Northwestern University

PatchAgent

A Practical Program Repair Agent Mimicking Human Expertise

○ About Me

My name is Zheng Yu. I am a final year PhD student in the Computer Science department at Northwestern University. My goal is to improve software security with minimal human intervention.

I am graduating in 2026 and actively seeking full-time opportunities!

□ What is PatchAgent?

PatchAgent is a fully automated program repair tool. It enables LLMs to effectively use coding tools and handle security information. We build a benchmark containing 178 cases, PatchAgent repaired 90+% of them. We also verified its repair ability in the real world.

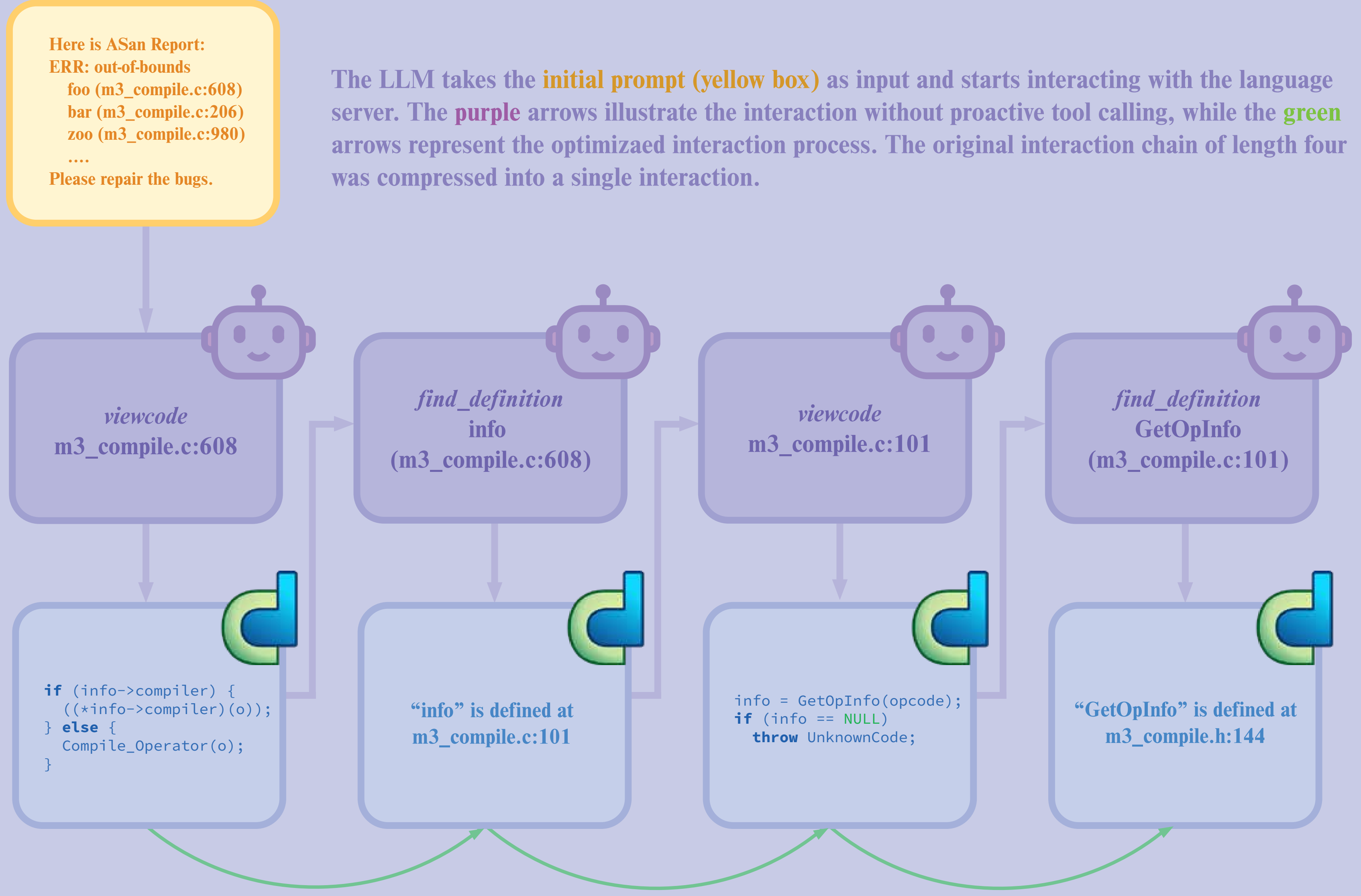
Model	Temporal	Spatial	Null Deref	Numeric	Total
GPT-4o	56.52%	76.80%	100.00%	100.00%	78.09%
GPT-4 Turbo	57.83%	69.60%	91.30%	100.00%	70.79%
Claude Opus 3	60.87%	86.40%	95.65%	100.00%	84.83%
Claude Sonnet 3	34.78%	61.60%	73.91%	85.71%	60.67%
Claude Haiku 3	39.13%	74.40%	82.61%	100.00%	71.91%
Union	86.96%	91.20%	100.00%	100.00%	92.13%

☆ Repair Real World Bugs

LIBSSH2 AIxCC HDF5

LibreDWG

PcapPlusPlus



Here is ASan Report:
ERR: out-of-bounds
foo (m3_compile.c:608)
bar (m3_compile.c:206)
zoo (m3_compile.c:980)
....
Please repair the bugs.

The LLM takes the initial prompt (yellow box) as input and starts interacting with the language server. The purple arrows illustrate the interaction without proactive tool calling, while the green arrows represent the optimizaed interaction process. The original interaction chain of length four was compressed into a single interaction.

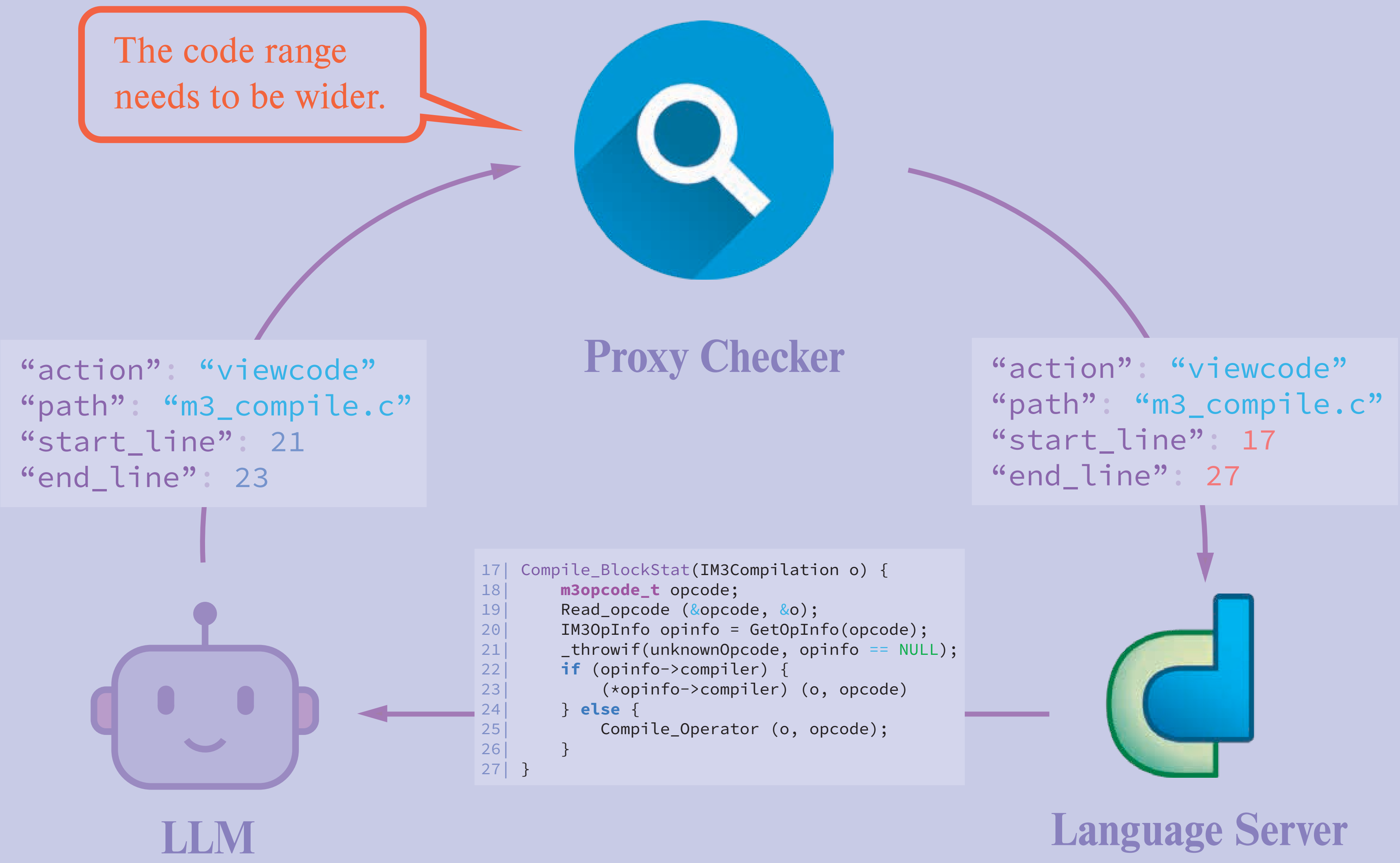
```
if (info->compiler) {
  ((*info->compiler)(o));
} else {
  Compile_Operator(o);
}
```

"info" is defined at m3_compile.c:101

```
info = GetOpInfo(opcode);
if (info == NULL)
  throw UnknownCode;
```

"GetOpInfo" is defined at m3_compile.h:144

Proactive Tool Calling



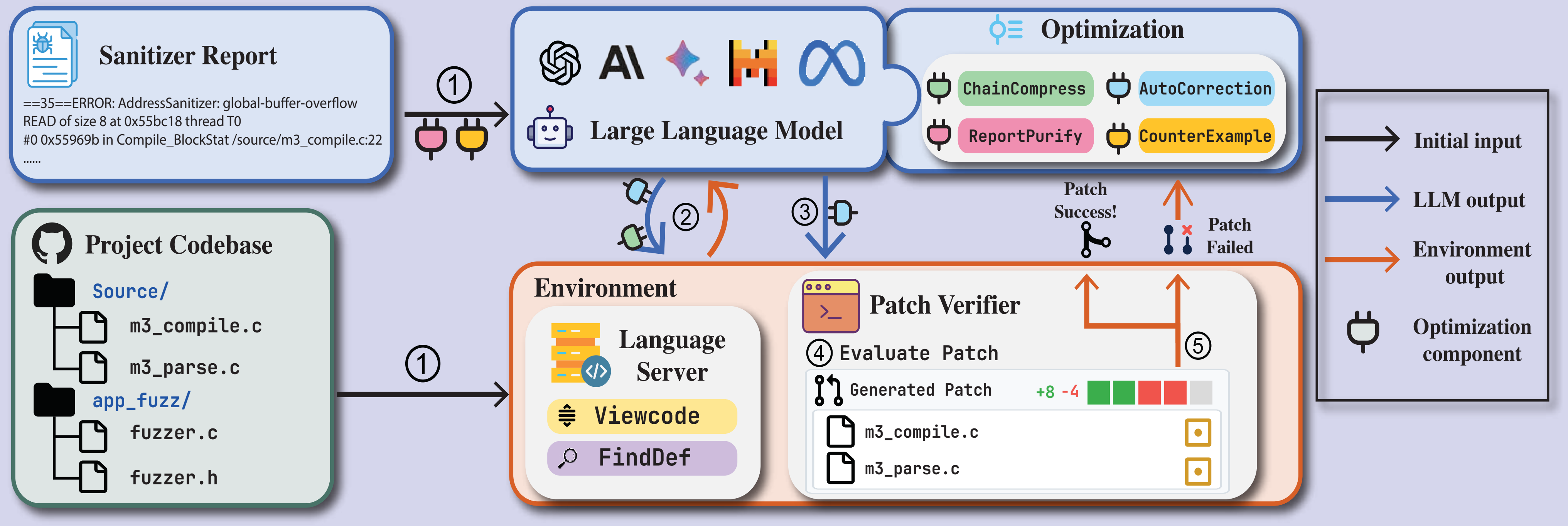
The code range needs to be wider.

```
"action": "viewcode"
"path": "m3_compile.c"
"start_line": 21
"end_line": 23
```

```
"action": "viewcode"
"path": "m3_compile.c"
"start_line": 17
"end_line": 27
```

```
Compile_BlockStat(IM3Compilation o) {
    m3opcode_t opcode;
    Read_opcode (&opcode, &o);
    IM3OpInfo opinfo = GetOpInfo(opcode);
    _throwif(unknownOpcode, opinfo == NULL);
    if (opinfo->compiler) {
        (*opinfo->compiler) (o, opcode)
    } else {
        Compile_Operator (o, opcode);
    }
}
```

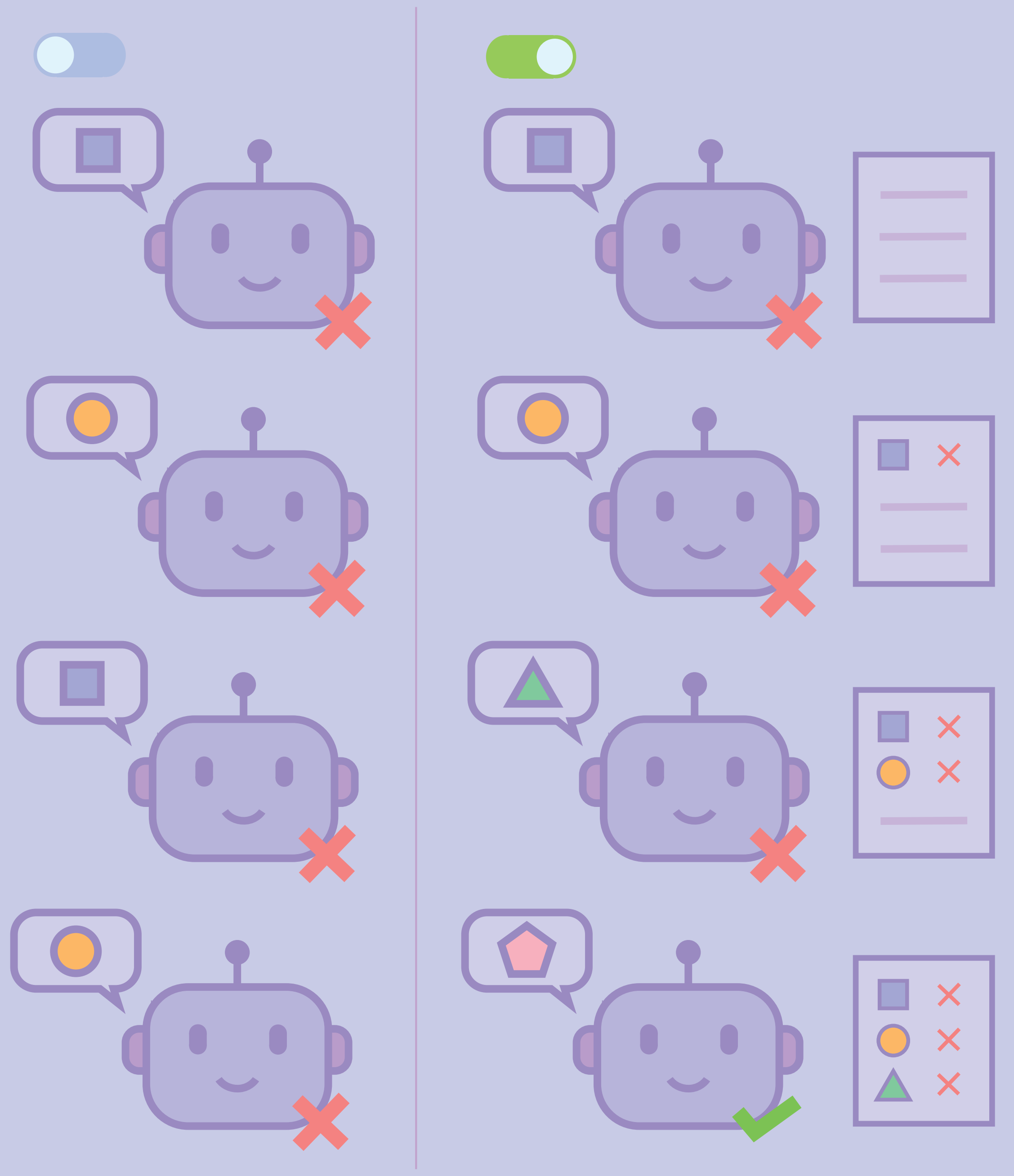
Automated Correction



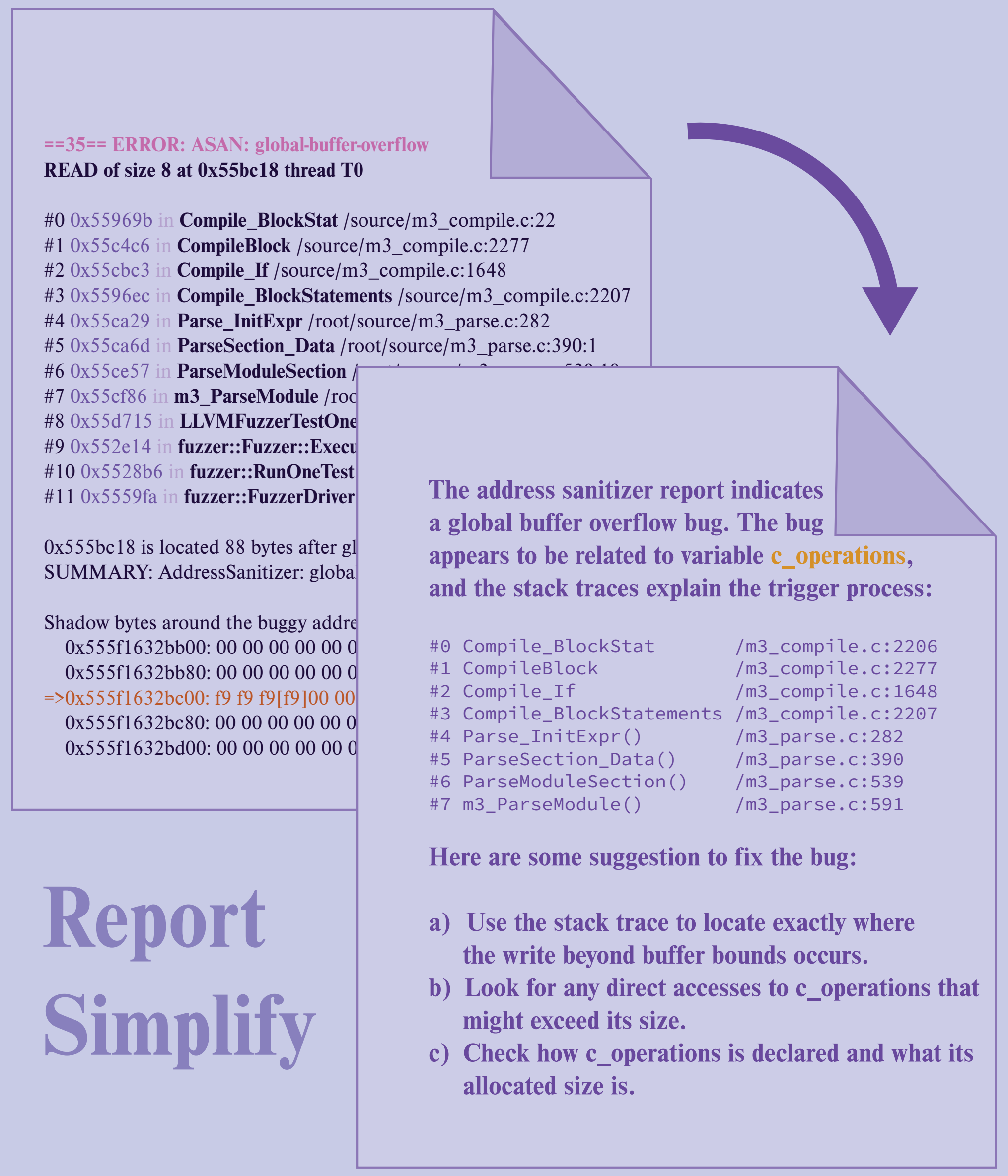
Sanitizer Report

```
==35==ERROR: AddressSanitizer: global-buffer-overflow
READ of size 8 at 0x55bc18 thread T0
#0 0x55969b in Compile_BlockStat /source/m3_compile.c:22
......
```

Self Reflection



Easy to Use



==35== ERROR: ASAN: global-buffer-overflow
READ of size 8 at 0x55bc18 thread T0

#0 0x55969b in Compile_BlockStat /source/m3_compile.c:22
#1 0x55c4c6 in CompileBlock /source/m3_compile.c:2277
#2 0x55cbc3 in Compile_If /source/m3_compile.c:1648
#3 0x5596ec in Compile_BlockStatements /source/m3_compile.c:2207
#4 0x55ca29 in Parse_InitExpr /root/source/m3_parse.c:282
#5 0x55ca6d in ParseSection_Data /root/source/m3_parse.c:390:1
#6 0x55ce57 in ParseModuleSection /
#7 0x55cf86 in m3_ParseModule /roo
#8 0x55d715 in LLVMFuzzerTestOne
#9 0x552e14 in fuzzer::Fuzzer::Execu
#10 0x5528b6 in fuzzer::RunOneTest
#11 0x5559fa in fuzzer::FuzzerDriver

0x555bc18 is located 88 bytes after gl
SUMMARY: AddressSanitizer: globa

Shadow bytes around the buggy addre
0x555f1632bb00: 00 00 00 00 00 0
0x555f1632bb80: 00 00 00 00 00 0
=>0x555f1632bc00: f9 f9 f9[f9]00 00
0x555f1632bc80: 00 00 00 00 00 0
0x555f1632bd00: 00 00 00 00 00 0

The address sanitizer report indicates a global buffer overflow bug. The bug appears to be related to variable c_operations, and the stack traces explain the trigger process:

```
#0 Compile_BlockStat        /m3_compile.c:2206
#1 CompileBlock             /m3_compile.c:2277
#2 Compile_If               /m3_compile.c:1648
#3 Compile_BlockStatements  /m3_compile.c:2207
#4 Parse_InitExpr()         /m3_parse.c:282
#5 ParseSection_Data()      /m3_parse.c:390
#6 ParseModuleSection()     /m3_parse.c:539
#7 m3_ParseModule()         /m3_parse.c:591
```

Here are some suggestion to fix the bug:

a) Use the stack trace to locate exactly where the write beyond buffer bounds occurs.
b) Look for any direct accesses to c_operations that might exceed its size.
c) Check how c_operations is declared and what its allocated size is.

Report Simplify

```
# Example Code
from patchagent.agent.generator import agent_generator
from patchagent.builder import OSSFuzzBuilder, OSSFuzzPoC
from patchagent.parser.sanitizer import Sanitizer
from patchagent.task import PatchTask

patchtask = PatchTask(
    [OSSFuzzPoC("poc.bin", "libpng_read_fuzzer")],
    OSSFuzzBuilder(
        "libpng",
        "/path/to/libpng",
        "/path/to/oss-fuzz",
        [Sanitizer.AddressSanitizer],
    ),
)
patchtask.initialize()
patch = patchtask.repair(agent_generator())
print(f"Generated patch: {patch}")
```

```
# Ready to Use Out of the Box
docker pull ghcr.io/cla7aye15i4nd/patchagent:latest
```

Please Give PatchAgent a Star ✨

Thank you for visiting ❤