Examining Zero-Shot Vulnerability Repair with Large Language Models

Research Questions

1. Can LLM generate repair?
2. How to design prompt?
3. How to deal with real-world problems?
4. How reliable is LLM-based repair?

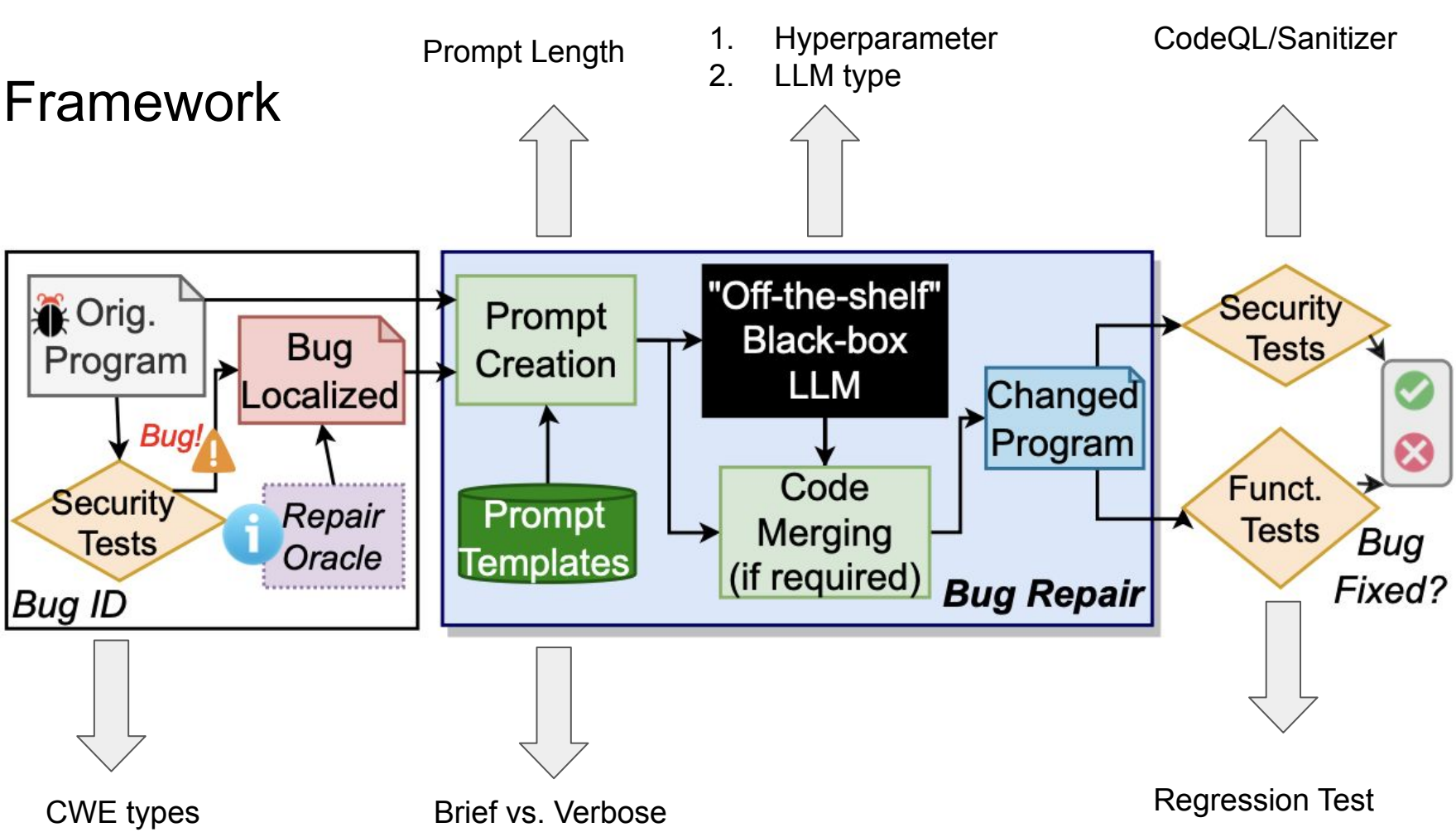
Neural Machine Translation for Repair

Train an encoder-decoder to predict a repair

Cons:

1. Fine-tuning
2. Overfitting
3. Restricted scenarios

Framework


Prompt Length
1. Hyperparameter
2. LLM type
CodeQL/Sanitizer
Orig. Program
Bug Localized
Bug!
Security Tests
Repair Oracle
Bug ID
Prompt Creation
Prompt Templates
"Off-the-shelf" Black-box LLM
Code Merging (if required)
Bug Repair
Changed Program
Security Tests
Funct. Tests
Bug Fixed?
CWE types
Brief vs. Verbose
Regression Test

Choose CWE types

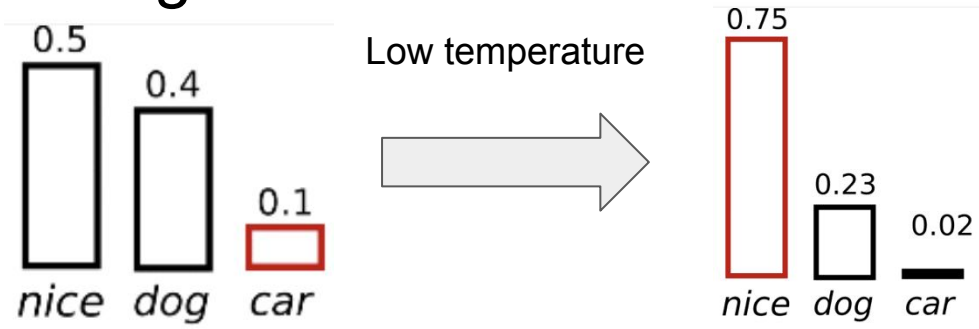
1. Important and common
2. Self-contained
3. High-level vs. Low-level

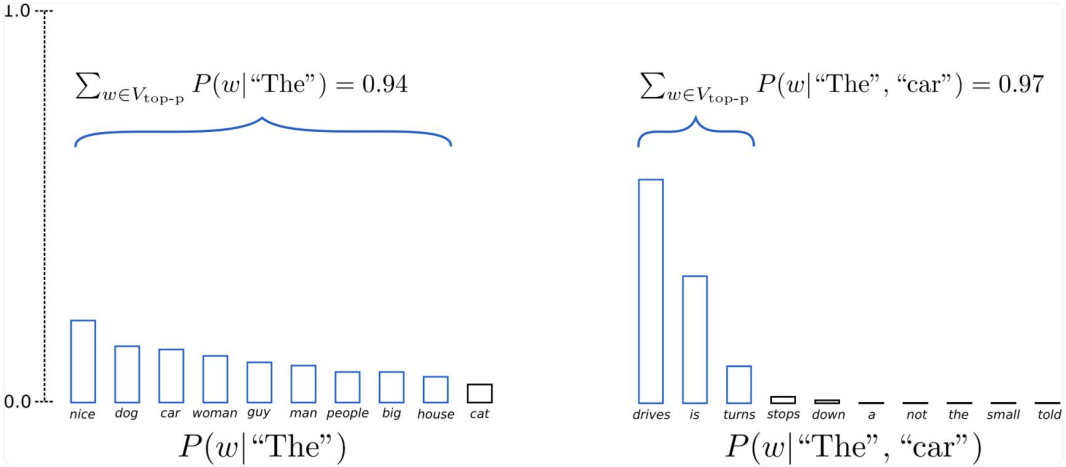
Choose LLM type

1. Closed-sourced vs. Open-sourced (Codex vs. PolyCoder)
2. Transparent dataset and training procedure vs. non-transparent (PolyCoder vs. gpt2-csrc)

Hyperparameter Tuning

1. Temperature
2. Top-p





Results

code-cushman-001

temperature \ top_p	0	0.25	0.5	0.75	1
0	-	-	-	-	-
0.25	-	-	-	2 /832	3 /818
0.5	-	2 /738	14 /688	13 /698	28 /709
0.75	-	6 /696	21 /652	32 /625	103 /776
1	-	3 /691	24 /608	56 /602	124 /655

code-davinci-001

temperature \ top_p	0	0.25	0.5	0.75	1
0	-	-	-	-	-
0.25	-	-	-	-	-
0.5	-	-	-	-	1 /419
0.75	-	-	-	-	20 /527
1	-	-	-	-	39 /405

(a) CWE-787: Out-of-bounds Write case study (C).

code-cushman-001

temperature \ top_p	0	0.25	0.5	0.75	1
0	49 /220	49 /220	49 /220	40 /220	49 /220
0.25	49 /220	46 /220	47 /220	48 /215	44 /214
0.5	49 /220	49 /220	49 /220	41 /210	59 /201
0.75	49 /220	49 /220	49 /220	41 /209	44 /198
1	49 /220	49 /220	49 /220	47 /209	29 /190

code-davinci-001

temperature \ top_p	0	0.25	0.5	0.75	1
0	84 /220	77 /220	87 /220	91 /220	91 /220
0.25	82 /220	92 /220	92 /220	93 /216	62 /212
0.5	90 /220	86 /220	90 /220	82 /209	62 /208
0.75	90 /220	90 /220	91 /220	82 /213	45 /203
1	89 /220	87 /220	87 /220	80 /212	23 /197

(b) CWE-89: ‘SQL injection’ case study for Python.

Results

2.2% of CWE-787; 29.6% of CWE-89 are fixed

SYNTHETIC PROGRAM REPAIR RESULTS. HIGHER VALID REPAIR PERCENTAGES (I.E. '# FN. & SAFE' / '# VLD.') ARE BETTER.

Scenario	# Gen.	# Vld.	# Fn.	# Vuln.	# Fn. & Vuln.	# Fn. & Safe.	% Vld. Repair
CWE-787	47500	22034	20029	21020	19538	491	2.2
CWE-89	11000	10796	7594	5719	4397	3197	29.6

Gen. (Generated), Vld. (compilable), Vuln. (Vulnerable), Fn. (Functional), Safe (Not Vulnerable)

Original generation:
4.4% of CWE-787; 93.6% of CWE-89

Prompt Template

Template ID	Description
`n.h.`	No Help - deletes the vulnerable code/function body and provides no additional context for regeneration.
`s.1`	Simple 1 - deletes the vulnerable code/function body and adds a comment '`bugfix: fixed [error name]`'.
`s.2`	Simple 2 - deletes the vulnerable code/function body and adds a comment '`fixed [error name] bug`'.
`c.`	Commented Code - After a comment '`BUG: [error name]`', this reproduces a 'commented-out' version of the vulnerable code/function body followed by a '`FIXED:`'. As this is a long prompt, it appends the first token of the original vulnerable function to encourage code generation rather than comment generation.
`c.m.`	Commented Code with Message - As with `c.`, but also includes a comment '`MESSAGE: [error message]`' and changes '`FIXED`' to '`FIXED VERSION`'. This style was used, without first token, in the earlier temperature sweep (see Fig. 4).
`c.a.`	Commented Code (alternative) - *Used for real-world examples, see Section V).* As with `c.`, but commented in the alternative style for C programs (i.e., in C commenting, `/*` and `*/` rather than `//`).
`c.n.`	Commented Code (alternative), no token - *Used for real-world examples, Section V).* As with `c.a.`, but with no 'first token' from vulnerable code.

Prompt Template

```
char* multiply_doubles_str(double d1, double d2) {
    // BUG: Potential buffer overflow
    // MESSAGE: This conversion may yield a string of
        length 318, which exceeds the allocated buffer size
         of 16
    //     //multiply the two doubles
    //
    //     double d3 = d1 * d2;
    //
    //     //convert the double to string
    //     char* d3_str = (char*)malloc(sizeof(char) *
        DBL_DIG + 1);
    //     sprintf(d3_str, "%lf", d3);
    //
    //     //return the string
    //     return d3_str;
    // }

    // FIXED VERSION:
```

Results

Scenario	Engine	n.h.	s.1	s.2	c.	c.m.
#2: CWE-79 (.py)	code-cushman-001	0/46	0/31	0/48	39/48	40/49
	code-davinci-001	0/49	0/47	0/48	38/49	40/46
	code-davinci-002	0/50	2/49	0/47	42/50	44/50
	j1-large	0/18	0/14	0/17	0/11	0/16
	j1-jumbo	0/19	0/14	0/15	0/16	0/13
	polycoder	0/14	0/9	0/3	0/8	0/5
#3: CWE-125 (.c)	code-cushman-001	31/50	25/49	28/47	45/50	39/50
	code-davinci-001	31/42	28/45	24/48	26/43	8/45
	code-davinci-002	32/48	31/49	27/49	36/50	13/50
	j1-large	1/16	4/20	4/15	0/17	1/12
	j1-jumbo	3/22	2/10	2/14	1/15	1/11
	gpt2-csrc	1/39	0/38	0/35	1/19	1/14
	polycoder	0/1	0/3	-	0/3	0/5
#4: CWE-20 (.py)	code-cushman-001	33/49	28/49	21/48	4/50	0/49
	code-davinci-001	34/49	27/43	21/45	1/50	3/50
	code-davinci-002	43/50	21/36	16/27	1/50	4/50
	j1-large	0/23	1/18	4/15	1/23	2/22
	j1-jumbo	12/25	9/22	7/23	0/24	0/24
	polycoder	9/19	1/7	0/13	2/11	0/9

Results

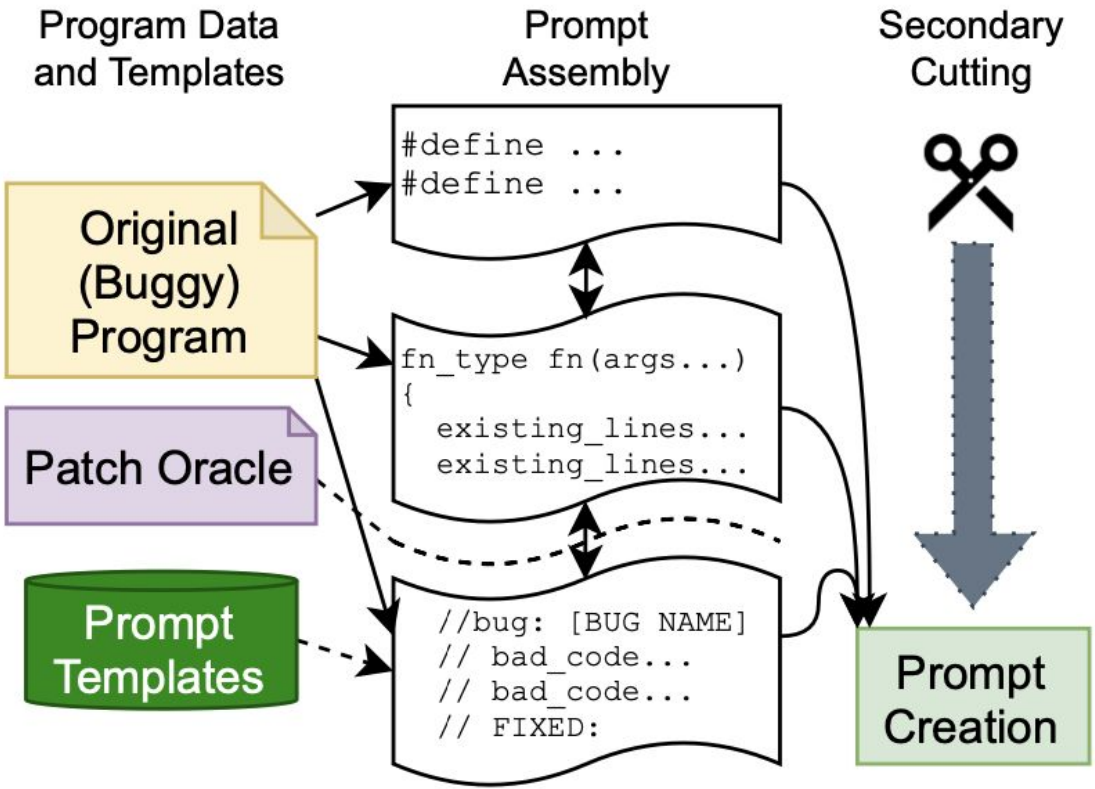
1. Results are diverse
2. Low context prompt may lead to secure but not functionally correct code
3. Overall, more robust prompt should include more details rather than fewer
4. Codex is better

HAND-CRAFTED PROGRAM REPAIR: TEMPLATE PERFORMANCE. HIGHER VALID REPAIR % (I.E. '# FN. & SAFE' / '# VLD.') ARE BETTER.

Template ID	# Gen.	# Vld.	# Vuln.	# Fn.	# Fn. & Vuln.	# Fn. & Safe	% Vld. Repair
`n.h.`	2000	1316	340	646	116	530	40.2
`s.1`	2000	1213	247	539	94	445	36.7
`s.2`	2000	1204	315	592	126	466	38.7
`c.`	2000	1345	561	1140	475	665	49.4
`c.m.`	2000	1315	478	1104	414	690	52.5

Gen. (Generated), Vld. (compilable), Vuln. (Vulnerable), Fn. (Functional), Safe (Not Vulnerable)

Real-world Repair



Challenges

1. long program with mutifile files vs. short context window (Sol: Localize)
2. Merge LLM-based repair into source codes (Sol: Augment prompt and find overlap)

```
/* Each tile contains only the data for a single plane
 * arranged in scanlines of tw * bytes_per_sample bytes.
 */
for (row = 0; row < imagelength; row += tl)
{
 nrow = (row + tl > imagelength) ? imagelength - row : tl;
 for (col = 0; col < imagewidth; col += tw)
 {
  for (s = 0; s < spp; s++)
  {  /* Read each plane of a tile set into srcbuffs[s] */
   tbytes = TIFFReadTile(in, srcbuffs[s], col, row, 0, s);
   if (tbytes < 0  && !ignore)
   {
    TIFFError(TIFFFileName(in),
    "Error, can't read tile for row %lu col %lu, "
...
```

(a) Buggy code from the large `tiffcrop.c` file. Error line 9 is shown in red.

```
for (row = 0; row < imagelength; row += tl)
{
 nrow = (row + tl > imagelength) ? imagelength - row : tl;
 for (col = 0; col < imagewidth; col += tw)
 {
/* BUG: stack buffer overflow
 * for (s = 0; s < spp; s++)
 *  { // Read each plane of a tile set into srcbuffs[s]
 * tbytes = TIFFReadTile(in, srcbuffs[s], col, row, 0, s);
 * FIXED:
 */
 for
```

(b) Prompt constructed according to Fig. 10 (shortened for brevity). The red highlighted line 7 is the original faulty line indicated by ASAN/the oracle. The template includes lines 8 and 9 (highlighted in grey) to encourage the LLMs to regenerate the safe code so the patch can be matched safely.

```
 (s = 0; (s < spp) && (s < MAX_SAMPLES); s++)
 {
   tbytes = TIFFReadTile(in, srcbuffs[s], col, row, 0, s);
   /* END BUG FIX */
   if (tbytes < (tsize_t)(tw * nrow * bytes_per_sample))
   {
   TIFFError ("readSeparateTilesIntoBuffer",
   ...
```

Results

1. The ensemble of LLMs is comparable to SOTA (questionable)
2. Memorization helps

Results

	Scenario, Engine	Prompt Template: n.h.	s.1	s.2	c.	c.a.	c.n.	LLMs / EF Pass?
EF01-libtiff CVE-2016-5321	code-cushman-001	3/4	2/4	4/8	1/44	3/49	2/48	
	code-davinci-001	3/13	0/4	4/9	6/43	5/24	4/15	
	code-davinci-002	20/21	21/22	9/13	6/48	1/44	4/43	✓
	j1-large	-	-	4/4	0/8	2/4	-	✓
	gpt2-csrc	1/2	20/20	21/21	1/5	2/29	2/9	
	polycoder	6/9	3/3	0/1	0/23	4/7	2/2	
EF02_1-libtiff CVE-2014-8128	code-cushman-001	-	-	-	0/4	0/40	0/37	
	code-davinci-001	0/2	-	-	0/44	0/45	0/42	
	code-davinci-002	-	-	-	0/48	0/48	0/44	✗
	j1-large	-	-	-	0/3	-	-	✓
	gpt2-csrc	-	-	-	0/3	0/1	0/1	
	polycoder	-	-	-	0/6	0/10	-	
EF02_2-libtiff CVE-2014-8128	code-cushman-001	0/50	0/50	0/50	0/50	0/50	0/50	
	code-davinci-001	0/50	0/50	0/50	0/50	0/50	0/50	
	code-davinci-002	0/50	0/50	0/50	0/50	0/50	0/50	✗
	j1-large	0/25	0/25	0/25	0/25	0/25	0/25	✗
	gpt2-csrc	0/50	0/50	0/50	0/50	0/50	0/50	
	polycoder	0/50	0/50	0/50	0/50	0/50	0/50	
EF07-libtiff CVE-2016-10094	code-cushman-001	-	-	-	3/26	0/1	-	
	code-davinci-001	-	-	-	0/1	0/1	-	
	code-davinci-002	-	-	-	2/3	-	-	✓
	j1-large	-	-	-	-	-	-	✓
	gpt2-csrc	-	-	-	-	-	-	
	polycoder	-	-	-	-	-	-	

Reliability

1. regression tests for a project are weak proxies for the correctness of the program
2. A pass guarantees that the failure case is repaired, however does not guarantee that the vulnerability is repaired
3. Complex real-world problems (dependency, multiple-files …)