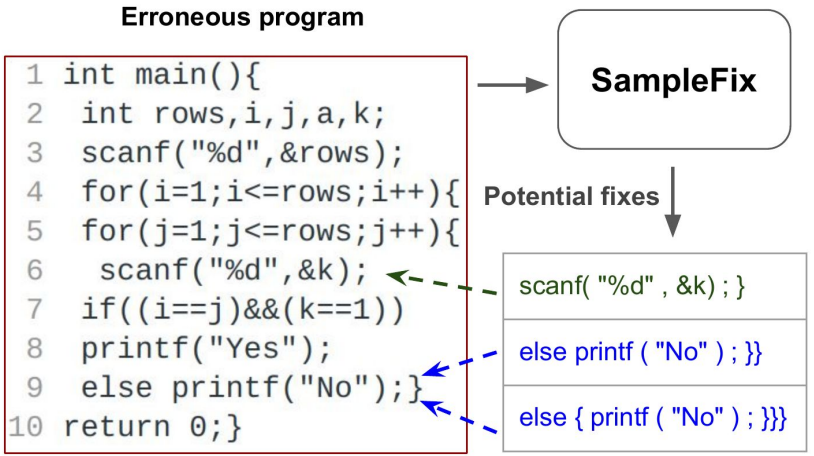
SampleFix: Learning to Generate Functionally Diverse Fixes

Hossein Hajipour[1], Apratim Bhattacharyya[2], Cristian-Alexandru Staicu[1], and Mario Fritz[1]

[1]CISPA Helmholtz Center for Information Security
[2]Max Planck Institute for Informatics

Introduction

- **Task:** Automatically correct common programming errors.
  - Missing scope delimiters
  - Missing variable declarations
  - Extraneous symbols

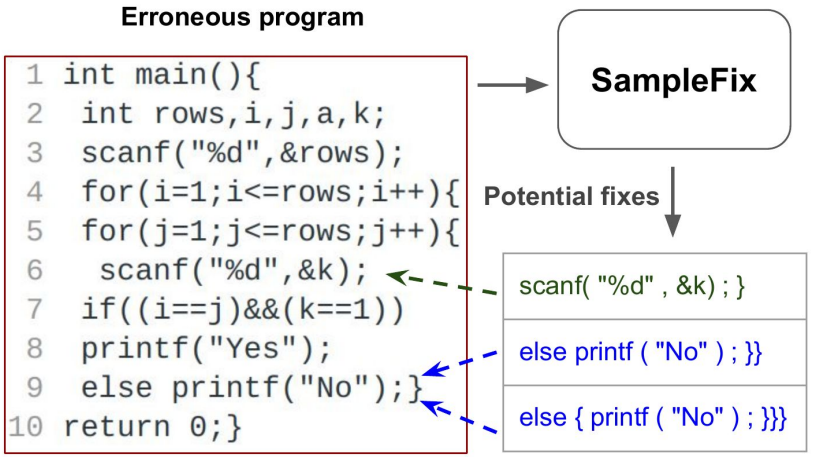


SampleFix captures the inherent ambiguity of the possible fixes by sampling multiple potential fixes for the given erroneous program.

Introduction

- **Task:** Automatically correct common programming errors.
  - Missing scope delimiters
  - Missing variable declarations
  - Extraneous symbols

- **Insight:** Multiple fixes can implement the same functionality, and there is uncertainty on the intention of the programmer.



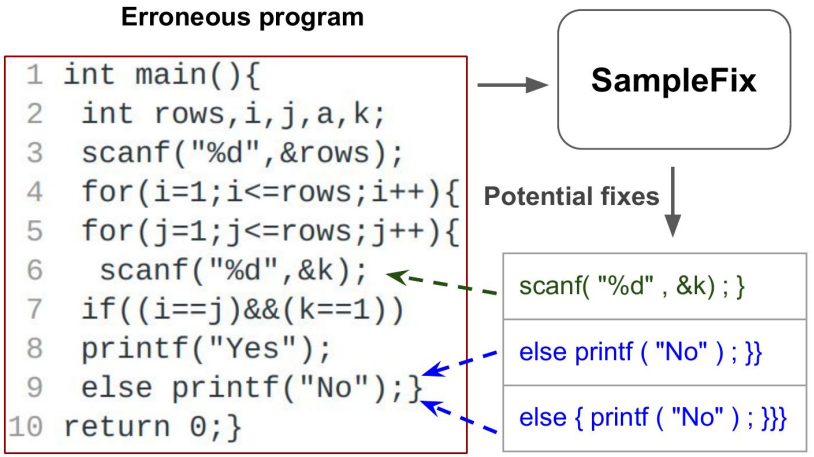
SampleFix captures the inherent ambiguity of the possible fixes by sampling multiple potential fixes for the given erroneous program.

Introduction

- **Task:** Automatically correct common programming errors.
  - Missing scope delimiters
  - Missing variable declarations
  - Extraneous symbols

- **Insight:** Multiple fixes can implement the same functionality, and there is uncertainty on the intention of the programmer.

- **Our approach:** We propose a generative framework to account for inherent ambiguity and lack of representative datasets

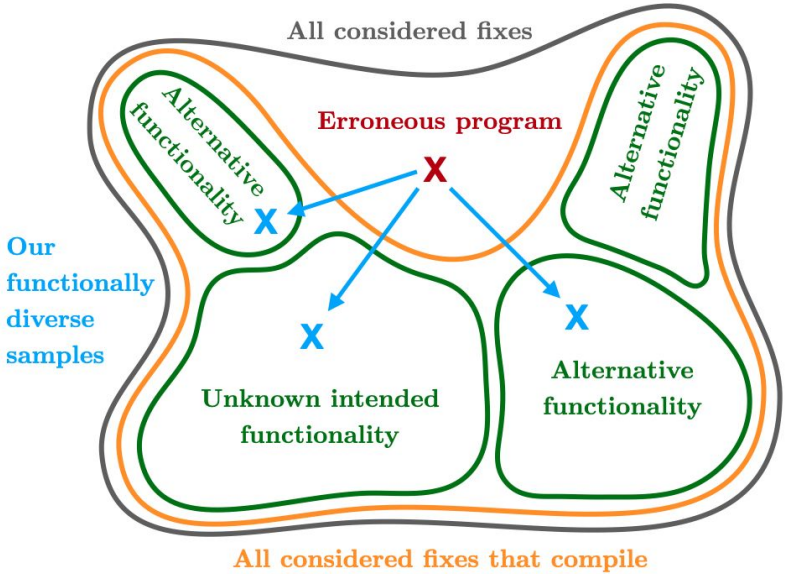


SampleFix captures the inherent ambiguity of the possible fixes by sampling multiple potential fixes for the given erroneous program.

Introduction

Generate Functionally Diverse Fixes

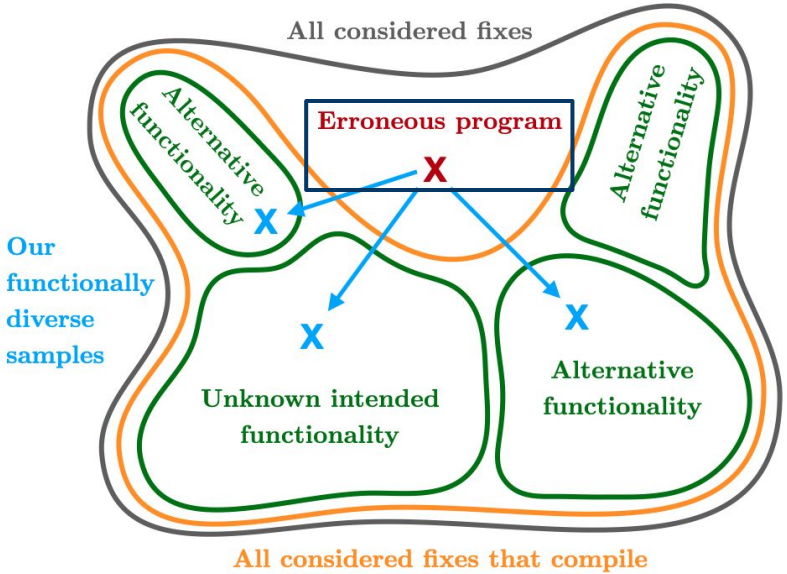
- Uncertainty about the intention of the programmer



Introduction

Generate Functionally Diverse Fixes

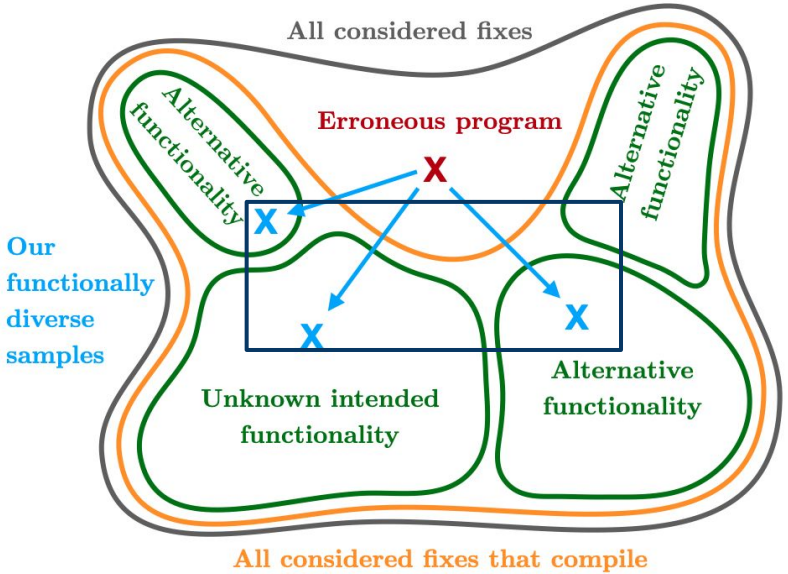
- Uncertainty about the intention of the programmer



Introduction

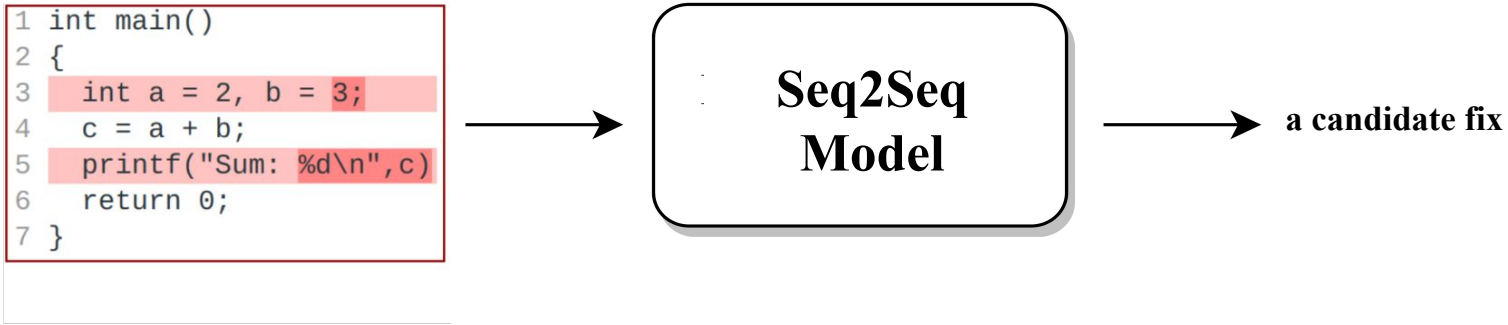
Generate Functionally Diverse Fixes

- Uncertainty about the intention of the programmer



Related Work

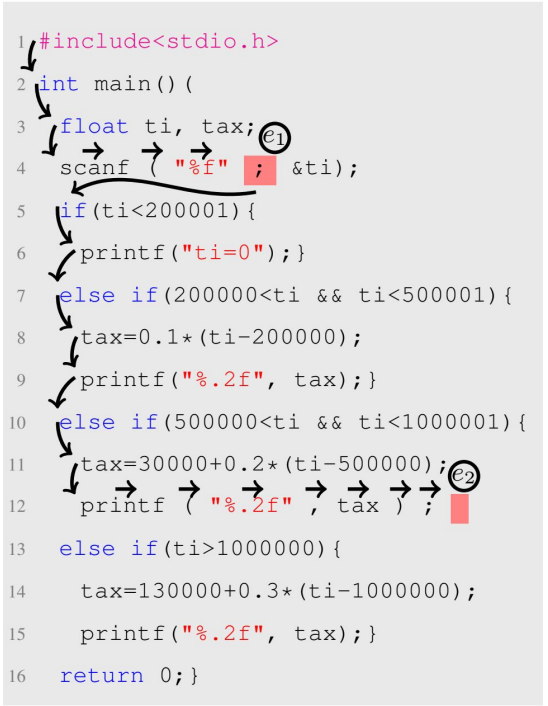
DeepFix [Gupta, Rahul, et al. *AAAI*. 2017.]



Related Work

RLAssist [Gupta, Rahul, et al. *AAAI*. 2019.]

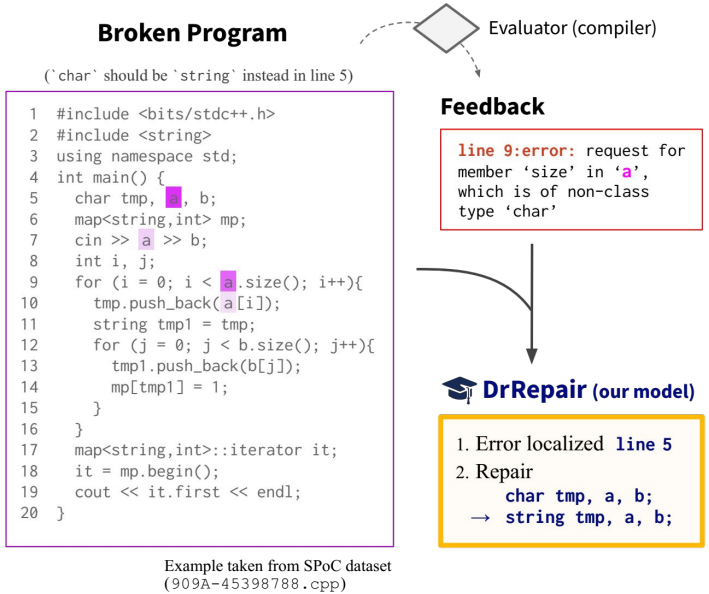
- Using reinforcement learning.
- An **agent** navigate over the program to **locate** and **resolve** the syntax errors.
- It relies on hand designed actions.



Related Work

DrRepair [Yasunaga, Michihiro, et al. *ICML*. 2020.]

- Using graph-attention mechanism to connects symbols relevant to program repair in source code and compile message.
- Utilizing the compiler output seems to be beneficial, it also limits the generality of the approach.

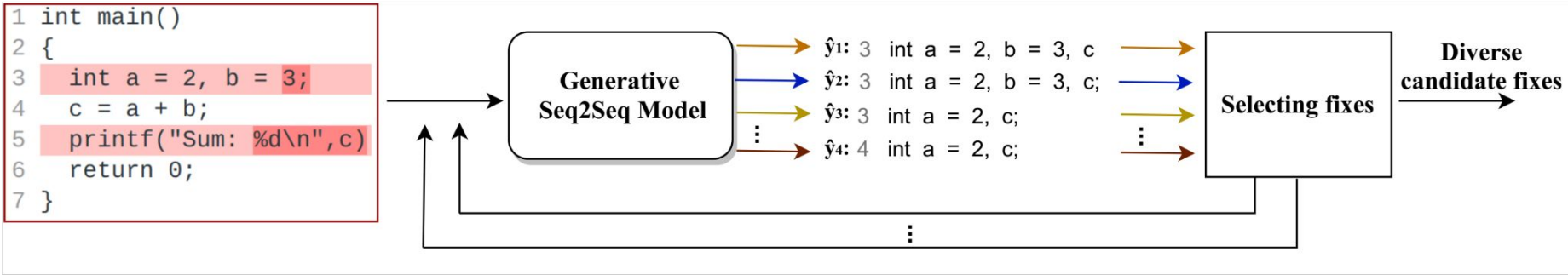


Example taken from SPoC dataset (909A-45398788.cpp)

SampleFix

Generative Model for Diversified Code Fixes

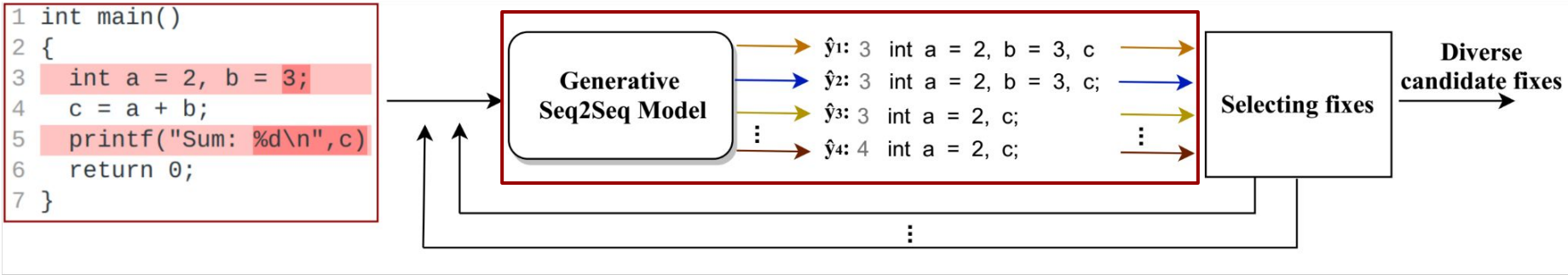
- Overview of SampleFix at inference time, highlighting the generation of diverse fixes.



SampleFix

Generative Model for Diversified Code Fixes

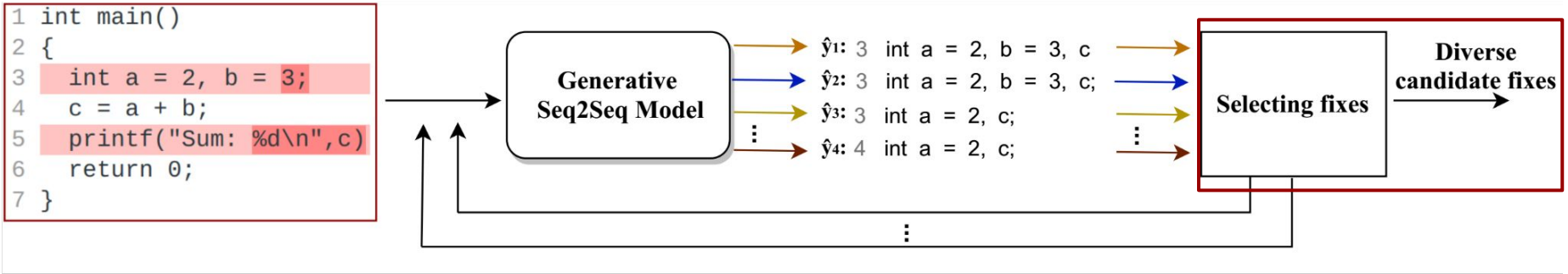
- Overview of SampleFix at inference time, highlighting the generation of diverse fixes.



SampleFix

Generative Model for Diversified Code Fixes

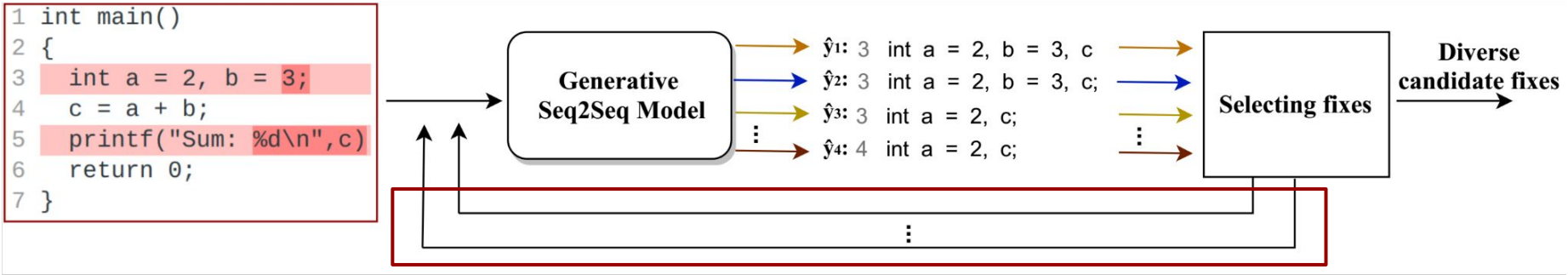
- Overview of SampleFix at inference time, highlighting the generation of diverse fixes.



SampleFix

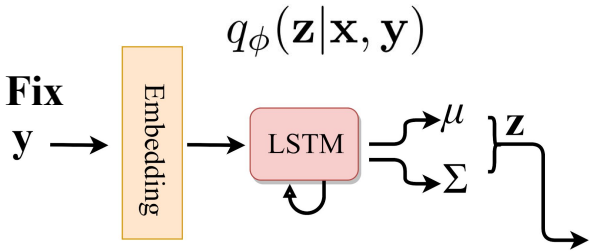
Generative Model for Diversified Code Fixes

- Overview of SampleFix at inference time, highlighting the generation of diverse fixes.



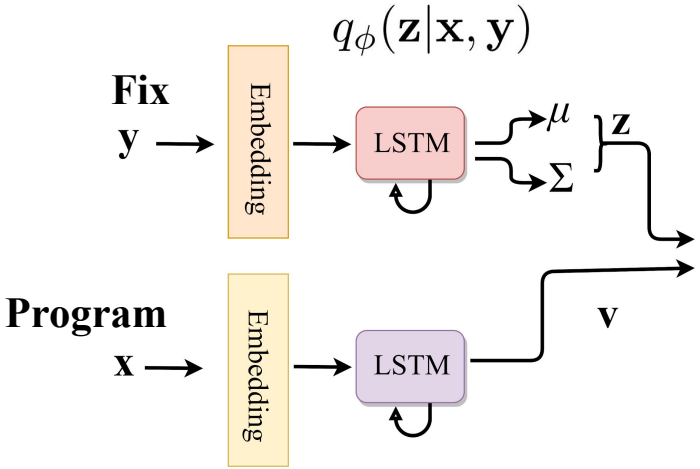
SampleFix

Conditional Variational Autoencoders for Generating Fixes



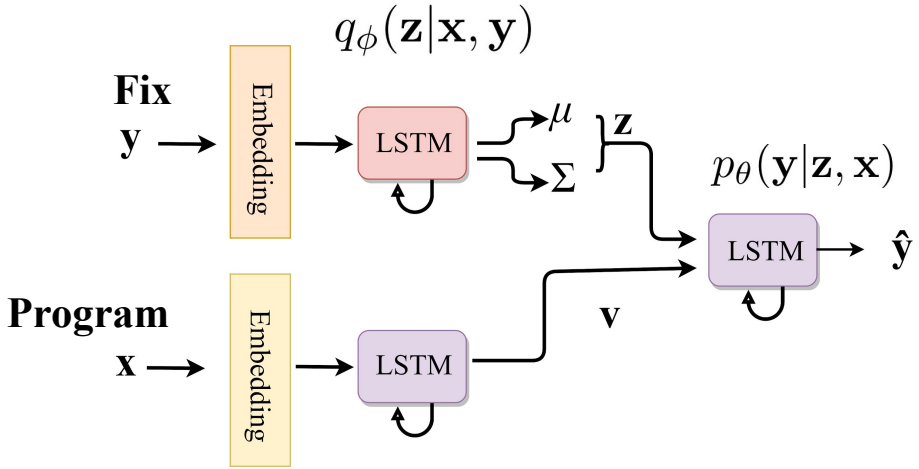
SampleFix

Conditional Variational Autoencoders for Generating Fixes



SampleFix

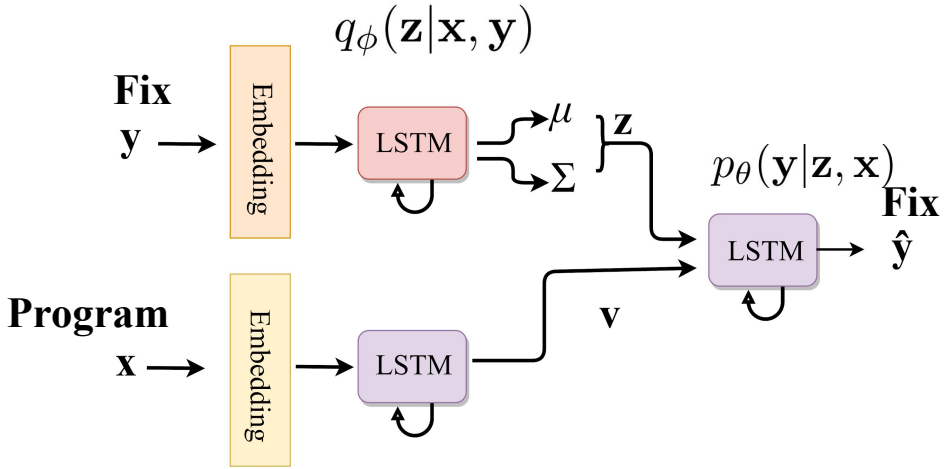
Conditional Variational Autoencoders for Generating Fixes



$$\hat{\mathcal{L}}_{\text{CVAE}} = \frac{1}{T}\sum_{\text{i}=1}^{T} \log(p_\theta(\mathbf{y}|\hat{\mathbf{z}}_\text{i},\mathbf{x})) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}),p(\mathbf{z}|\mathbf{x})) \quad .$$

SampleFix

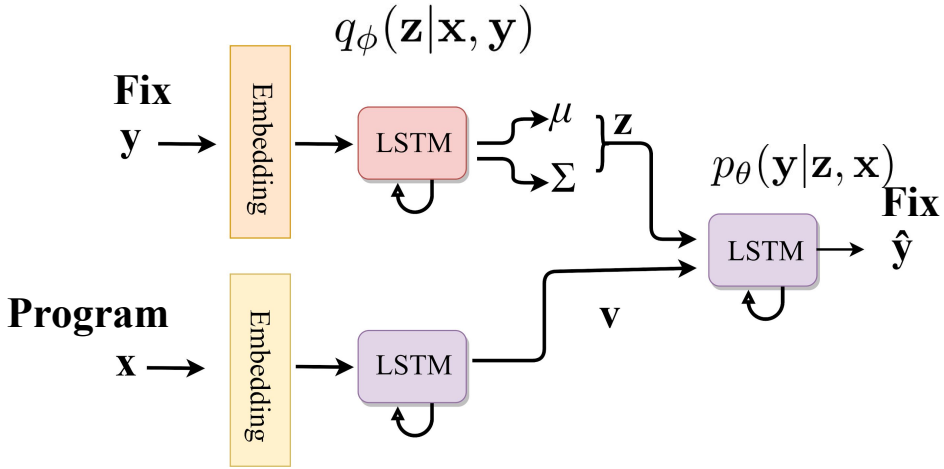
Enabling diverse samples using the Best of Many objective (BMS).



$$\hat{\mathcal{L}}_{\text{BMS}} = \max_i \left(\log(p_\theta(\mathbf{y}|\hat{\mathbf{z}}_i, \mathbf{x})) \right) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}), p(\mathbf{z}|\mathbf{x})) \quad .$$

SampleFix

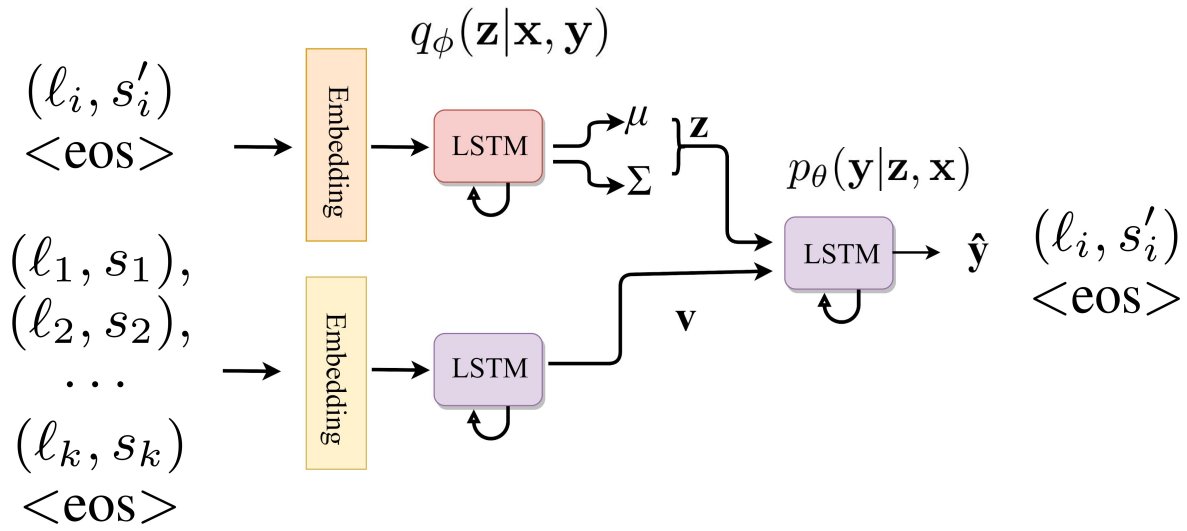
DS-SampleFix: Encouraging diversity with a diversity-sensitive regularizer



$$\hat{\mathcal{L}}_{\text{DS-BMS}} = \max_{i} \Big(\log(p_\theta(\mathbf{y}|\hat{\mathbf{z}}_i, \mathbf{x})) \Big) + \min_{i,j} d(\hat{\mathbf{y}}^i, \hat{\mathbf{y}}^j) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}), p(\mathbf{z}|\mathbf{x})) \ .$$

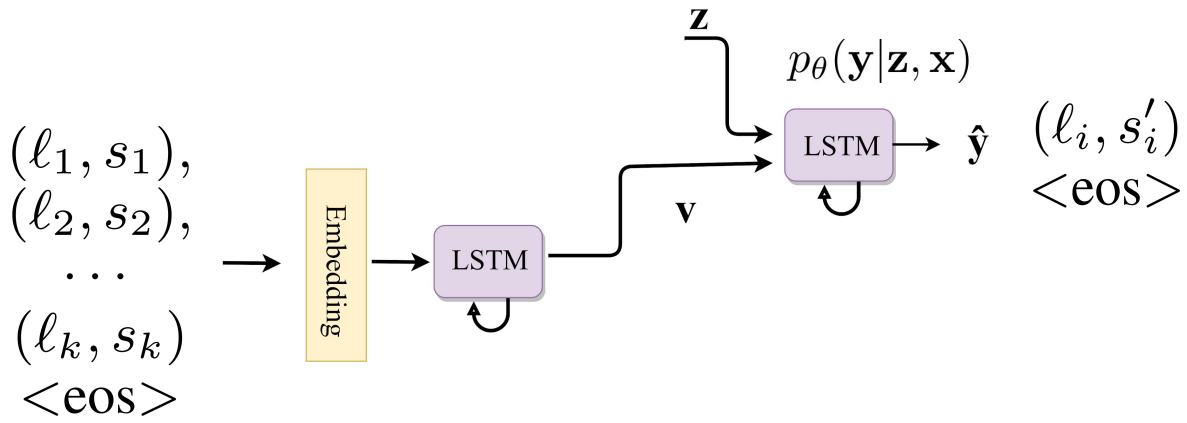
SampleFix

Training and inference



SampleFix

Training and inference



Experiments

- **Data**
  - Training Data: 500k pairs of data by mutating the correct programs
  - Test Data: Real-world data contains 6975 erroneous programs with 16766 error messages written by students
  - All of these program were written for 93 different programming tasks

Experiments

- **Data**
  - Training Data: 500k pairs of data by mutating the correct programs
  - Test Data: Real-world data contains 6975 erroneous programs with 16766 error messages written by students
  - All of these program were written for 93 different programming tasks
- **Type of data**
  - **Synthetic Data:** Generating training and validation set by mutating the correct program
  - **Real-World Data:** 6975 erroneous programs written by students

Experiments

- **Data**
  - Training Data: 500k pairs of data by mutating the correct programs
  - Test Data: Real-world data contains 6975 erroneous programs with 16766 error messages written by students
  - All of these program were written for 93 different programming tasks
- **Type of data**
  - **Synthetic Data:** Generating training and validation set by mutating the correct program
  - **Real-World Data:** 6975 erroneous programs written by students
- **Type of errors**
  - Typographic errors
  - Missing variable declaration

Experiments

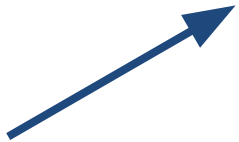
Training data

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (){
4  int n ;
5  double a , b , dx , areai ;
6  for ( i = 0 ; i <= n - 0 , i ++){
7  dx =( b - a )/ n ;
8  k =( 0 * i - 0 * i * i * i );
9  while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Experiments

Training data

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (){
4  int n ;
5  double a , b , dx , areai ;
6  for ( i = 0 ; i <= n - 0 , i ++){
7  dx =( b - a )/ n ;
8  k =( 0 * i - 0 * i * i * i );
9  while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```



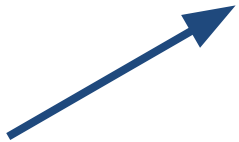
Input

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (){
4  int n ;
5  double a , b , dx , areai ;
6  for ( i = 0 ; i <= n - 0 , i ++){
7  dx =( b - a )/ n ;
8  k =( 0 * i - 0 * i * i * i );
9  while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Experiments

Training data

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```



Input

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Output

```
6 for ( i = 0 ; i <= n - 0 ; i ++){
```

Experiments

Training data

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Input

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Output

```
6 for ( i = 0 ; i <= n - 0 ; i ++){
```

Input

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 ; i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Experiments

Training data

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Input

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 , i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Output

```
6 for ( i = 0 ; i <= n - 0 ; i ++){
```

Input

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main (){
4 int n ;
5 double a , b , dx , areai ;
6 for ( i = 0 ; i <= n - 0 ; i ++){
7 dx =( b - a )/ n ;
8 k =( 0 * i - 0 * i * i * i );
9 while ( k >= 0 )
10 k = k ;
11 else
12 k =- k ;
13 return 0 ;}
```

Output

```
12 k =- k ;}
```

Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Results on synthetic data

- To evaluate our approach on the synthetic test set we randomly select 20k pairs of the data.
- Results of DeepFix, Beam Search(BS), SampleFix, DS-SampleFix.
- **Typo**, **Miss Dec**, and **All** refer to typographic, missing variable declarations, and all of the error messages respectively.

Models	Typo	Miss Dec	All
DeepFix	84.7%	78.8%	82.0%
Beam search (BS)	91.8%	89.5%	90.7%
SampleFix	86.8%	86.5%	86.6%
DS-SampleFix	95.6%	88.1%	92.2%

Experiments

Results on real-world data

- Results of DeepFix, RLAssist, DrRepair, Beam Search(BS), SampleFix, DS-SampleFix, and DS-SampleFix + BS (Beam Search).
- **Typo**, **Miss Dec**, and **All** refer to typographic, missing variable declarations, and all of the error messages respectively.
- ✔ denotes successfully compiled programs, while 🐞 refers to resolved error messages.

Models	Typo		Miss Dec		All		Speed (s)
	✔	🐞	✔	🐞	✔	🐞	
DeepFix [15]	23.3%	30.8%	10.1%	12.9%	33.4%	40.8%	-
RLAssist [14]	*26.6%*	*39.7%*	-	-	-	-	-
DrRepair [38]	-	-	-	-	34.0%	-	-
Beam search (BS)	25.9%	42.2%	**20.3%**	47.0%	44.7%	63.9%	4.82
SampleFix	24.8%	38.8%	16.1%	22.8%	40.9%	56.3%	0.88
DS-SampleFix	27.7%	40.9%	16.7%	24.7%	44.4%	61.0%	0.88
DS-SampleFix + BS	**27.8%**	**45.6%**	19.2%	**47.9%**	**45.2%**	**65.2%**	1.17

Experiments

Results on real-world data

- Results of DeepFix, RLAssist, DrRepair, Beam Search(BS), SampleFix, DS-SampleFix, and DS-SampleFix + BS (Beam Search).
- **Typo**, **Miss Dec**, and **All** refer to typographic, missing variable declarations, and all of the error messages respectively.
- ✔ denotes successfully compiled programs, while 🐞 refers to resolved error messages.

Models	Typo		Miss Dec		All		Speed (s)
	✔	🐞	✔	🐞	✔	🐞	
DeepFix [15]	23.3%	30.8%	10.1%	12.9%	33.4%	40.8%	-
RLAssist [14]	*26.6%*	*39.7%*	-	-	-	-	-
DrRepair [38]	-	-	-	-	34.0%	-	-
Beam search (BS)	25.9%	42.2%	**20.3%**	47.0%	44.7%	63.9%	4.82
SampleFix	24.8%	38.8%	16.1%	22.8%	40.9%	56.3%	0.88
DS-SampleFix	27.7%	40.9%	16.7%	24.7%	44.4%	61.0%	0.88
DS-SampleFix + BS	**27.8%**	**45.6%**	19.2%	**47.9%**	**45.2%**	**65.2%**	1.17

Experiments

Results on real-world data

- Results of DeepFix, RLAssist, DrRepair, Beam Search(BS), SampleFix, DS-SampleFix, and DS-SampleFix + BS (Beam Search).
- **Typo**, **Miss Dec**, and **All** refer to typographic, missing variable declarations, and all of the error messages respectively.
- ✔ denotes successfully compiled programs, while 🐞 refers to resolved error messages.

Models	Typo		Miss Dec		All		Speed (s)
	✔	🐞	✔	🐞	✔	🐞	
DeepFix [15]	23.3%	30.8%	10.1%	12.9%	33.4%	40.8%	-
RLAssist [14]	*26.6%*	*39.7%*	-	-	-	-	-
DrRepair [38]	-	-	-	-	34.0%	-	-
Beam search (BS)	25.9%	42.2%	**20.3%**	47.0%	44.7%	63.9%	4.82
SampleFix	24.8%	38.8%	16.1%	22.8%	40.9%	56.3%	0.88
DS-SampleFix	27.7%	40.9%	16.7%	24.7%	44.4%	61.0%	0.88
DS-SampleFix + BS	**27.8%**	**45.6%**	19.2%	**47.9%**	**45.2%**	**65.2%**	1.17

Experiments

Qualitative example

- An example illustrating that our DS-SampleFix can generate diverse fixes.

Erroneous program

```
1  #include <stdio.h>
2  int main (){
3  int a, i;
4  scanf("%d\n", &a);
5  int s[a], p[a], g[a];
6  for (i = 0; i < a; i++){
7  scanf("%d", &s[i]);}
8  for (i = 0; i < a; i++){
9  scanf("%d", &p[i]);}
10 for (i = 0; i < a; i++){
11 g[p[i]] = s[i];}
12 for (i = 0; i < a; i++){
13 printf("%d", g[i]);
14 printf("end");
15 return 0 ;}
```

Id	Action	New Code
P_1	replace line 13	`printf("%d", g[i]);}`
P_2	replace line 14	`printf("end");}`

Experiments

Qualitative example

- Example of resolving typographic errors, and missing variable declaration errors

Erroneous program

```
1  #include <stdio.h>
2  int main (){
3
4  int s [ 0 ];
5  int k , n , i ;
6  scanf ( "%d%d" ,& k ,& n );
7  for ( i = 0 ; i < n ; i ++){
8  scanf ( "%d" ,& value );
9  s [ i ]= value ;}
10 for ( i = 0 ; i < n ; i ++) {
11 if ( k == s [ i ]+ s [ k - i ])} {
12 printf ( "lucky" );
13 break ;}
14 else { printf ( "unlucky" );
15 break ;}
16 return 0 ;}
```

Repaired program

```
1  #include <stdio.h>
2  int main (){
3  int value ;
4  int s [ 0 ];
5  int k , n , i ;
6  scanf ( "%d%d" ,& k ,& n );
7  for ( i = 0 ; i < n ; i ++){
8  scanf ( "%d" ,& value );
9  s [ i ]= value ;}
10 for ( i = 0 ; i < n ; i ++) {
11 if ( k == s [ i ]+ s [ k - i ]) {
12 printf ( "lucky" );
13 break ;}
14 else { printf ( "unlucky" );
15 break ;}}
16 return 0 ;}
```

Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Generating Functionally Diverse Programs

- Given an erroneous program, our approach can generate multiple potential fixes.
- It is desirable to suggest multiple potential fixes with diverse functionalities.

Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Generating Functionally Diverse Programs

- Given an erroneous program, our approach can generate multiple potential fixes.
- It is desirable to suggest multiple potential fixes with diverse functionalities.
- In order to assess different functionalities, we generate 10 input example for 93 task in dataset.
- We consider two programs to have different functionalities if they return different outputs given the same input examples.

Experiments

Generating Functionally Diverse Programs

- **Diverse programs** refers to the percentage of cases where the models generate at least two or more successfully compiled unique programs.
- **Diverse functionality** denotes the percentage of cases where the models generate at least two or more programs with different functionalities.

Models	Diverse programs	Diverse functionality
Beam search	55.6%	45.1%
SampleFix	44.6%	34.9%
DS-SampleFix	68.8%	53.4%
DS-SampleFix + BS	**69.5%**	**60.4%**

Experiments

Generating Functionally Diverse Programs

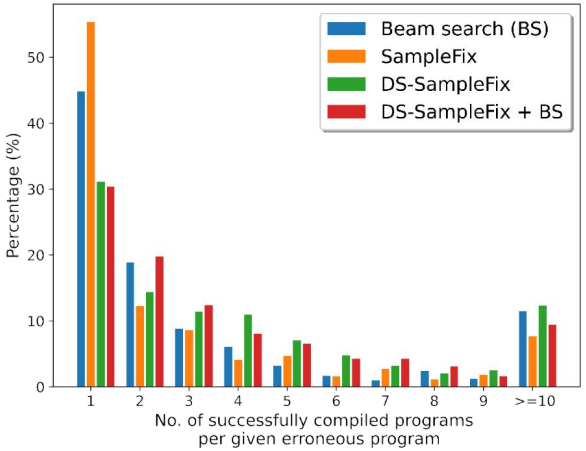
- **Diverse programs** refers to the percentage of cases where the models generate at least two or more successfully compiled unique programs.
- **Diverse functionality** denotes the percentage of cases where the models generate at least two or more programs with different functionalities.

Models	Diverse programs	Diverse functionality
Beam search	55.6%	45.1%
SampleFix	44.6%	34.9%
DS-SampleFix	68.8%	53.4%
DS-SampleFix + BS	**69.5%**	**60.4%**

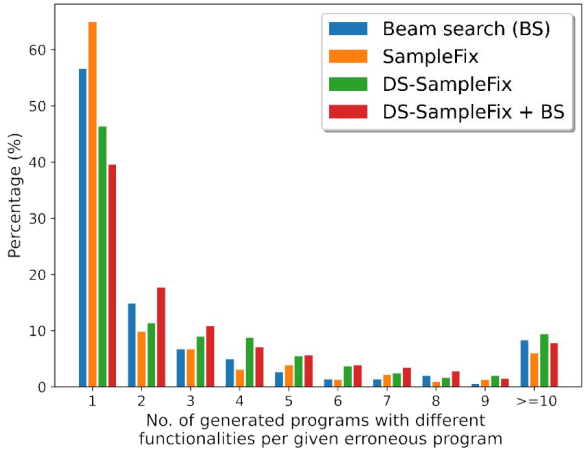
Experiments

Generating Functionally Diverse Programs

- **Left:** Percentage of the number of the generated successfully compiled, unique programs for the given erroneous programs.
- **Right:** Percentage of the successfully compiled programs with different functionalities for the given erroneous programs.



(a) Diversity of the generated programs.

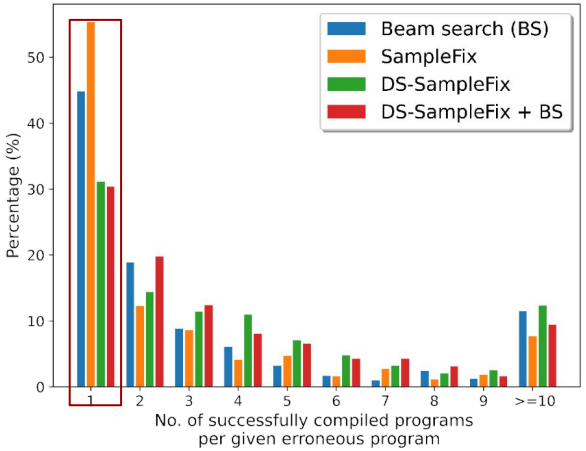


(b) Diversity of the functionality of the generated programs.

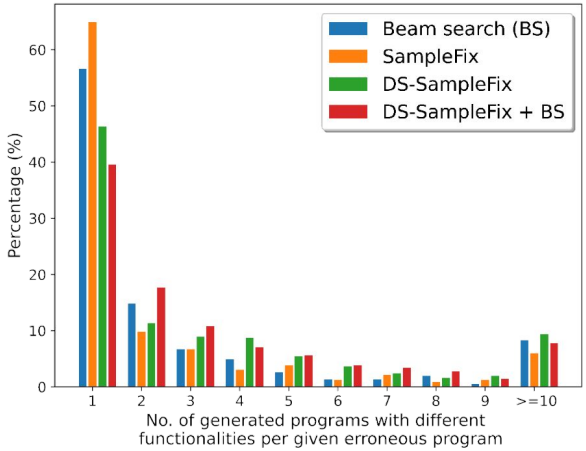
Experiments

Generating Functionally Diverse Programs

- **Left:** Percentage of the number of the generated successfully compiled, unique programs for the given erroneous programs.
- **Right:** Percentage of the successfully compiled programs with different functionalities for the given erroneous programs.



(a) Diversity of the generated programs.

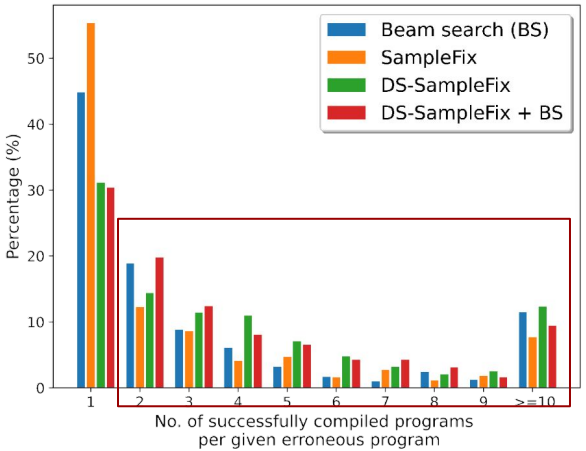


(b) Diversity of the functionality of the generated programs.

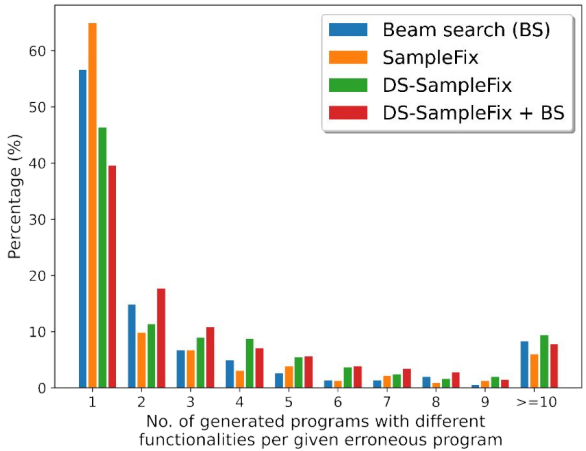
Experiments

Generating Functionally Diverse Programs

- **Left:** Percentage of the number of the generated successfully compiled, unique programs for the given erroneous programs.
- **Right:** Percentage of the successfully compiled programs with different functionalities for the given erroneous programs.



(a) Diversity of the generated programs.

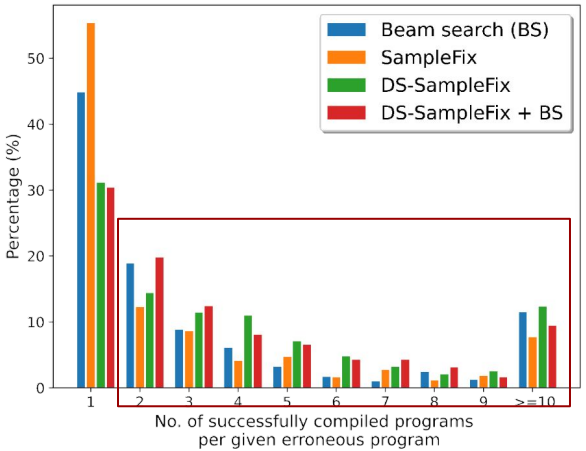


(b) Diversity of the functionality of the generated programs.

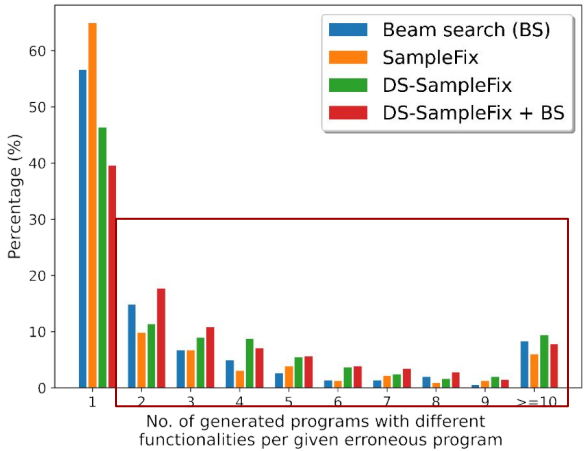
Experiments

Generating Functionally Diverse Programs

- **Left:** Percentage of the number of the generated successfully compiled, unique programs for the given erroneous programs.
- **Right:** Percentage of the successfully compiled programs with different functionalities for the given erroneous programs.



(a) Diversity of the generated programs.



(b) Diversity of the functionality of the generated programs.

Conclusion

- We propose a novel generative framework to generate functionally diverse code fixes.
- In order to overcome the inherent limitations of the datasets we propose a novel diversity-sensitive regularizer.
- Our evaluations on common programming errors show strong improvements over the state-of-the-art approaches .
- We additionally show that for the 65%of the repaired programs, our approach was able to generate multiple programs with diverse functionalities