Automated Program Repair via Conversation: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT

University of Illinois at Urbana-Champaign (UIUC), USA

Presenter: Thi Van Anh Dau

Motivation

- APR: automatically generate patches for bugs in software
- Limitations of existing approaches:
  - Traditional APR tools: lack of patch variety
  - Learning-based APR:
    - rely heavily on the training data
    - may not generalize to unseen bugs

Motivation

- Limitations of existing approaches:
  - Recent advanced LLMs for APR:
    - Missing test failure information

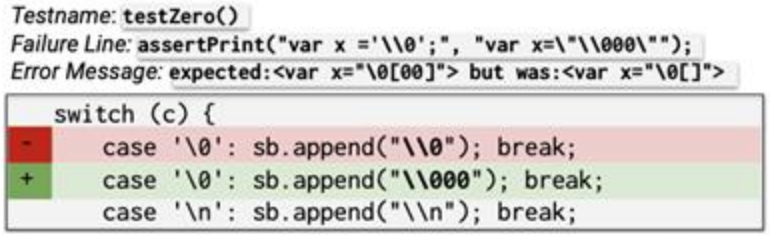


Figure 1: Example bug fix with original testcase information

    - Repeated sampling
    - Ignorance of valuable plausible patches

Contribution

- ChatRepair, a conversation-driven APR tool using the ChatGPT model
  - They leverage previously ignored test failure information and test success in a conversion manner
- ChatRepair obtains the new state-of-the-art repair result on Defects, QuixBugs and ConDefect datasets with high performance and low cost

Overview

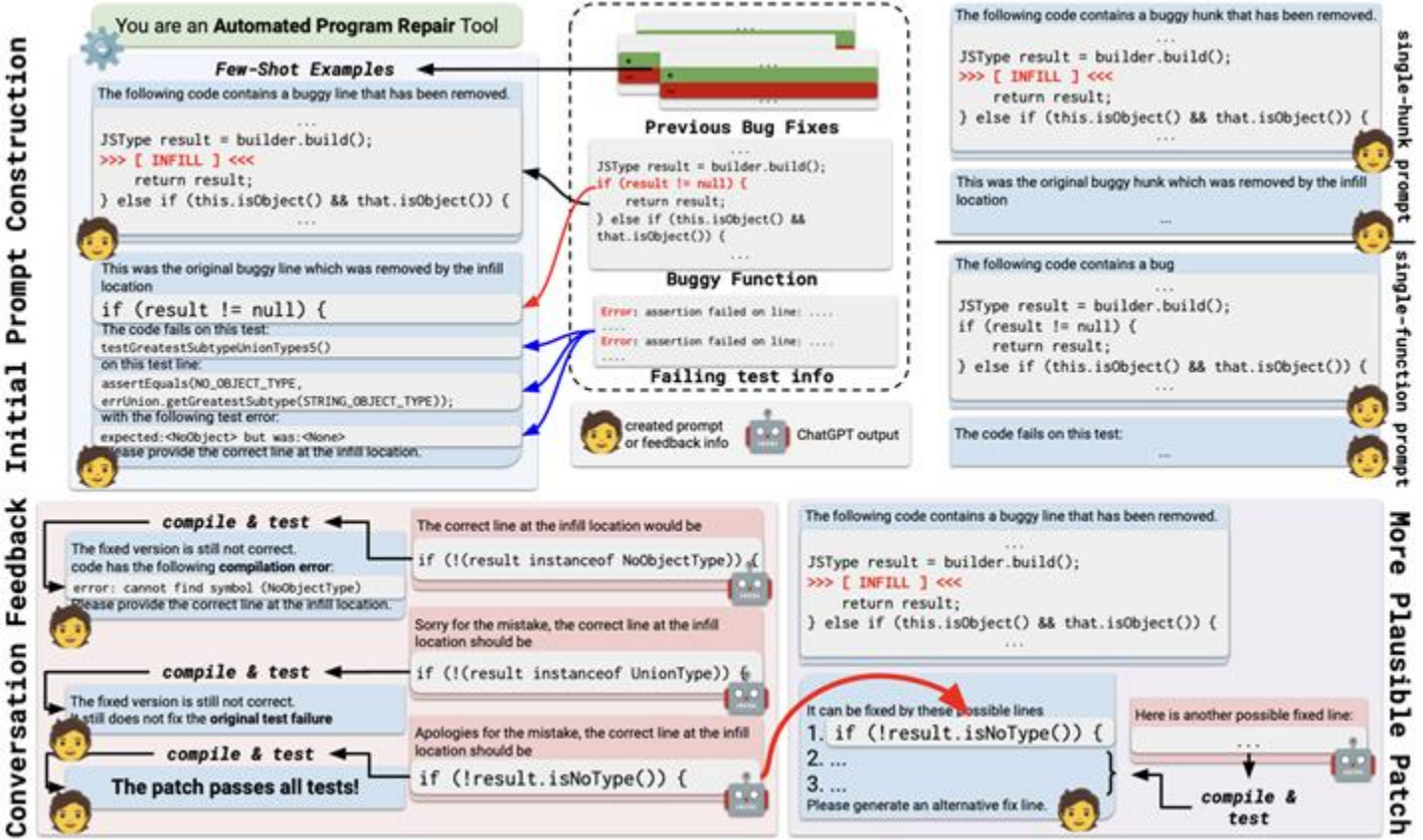
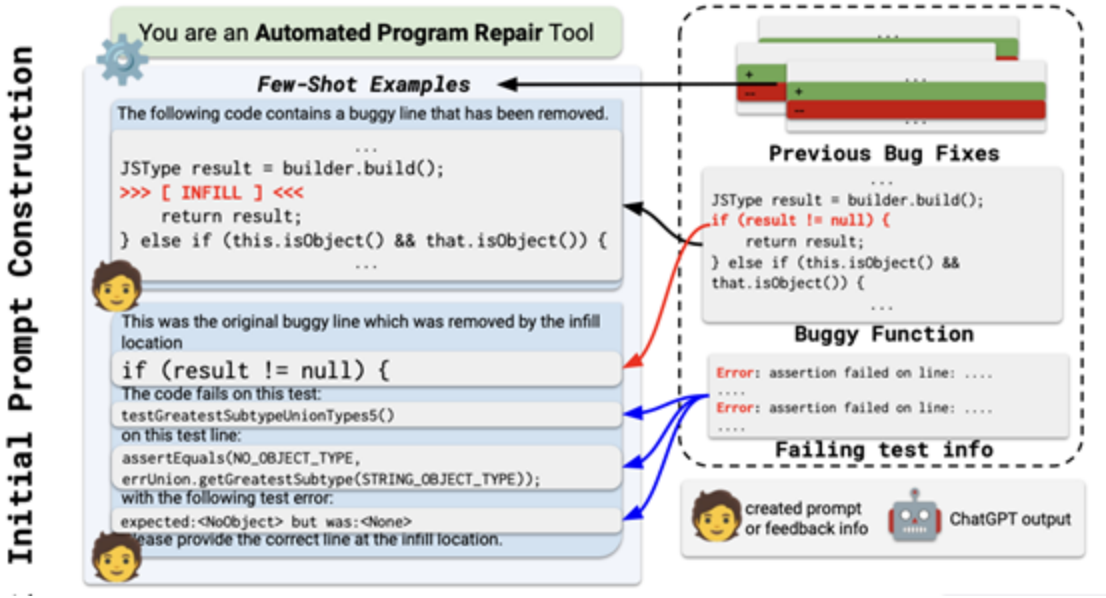


Figure 2: Overview of ChatRepair

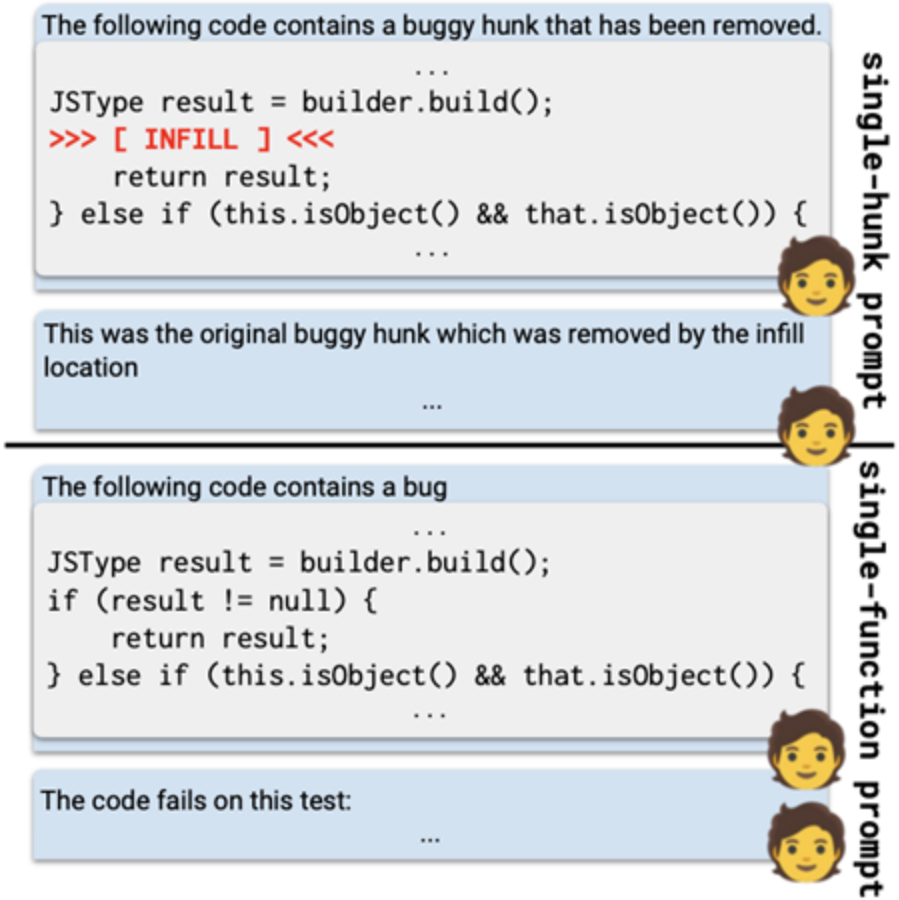
Initial Input

- the buggy code within the function with an infill location indicator
- buggy line
- failing tests
  - its name
  - failure code line
  - error message



Initial Input

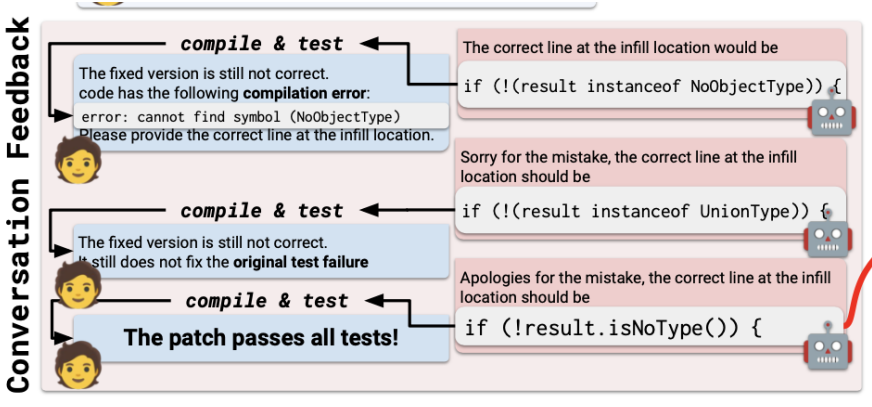
- repair strategy
  - single-line
  - single-hunk
  - single-function



Conversational Repair

Algorithm 1: Conversational Repair

```
1  Function ConversationalRepair:
       Input  : initialPrompt (initial prompt), oFailure (original failing
                test info), testSuite (test suite), ChatGPT,
                maxConvLength (max conversation length), maxTries
                (max tries), AltInstruct (plausible patch prompt)
       Output : pPatches (plausible patches), cost (total cost)
2      pPatches, currentTries, cost ← NONE, 0, $0
3      while currentTries < maxTries and pPatches is NONE do
4          currentLength ← 0
5          input ← initialPrompt
6          while currentLength < maxConvLength do
7              patch, cost ← ChatGPT (input)
8              testResult ← Validate (patch, testSuite)
9              if testResult is PASS then
10                 pPatches ← [ patch ]
11                 break
12             else if testResult is oFailure then
13                 feedback ← "still doesn't fix original failure"
14             else
15                 feedback ← ConstructPrompt (testResult)
16             input ← { input, patch, feedback }
17             currentTries ← currentTries +1
18             currentLength ← currentLength +1
```



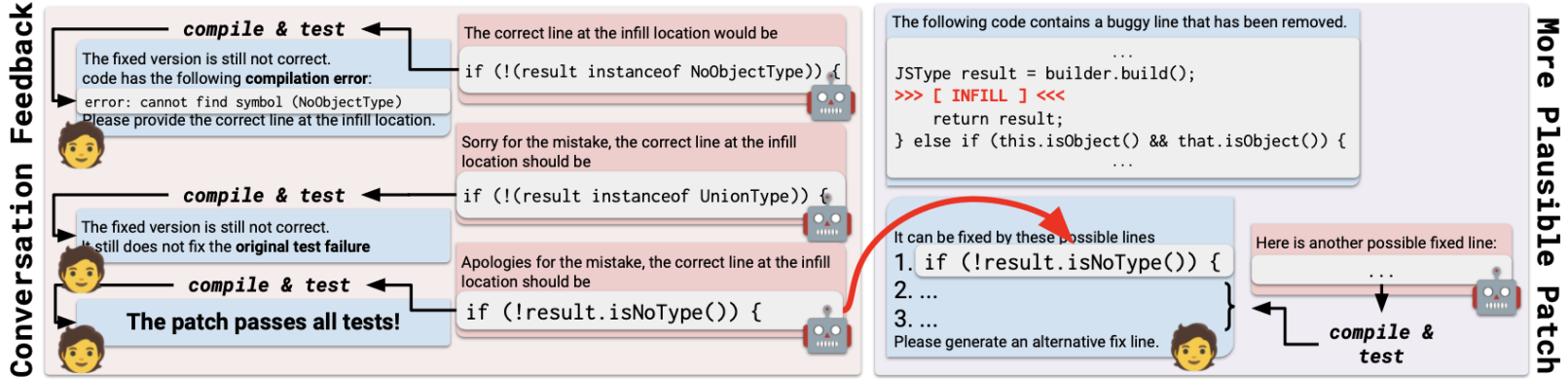
Plausible Patch Generation

```
19  if pPatches is not NONE then
20      while currentTries < maxTries do
21          input ← { initialPrompt, pPatches, AltInstruct }
22          patch, cost ← ChatGPT (input)
23          testResult ← Validate (patch, testSuite)
24          if testResult is PASS and patch not in pPatches then
25              pPatches ← pPatches | [ patch ]
26          currentTries ← currentTries +1
27  return pPatches, cost
```

```
+ if (fnType != null && fnType.isConstructor()) {
```
Plausible Patch

```
+ if (fnType != null && (fnType.isConstructor() ||
+     fnType.isInterface())) {
```
Correct Patch

Figure 6: Plausible generation example



Experiment Setup

- Repair scenarios
  - **single-line**–fixed by replacing/adding a single line
  - **single-hunk**–fixed by replacing/adding a continuous code hunk
  - **single-function**–fixed by generating a new function to replace the original buggy version

Experiment Setup

- Temperature = 1
- Maximum conversation length = 3
- Maximum number of repair attempts
  - 200 for single-line and single-hunk
  - 100 for single-function

Benchmarks

- 2 popular repair benchmarks: Defect4j, QuixBug
- ConDefect: single-line programming contest bugs collected after October 2021

Baseline

- 10 learning-based and LLM-based technique: FitRepair, SelfAPR, AlphaRepair , RewardRepair, Recoder, CURE, CoCoNuT, DLFix, SequenceR, CodexRepair, BaseChatGPT

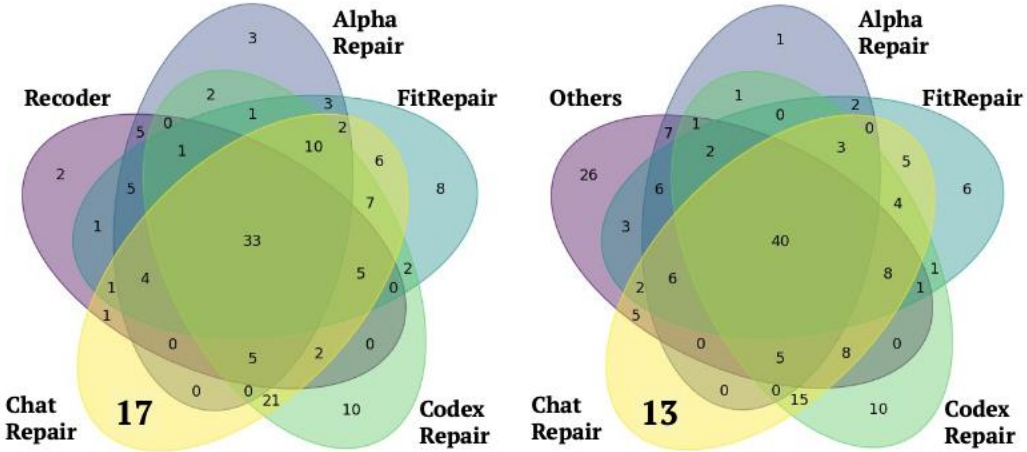
- 12 traditional APR tools: TBar, PraPR, AVATAR, SimFix, FixMiner, CapGen, JAID, SketchFix, NOPOL, jGenProg, jMutRepair, and jKali

RQ1: Effectiveness of ChatRepair compared to other baselines

Table 1: Correct fixes on Defects4j

Dataset	ChartRepair	BaseChatGPT	CodexRepair	FitRepair	AlphaRepair	SelfAPR	RewardRepair	Recoder	TBar	CURE
Chart	15	9	9	8	9	7	5	10	11	10
Closure	37	23	30	29	23	19	15	21	16	14
Lang	21	15	22	19	13	10	7	11	13	9
Math	32	25	29	24	21	22	19	18	22	19
Mockito	6	6	6	6	5	3	3	2	3	4
Time	3	2	3	3	3	3	1	3	3	1
D4J 1.2	**114**	80	99	89	74	64	50	65	68	57
D4J 2.0	**48**	25	31	44	36	31	25	11	8	-

RQ1: Effectiveness of ChatRepair compared to other baselines



(a) Learning-based APR tools (b) All APR tools

Figure 3: Bug fix Venn diagram on Defects4j 1.2

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

Testname: `testNonFiniteDoublesWhenLenient()`
Failure Line: `jsonWriter.value(Double.NaN);`
Error Message: `Numeric values must be finite, but was NaN`

```
    writeDeferredName();
-   if (Double.isNaN(value) || Double.isInfinite(value)){
+   if (!isLenient() && (Double.isNaN(value) ||
+       Double.isInfinite(value))) {
        throw new IllegalArgumentException("Numeric
```

Figure 5: Unique bug fixed in Defects4j 2.0

RQ2: Repair Scenarios

Table 3: Correct fixes using three repair settings

Tools	D4J 1.2			Quixbugs-Py			Quixbugs-J		
	SL	SH	SF	SL	SH	SF	SL	SH	SF
ChatRepair	**57**	**79**	**76**	**39**	**40**	**40**	**36**	**37**	**39**
BaseChatGPT	41	55	45	38	37	35	33	36	**39**
CodexRepair	39	62	63	**39**	39	37	34	34	32

RQ3: Ablation study

Table 4: Initial prompt variations

Initial Prompt	#P	Avg. # tries	Avg. $
BasePrompt	55	22.53	$0.069
TestName+ErrMsg	59	22.47	$0.072
TestName+ErrMsg+FailLine	**64**	**21.86**	**$0.061**
TestName+ErrMsg+TestBody	61	23.42	$0.083
`You are a helpful assistant`	**64**	24.17	$0.074
`You are an APR tool`	**64**	**21.86**	**$0.061**
0-shot	64	21.86	**$0.061**
1-shot	**65**	9.91	$0.072
2-shot	**65**	**9.87**	$0.085

Table 5: Feedback response variations

Feedback Response	#P	Avg. # tries	Avg. $
BaseFeedback	58	23.12	$0.071
TestName+ErrMsg	61	22.48	$0.073
TestName+ErrMsg+FailLine	62	24.71	$0.074
Dynamic	**64**	**21.86**	**$0.061**

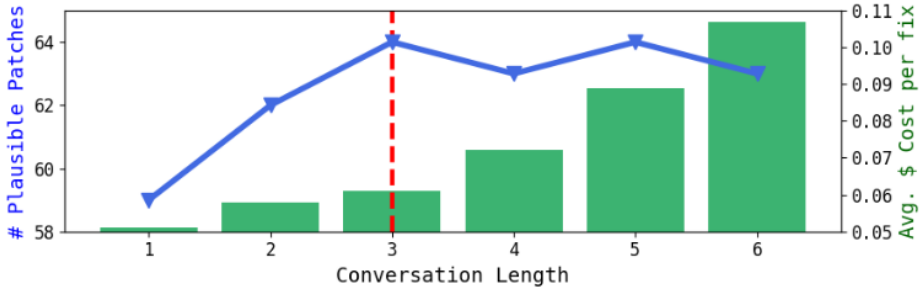


Figure 7: Effect of maximum conversation length

RQ4: Evaluation on the recent bugs

Table 6: Correct and Plausible fixes on ConDefect

Tools	ConDefects-Java	ConDefects-Python
ChatRepair	**243 / 250**	**241 / 249**
BaseChatGPT	170 / 189	165 / 171
AlphaRepair	154 / 158	142 / 160

Limitation

- ChatRepair depends heavily on test suite
- Lack of evaluation on the open-source models
- Scalability Concerns
- Limited Exploration of Multi-Hunk Bugs

Future Work

- improve ChatRepair with LLM-based agents
  - identify potential buggy locations
  - obtain useful repair-specific information
  - explore multi-hunk repair