Context Fusioning Multi Line Patch Generator

Team: BugZero

- Ayesh Vininda (190649F)
- Chathuranga Jayanath (190585E)
- Prasad Sandaruwan (190558B)

Supervisors:

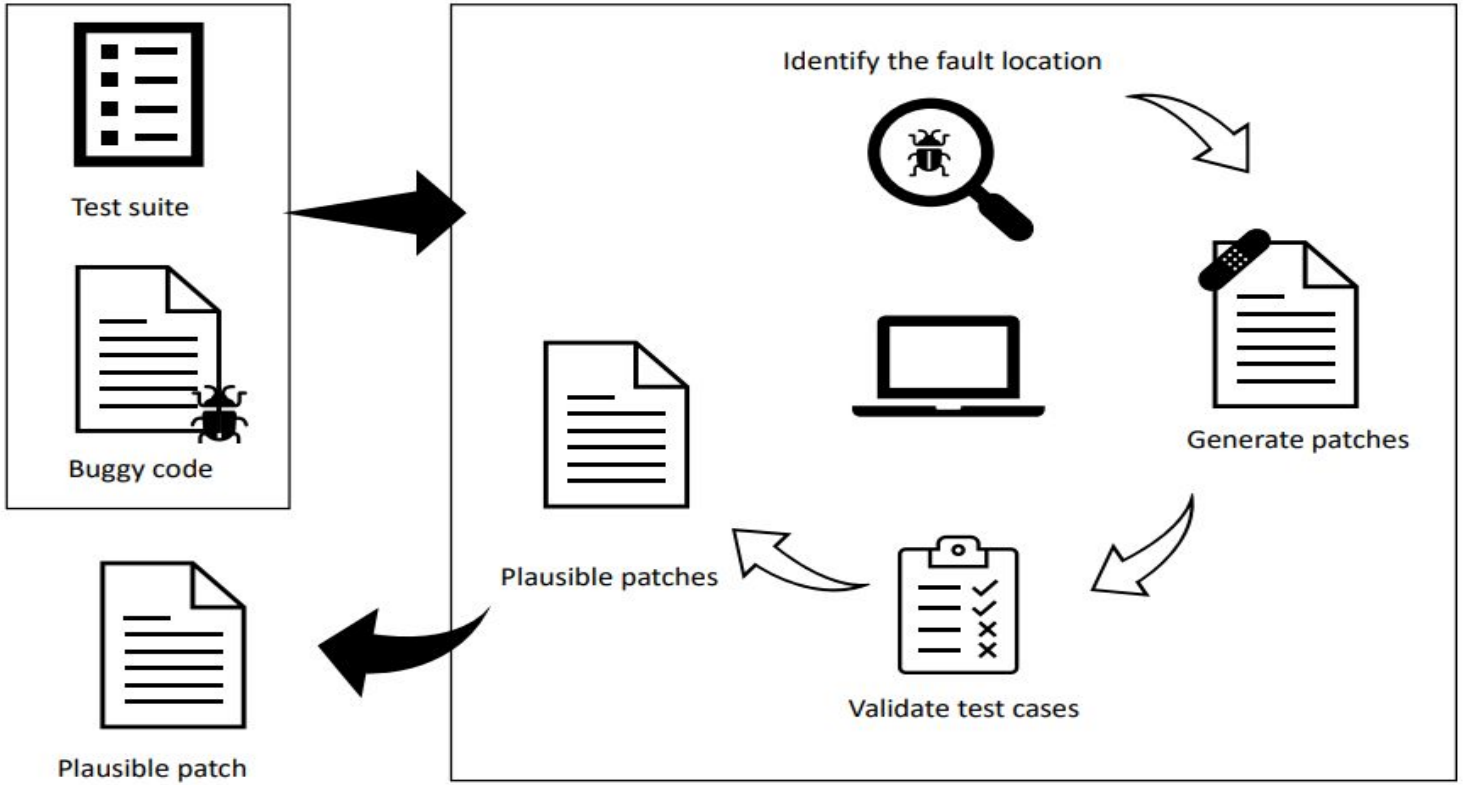
- Dr. Sandareka Wickramanayake, UoM
- Dr. Nisansa de Silva, UoM
- Dr. Ridwan Shariffdeen, NUS

Automated Program Repair (APR)

- Developers waste 50% of programming time finding and fixing bugs. (K. Saha et al. CASE 2017)

- Automates the process of identifying and fixing bugs, errors, or vulnerabilities in computer programs without manual intervention.

APR Workflow



Motivation

- Currently there are many deep learning based APR tools with single line patch generation capabilities such as:

  Eg: DLFix(Yi et al. ICSC 2020), SequenceR(Chen et al. TSE 2021), CoCoNuT (Lutellier et al. ISSTA 2020), CURE (Jiang et al. ICSE 2021)

- They can't fix bugs that need multi line patches.
- There have been only few attempts on deep learning based APR tools with multi line patch generation capabilities such as:

  Eg: ITER (He et al),DEAR (Yi et al. ICSE 2022),HERCULES (Saha et al. ICSE 2019)

- Limitations in current DL based APR tools with multi line patch generation capabilities
  - Can’t correctly identify partial patch correctness
  - Can’t provide all buggy lines in the same context
  - Can’t capture semantic dependencies between patches

Partial patch correctness

```
public final class MathUtils {
      * @throws IllegalArgumentException if n < 0
      */
     public static long factorial(final int n) {
+        if (n < 0) {
+           throw new IllegalArgumentException("must have n >= 0 for n!");
+        }
+        if (n > 20) {
-        long result = Math.round(factorialDouble(n));
-        if (result == Long.MAX_VALUE) {
             throw new ArithmeticException(
                     "factorial value is too large to fit in a long");
         }
…
     public static double factorialDouble(final int n) {
         if (n < 0) {
             throw new IllegalArgumentException("must have n >= 0 for n!");
         }
+        if (n < 21) {
+            return factorial(n);
+        }
         return Math.floor(Math.exp(factorialLog(n)) + 0.5);
     }
    public static double factorialLog(final int n) {
             if (n < 21) {
-                return Math.log(factorial(n));
-             }

}
```

Example from the Maths-93 in Defects4J

- Three locations to be fixed
- Only one failing test case given

Have all buggy lines in the same context.

```
1    public final class MathUtils {

539    public static int gcd(final int p, final int q) {
540                int u = p;
541                int v = q;
542                if ((u == 0) || (v == 0)) {
543        +           if ((u == Integer.MIN_VALUE) || (v == Integer.MIN_VALUE)) {
544        +               throw MathRuntimeException.createArithmeticException(
545        +                       "overflow: gcd({0}, {1}) is 2^31",
546        +                       new Object[] { p, q });
547        +           }
548                    return (Math.abs(u) + Math.abs(v));
549                }

714  public static int lcm(int a, int b) {

718              int lcm = Math.abs(mulAndCheck(a / gcd(a, b), b));
719        +         if (lcm == Integer.MIN_VALUE){
720        +             throw new ArithmeticException("overflow: lcm is 2^31");
721        +         }
722               return lcm;
723           }


1228      }
```

Defects4j Math 99 : which need fixes in 540 and 720 line in the MathUtils Class. MathUtils Class contain over 1200+ lines

Semantic dependencies between patches

```
public class Compiler extends AbstractCompiler {

          // if LanguageMode is ECMASCRIPT5_STRICT, only print 'use strict'
          // for the first input file
+         String code = toSource(root, sourceMap, inputSeqNum == 0);
-         String code = toSource(root, sourceMap);
          if (!code.isEmpty()) {
            cb.append(code);

   @Override
   String toSource(Node n) {
     initCompilerOptionsIfTesting();
+    return toSource(n, null, true);
-    return toSource(n, null);
   }

   /**
    * Generates JavaScript source code for an AST.
    */
+  private String toSource(Node n, SourceMap sourceMap, boolean firstOutput) {
-  private String toSource(Node n, SourceMap sourceMap) {
     CodePrinter.Builder builder = new CodePrinter.Builder(n);
     builder.setPrettyPrint(options.prettyPrint);
     builder.setLineBreak(options.lineBreak);
     builder.setSourceMap(sourceMap);
     builder.setSourceMapDetailLevel(options.sourceMapDetailLevel);

   }
}
```

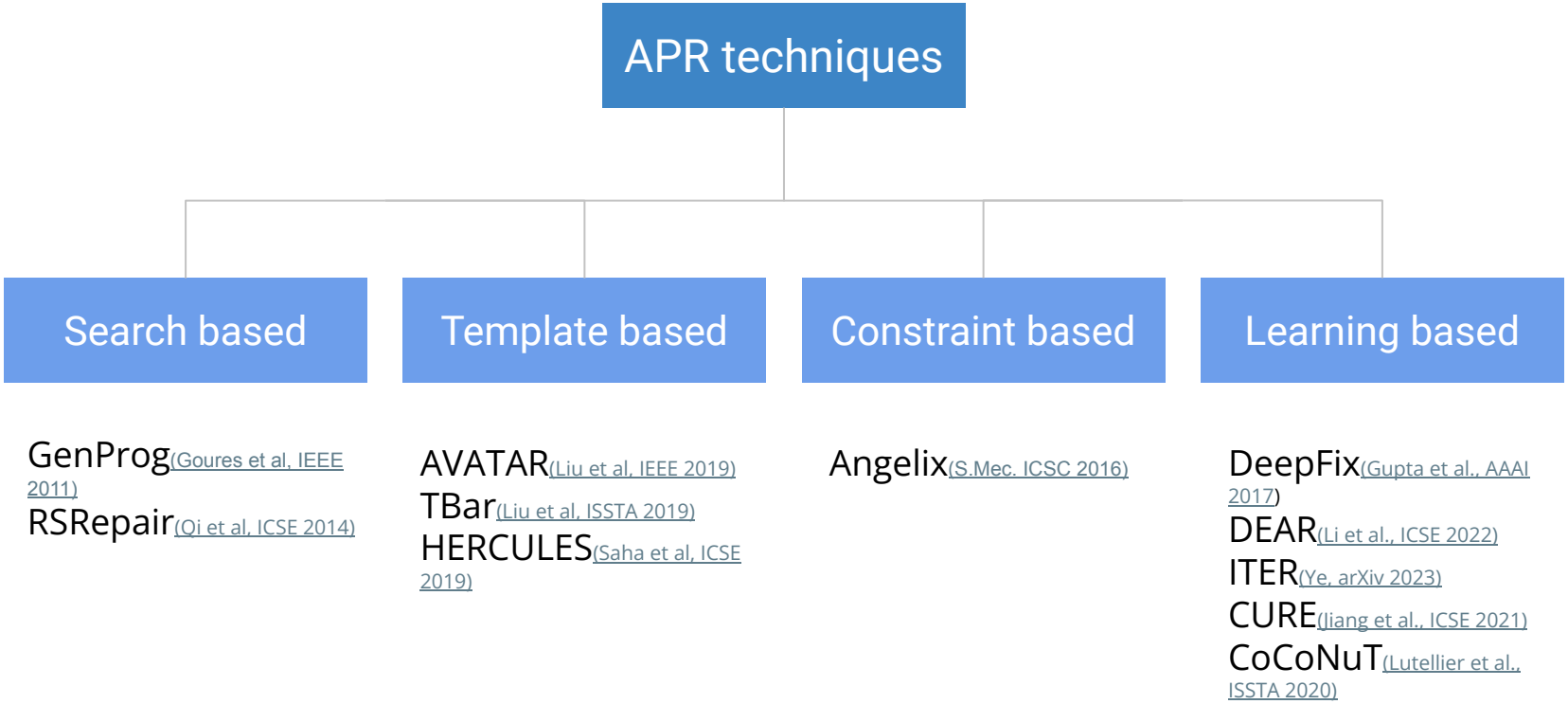
Defects4j Closure 64

Problem Statement

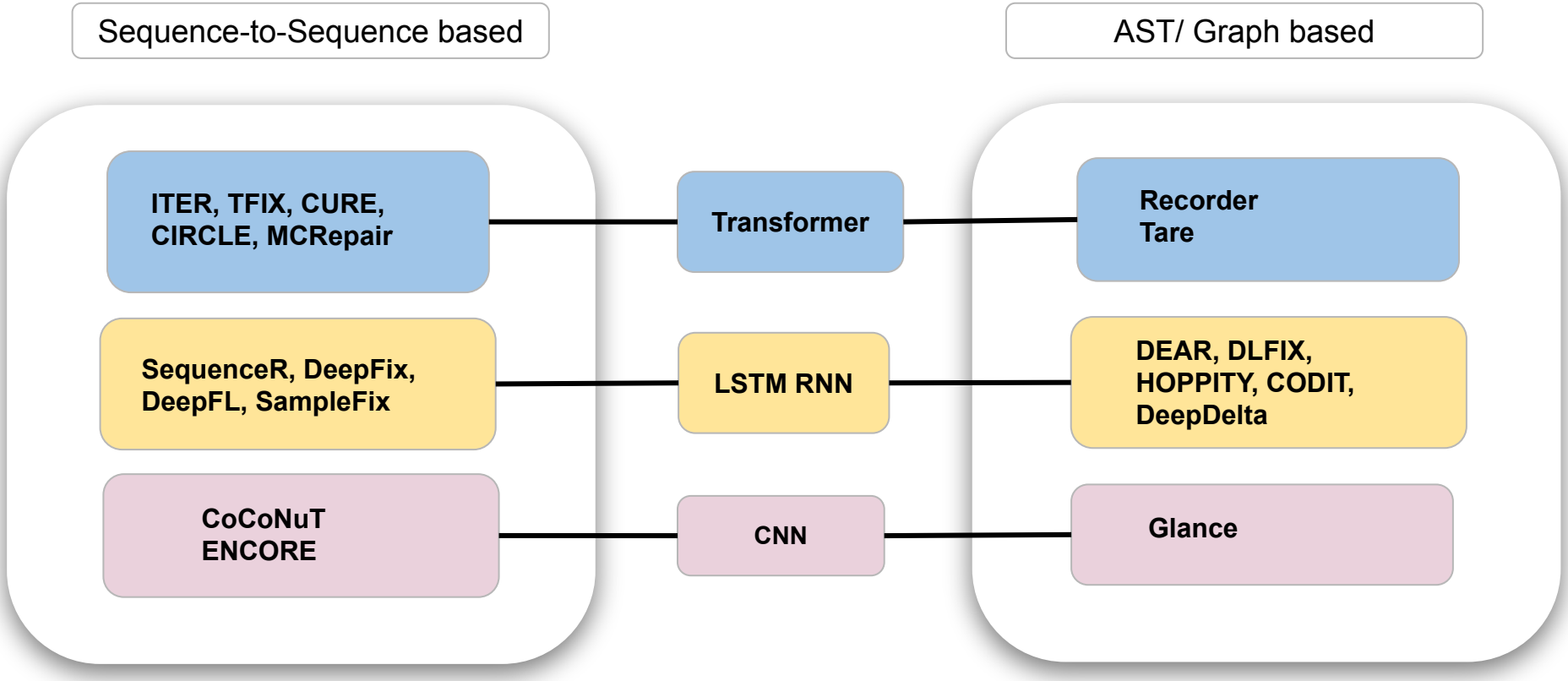
Develop a transformer-based Automated Program Repair tool to generate single line and multi line patches in a single file.

Objectives

1. Use generative AI to generate more test cases using code comments.
2. To develop a transformer based APR tool to fix bugs by generating single file single-line & single file multi-line patches.
3. Analyse and compare the results with existing techniques.


APR techniques
Search based
Template based
Constraint based
Learning based
GenProg(Goures et al, IEEE 2011)
RSRepair(Qi et al, ICSE 2014)
AVATAR(Liu et al, IEEE 2019)
TBar(Liu et al, ISSTA 2019)
HERCULES(Saha et al, ICSE 2019)
Angelix(S.Mec. ICSC 2016)
DeepFix(Gupta et al., AAAI 2017)
DEAR(Li et al., ICSE 2022)
ITER(Ye, arXiv 2023)
CURE(Jiang et al., ICSE 2021)
CoCoNuT(Lutellier et al., ISSTA 2020)

DL-Based APR Tools Categorization



Papers that inspired our approach

- Hercules: harnessing evolution for multi-hunk program repair (Seemanta Saha, Ripon K. Saha, and Mukul R. Prasad. 2019. Harnessing evolution for multi-hunk program repair. In Proceedings of the 41st International Conference on Software Engineering (ICSE '19). IEEE Press, 13–24)
- DEAR: a novel deep learning-based approach for automated program repair (Yi Li, Shaohua Wang, and Tien N. Nguyen. 2022. DEAR: a novel deep learning-based approach for automated program repair. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). Association for Computing Machinery, New York, NY, USA, 511–523.)
- ITER: Iterative Neural Repair for Multi-Location Patches (H. Ye and M. Monperrus, “ITER: Iterative neural repair for multi-location patches,” arXiv [cs.SE], 2023.)
- Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering (G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering,” arXiv [cs.CL], 2020.)

ITER: Iterative Neural Repair for Multi-Location Patches

- Focus on generating multi line patches with an iterative approach.
- Use iterative approach to improve partial patches.
- Results indicate that with iteration, partial patches can be improved to plausible patches.
- May generate 100k+ patches for a single buggy line.
- Max time taken to fix a bug 84 hours.
- Identify partial patch correctness with decreased number of failing test cases.

H. Ye and M. Monperrus, “ITER: Iterative neural repair for multi-location patches,” arXiv [cs.SE], 2023.

ITER: Iterative Neural Repair for Multi-Location Patches

- Focus on generating multi line patches with an iterative approach.
- Use iterative approach to improve partial patches.
- Results indicate that with iteration, partial patches can be improved to plausible patches.
- For each partial patch with decreased number of failing test cases, fault localization is re executed to identify next buggy line to be fixed.
- At inference time, fault localization, patch generation and patch validation run as a repair loop.

H. Ye and M. Monperrus, “ITER: Iterative neural repair for multi-location patches,” arXiv [cs.SE], 2023.

Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering

- Need to find answer for a question from many passages.
- Since there are many passages, they have come up with a better way to provide that input.
- This approach has achieved promising results.

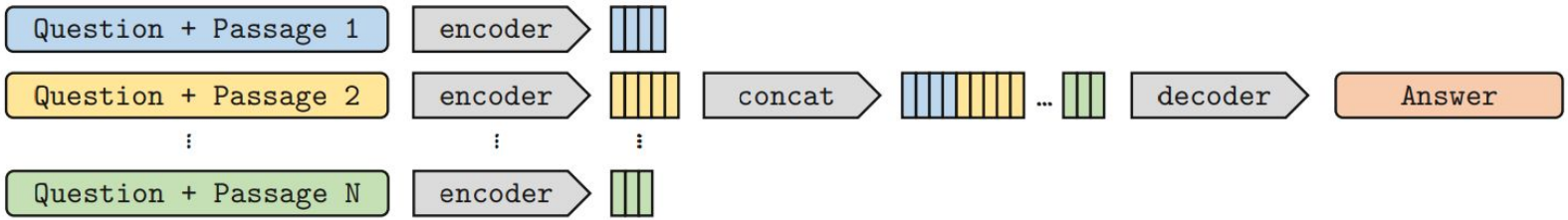


Figure 2: Architecture of the Fusion-in-Decoder method.

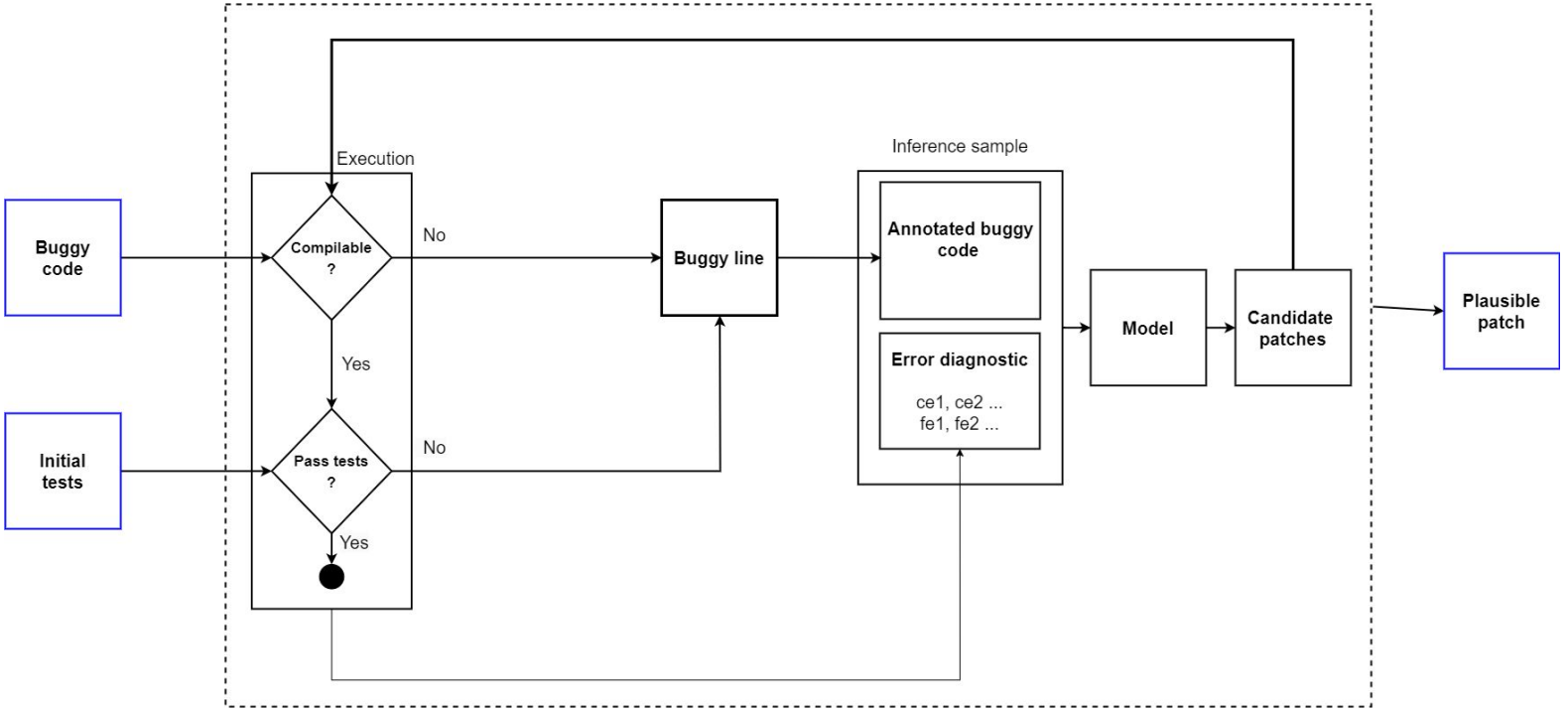
G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," arXiv [cs.CL], 2020.

Methodology

- Implementation of inference phase
- Dataset Creation
- Model development
- Implementation of test generation model

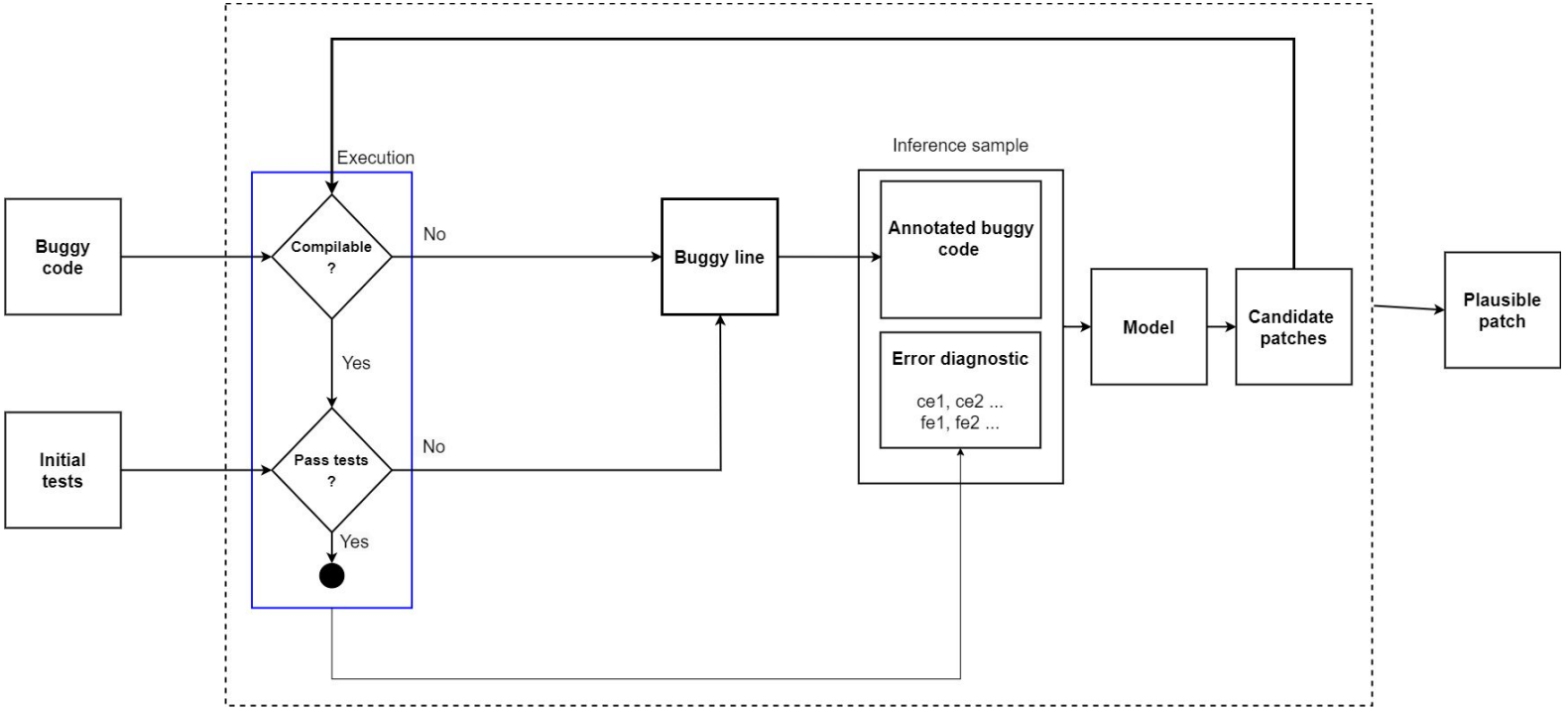
Implementation of inference phase

Inference Phase



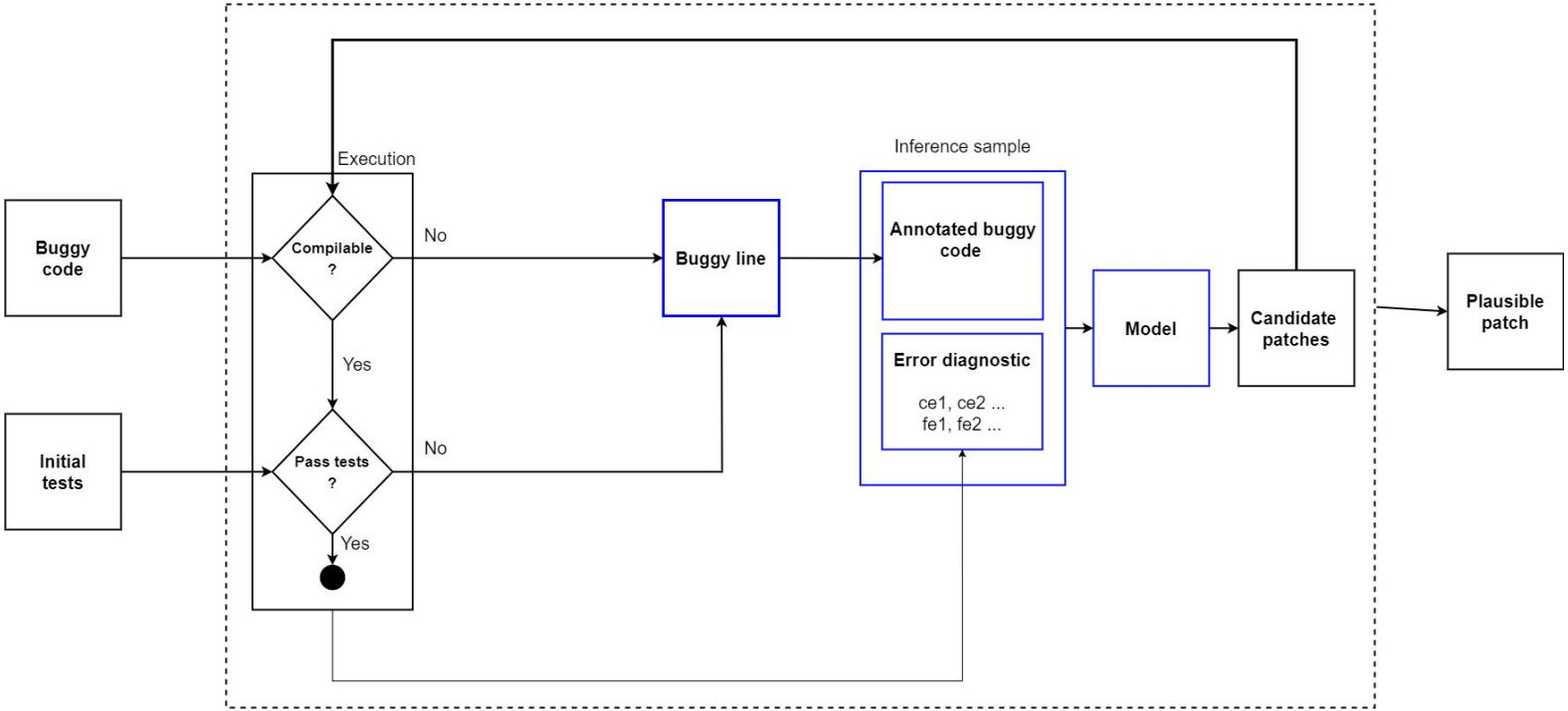
ce-compilation error, fe-functional error

Inference Phase



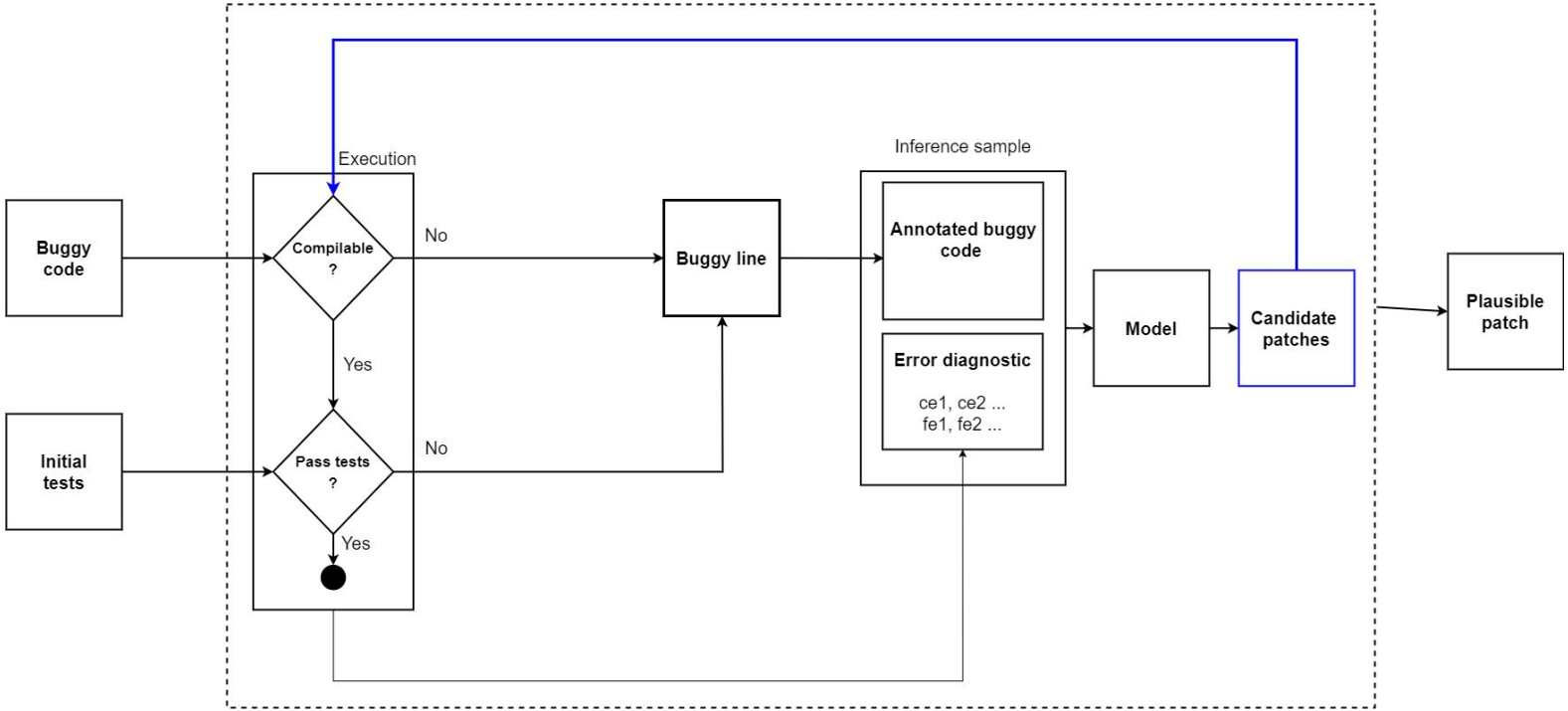
ce-compilation error, fe-functional error

Inference Phase



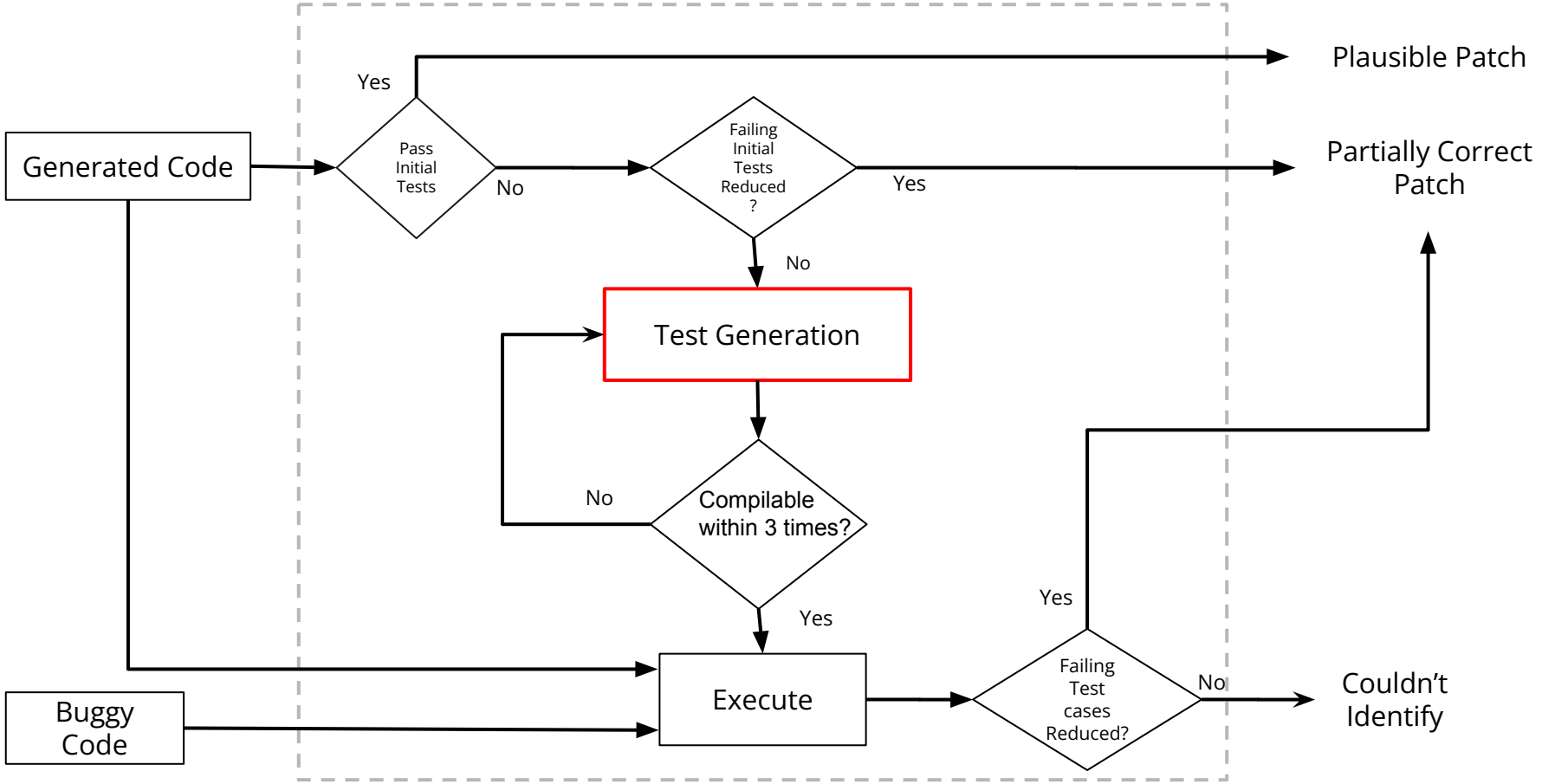
ce-compilation error, fe-functional error

Inference Phase



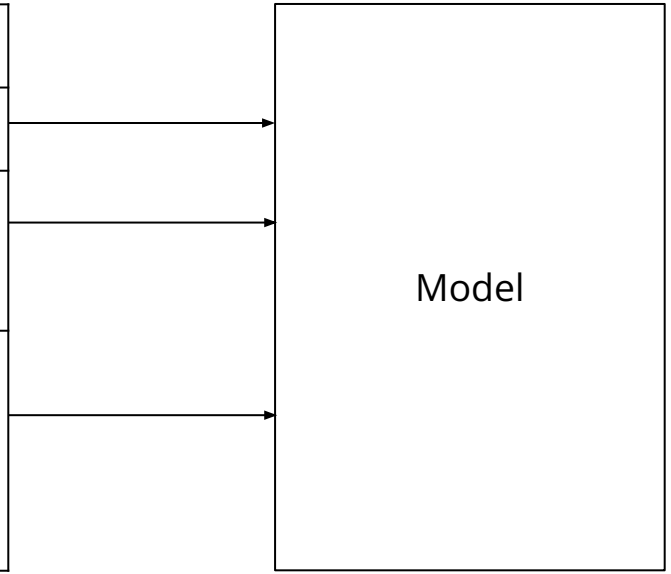
ce-compilation error, fe-functional error

Patch Correctness Identification

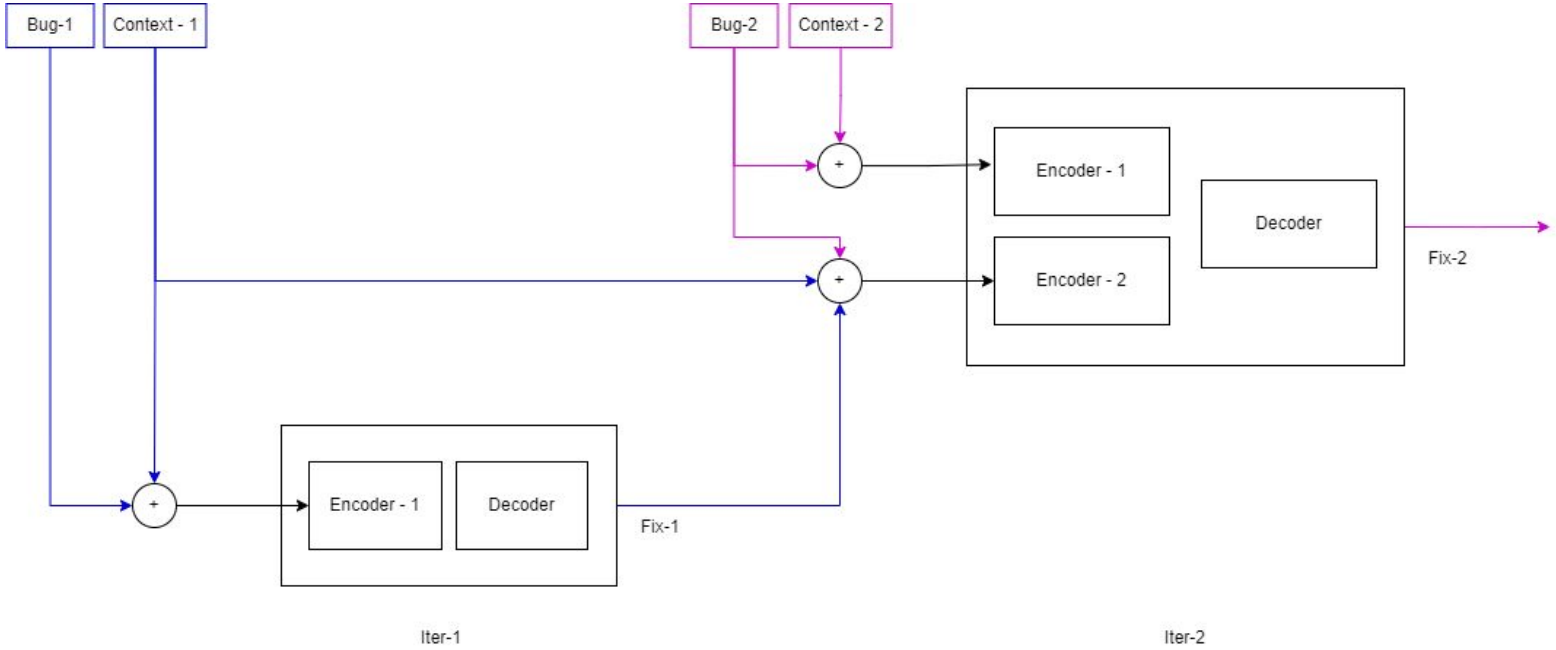


Capture Semantic Dependencies Between Generated Patches

Iteration	Annotated buggy code
1	Buggy Line 01 + context 01
2	Patch for Line 01 + context 01 Buggy Line 02 + context 02
3	Patch for Line 01 + context 01 Patch for Line 02 + context 02 Buggy Line 03 + context 03



Context Improvement in FiD



Transformer based model - Architecture of Fusion in decoder method

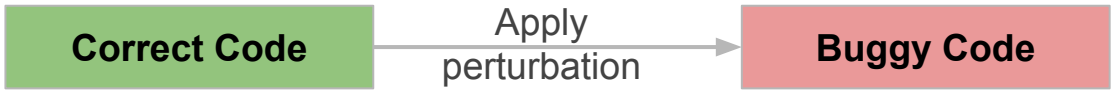
Dataset Creation

Dataset Creation

- Existing datasets
  - SelfAPR
  - CPatMiner
  - BigFix
  - Defects4j
- Why dataset creation?
  - Missing Error diagnostic
  - Need multi line train samples
- What is perturbation?
  - Generate buggy code from correct code

Multi Line Train Sample

Fixed line 01 + context 01
Fixed line 02 + context 02
Bug line 03 + context 03



Dataset Creation - Repo Selection

- Select java repositories with test cases
- Selected java repositories

Project	Repository	Created data points
Guava	https://github.com/google/guava	47000+
Time4J	https://github.com/MenoData/Time4J	~2000
Htmlunit	https://github.com/HtmlUnit/htmlunit	~50000
Maven-doxia	https://github.com/apache/maven-doxia.git	~400
Wro4j	https://github.com/wro4j/wro4j	20000+
Super Csv	https://github.com/super-csv/super-csv	~6000
finmath-lib	https://github.com/finmath/finmath-lib.git	210000+
	Total	760000+

Dataset Creation (Cont.)

- Applied perturbation rules - 16
  - Swap arguments

Correct Code

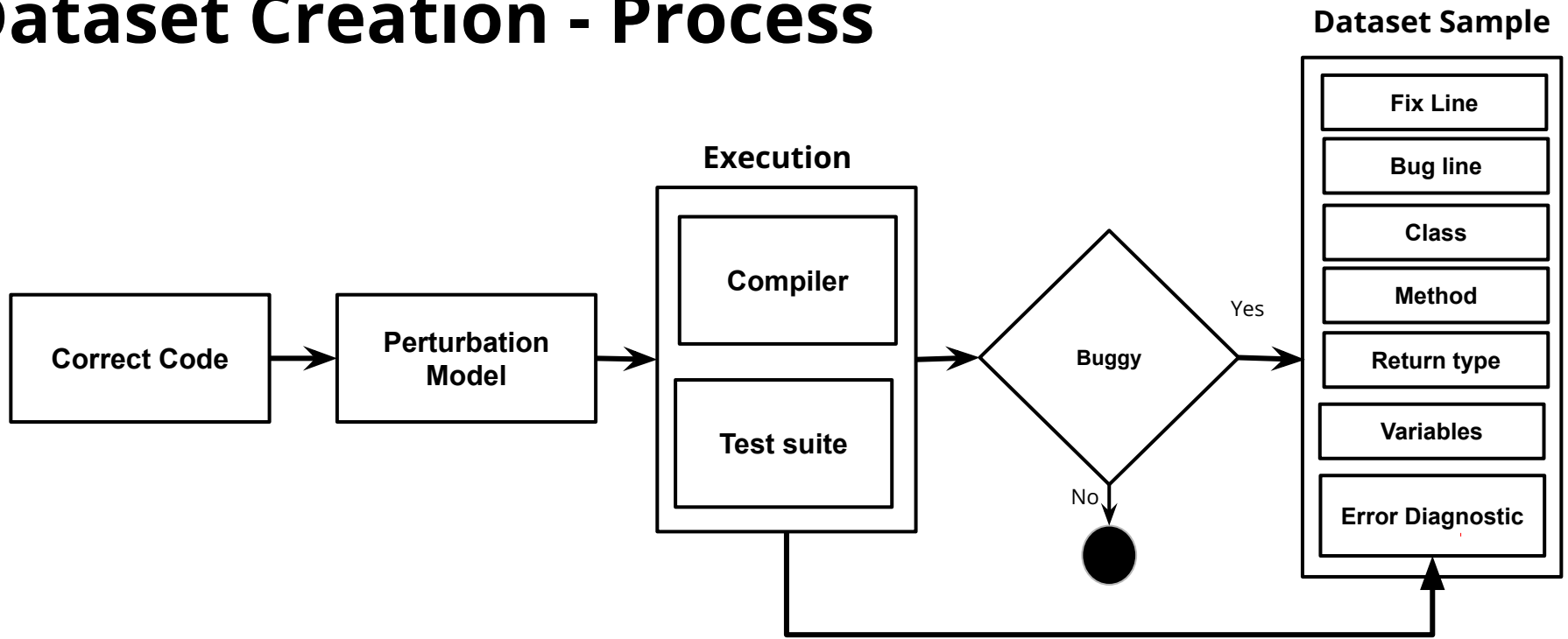
```
public void setInfo(int age, String name){
     this.age = age;
     this.name = name;
}
```

Buggy Code

```
public void setInfo(String name, int age){
     this.age = age;
     this.name = name;
}
```

- Create dataset by extracting relevant details according to the buggy line
  - Error diagnostic
  - Class
  - Method
  - Return type
  - Variables

Dataset Creation - Process



Model Development

Model Selection - Why code based LLMs?

- Most of the previous researches have used text based LLMs.
- Text based LLMs doesn't specifically support code based tokens.
- Generated identical patches to Defects4J bugs

Model	Number of return sequences							
	2		5		10		50	
	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line
T5-small	0	0	0	0	0	0	0	0
CodeT5-Small	2	1	3	1	6	1	7	1

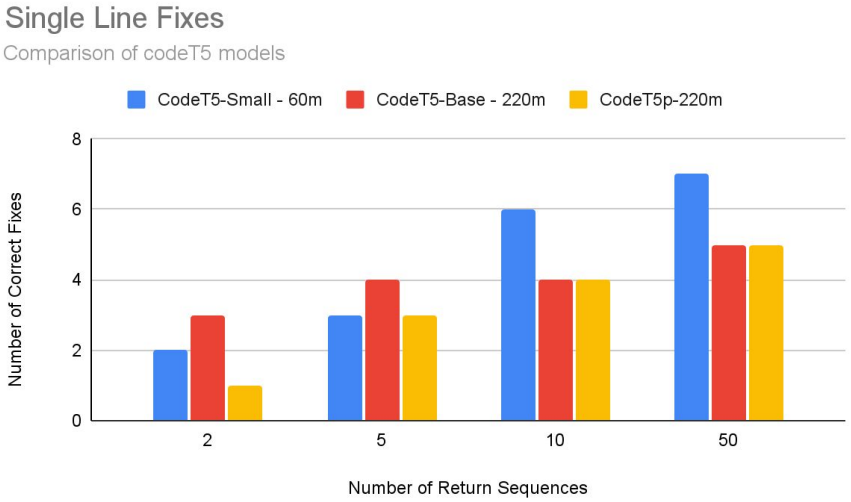
Model Selection - Why CodeT5?

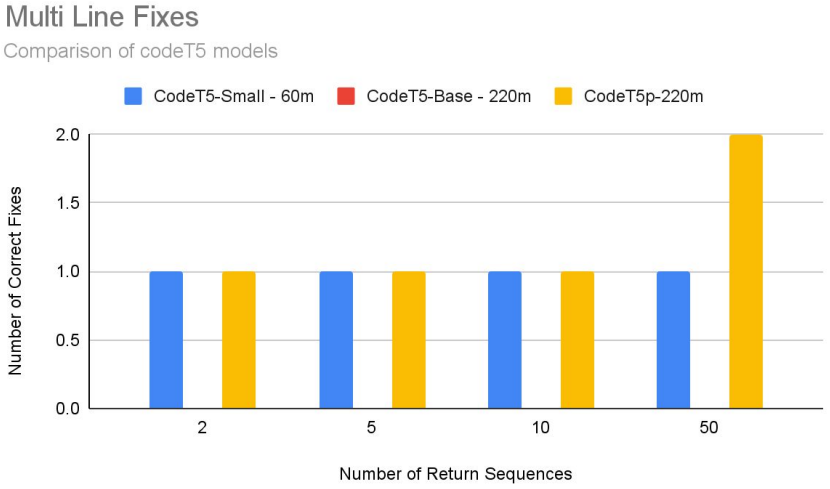
Model	CodeT5[1]	CodeLlama[2]	SantaCoder[3]	StarCoder[4]
Minimum learnable parameters	60M (CodeT5-small)	7B	1.1B	15.5B
Model Architecture	Encoder-Decoder	Encoder-decoder	Decoder-only	Decoder-only

[1]W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, “Detecting code clones with graph neural network and flow-augmented abstract syntax tree,” in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020.
[2]B. Rozière *et al.*, “Code Llama: Open foundation models for code,” *ArXiv*, vol. abs/2308.12950, 2023
[3]L. B. Allal *et al.*, “SantaCoder: don’t reach for the stars!,” *arXiv [cs.SE]*, 2023.
[4]R. Li *et al.*, “StarCoder: may the source be with you!,” *ArXiv*, vol. abs/2305.06161, 2023.

Model Selection - CodeT5 models comparison

- Used input prompt,
  - … **<extra_id_0>**...
- Context contains five lines above and below the buggy line
- Generated identical patches to Defects4J bugs





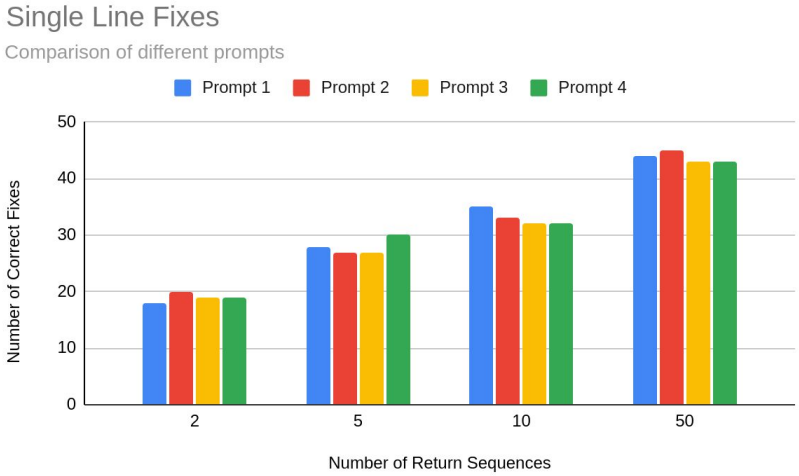
Prompt Selection

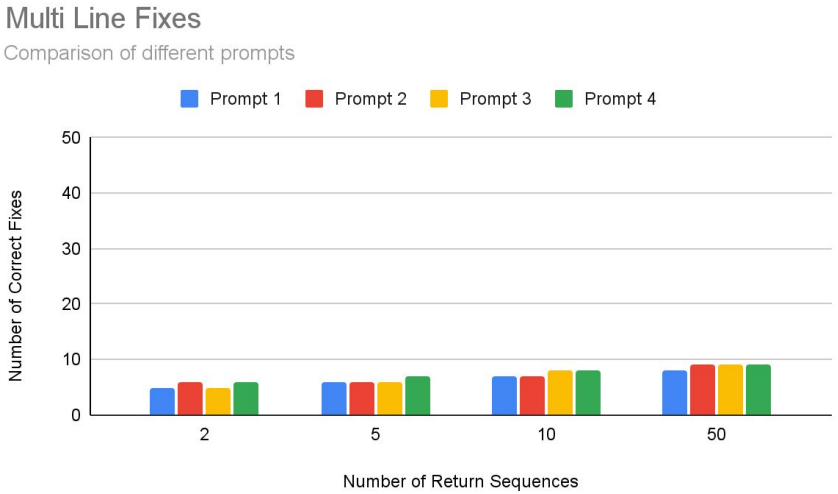
- Evaluated improvement of results with the change of prompt.
- Same 10k training samples used in model fine tuning process
- Evaluated prompts

Prompt 1	Prompt 2	Prompt 3	Prompt 4
[BUG]... [CONTEXT]...	[BUG]... **[ERROR]**... [CONTEXT]...	[BUG]... [ERROR]... [CONTEXT]... **[VARIABLES]**...	[BUG]... [ERROR]... [CONTEXT]... **[CLASS]...** **[METHOD]...** **[RETURN_TYPE]**... [VARIABLES]...

Prompt Selection - Results

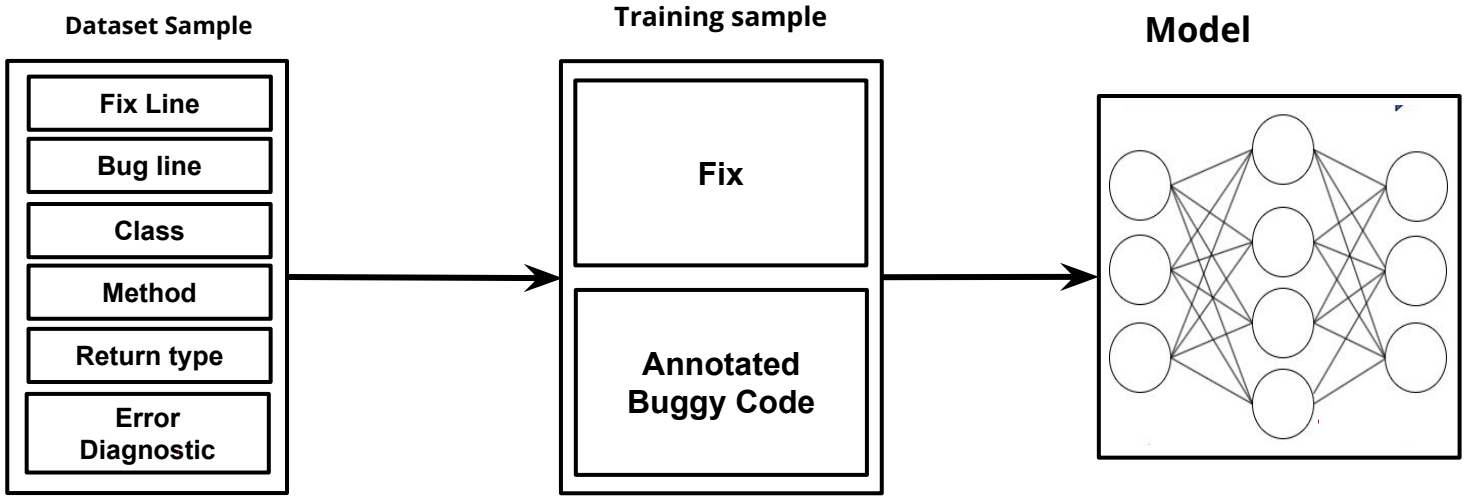
- Generated identical patches to Defects4J bugs



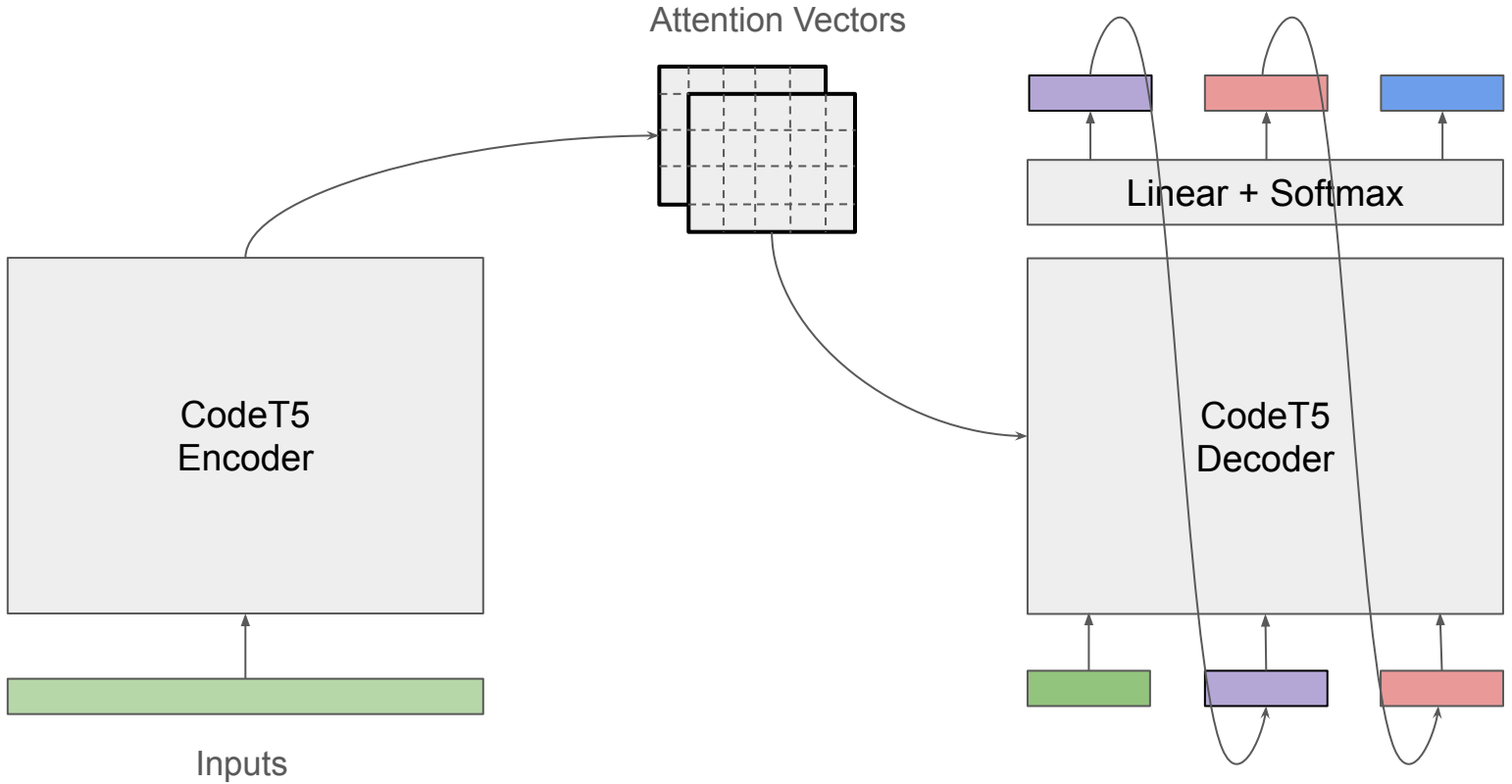


Training Phase

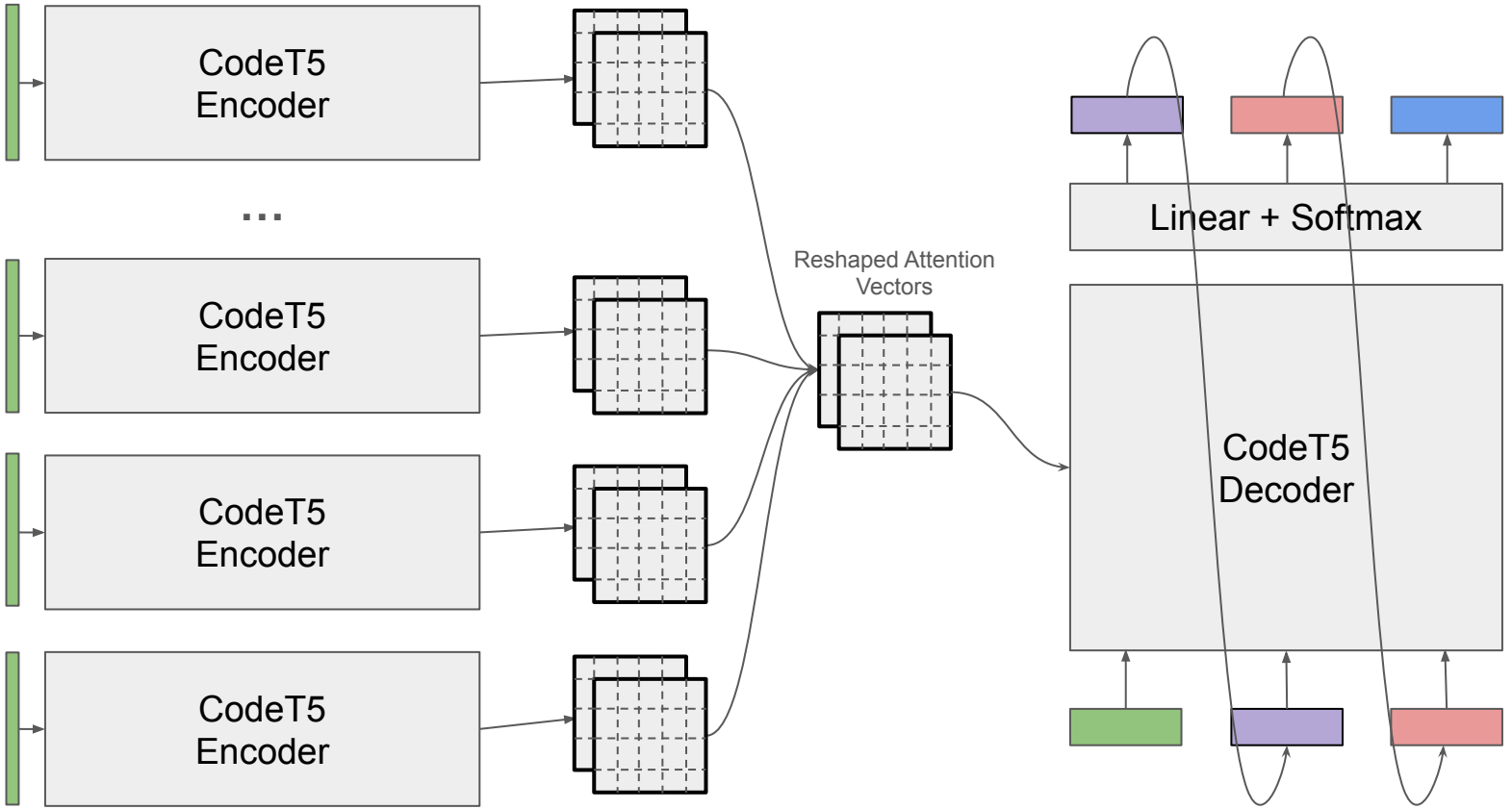
- There are two steps in training phase,
  - Fine tuning CodeT5-small model
  - Training FiD model



CodeT5 model Architecture



FiD Architecture



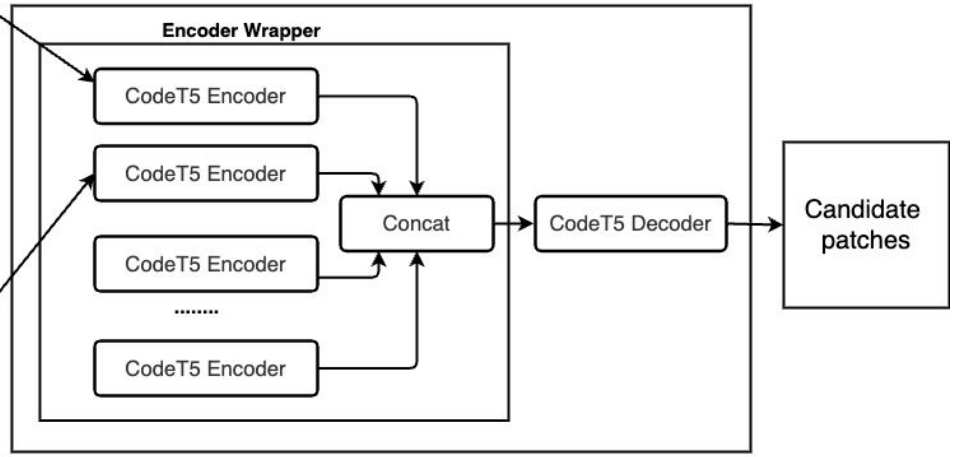
FiD Model Training

- Used fine tuned CodeT5 as base model.
- Trained FiD model

```
[BUG]
int partialEntropy = primeF(x_value,beta_coefficient,beta_coefficient);
[CONTEXT]
public double computeTotalEnergey(){
    double x_value = 3.0;
    double alpha_coefficient = 2.0;
    double beta_coefficient = 1.5;
    int gamma_coefficient = -3;

    <extra_id_0>
    return partialEntropy*gamma_coefficient;
}
[ERROR]
java: incompatible types: possible lossy conversion from double to int
```

```
[BUG]
int partialEntropy = primeF(x_value,beta_coefficient,beta_coefficient);
[CONTEXT]
public double primeF(double initial, double alpha, double beta) {
    double h = 0.01;
    double y = initial;
    for (int i = 0; i < 100; i++) {
        double y_prime = alpha * y + beta;
        y = y + h * y_prime;
    }
    return y;
}
[ERROR]
java: incompatible types: possible lossy conversion from double to int
```



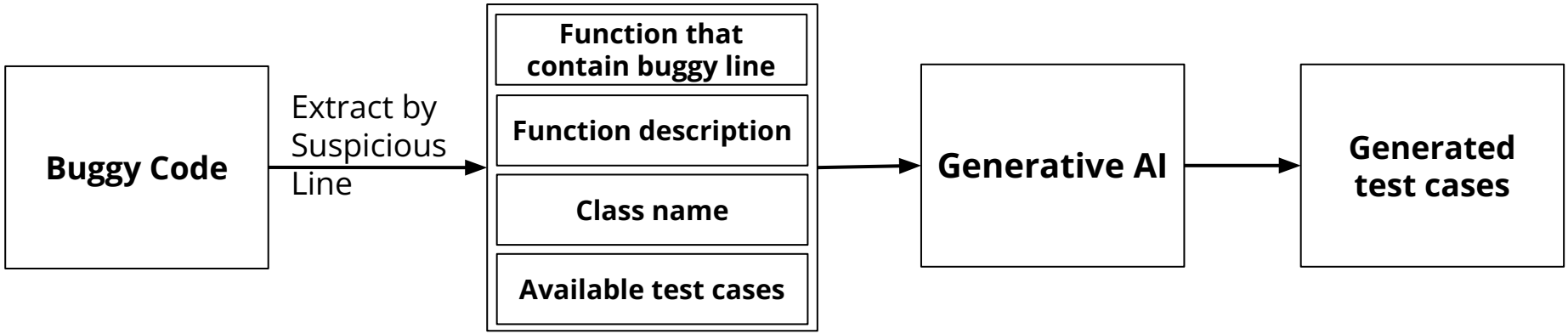
Test Generation

Test Case Generation - Why?

- Importance of partial patch correctness

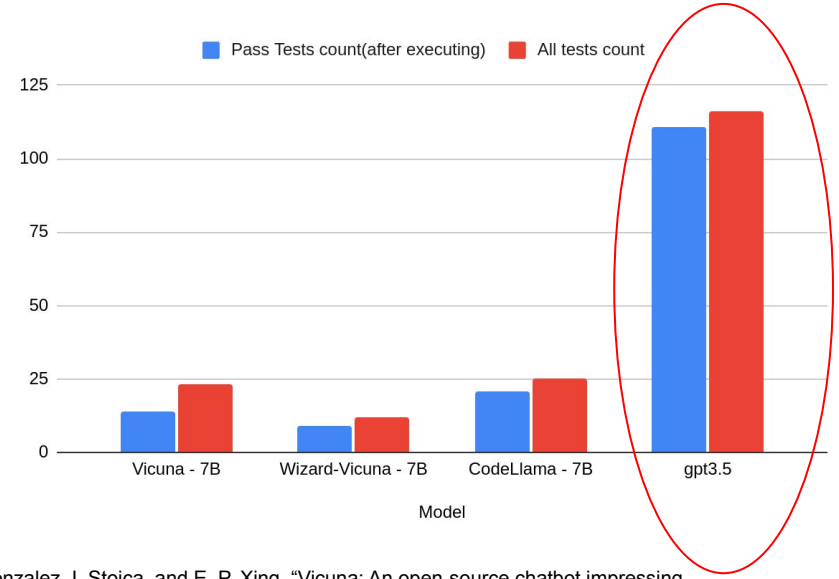
- Available tools identify partial patch correctness using existing test cases
- Limitation - Not enough test cases
- Importance of test generation
  - Early identification of partial patch correctness
  - Reduce computation time

Test Generation Process



Test Generation - Model Evaluation

- Compared Models
  - Vicuna 7B [1]
  - Wizard-Vicuna-7B-Uncensored-GPTQ
  - CodeLlama-7b-hf [2]
  - GPT-3.5-turbo-1106 [3]
- Generate test cases for 13 different prompts and for each prompt generate test cases for 3 times.



[1] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, “Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality,” March 2023
[2] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G.Synnaeve, “Code llama: Open foundation models for code,” 2024
[3]T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter,C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,”

Test Generation Implementation

- Relevant detail extraction
  - Extract using Java parser
  - Buggy class name, method, method description and test cases which use that buggy function
- Prompt creation

```
public class {{class_name}} {
        {{method}}
}
{{sample_test_case}}
Only give the completed below test case code.
public class {{test_class_name}} {
@Test
public void generated_test_case_{{test_case_number}}(
```

Prompt template

Test Generation Implementation (cont.)

- Test Case Generation

```
Need to generate test cases for a given function description.
public class Calculator{
    /**
    * Calculates the modulus (remainder) of dividing one number by another.
    * @param a The dividend (the number to be divided).
    * @param b The divisor (the number by which the dividend is divided).
    * @return The remainder when dividing the dividend by the divisor.
    * @throws ArithmeticException If the divisor is zero, as division by zero is undefined.
    */
    public static double modulus(double a, double b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot calculate modulus with zero divisor.");
        }
        return a % b;
    }
}

sample test case:
@Test
void testModulus() {
    assertEquals(1.0, Calculator.modulus(5.0, 2.0));
    assertEquals(0.0, Calculator.modulus(10.0, 5.0));
    assertThrows(ArithmeticException.class, () -> Calculator.modulus(8.0, 0.0));
}

Only give the completed below test case code.
public class CalculatorTest {
    @Test
    public void generated_test_case_2158(
```

Test Generation Implementation (cont.)

- Test case generation result

```
public class CalculatorTest {
    @Test
    public void generated_test_case_2158() {
        assertEquals(1.0, Calculator.modulus(5.0, 2.0));
        assertEquals(0.0, Calculator.modulus(10.0, 5.0));
        assertThrows(ArithmeticException.class, () -> Calculator.modulus(8.0, 0.0));
        assertEquals(2.0, Calculator.modulus(8.0, 3.0));
        assertEquals(0.0, Calculator.modulus(0.0, 7.0));
        assertThrows(ArithmeticException.class, () -> Calculator.modulus(9.0, 0.0));
        assertEquals(3.5, Calculator.modulus(7.0, 2.0));
        assertEquals(-3.0, Calculator.modulus(-10.0, 7.0));
        assertThrows(ArithmeticException.class, () -> Calculator.modulus(Double.POSITIVE_INFINITY, 2.0));
    }
}
```

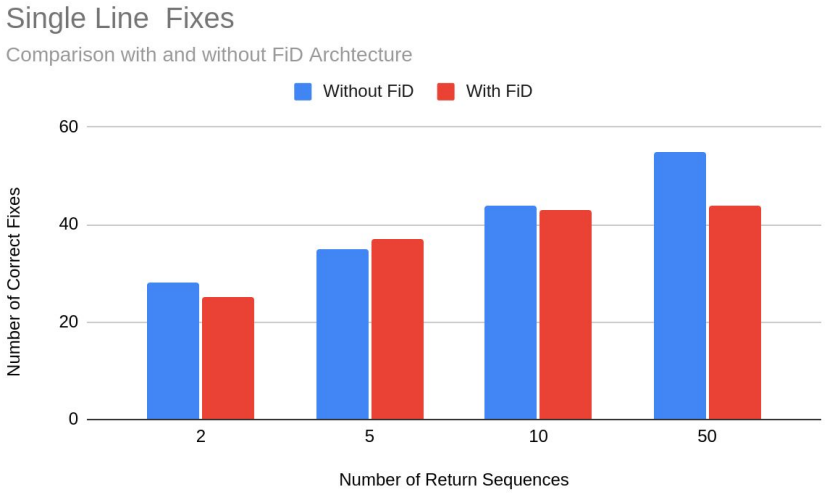
- Test Case Injection
  - Used a java parser
  - If extracted test cases available, inject new test cases into the same test file
  - Otherwise, create new test file with all import statements and inject test cases

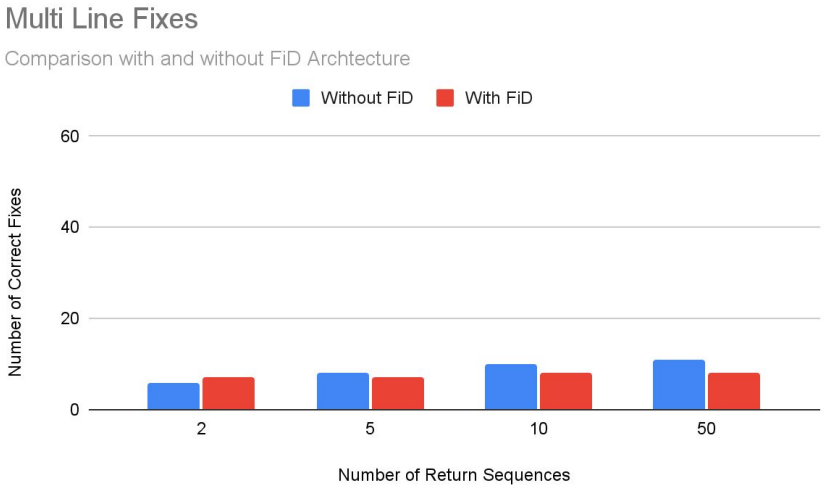
Evaluations

Evaluations

- All evaluation conducted on,
  - Defects4J dataset (835 samples)
  - Identical patches
  - Used prompt : Prompt 4
  - Evaluate for one Iteration only
- Our tool without FiD and with FiD model Evaluation

[BUG]...
[ERROR]...
[CONTEXT]...
[CLASS]...
[METHOD]...
[RETURN_TYPE]...
[VARIABLES]...





Evaluations (cont.)

- Comparison with available apr tools
- Generated identical patches to Defects4J bugs

Model	DEAR	VarFix (Math and Closure only)	Hercules	ITER (Iter. 1) T5-Base 220m	ITER (Iter. 2)	ITER (Iter. 3)	Our Tool (iterations 1) codeT5-small 60m	
							Without FiD	With FiD
Single Line	37	12	37	-	-	61	52	44
Multi Line	16	5	12	-	-	15	11	8
All Fixes	**53**	**17**	**49**	**41**	**67**	**76**	**63**	**52**

Evaluations (cont.)

- Comparison with available apr tools
- Generated identical patches to Defects4J bugs

Model	DEAR	VarFix (Math and Closure only)	Hercules	ITER (Iter. 1) T5-Base-220m	ITER (Iter. 2)	ITER (Iter. 3)	Our Tool (iterations 1) codeT5-small 60m	
							Without FiD	With FiD
Single Line	37	12	37	-	-	61	52	44
Multi Line	16	5	12	-	-	15	11	8
All Fixes	**53**	**17**	**49**	**41**	**67**	**76**	**63**	**52**

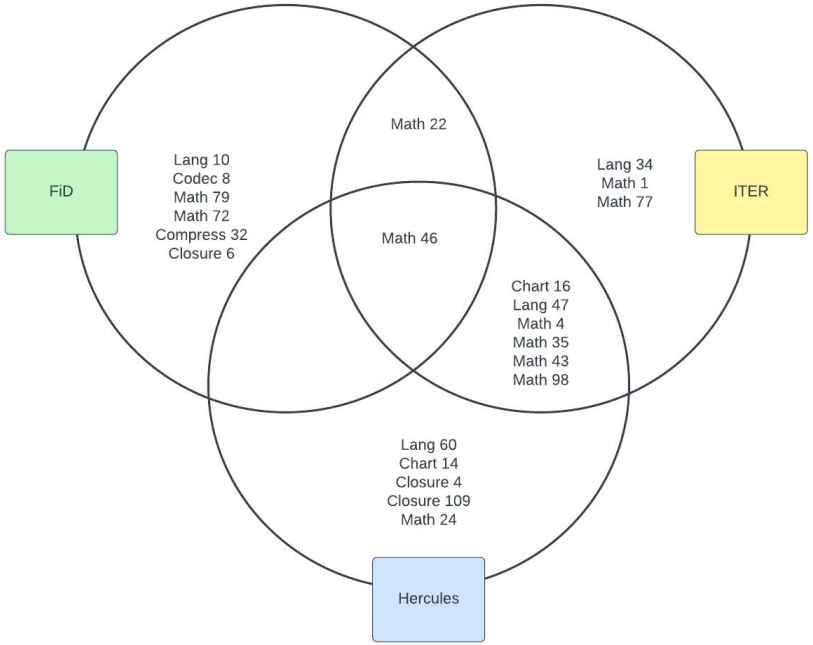
Evaluations (cont.)

- Comparison with available apr tools
- Generated identical patches to Defects4J bugs

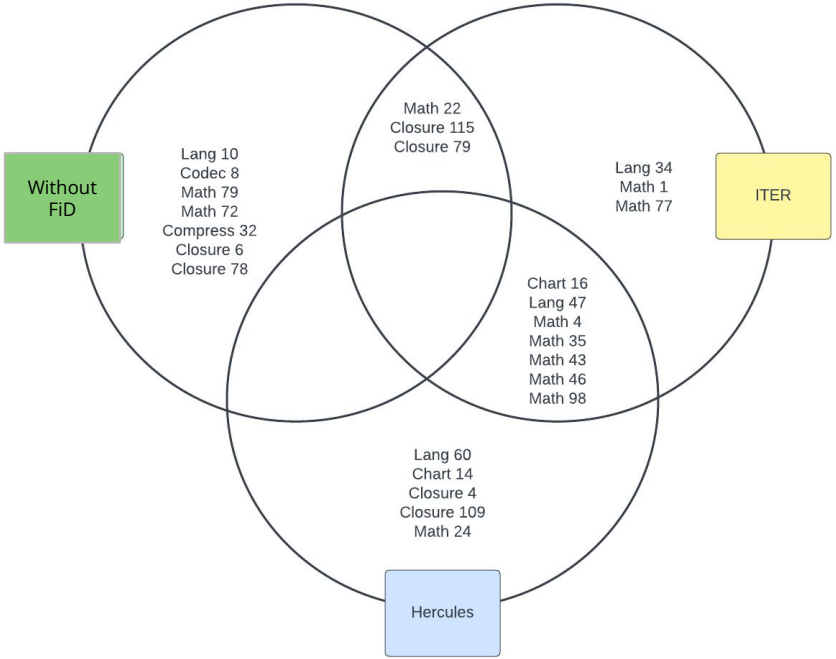
Model	DEAR	VarFix (Math and Closure only)	Hercules	ITER (Iter. 1) T5-Base-220m	ITER (Iter. 2)	ITER (Iter. 3)	Our Tool (iterations 1) codeT5-small 60m	
							Without FiD	With FiD
Single Line	37	12	37	-	-	61	52	44
Multi Line	16	5	12	-	-	15	11	8
All Fixes	**53**	**17**	**49**	**41**	**67**	**76**	**63**	**52**

Evaluations (cont.) - Multi Line fixes comparison

- FiD with ITER and Hercules



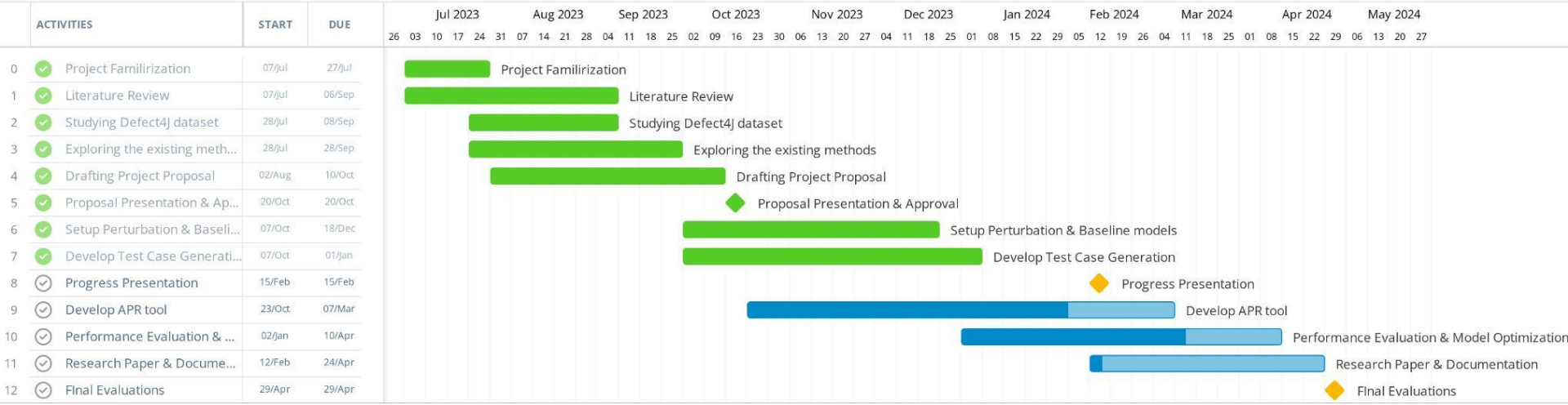
- Without FiD with ITER and Hercules



What we have done?

- Creating datasets
- Complete implementation of,
  - Training phase
  - Test generation
  - Applying FiD architecture on CodeT5
  - Inference phase
- Evaluations for,
  - Selecting proper Generative AI technique.
  - Selecting proper CodeT5 model as the baseline model.
  - Training CodeT5 models on different formats to identify the better prompt format.
  - Our tool with and without FiD

Timeline


ACTIVITIES
START
DUE
Jul 2023
Aug 2023
Sep 2023
Oct 2023
Nov 2023
Dec 2023
Jan 2024
Feb 2024
Mar 2024
Apr 2024
May 2024
0 Project Familirization 07/Jul 27/Jul
1 Literature Review 07/Jul 06/Sep
2 Studying Defect4J dataset 28/Jul 08/Sep
3 Exploring the existing meth... 28/Jul 28/Sep
4 Drafting Project Proposal 02/Aug 10/Oct
5 Proposal Presentation & Ap... 20/Oct 20/Oct
6 Setup Perturbation & Baseli... 07/Oct 18/Dec
7 Develop Test Case Generati... 07/Oct 01/Jan
8 Progress Presentation 15/Feb 15/Feb
9 Develop APR tool 23/Oct 07/Mar
10 Performance Evaluation & ... 02/Jan 10/Apr
11 Research Paper & Docume... 12/Feb 24/Apr
12 Final Evaluations 29/Apr 29/Apr
Project Familirization
Literature Review
Studying Defect4J dataset
Exploring the existing methods
Drafting Project Proposal
Proposal Presentation & Approval
Setup Perturbation & Baseline models
Develop Test Case Generation
Progress Presentation
Develop APR tool
Performance Evaluation & Model Optimization
Research Paper & Documentation
Final Evaluations

Future Work

- Created an improved dataset.
- Improve performance of the FiD model.
- Evaluate for plausible patches
- Evaluate the impact of test generation and without test generation.
- Complete research paper.
- Complete VS Code extension

Thank you

References

H. Ye and M. Monperrus, “ITER: Iterative neural repair for multi-location patches,” arXiv [cs.SE], 2023.

Berabi, B., He, J., Raychev, V. & Vechev, M.. (2021). TFix: Learning to Fix Coding Errors with a Text-to-Text Transformer. Proceedings of the 38th International Conference on Machine Learning, in Proceedings of Machine Learning Research139:780-791 Available from https://proceedings.mlr.press/v139/berabi21a.html.

Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. CURE: Code-Aware Neural Machine Translation for Automatic Program Repair. In Proceedings of the 43rd International Conference on Software Engineering (ICSE '21). IEEE Press, 1161–1173. https://doi.org/10.1109/ICSE43902.2021.00107

W. Yuan et al., “CIRCLE: continual repair across programming languages,” in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022.

J. Kim and B. Lee, “MCRepair: Multi-chunk program repair via patch optimization with buggy block,” in Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 2023.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Fu, Michael & Tantithamthavorn, Chakkrit & Le, Trung & Le, Trung & Nguyen, Van & Phung, Dinh. (2022). VulRepair: A T5-Based Automated Software Vulnerability Repair. 10.1145/3540250.3549098.

Z. Chen, S. Kommrusch, M. Tufano, L.-N. Pouchet, D. Poshyvanyk, and M. Monperrus, “SequenceR: Sequence-to-sequence learning for end-to-end program repair,” arXiv [cs.SE], 2018.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. DeepFix: fixing common C language errors by deep learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17). AAAI Press, 1345–1351.

X. Li, W. Li, Y. Zhang, and L. Zhang, “DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization,” in Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2019.

Hajipour, H., Bhattacharyya, A., Staicu, C.A. and Fritz, M., 2021, September. Samplefix: learning to generate functionally diverse fixes. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 119-133). Cham: Springer International Publishing.

Lutellier, T., Pham, H. V., Pang, L., Li, Y., Wei, M., & Tan, L. (2020, July 18). CoCoNuT: combining context-aware neural translation models using ensemble for program repair. Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. https://doi.org/10.1145/3395363.3397369

Lutellier, Thibaud & Pang, Lawrence & Pham, Viet & Wei, Moshi & Tan, Lin. (2019). ENCORE: Ensemble Learning using Convolution Neural Machine Translation for Automatic Program Repair.

Y. Li, S. Wang, and T. N. Nguyen, “DEAR: A novel deep learning-based approach for automated program repair,” in Proceedings of the 44th International Conference on Software Engineering, 2022.

Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. DLFix: context-based codetransformation learning for automated program repair. In ICSE ’20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June 19 July, 2020, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 602–614. https://doi.org/10.1145/3377811.3380345

Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang.2020. Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. https://openreview.net/forum?id=SJeqs6EFvB

Wang, S., Liu, K., Lin, B., Li, L., Klein, J., Mao, X. and Bissyandé, T.F., 2021. Beep: Fine-grained fix localization by learning to predict buggy code elements. arXiv preprint arXiv:2111.07739.

Chakraborty, S., Ding, Y., Allamanis, M. and Ray, B., 2020. Codit: Code editing with tree-based neural models. IEEE Transactions on Software Engineering, 48(4), pp.1385-1399.

Ali Mesbah, Andrew Rice, Emily Johnston, Nick Glorioso, and Edward Aftandilian. Deepdelta: learning to repair compilation errors. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 925–936, 2019 (https://dl.acm.org/doi/pdf/10.1145/3338906.3340455)

N. Nashid, M. Sintaha and A. Mesbah, "Embedding Context as Code Dependencies for Neural Program Repair," 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), Dublin, Ireland, 2023, pp. 95-106, doi: 10.1109/ICST57152.2023.00018.

ChatGPT

- Common bug solved by almost every model we trained.
- Bug line - if (dataset != null) {
- Fix line - if (dataset == null) {

You

consider the below buggy code segment extracted from a java file. There is a tag indicated with [BUG]. The code within the 2 [BUG] tags is buggy. generate a fix for that part. once it is pasted between 2 [BUG] tags and removed [BUG] tags, the code must work without any errors. provide only the part to be pasted between 2 [BUG] tags

```
LegendItemCollection result = new LegendItemCollection();
  if (this.plot == null) {
  return result;
}
int index = this.plot.getIndexOf(this);
CategoryDataset dataset = this.plot.getDataset(index);
[BUG] if (dataset != null) { [BUG]
  return result;
}
int seriesCount = dataset.getRowCount();
if (plot.getRowRenderingOrder().equals(SortOrder.ASCENDING)) {
  for (int i = 0; i < seriesCount; i++) {
    if (isSeriesVisibleInLegend(i)) {
```

ChatGPT

java — Copy code

```java
if (dataset == null) { // [BUG]
    return result;
} // [BUG]
```

Literature Review - Datasets

Dataset Name	Description
Defects4J	Benchmark dataset. 357 real bugs from 5 open sources projects. Newer version having 835 bugs.
BigFix	26k+ bugs Real world bugs from 8 projects.
CPatMiner	44k+ bugs
SelfAPR	Perturbed 17 repositories. 800k+ bugs.

Hercules: Harnessing evolution for multi-hunk program repair

- Uses a template based approach.
- Focus on generating patches that require edits in multiple locations.
- Generate patches for similar looking code(evolutionary siblings) that is in similar contexts and needed similar changes.
- Generalize single line patch generation to bugs that may require similar patches at number of locations.
- Limited to this kind of bugs.

S. Saha, R. k. Saha, and M. r. Prasad, “Harnessing evolution for multi-hunk program repair,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019.

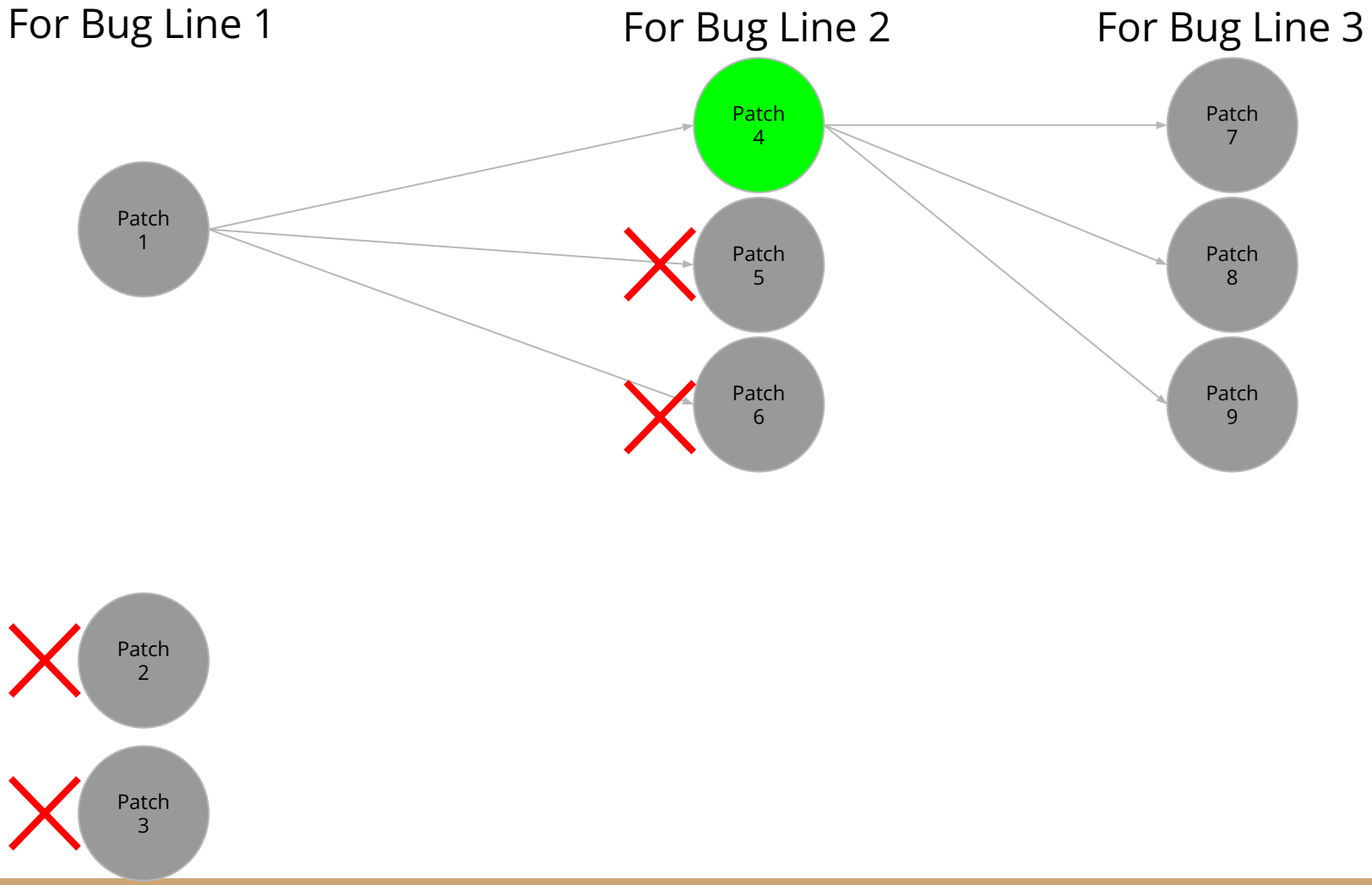
DEAR: a novel deep learning-based approach for automated program repair

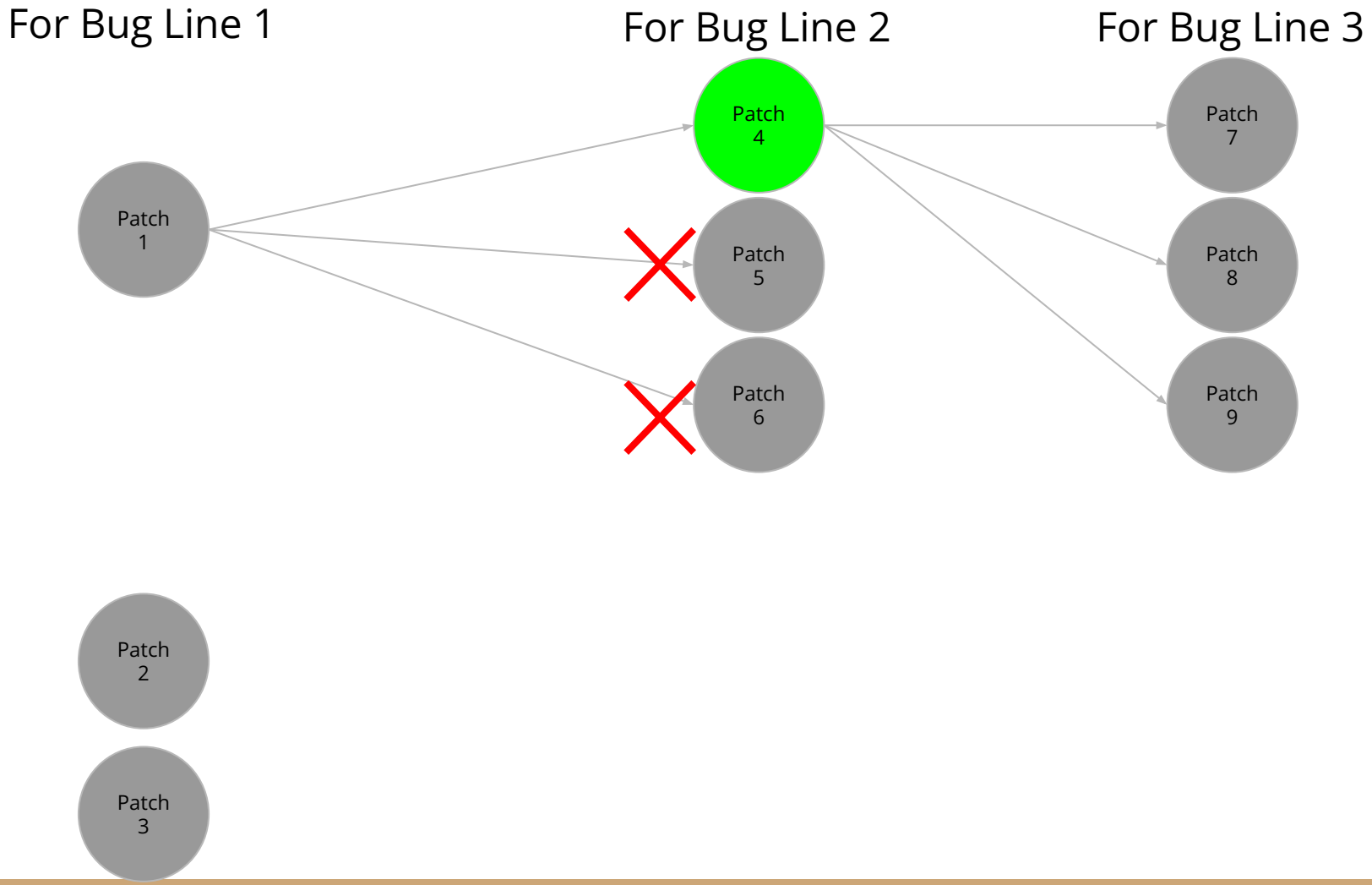
- Focus on generating multi line patches.
- Code is represented in a AST and buggy subtrees are marked.
- One model is used to capture context of buggy subtrees.
- Another model is used to transform buggy subtrees into fixed subtrees.
- By fixing many buggy subtrees, DEAR can generate multi line patches.
- Not focused on improving partially corrected patches.

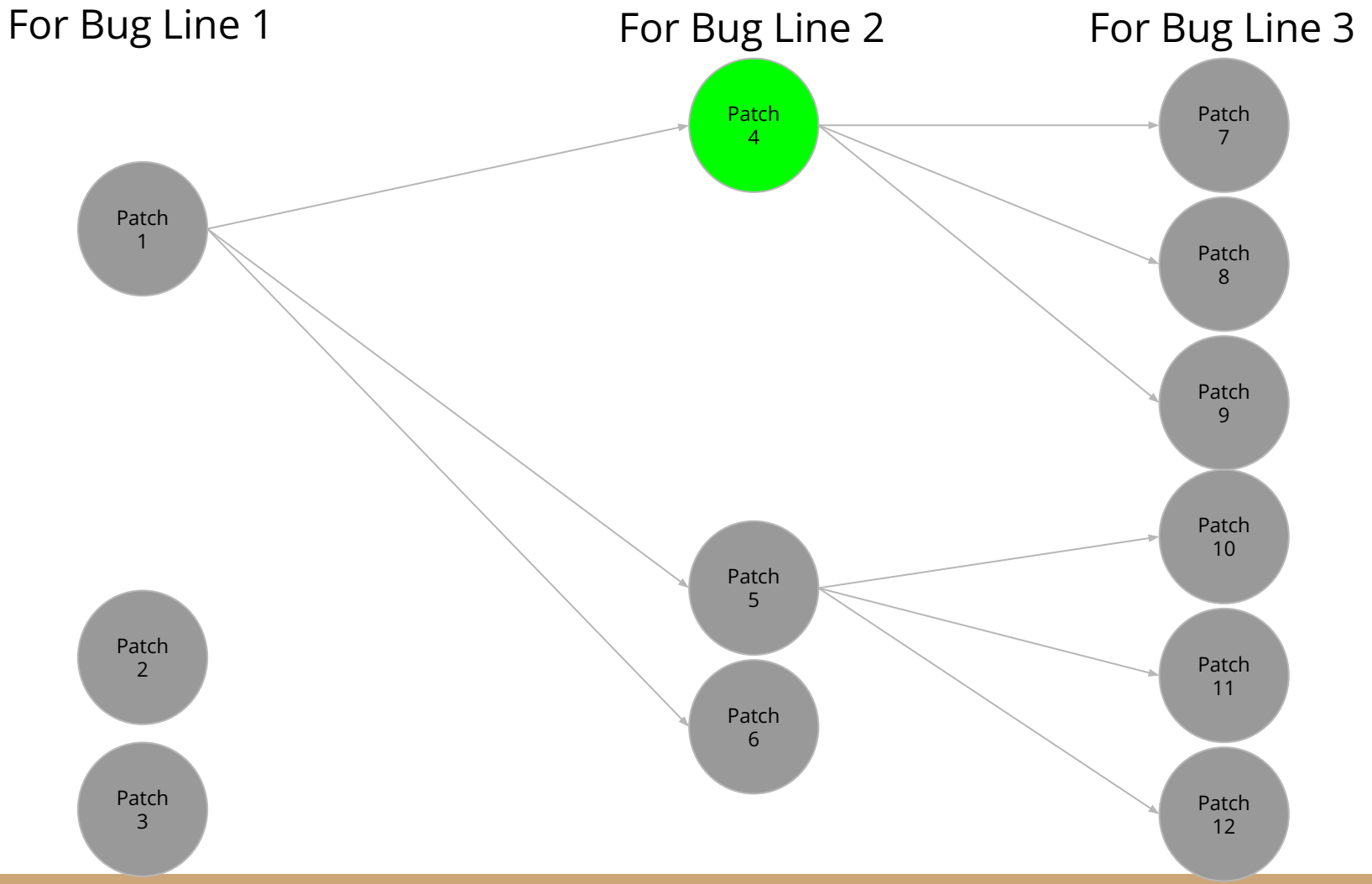
Y. Li, S. Wang, and T. N. Nguyen, “DEAR: A novel deep learning-based approach for automated program repair,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022.

Continuation After Partial Patch Identification

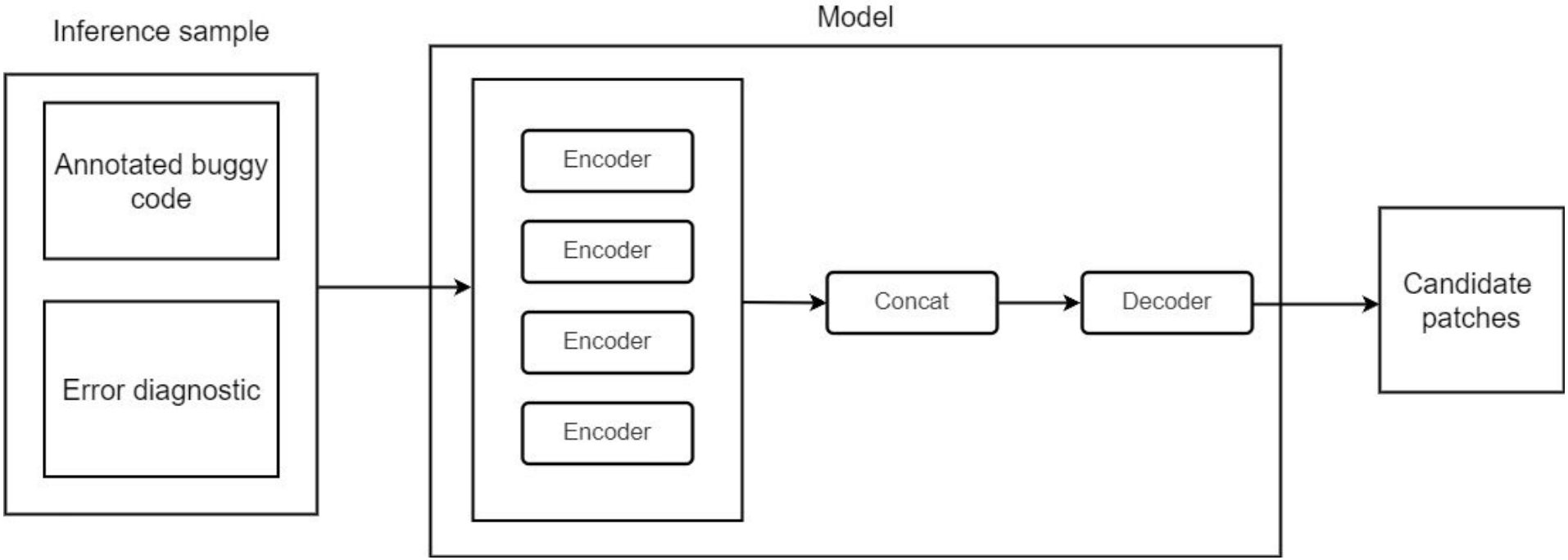
- If partially correct, ways to continue
  - Method 1 - Continue with that patch without ignoring others
  - Method 2 - Ignore patches in same level
  - Method 3 - Ignore all previously generated patches
- Otherwise always continue


For Bug Line 1
For Bug Line 2
For Bug Line 3
Patch 1
Patch 2
Patch 3
Patch 4
Patch 5
Patch 6
Patch 7
Patch 8
Patch 9


For Bug Line 1
For Bug Line 2
For Bug Line 3
Patch 1
Patch 2
Patch 3
Patch 4
Patch 5
Patch 6
Patch 7
Patch 8
Patch 9


For Bug Line 1
For Bug Line 2
For Bug Line 3
Patch 1
Patch 2
Patch 3
Patch 4
Patch 5
Patch 6
Patch 7
Patch 8
Patch 9
Patch 10
Patch 11
Patch 12

FiD Architecture



Transformer based model - Architecture of Fusion in decoder method

Model Selection - CodeT5 models comparison

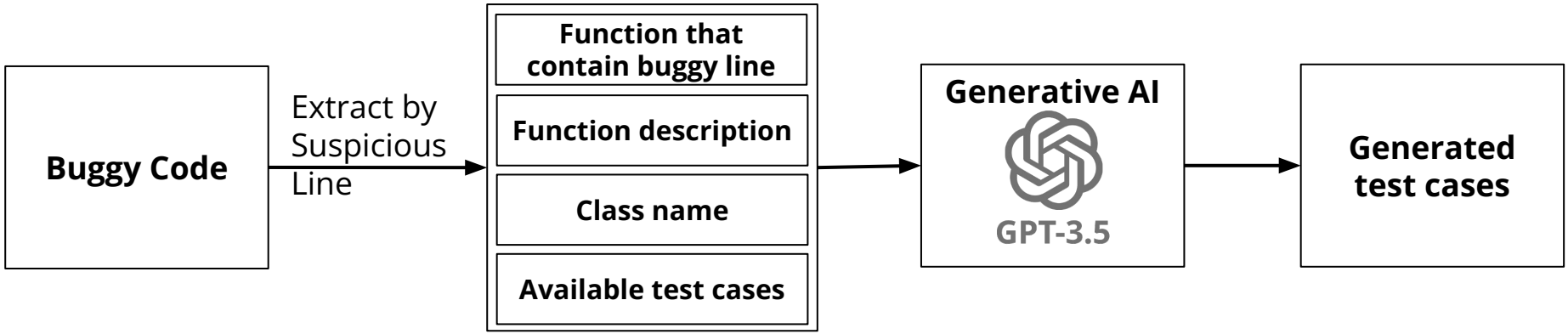
- Used input prompt,
  - … **<extra_id_0>**...
- Context contains five lines above and below the buggy line

Model	Number of return sequences							
	2		5		10		50	
	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line
CodeT5-Base	3	0	4	0	4	0	5	0
CodeT5p-220m	1	1	3	1	4	1	5	2
CodeT5-Small	2	1	3	1	6	1	7	1

Prompt Selection - Results

Prompt Number	Number of return sequences							
	2		5		10		50	
	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line
Prompt 1	18	5	28	6	35	7	44	8
Prompt 2	20	6	27	6	33	7	45	9
Prompt 3	19	5	27	6	32	8	43	9
Prompt 4	19	6	30	7	32	8	43	9

Test Generation Process



Evaluations

- Our tool without FiD and with FiD model Evaluation
  - Used prompt : [BUG]... [ERROR]... [CONTEXT]... [CLASS]... [METHOD]... [RETURN_TYPE]... [VARIABLES]...
  - Evaluate for one Iteration only

Our Tool	Number of return sequences							
	2		5		10		50	
	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line	Single Line	Multi Line
Without FiD	28	6	35	8	44	10	52	11
With FiD	25	7	37	7	43	8	44	8