

Automated program repair using LLMs

Michael-Raphael Kostagiannis

Undegraduate student at DIT NKUA

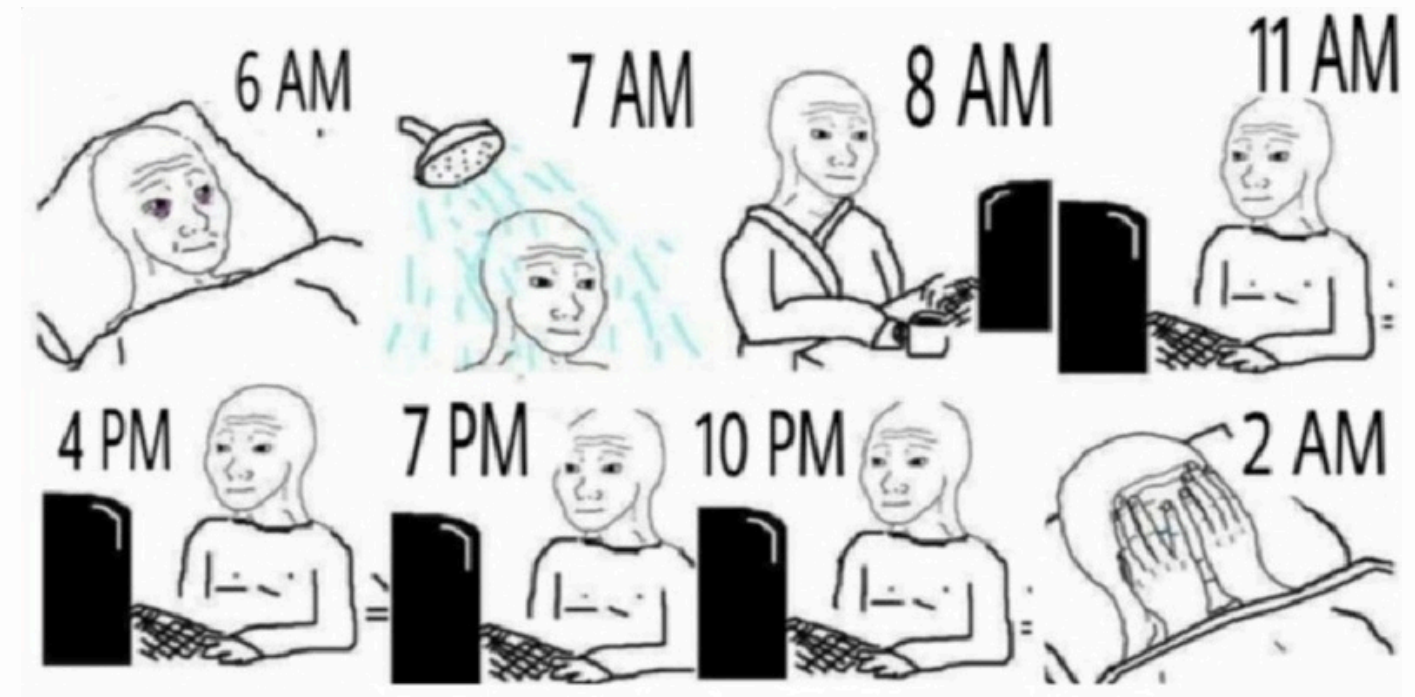
(hopefully not for long)

What is software repair?

- **Software repairing** is the process of detecting software failures and applying fixes at the source code level.
- Software failures include bugs, performance issues, security vulnerabilities, and compatibility problems.
- Consists of three phases:
 - **Fault localization**: Spot the code that needs to be fixed.
 - **Patch generation**: Generate a fix.
 - **Patch validation**: Make sure fix isn't **rm -rf /**
- Is tiring and boring.

Why automate it?

- **Bugs are expensive** – Software failures cost companies billions (e.g., NASA's Mars Climate Orbiter failure due to a unit conversion bug).
- **Security vulnerabilities** – Delayed bug fixes can be exploited.
- **Time-consuming** – Developers spend **50%+** of their time debugging rather than building new features.
- **Human error in fixes** – Manually written patches can introduce new bugs.
- Is **really** tiring and boring.



Keywords & Terminology

- **Software Repairing:** Detect failure and apply fix at the source code level.
- **Software Healing:** Detect failure and apply fix at runtime.
- **Automatic software repairing (APR):** No human is involved in repairing.

- **Patch:** A permanent solution to a software bug.
- **Workaround:** A temporary solution to a software bug.

- **Fault localization:** Process of identifying faulty statements inside a program.
- **Patch generation:** Generate program variants for plausible fixes.
- **Patch validation:** Utilize test suites to identify correct patches.

- **Large Language Model (LLM):** A pre-trained deep learning model designed to generate and understand natural language.
- **LLM Agent:** A LLM equipped with tools like Google, read/write files etc.

GenProg👴(2009)

GenProg is one of the first APR tools that uses **genetic programming** to search for repairs. It works by mutating, recombining, and selecting patches that pass test cases.

```
1 char* ProcessRequest() {
2     ...
3     while(l=sgets(l,sock)) {
4         if(l=="Request:")
5             strcpy(req_type,l+12)
6         if(l=="Content-Length:")
7             len=atoi(l+16);
8     }
9     if(req_type=="GET")
10        buff=DoGETReq(sock,len);
11    if(req_type=="POST") {
12        sz=sizeof(char);
13        buff=calloc(len,sz);
14        rc=recv(sock,buff,len)
15        buff[len]='\0';
16    }
17    return buff;
18 }
```

(a) Webserver code snippet.

```
3 ...
4     if(l=="Request:")
5         strcpy(req_type,l+12)
6     if(l=="Content-Length:")
7         len=atoi(l+16);
8 }
9 if(req_type=="GET")
10    buff=DoGETReq(sock,len);
11 if(req_type=="POST") {
12 +   if (len <= 0)
13 +       return null;
14     sz=sizeof(char);
15     buff=calloc(len,sz);
16     rc=recv(sock,buff,len)
17     buff[len]='\0';
18 }
19 return buff;
20 }
```

(b) Patched webserver.

How GenProg works

1) Fault localization: Uses test cases to find faulty behavior.

2) Mutation: Randomly modifies the buggy program (e.g., inserting, deleting, or swapping code).

3) Crossover: Combines multiple mutated versions to create candidate patches.

4) Fitness Evaluation: Runs tests to check which patches fix the bug without breaking functionality.

5) Selection: Best patches are evolved over generations until a working fix is found.

Input: Full fitness predicate $\text{FullFitness} : \text{Patch} \rightarrow \mathbb{B}$

Input: Sampled fitness $\text{SampleFit} : \text{Patch} \rightarrow \mathbb{R}$

Input: Mutation operator $\text{mutate} : \text{Patch} \rightarrow \text{Patch}$

Input: Crossover operator $\text{crossover} : \text{Patch}^2 \rightarrow \text{Patch}^2$

Input: Parameter PopSize

Output: Patch that passes FullFitness

1: **let** $\text{Pop} \leftarrow \text{map mutate over } \text{PopSize} \text{ copies of } \langle \rangle$

2: **repeat**

3: **let** $\text{parents} \leftarrow \text{tournSelect}(\text{Pop}, \text{Popsizes}, \text{SampleFit})$

4: **let** $\text{offspr} \leftarrow \text{map crossover over parents, pairwise}$

5: $\text{Pop} \leftarrow \text{map mutate over } \text{parents} \cup \text{offspr}$

6: **until** $\exists \text{ candidate} \in \text{Pop}. \text{FullFitness}(\text{candidate})$

7: **return** candidate

Is GenProg good?

Repairs all of these bugs in 356.5 seconds.

Program	LOC	Description	Fault
gcd	22	example	infinite loop
zune	28	example [35]	infinite loop†
uniq utx	1146	duplicate text processing	segmentation fault
look utx	1169	dictionary lookup	segmentation fault
look svr	1363	dictionary lookup	infinite loop
units svr	1504	metric conversion	segmentation fault
deroff utx	2236	document processing	segmentation fault
nullhttpd	5575	webserver	remote heap buffer overflow (code)†
openldap	292598	directory protocol	non-overflow denial of service†
ccrypt	7515	encryption utility	segmentation fault†
indent	9906	source code processing	infinite loop
lighttpd	51895	webserver	remote heap buffer overflow (vars)†
flex	18775	lexical analyzer generator	segmentation fault
atris	21553	graphical tetris game	local stack buffer exploit†
pphp	764489	scripting language	integer overflow†
wu-ftp	67029	FTP server	format string vulnerability†
total	1246803		

Repaired approximately half of these bugs in 12 hours.

Program	Description	LOC	Tests	Bugs
fbc	legacy compiler for Basic	97,000	773	3
gmp	precision math library	145,000	146	2
gzip	data compression utility	491,000	12	5
libtiff	image manipulation library	77,000	78	24
lighttpd	lightweight web server	62,000	295	9
php	web programming language interpreter	1,046,000	8,471	44
python	general programming language interpreter	407,000	355	11
wireshark	network packet analyzer	2,814,000	63	7
total		5,139,000	10,193	105

All with an unprecedented cost of **0 cents!** 😎

Why bother with LLMs?

- While GenProg might seem perfect, it is heavily dependent on test cases. If tests are incomplete, GenProg will generate incorrect patches.
- Older APR tools also **lack deep reasoning**. Complicated bugs like SQL injections or synchronization primitives prove impossible to fix - random mutations can only take you so far.
- **LLMs** are equipped with code reasoning, context awareness and natural language understanding. We can use these to our advantage in order to enhance APR techniques!



AGENTLESS (July 2024)

- An agentless LLM APR approach to automatically resolve software development issues.
- Uses prompt-based reasoning and can handle multiple independent fixes simultaneously.
- High performance (**32.00%**, 96 correct fixes) and low cost (**\$0.70**) on the SWE-bench benchmark.
- Has already been adopted by OpenAI for both **GPT-4o** and the new OpenAI **o1 models**.
- Open-source.

How AGENTLESS🐱 works

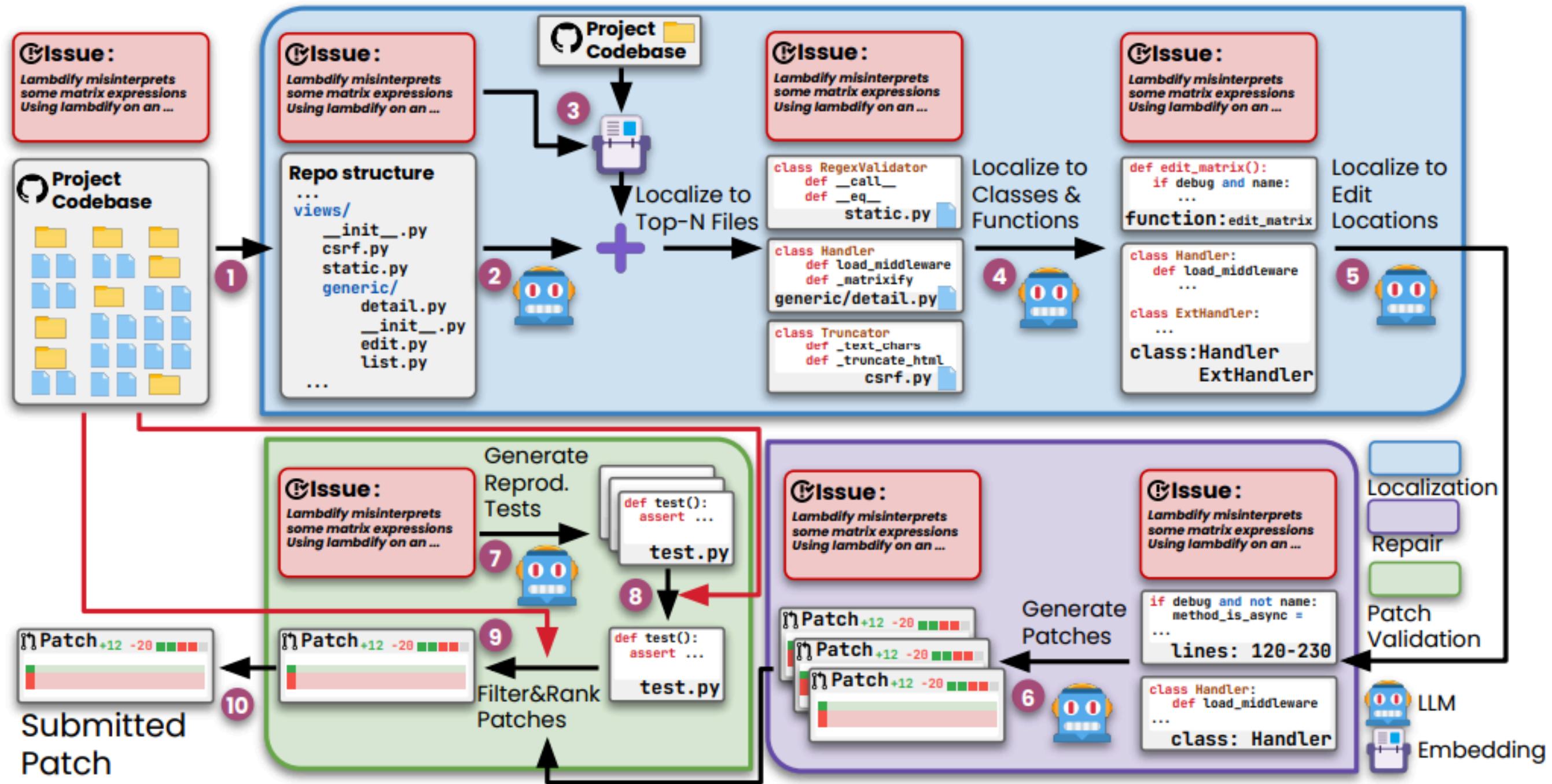













































Figure 1: Overview of AGENTLESS.

AGENTLESS benchmarks

Tool	LLM	SWE-bench Lite		SWE-bench Lite-S	
		% Resolved	Rank	% Resolved	Rank
CodeStory Aide [2] 	 GPT-4o+  Claude 3.5 S	129 (43.00%)	1	114 (45.78%)	1
Bytedance MarsCode [58] 	NA	118 (39.33%)	2	106 (42.57%)	2
Honeycomb [10] 	NA	115 (38.33%)	3	98 (39.36%)	3
MentatBot [14] 	 GPT-4o	114 (38.00%)	4	96 (38.55%)	4
Gru [20] 	NA	107 (35.67%)	5	94 (37.75%)	5
Isoform [12] 	NA	105 (35.00%)	6	91 (36.55%)	6
SuperCoder2.0 [22] 	NA	102 (34.00%)	7	87 (34.94%)	7
Alibaba Lingma Agent [13] 	 GPT-4o+  Claude 3.5 S	99 (33.00%)	8	86 (34.54%)	8
Factory Code Droid [9] 	NA	94 (31.33%)	10	82 (32.93%)	10
Amazon Q Developer-v2 [4] 	NA	89 (29.67%)	12*	76 (30.52%)	13
CodeR [30] 	 GPT-4	85 (28.33%)	14	71 (28.51%)	14*
MASAI [27] 	NA	84 (28.00%)	15	70 (28.11%)	16
SIMA [3] 	 GPT-4o	83 (27.67%)	16	71 (28.51%)	14*
IBM Research Agent-101 [1] 	NA	80 (26.67%)	17*	66 (26.51%)	18*
OpenCSG StarShip [16] 	 GPT-4	71 (23.67%)	22	56 (22.49%)	23*
Amazon Q Developer [4] 	NA	61 (20.33%)	26	51 (20.48%)	25*
RepoUnderstander [64] 	 GPT-4	64 (21.33%)	25	51 (20.48%)	25*
AutoCodeRover-v2 [6]	 GPT-4o	92 (30.67%)	11	79 (31.73%)	11
RepoGraph [19]	 GPT-4o	89 (29.67%)	12*	77 (30.92%)	12
Moatless [15]	 Claude 3.5 S	80 (26.67%)	17*	67 (26.91%)	17
	 GPT-4o	74 (24.67%)	21	62 (24.90%)	21
OpenDevin+CodeAct v1.8 [17]	 Claude 3.5 S	80 (26.67%)	17*	65 (26.10%)	20
Aider [37]	 GPT-4o+  Claude 3.5 S	79 (26.33%)	20	66 (26.51%)	18*
SWE-agent [101]	 Claude 3.5 S	69 (23.00%)	23	58 (23.29%)	22
	 GPT-4o	55 (18.33%)	28	45 (18.07%)	27*
	 GPT-4	54 (18.00%)	29	42 (16.87%)	29
AppMap Navie [5]	 GPT-4o	65 (21.67%)	24	56 (22.49%)	23*
AutoCodeRover [108]	 GPT-4	57 (19.00%)	27	45 (18.07%)	27*
RAG [101]	 Claude 3 Opus	13 (4.33%)	30	10 (4.02%)	30
	 GPT-4	8 (2.67%)	32	5 (2.01%)	32
	 Claude-2	9 (3.00%)	31	6 (2.41%)	31
	 GPT-3.5	1 (0.33%)	33	0 (0.00%)	33
AGENTLESS	 GPT-4o	96 (32.00%)	9	84 (33.73%)	9

Pros:

- ✓ Faster execution
- ✓ Low cost
- ✓ Scalable
- ✓ Builds its own unit tests

Cons:

- ✗ Lack of memory
- ✗ No self-correction mechanism
- ✗ Struggles with complex bugs

RepairAgent (May 2024)

- The first work to address APR through autonomous LLM agents.
- Key Features:
 - Queries the agent
 - Post-processes the response
 - Executes the command suggested by the agent
 - Updates the dynamic prompt based on the command's output
- Successfully fixes **164 out of 835 bugs**. 39 of those have not been fixed by prior work on the Defects4J benchmark. Costs **~14 cents per bug**.
- Open-source.

How RepairAgent works

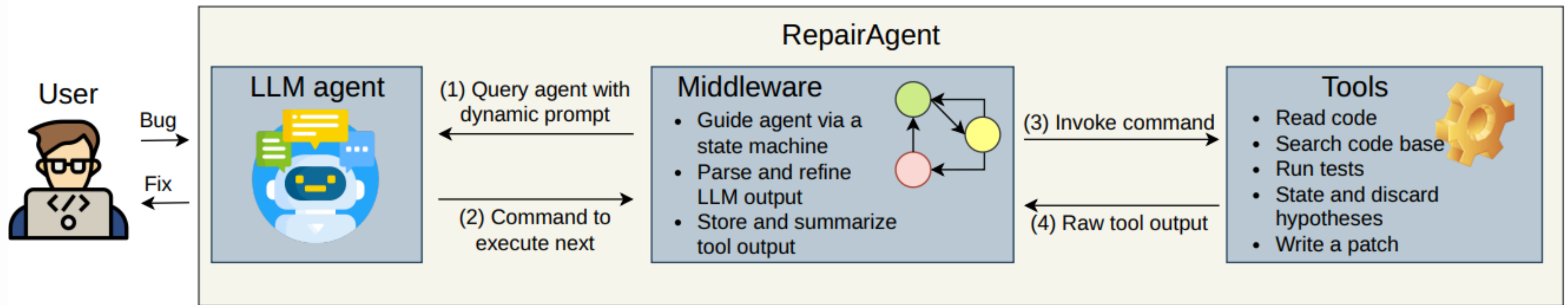


Fig. 1: Overview of RepairAgent.

How Middleware works

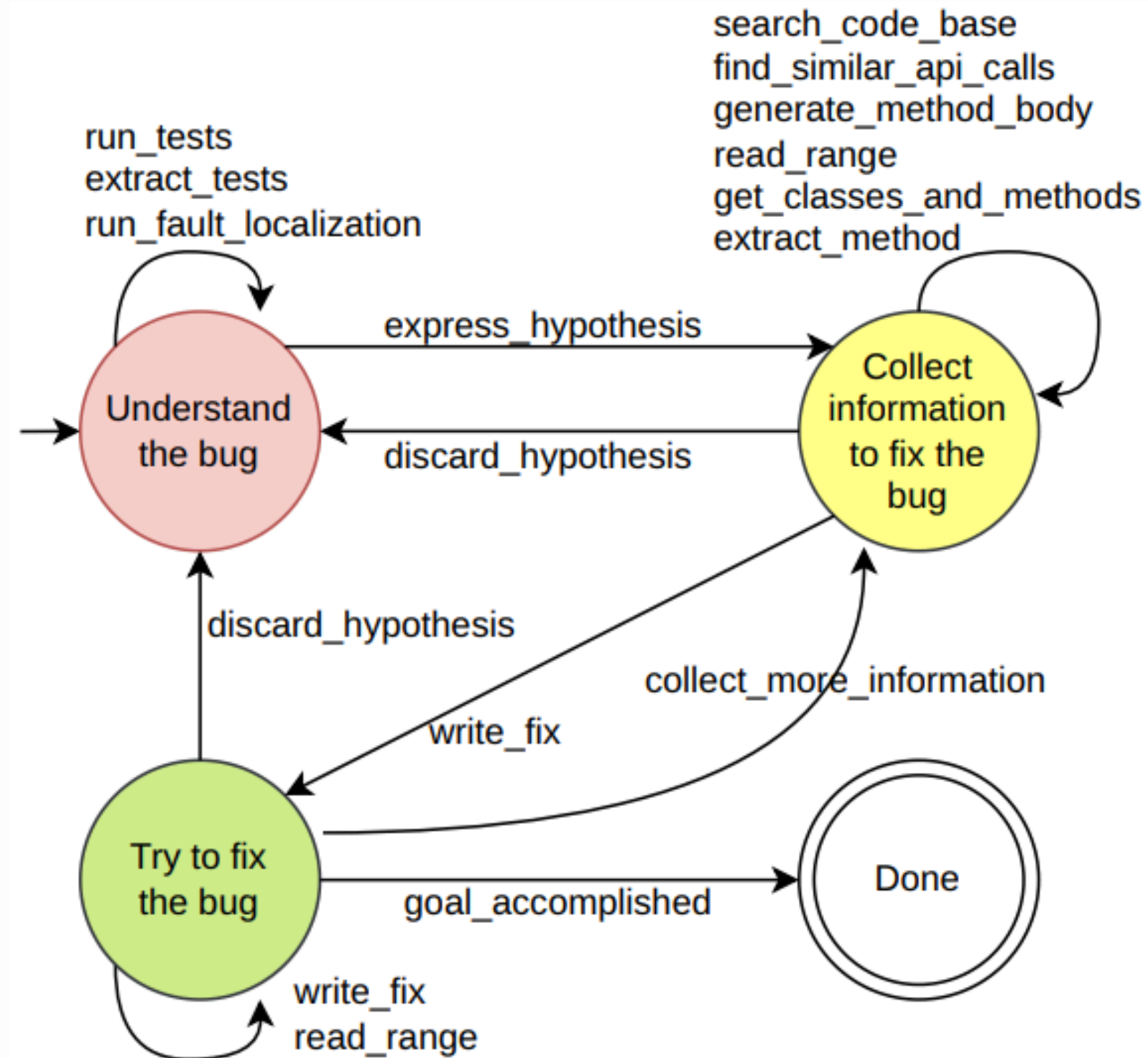
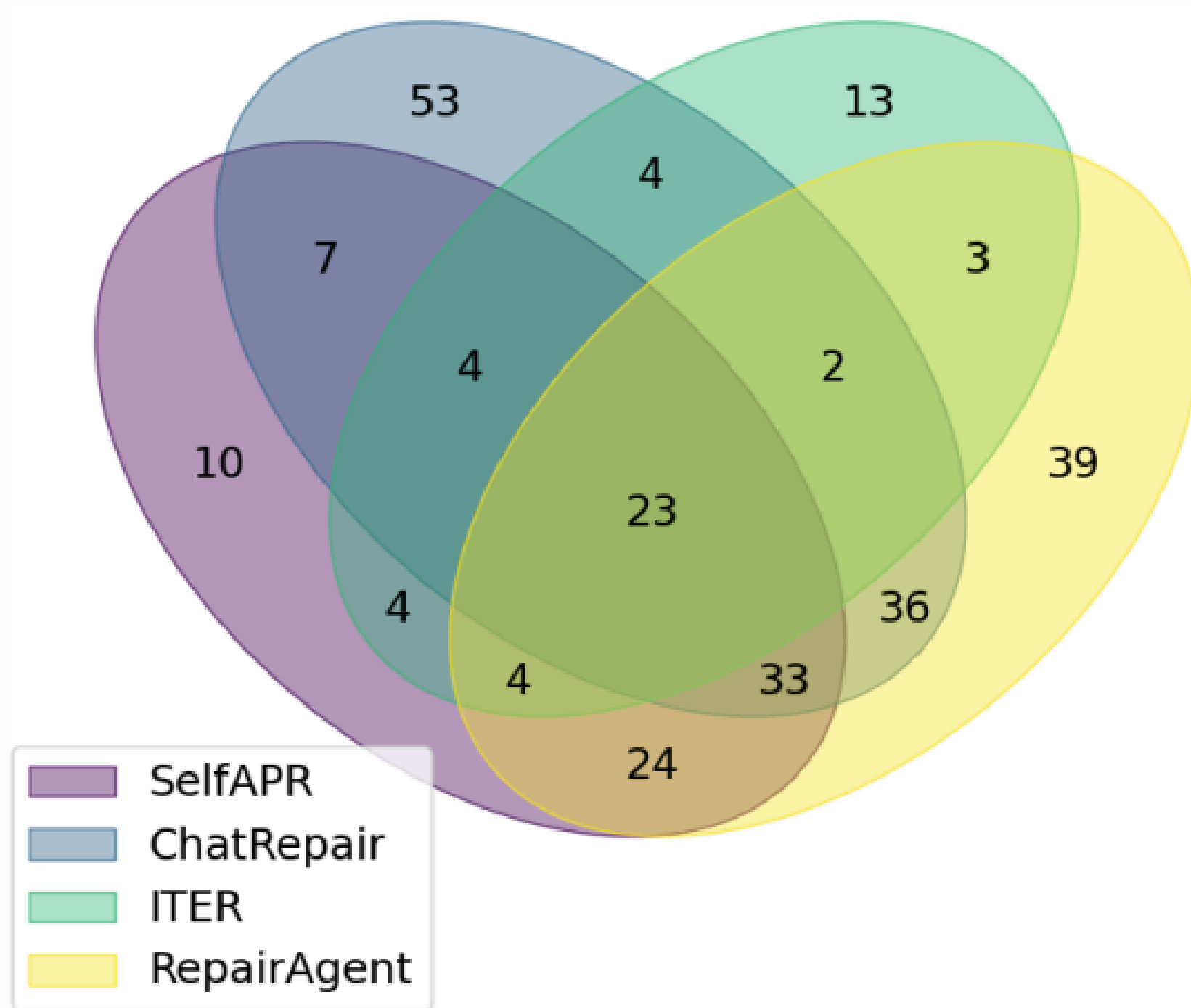


Fig. 2: State machine to guide selection of tools.

RepairAgent benchmarks



Pros:

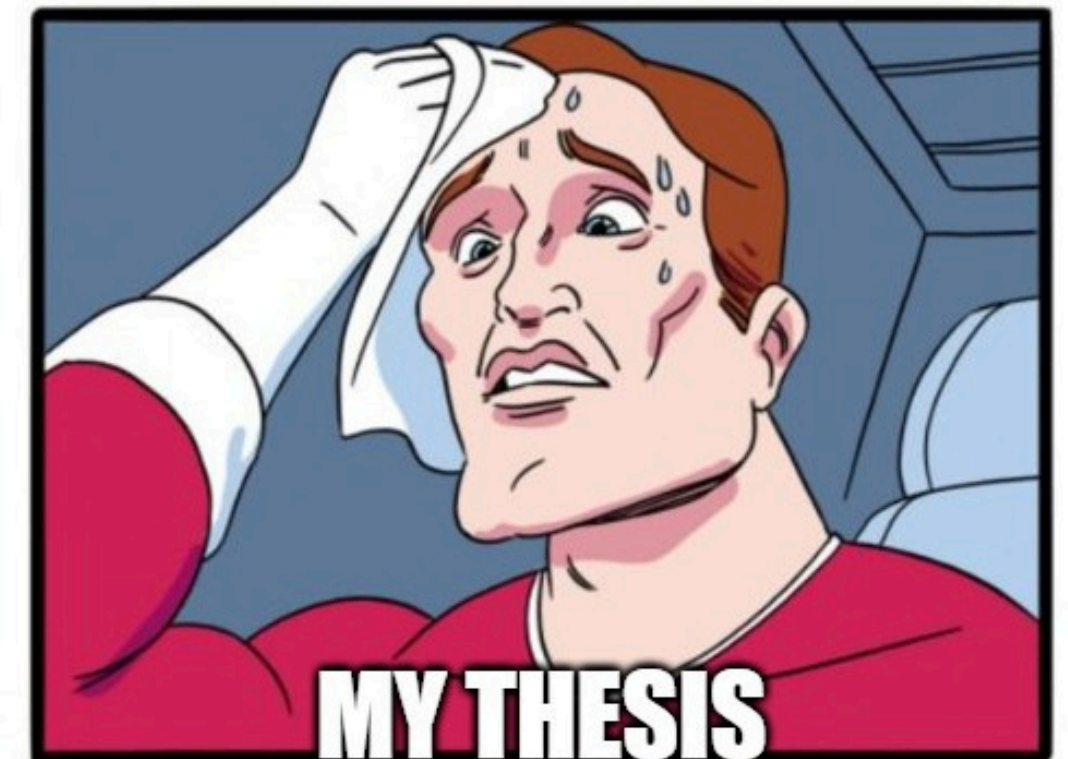
- ✓ Fully autonomous
- ✓ Understands software context
- ✓ Scalable
- ✓ Builds its own unit tests

Cons:

- ✗ Requires multiple tool calls
- ✗ Can repeat the same mistakes
- ✗ If error messages are unclear, it generates incorrect patches

ChatRepair (September 2024)

- State-of-the-art APR tool that uses LLMs to generate high-quality bug fixes.
- Uses **conversation** to produce and refine patches.
- **Learns** from both failures and successes of earlier patching attempts by keeping track of the chat history.
- Fixes **162 out of 337 bugs** for **\$0.42 each**.
- Not open-source 🧐

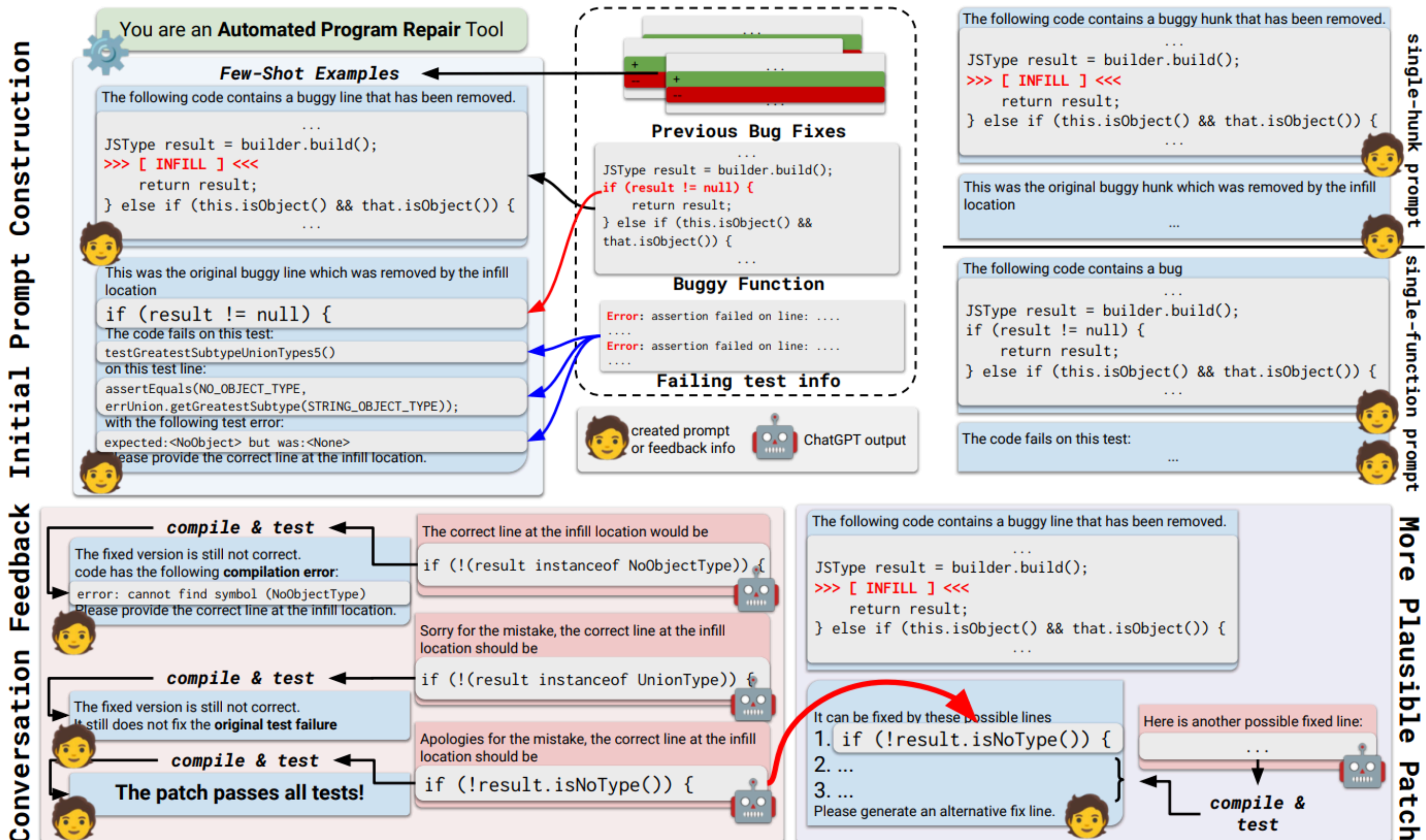


ChatRepair vs RepairAgent

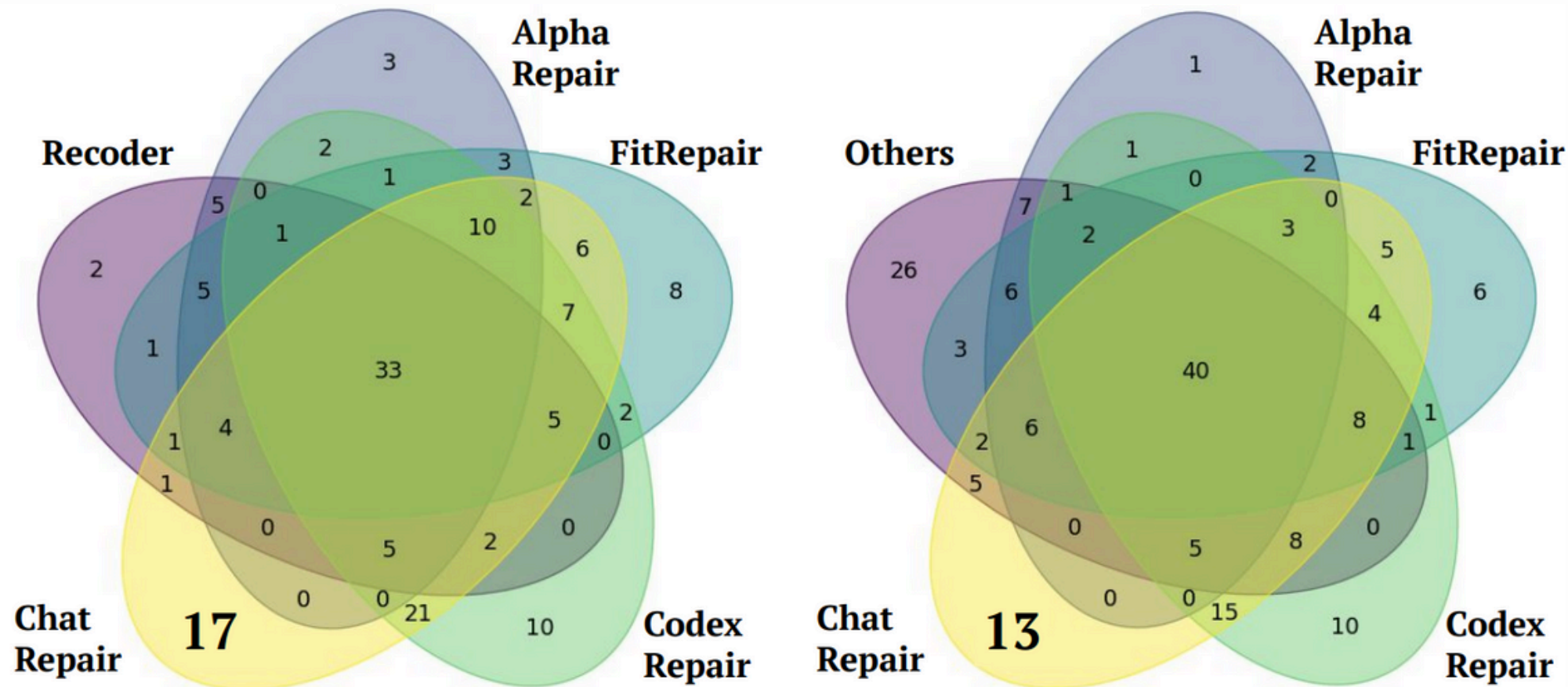


<u>ChatRepair</u>	<u>RepairAgent</u>
Chat-based	Agent-driven
Takes any form of natural language and code as input	Strictly takes code and test failures or logs as input
Learns from both failures and successes of earlier patching	Is susceptible to repeating the same mistake
Semi-autonomous – depends on user input	Fully autonomous
Fixes 162 out of 337 bugs for ~42 cents each	Fixes 164 out of 835 bugs for ~14 cents each
Not open-source	Open-source

How ChatRepair works



ChatRepair benchmarks



(a) Learning-based APR tools

(b) All APR tools

Testname: `testCreateNumber()`

Failure Line: `0xFADE == NumberUtils.createNumber("0Xfade").intValue()`

Error Message: `0Xfade is not a valid number.`

```
}  
- if (str.startsWith("0x") || str.startsWith("-0x")) {  
+ if (str.startsWith("0x") || str.startsWith("0X") ||  
+   str.startsWith("-0x") || str.startsWith("-0X")) {  
    case '\n': sb.append("\n"); break;
```

Figure 4: Unique bug fixed in Defects4j 1.2

Pros:

- ✓ Learns by asking for clarification
- ✓ Retains chat history
- ✓ Scalable and lightweight
- ✓ Versatile across bug types

Cons:

- ✗ Requires human interaction
- ✗ No tool integration
- ✗ Highly dependent on prompts

Final thoughts

- Automated Program Repair is a new and hot branch that is rapidly evolving. LLMs can reason about code, generate meaningful fixes, and adapt to a wide variety of bugs.
- **LLMs are replacing developers?????** 🤖 No. Human oversight is still crucial. Especially for patch validation and integration.
- Lots of tools already exist, each with its own trade-off:
 - **GenProg** is an old-school, cost-efficient and reliable implementation.
 - **AGENTLESS** 😸 is fast and stateless — ideal for simple one-shot fixes.
 - **RepairAgent** offers depth and iteration.
 - **ChatRepair** offers speed and interactivity.
- The future is hybrid: LLMs are proving to be valuable tools that enhance developers' toolkits.

Demo



Thank you!

Michael-Raphael Kostagiannis

Undegraduate student at DIT NKUA

(hopefully not for long)