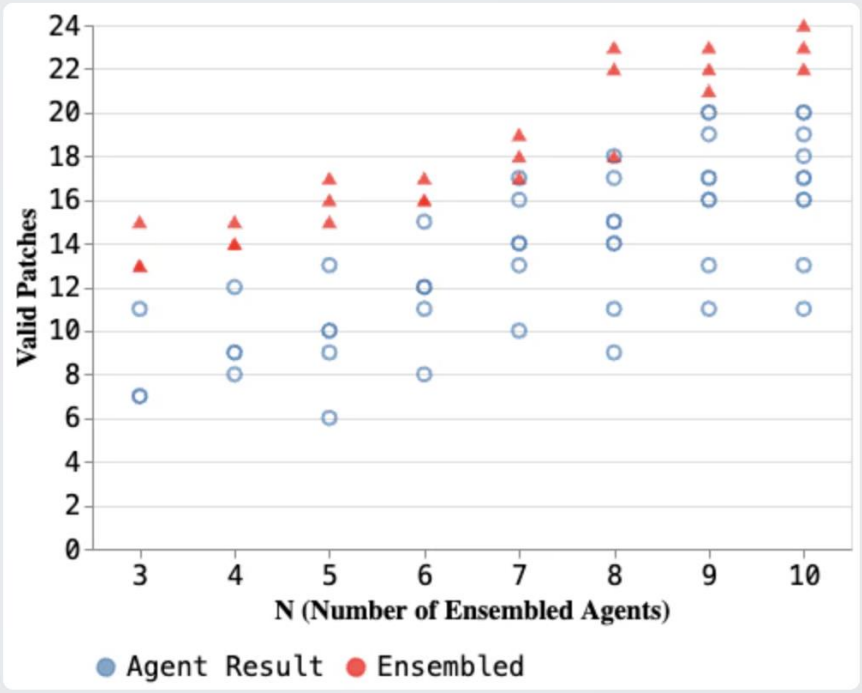
Scaffolding matters more than models

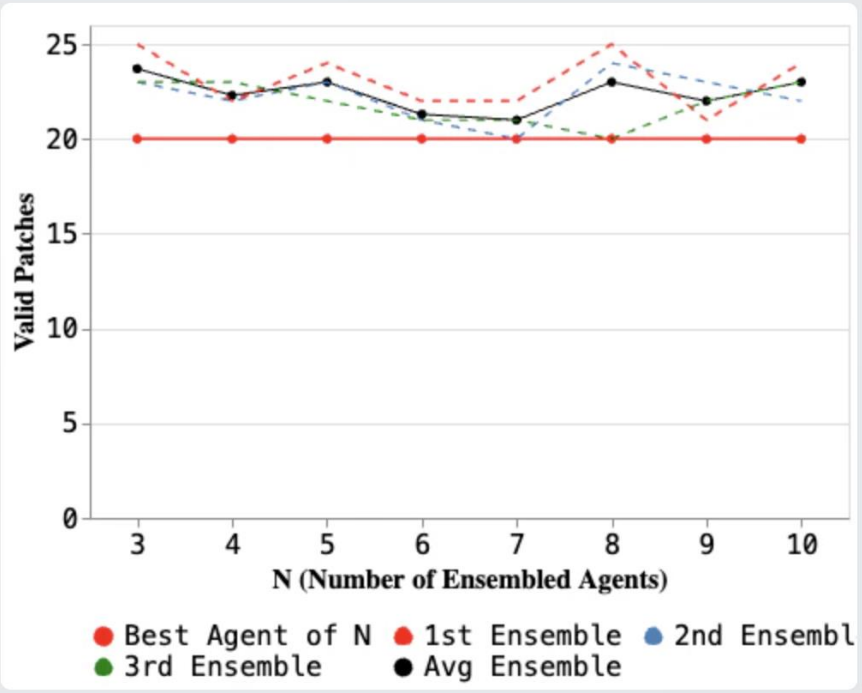






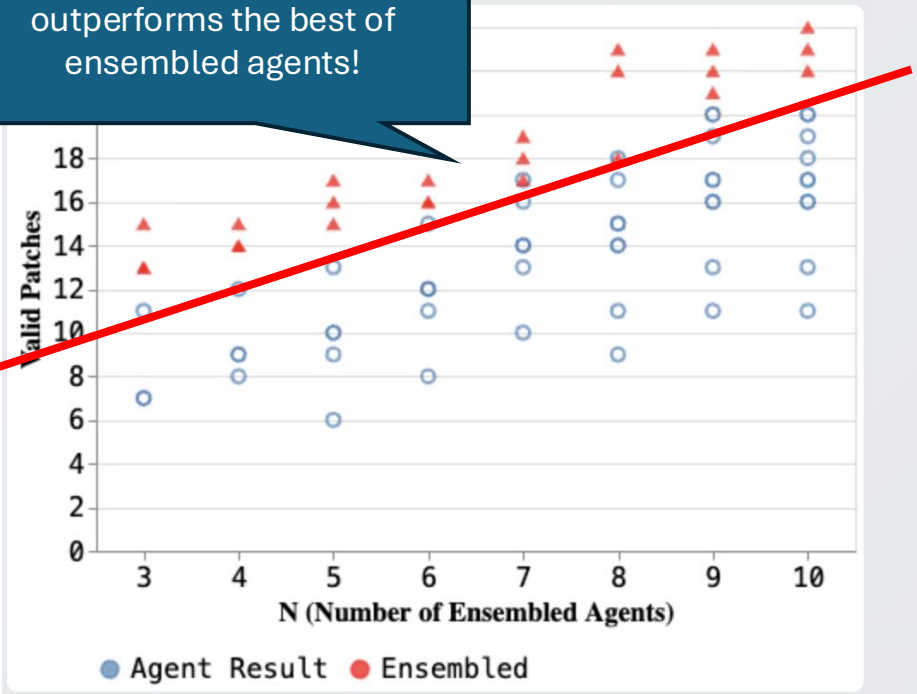
Q. Does “ensembling” still work?

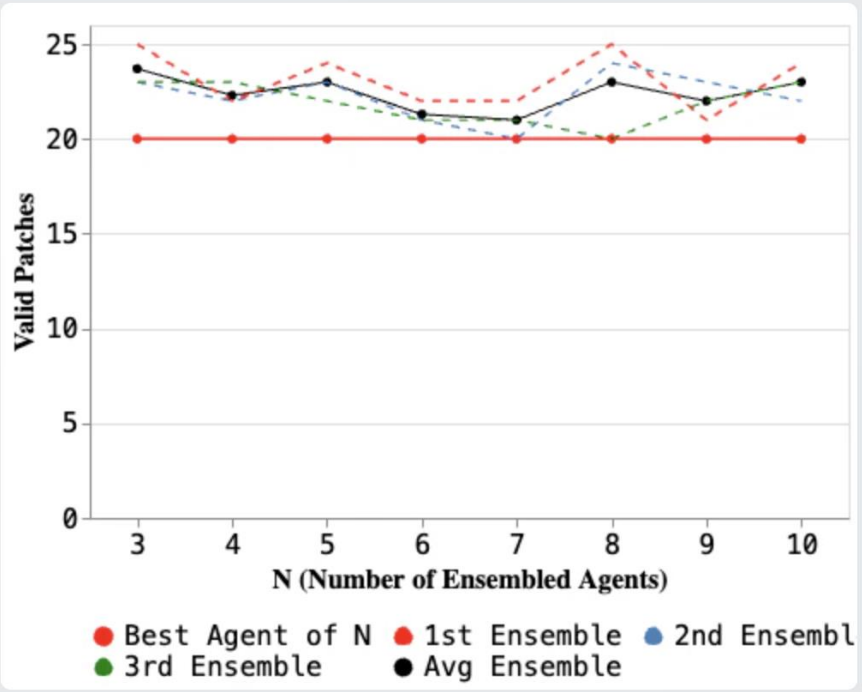




Q. Does “ensembling” still work?

The combined result always outperforms the best of ensembled agents!





Ref. https://github.com/ossf/oss-crs

What's Next?

1) ***oss-crs:*** Standard CRS interface (RFC)
   (selected for an official Linux Foundation project)
2) ***crs-bench:*** Standard benchmark (RFC)
3) **0-day** bug hunting ☺

It's time for real-world AIxCC!

Ref. https://github.com/ossf/oss-crs

OSS-CRS: Liberating AIxCC CRS for Real-World Use (WOOT'26)

CRS	Comp.	Infra	Middleware	Local	Composable
ATLANTIS [23]	9+	TF+K8s	Ka/PG+R*	×	×
BUTTERCUP [8]	14+	TF+Helm	R/M	○	×
ROBODUCK [4]	1	TF+VMs	–/S*	○	×
FUZZINGBRAIN [44]	4	TF+K8s	–/–*	○	×
ARTIPHISHELL [6]	53	TF+Helm	RMQ/PG+N*	△	×
BUGBUSTER [7]	16	TF+Helm	RMQ/PG+R*	×	×
LACROSSE [27]	3+	TF+VMs	RMQ/–	×	×

Standardized interface

Compose multiple AIxCC CRSs and simply all together

OpenSSF

OPEN SOURCE SECURITY FOUNDATION

From AIxCC to OpenSSF

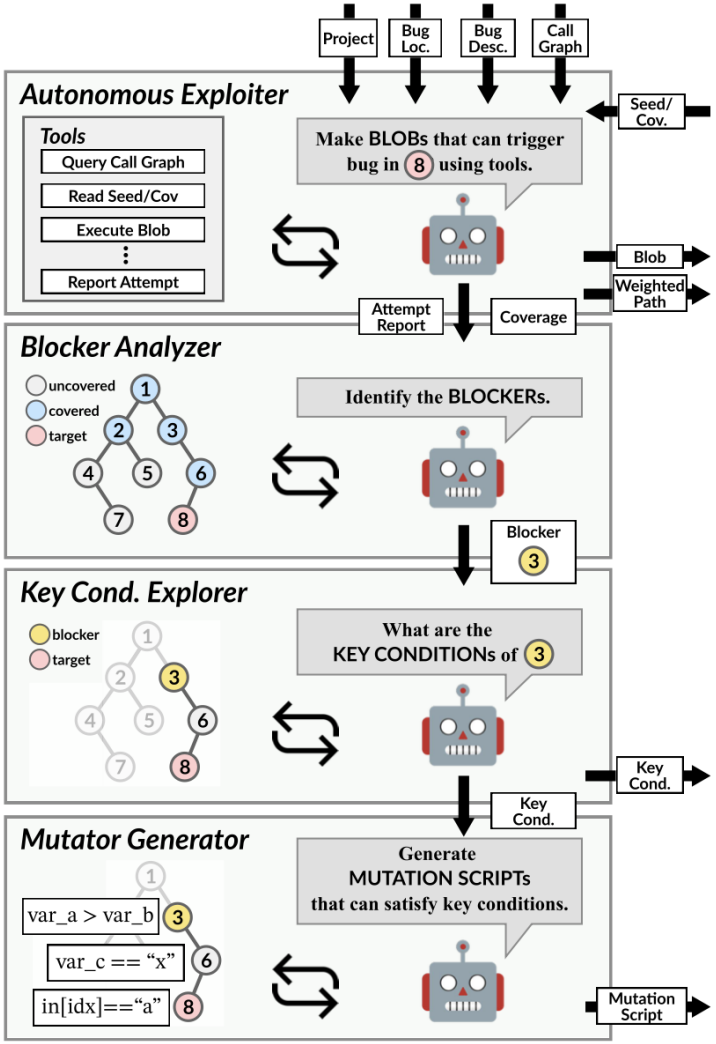
Welcoming OSS-CRS to Advance AI Driven Open Source Security

Ref. https://github.com/ossf/oss-crs

CRS-Bench: AIxCC artifacts to evaluate CRS

Name	Year	Objective	Bug-finding Verification	Bug-fixing Verification		Language	#CWE	#Projects	#Samples	Source
				Security	Functionality					
Magma [28]	2020	Finding (F)	Canary	–	–	C/C++	–	7	117	Real-world
ExtractFix [26]	2021	Fixing	–	Single PoC	✗	C/C++	8	7	30	Real-world
FuzzBench [34]	2021	Finding (F)	Sanitizer-based	–	–	C/C++	–	22	–	Real-world
UniFuzz [32]	2021	Finding (F)	–	–	–	C/C++	–	20	–	Real-world
Vul4J [16]	2022	Fixing	–	Multiple PoC	✗	Java	25	51	79	Real-world
VJBench [45]	2023	Fixing	–	Multiple PoC	✗	Java	23	30	42	Real-world
NYU-CTF-Bench [38]	2024	Finding (L)	Flag-based	✗	✗	C/C++/Python/Go/JS/Java/etc	–	200	200	CTF
CVE-Bench [55]	2025	Finding (L)	Goal-based	–	–	PHP/Python/JS/Go/Rust	26	40	40	Real-world
CVE-Bench [42]	2025	Fixing	–	Single PoC	✗	Java/JS/Python/PHP	103	120	509	Real-world
Arvo [33]	2025	Finding (F)	Sanitizer-based	–	–	C/C++	–	273	5001	Real-world
CyBench [50]	2025	Finding (L)	Flag-based	✗	✗	C/C++/Python/PHP/JS/Java/etc	–	40	40	CTF
AutoPatchBench [18]	2025	Fixing	–	Single PoC, Fuzzing	Differential Testing	C/C++	–	46	136	Real-world
PatchEval [44]	2025	Fixing	–	Single Test	LLM-as-Judge, Unit test	Go/Python/JS	39	175	230	Real-world
CyberGym [43]	2025	Finding (L)	Patch-based	–	–	C/C++	–	188	1507	Real-world
SEC-Bench [31]	2025	Finding (L) Fixing	Sanitizer-based	Single PoC	✗	C/C++	17	29	200	Real-world
BountyBench [49]	2025	Finding (L) Fixing	Patch-based	Single PoC	Unit test	C/Python/JS/etc	27	25	40	Real-world
PVBench [48]	2026	Fixing	–	Single PoC	Unit test, PoC+ test	C/C++	12	20	209	Real-world
CRSBench (Ours)	2026	Finding (F/L) Fixing	Patch-based	Multiple PoC	Unit test w/ optimization	C/C++, Java	TBD	82 /124	319	Real-world, Synthetic

CoTFuzz
Context-Aware Fuzzing (ASE'26)

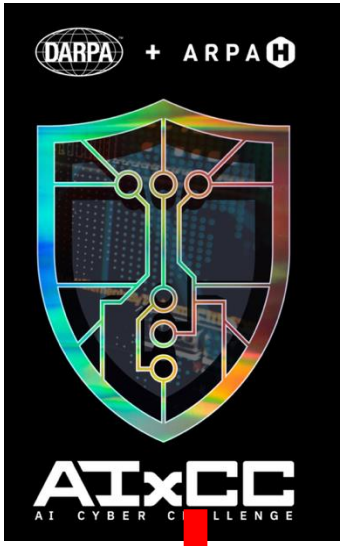


0-day (under disclosure)

- A week of pilot run: $10k per bug ($170k by DARPA)

- memcached, php, questdb, wamr, wasm3, libyang, monetdb, etc, jq, h2database, hsqldb, questdb, OpenLDAP, Eclipse Mosquitto, arp-scan, FR Routing (frr) ...

Team Atlanta leading Real-world AIxCC



+

Five of us joined MSFT!

Microsoft

Google

OpenAI

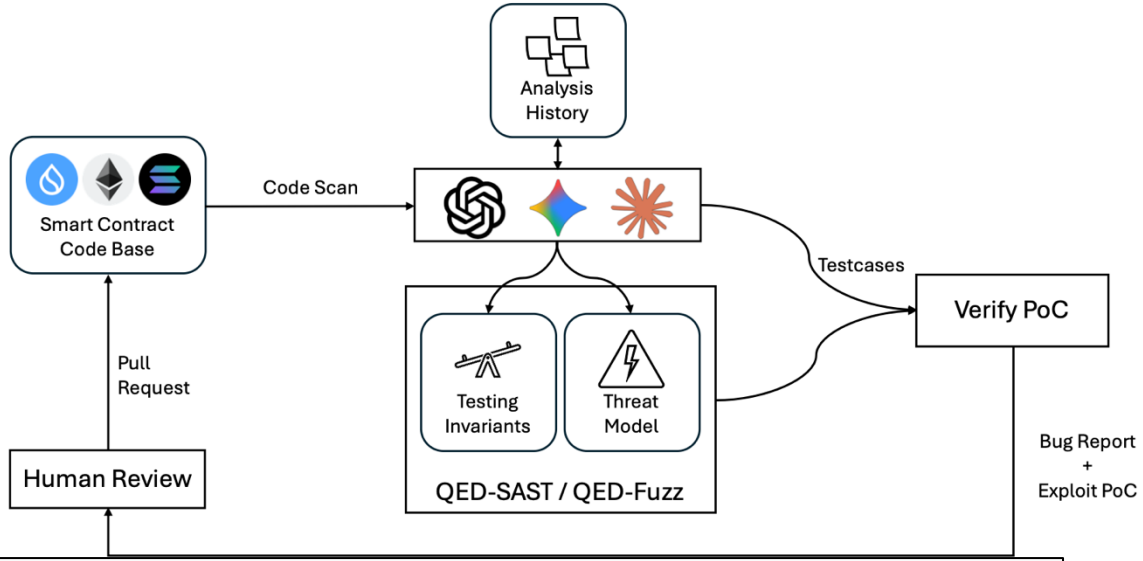
ARMADIN

XBOW

QED

QED Auditing: hyperscale, autonomous auditing and pen-testing for web3

QED



Tachyon: Saving $600M from a time-warp attack

Sunghyeon Jo & Yonghwi Jin · January 28, 2026 · 9 min read

Woosung Son

DEFCON CTF finalist
('19,'22,'23,'24)
$300k+ in bug bounty

Gyejin Lee

DEFCON CTF finalist ('21)
KMO Gold Medalist

Yongwhi Jin

DEFCON CTF winner
('15,'18,'22,'23,'24,'25)
Pwn2Own '20
(Safari/macOS)
Bug bounty on Browser, OS, Blockchain

Sunghyeon Jo

IOI Gold Medalist ('15)
ICPC World Finals Gold Medalist ('17 '20)
#1 Korean on Codeforces

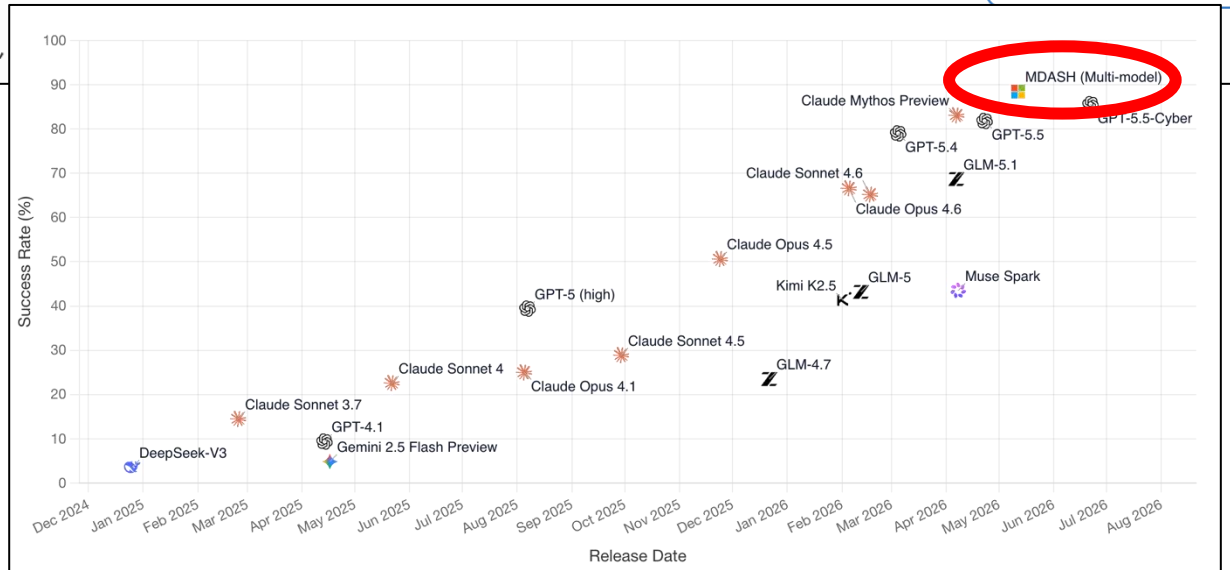
Forbes

Microsoft MDASH Beats A Key Mythos Benchmark. Here's Why That Matters

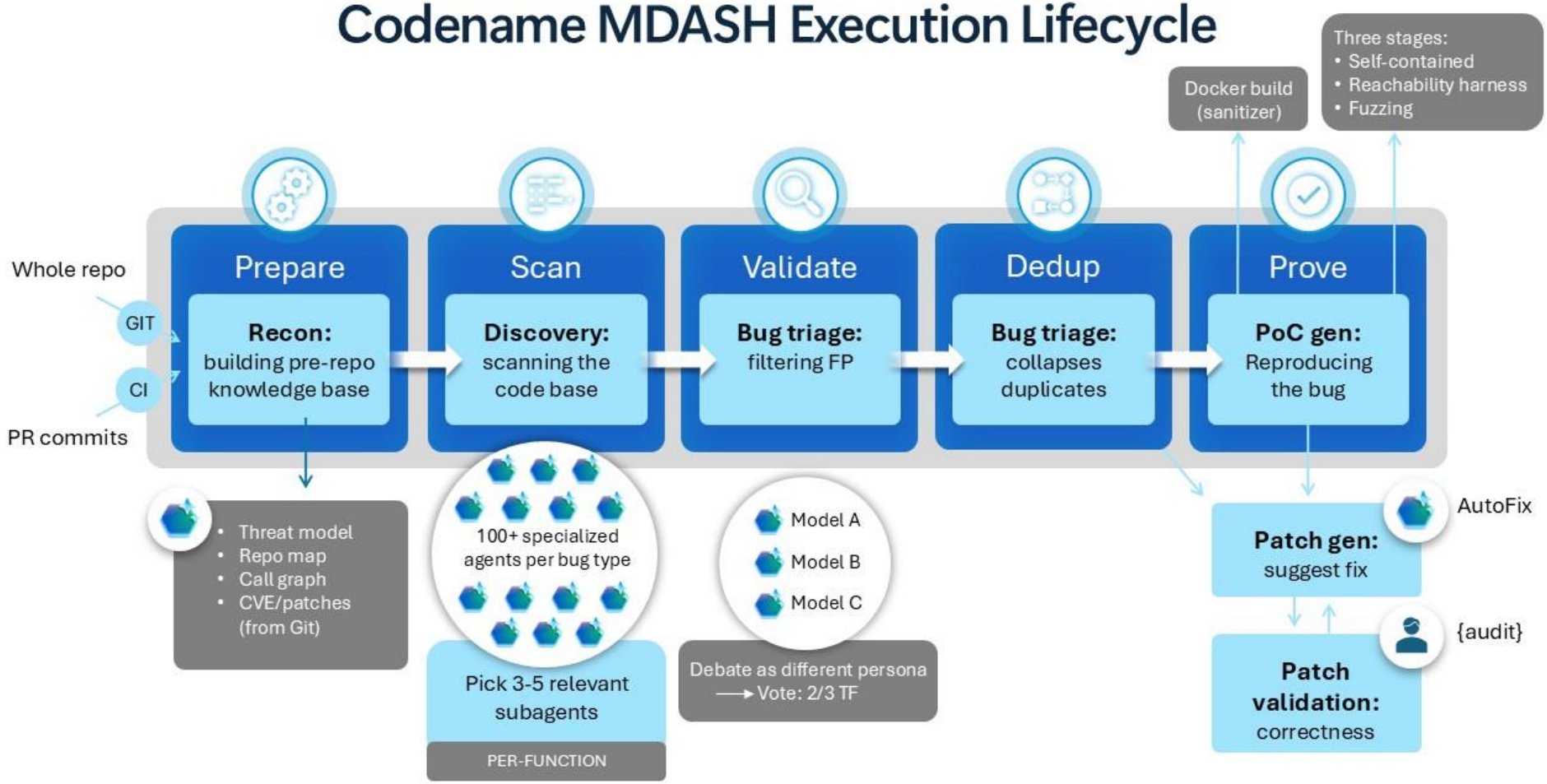
By Tim Keary, Contributor. ⓘ Tim Keary is a reporter covering enterprise AI adoption.

Follow Author

Published May 15, 2026,



Codename MDASH Execution Lifecycle



SECURITYWEEK
CYBERSECURITY NEWS, INSIGHTS & ANALYSIS

VULNERABILITIES

19-Year-Old Linux Kernel Vulnerability Exposes Systems to Root Access

Proof-of-concept (PoC) exploit code has been released for the CIFSwitch flaw, which allows low-privileged users to escalate to root on vulnerable Linux systems.

VULNERABILITIES & THREATS | CLOUD SECURITY | CYBER RISK | ENDPOINT SECURITY | NEWS

Another AI-Assisted Software Scan Yields 9-Year-Old Linux Bug

The proof-of-concept exploit code runs only 10 lines long, but luckily, a patch is already available.

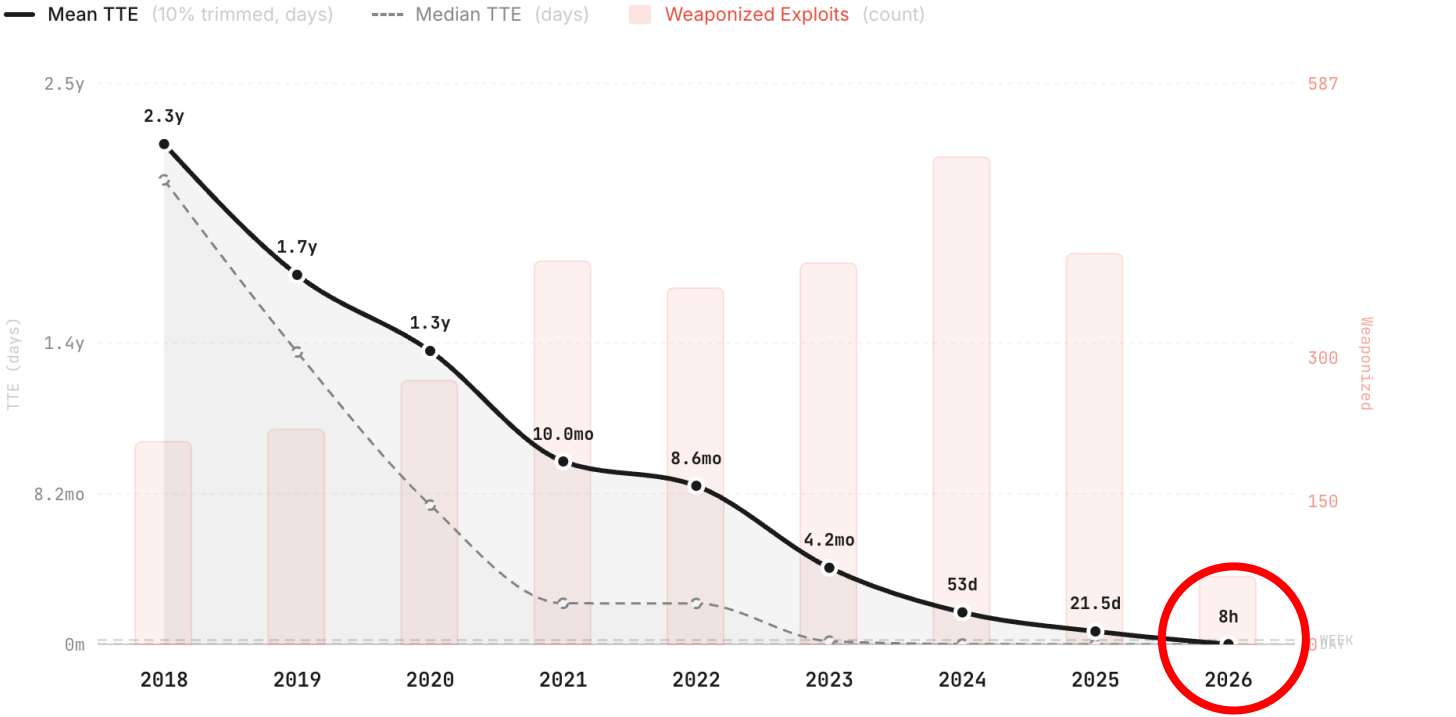
Nate Nelson, Contributing Writer
April 30, 2026

4 Min Read

Editor's Choice

From Vulnerability to Exploitation

TTE measures the gap between CVE public disclosure and first confirmed in-the-wild exploitation. Zero = same-day.



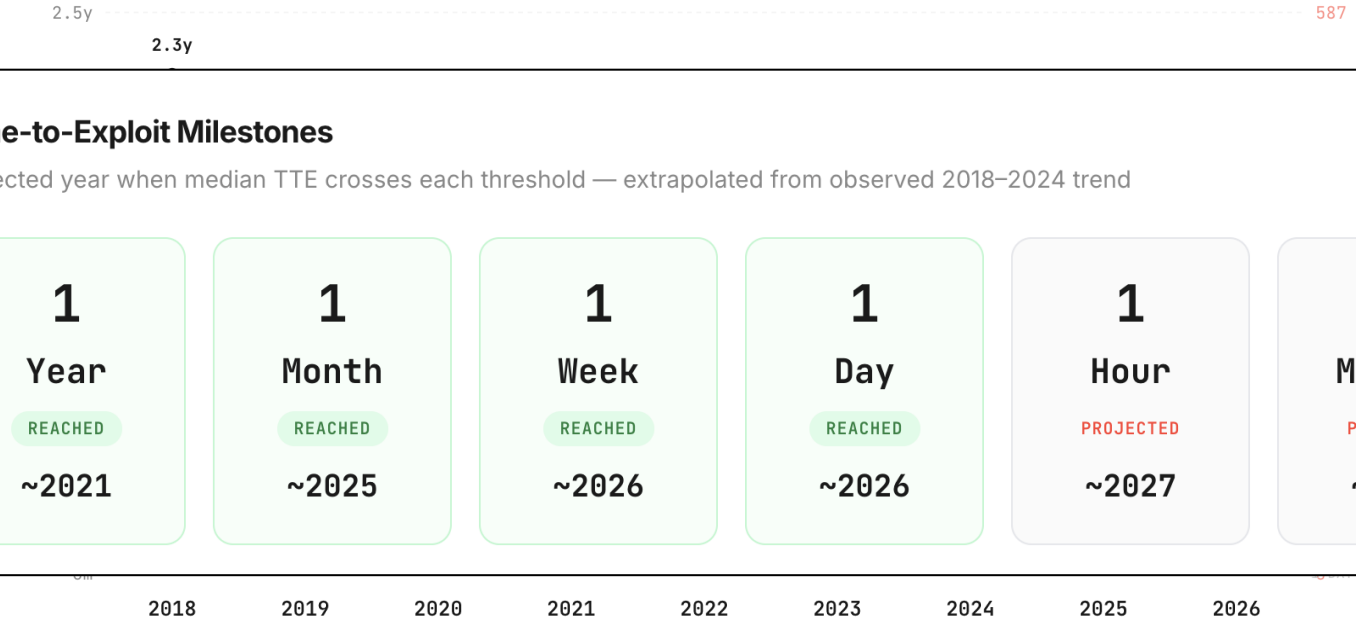
Based on 3,500+ confirmed-exploited CVEs (CISA KEV + VulnCheck KEV, with VulnCheck XDB timestamps for early-year CVEs)

zerodayclock.com

From Vulnerability to Exploitation

TTE measures the gap between CVE public disclosure and first confirmed in-the-wild exploitation. Zero = same-day.

Mean TTE (10% trimmed, days) ---- Median TTE (days) Weaponized Exploits (count)



Time-to-Exploit Milestones

Projected year when median TTE crosses each threshold — extrapolated from observed 2018–2024 trend

1 Year	1 Month	1 Week	1 Day	1 Hour	1 Minute
REACHED	REACHED	REACHED	REACHED	PROJECTED	PROJECTED
~2021	~2025	~2026	~2026	~2027	~2027

Based on 3,500+ confirmed-exploited CVEs (CISA KEV + VulnCheck KEV, with VulnCheck XDB timestamps for early-year CVEs)

zerodayclock.com

Announcements

Statement on the US government directive to suspend access to Fable 5 and Mythos 5

Jun 12, 2026

The US government, citing national security authorities, has issued an export control directive to suspend all access to Fable 5 and Mythos 5 by any foreign national, whether inside or outside the United States, including foreign national Anthropic employees. The net effect of this order is that we must abruptly disable Fable 5 and Mythos 5 for **all** our customers to ensure compliance. **Access to all other Anthropic models will not be affected.**

We received the directive from the government today at 5:21pm (ET). The letter did not provide specific details of its national security concern. Our understanding is that the government believes it has become aware of a method of bypassing, or "jailbreaking" Fable 5. We reviewed a demonstration of this specific technique being used to identify a small number of previously known, minor vulnerabilities. These vulnerabilities all appear relatively simple, and we have found that other publicly-available models are able to discover them as well without requiring a bypass.

RESCANA

Solution About Blog Contact Book a Demo →

Active Exploitation Alert Jun 16, 2026 7 min read ← All posts

Critical LiteLLM Vulnerability Chain Enables Remote Code Execution and Full AI Gateway Server Takeover (CVE-2026-42271, CVE-2026-47101, CVE-2026-47102, CVE-2026-40217)

Over 140 popular Mastra npm Packages Hit by Supply Chain Attack

Published on: Jun 17, 2026

Attack targeting OpenAI Codex users exposes AI software supply chain risks

News

Jun 2, 2026 • 4 mins

Security for AI

- Supply chain issue
- Prompt injection (XPIA)
- Model abusing
- Model backdoor
- Model extraction
- …

IBM and Red Hat Commit $5 Billion to Redefine the Future of Open Source in the AI Era

Project Lightwell establishes a trusted enterprise clearinghouse for open source software with a new AI-driven model for securing the software supply chain

May 28, 2026

IBM and Red Hat Commit $5 Billion to Redefine the Future of Open Source in the AI Era

Project Lightwell establishes a trusted enterprise clearinghouse for open source software with a new AI-driven model for securing the software supply chain

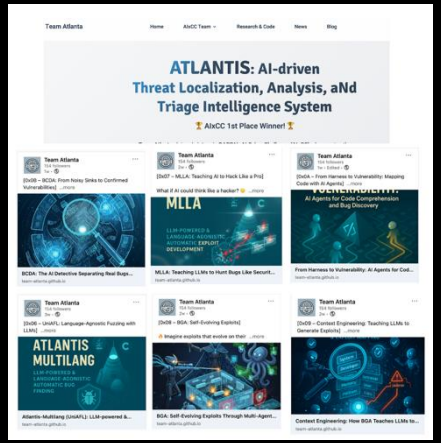
From AIxCC to OpenSSF: Welcoming OSS-CRS to Advance AI Driven Open Source Security

April 2, 2026 | AI, Blog, Global Cyber Policy

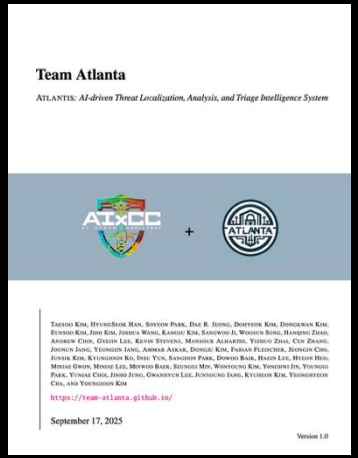
June 22, 2026 Security

Patch the Planet: a Daybreak initiative to support open source maintainers

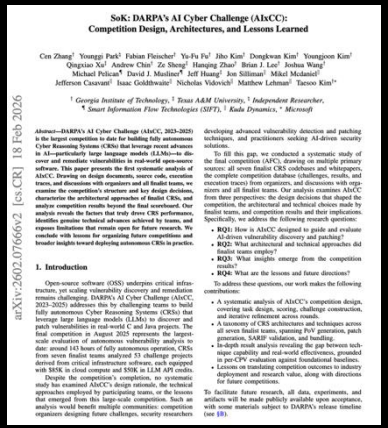
TL;DR. Open Source + Technical Report!



Blog Postings



150+pg TR

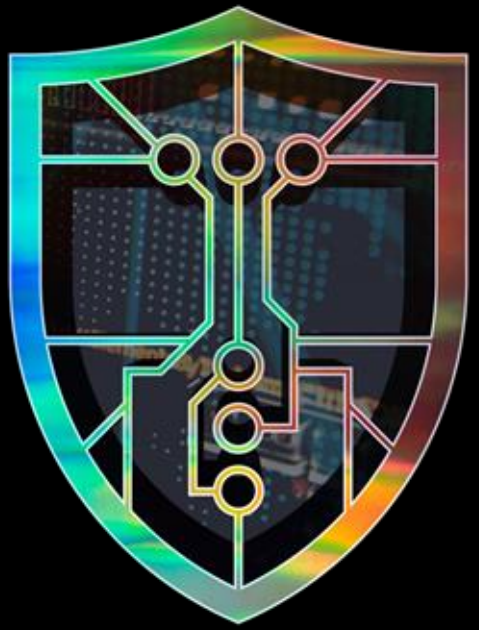


SoK: AIxCC CRS

OpenSSF

https://team-atlanta.github.io/

DARPA + ARPA-H



AIxCC
AI CYBER CHALLENGE

The world changes today.

Automated patch development is:

Fast
Scalable
Cost-effective
Available / Open-source

AI + CRS = The ~~Future~~ Present

https://team-atlanta.github.io/


Thank You!
Questions?