Graph-based, Self-Supervised Program Repair from Diagnostic Feedback

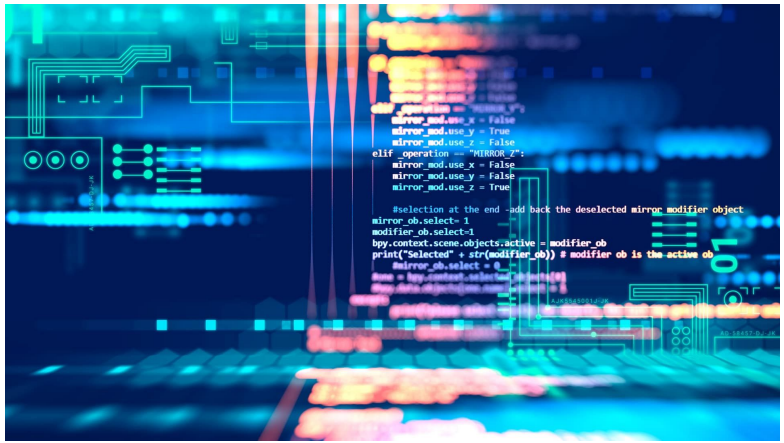
ICML 2020

Michihiro Yasunaga, Percy Liang
Stanford University

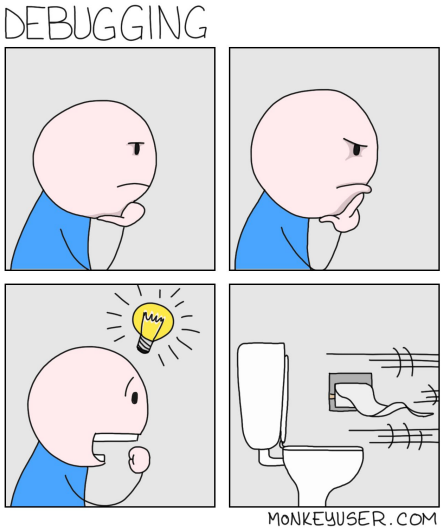
Why program repair?

Programmers spend 75% of time fixing source code errors

Automatic program repair can dramatically enhance programming productivity



Dream



Reality

Theme: Learning from Feedback

Humans learn from feedback

Write code → compile/execute → **repair code based on feedback**

Theme: Learning from Feedback

Humans learn from feedback

Write code → compile/execute → **repair code based on feedback**

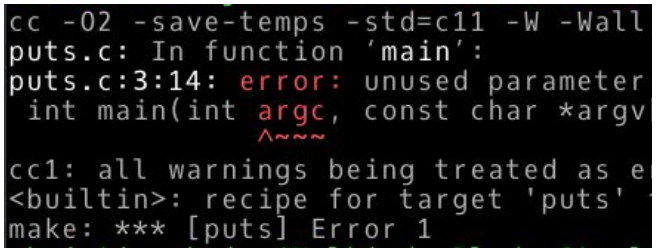
Theme: Learning from Feedback

Humans learn from feedback

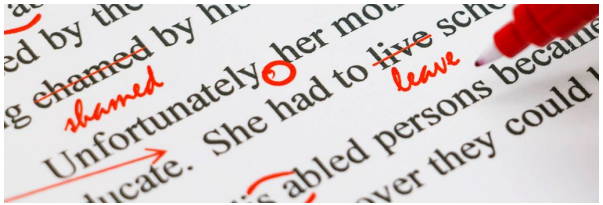
Write code → compile/execute → **repair code based on feedback**

General framework with many other applications

- Edit essays based on written feedback
- Learn from user inputs in interactive dialogue



Repair programs based on errors



Edit essays based on feedback

Example

Broken program

```
 1  #include <bits/stdc++.h>
 2  #include <string>
 3  using namespace std;
 4  int main() {
 5    char tmp,  a , b;
 6    map<string,int> mp;
 7    cin >>  a  >> b;
 8    int i, j;
 9    for (i = 0; i <  a .size(); i++){
10      tmp.push_back( a [i]);
11      string tmp1 = tmp;
12      for (j = 0; j < b.size(); j++){
13        tmp1.push_back(b[j]);
14        mp[tmp1] = 1;
15      }
16    } ...
```

Example

Should be `string`

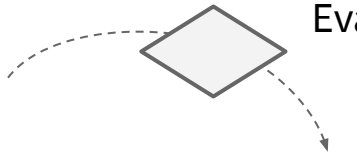
Broken program

```
 1  #include <bits/stdc++.h>
 2  #include <string>
 3  using namespace std;
 4  int main() {
 5    char tmp,  a , b;
 6    map<string,int> mp;
 7    cin >>  a  >> b;
 8    int i, j;
 9    for (i = 0; i <  a .size(); i++){
10      tmp.push_back( a [i]);
11      string tmp1 = tmp;
12      for (j = 0; j < b.size(); j++){
13        tmp1.push_back(b[j]);
14        mp[tmp1] = 1;
15      }
16    } ...
```

Example

Broken program

```
 1  #include <bits/stdc++.h>
 2  #include <string>
 3  using namespace std;
 4  int main() {
 5    char tmp,  a , b;
 6    map<string,int> mp;
 7    cin >>  a  >> b;
 8    int i, j;
 9    for (i = 0; i <  a .size(); i++){
10      tmp.push_back( a [i]);
11      string tmp1 = tmp;
12      for (j = 0; j < b.size(); j++){
13        tmp1.push_back(b[j]);
14        mp[tmp1] = 1;
15      }
16    } ...
```



Evaluator (compiler)

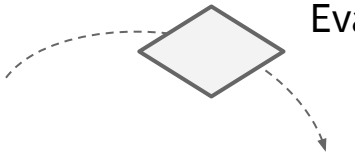
Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```



Evaluator (compiler)

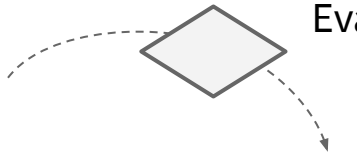
Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```



Evaluator (compiler)

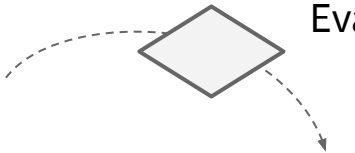
Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```



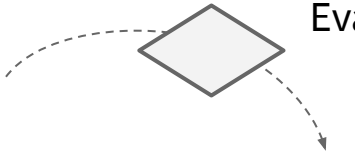
Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```



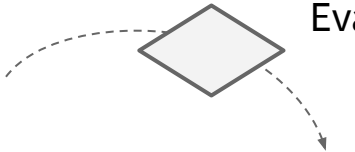
Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```



Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Repair!

1. Error localized **line 5**
2. Repair

```
  char tmp, a, b;
→ string tmp, a, b;
```

Goal: learn to repair programs from error messages

Goal: learn to repair programs from error messages

Broken program

```
 1  #include <bits/stdc++.h>
 2  #include <string>
 3  using namespace std;
 4  int main() {
 5    char tmp,  a , b;
 6    map<string,int> mp;
 7    cin >>  a  >> b;
 8    int i, j;
 9    for (i = 0; i <  a .size(); i++){
10      tmp.push_back( a [i]);
11      string tmp1 = tmp;
12      for (j = 0; j < b.size(); j++){
13        tmp1.push_back(b[j]);
14        mp[tmp1] = 1;
15      }
16    } ...
```

Diagnostic Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Goal: learn to repair programs from error messages

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp,  a , b;
6    map<string,int> mp;
7    cin >>  a  >> b;
8    int i, j;
9    for (i = 0; i <  a .size(); i++){
10     tmp.push_back( a [i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```

Diagnostic Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

DrRepair (our system)

1. Error localized **line 5**

Goal: learn to repair programs from error messages

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp,  a , b;
6    map<string,int> mp;
7    cin >>  a  >> b;
8    int i, j;
9    for (i = 0; i <  a .size(); i++){
10     tmp.push_back( a [i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```

Diagnostic Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

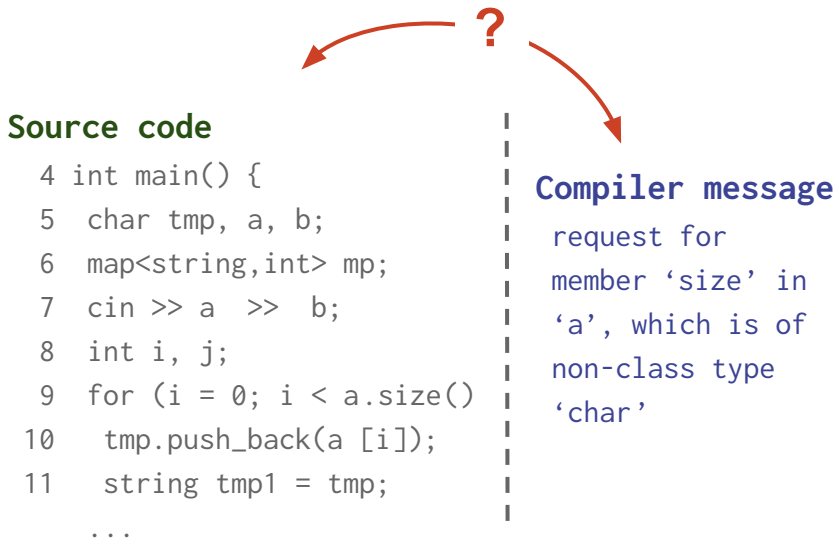
DrRepair (our system)

1. Error localized **line 5**
2. Repair

```
  char tmp, a, b;
→ string tmp, a, b;
```

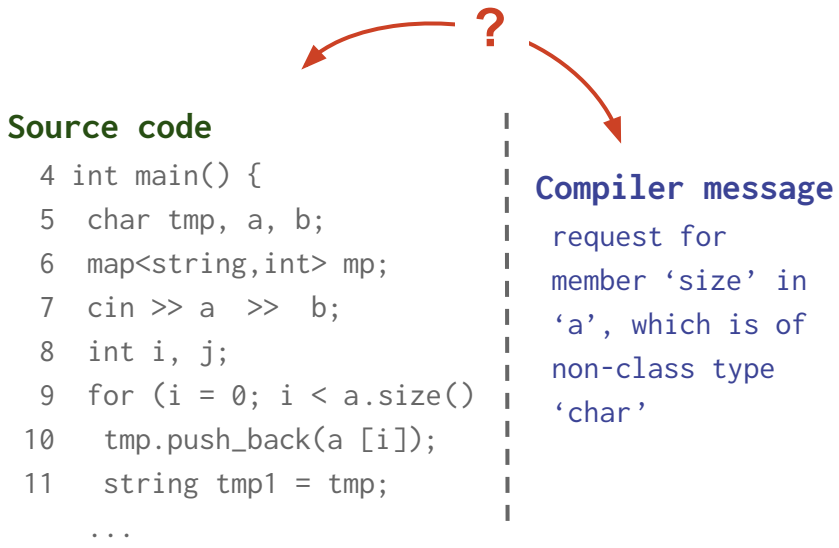
Challenges

1. **Modeling**
   - How to connect two modalities: **program** and **feedback**?
   - How to model the **reasoning** of repair (e.g. tracking symbols)?



Challenges

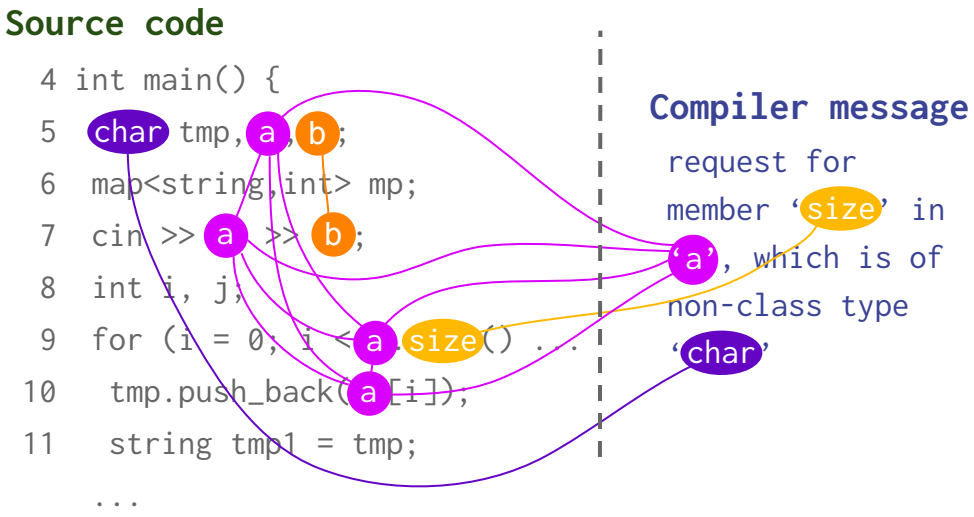
1. **Modeling**
   - How to connect two modalities: **program** and **feedback**?
   - How to model the **reasoning** of repair (e.g. tracking symbols)?



2. **Data**
   - Existing works rely on labeled datasets of <broken code, fixed code>
   - Relatively small (10–100K data points). How to **scale up**?

Our contributions

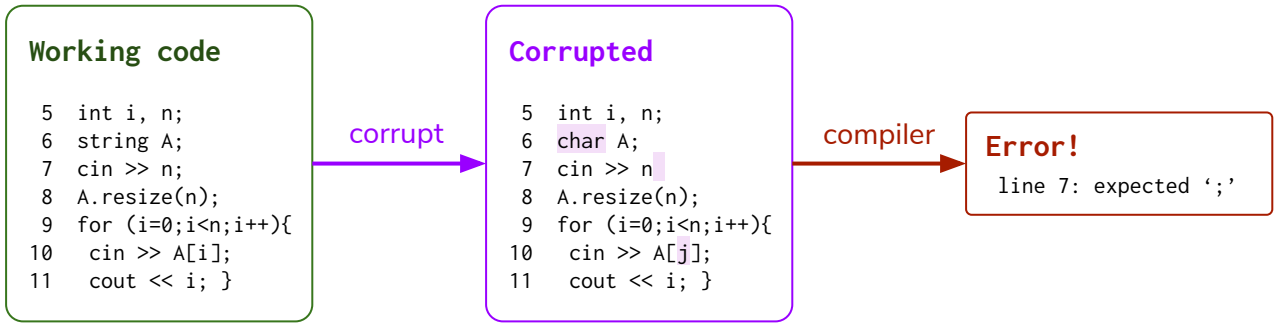
1. **Program-feedback graph**
   - Connect symbols across **program** & **feedback**
   - Performs reasoning via **graph-attention**



Our contributions

2. **Self-supervised learning**
   - Collect unlabeled programs
   - Corrupt and get diagnostic feedback (e.g. run compiler)

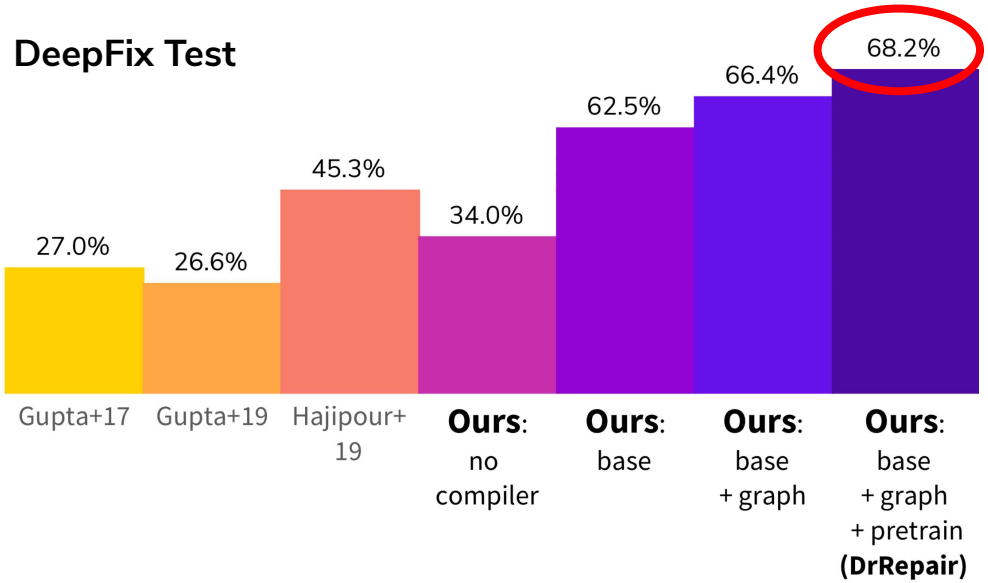
⇒ **Extra training data**: <broken code, feedback, fixed code>

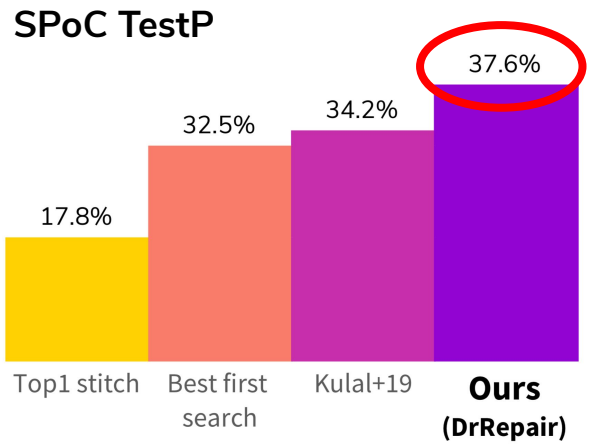


Our results

Improved performance on two applications

- DeepFix: correct intro programming assignments in C
- SPoC: correct output of C++ program synthesis





Outline

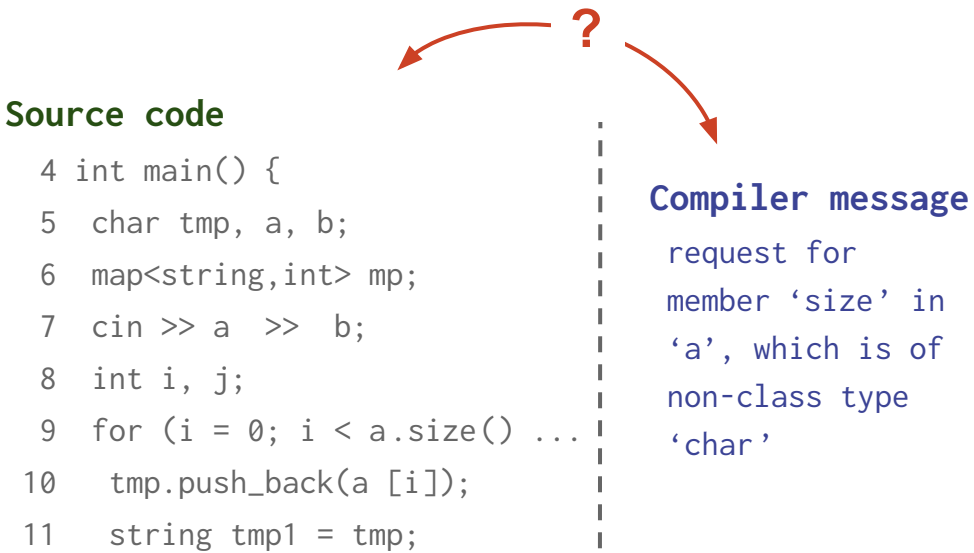
- Innovations
    1. Reasoning via program-feedback graph
    2. Self-supervised learning
- Evaluations
    1. DeepFix
    2. SPoC
- Analysis & Examples
- Takeaways

1. Reasoning via program-feedback graph

1. Reasoning via program-feedback graph

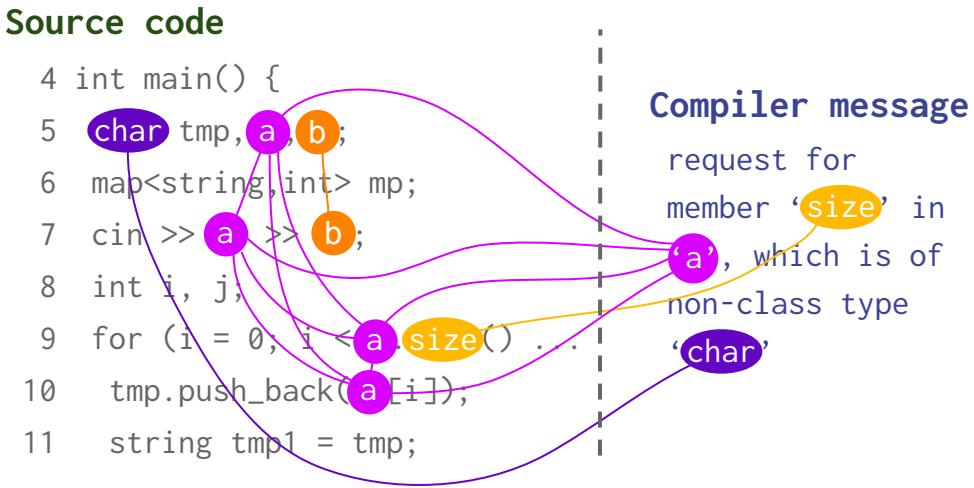
Challenges

- How to connect two modalities: **program** and **feedback**?
- How to model the **reasoning** of repair (e.g. tracking symbols)?



1. Reasoning via program-feedback graph

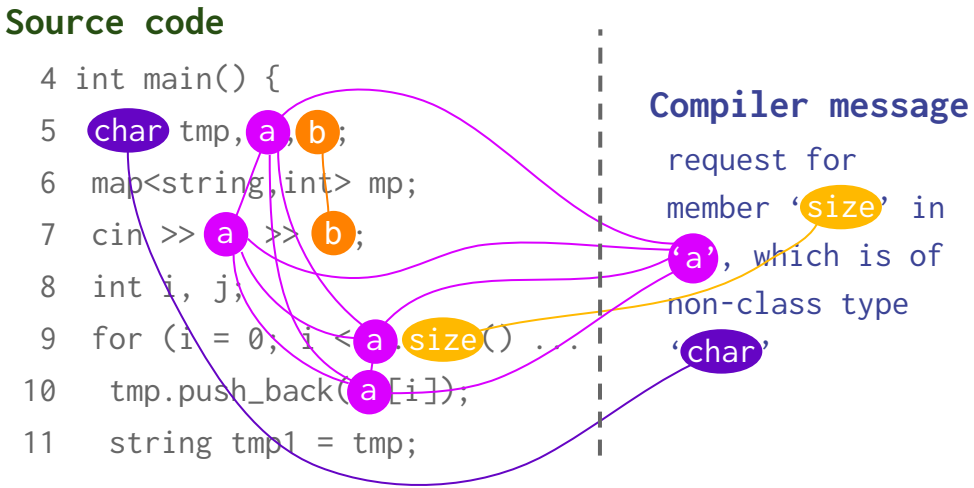
Our solution: program-feedback graph



1. Reasoning via program-feedback graph

Our solution: program-feedback graph

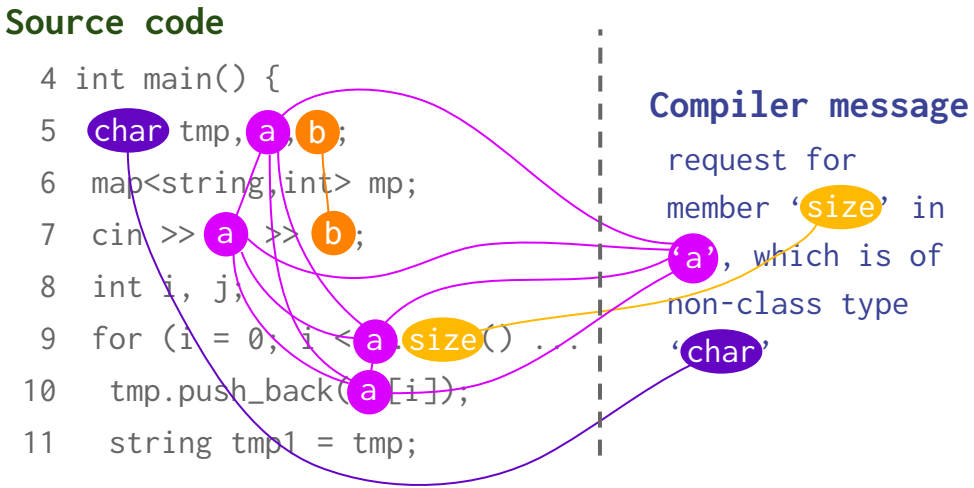
- Join program & feedback through symbols relevant to program repair
  → shared/abstracted semantic space



1. Reasoning via program-feedback graph

Our solution: program-feedback graph

- Join program & feedback through symbols relevant to program repair
  → shared/abstracted semantic space
- Reason over this space using graph attention



1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types

Source code

```
 4 int main() {
 5  char tmp, a, b;
 6  map<string,int> mp;
 7  cin >> a  >>  b;
 8  int i, j;
 9  for (i = 0; i < a.size() ...
10   tmp.push_back(a [i]);
11   string tmp1 = tmp;
```

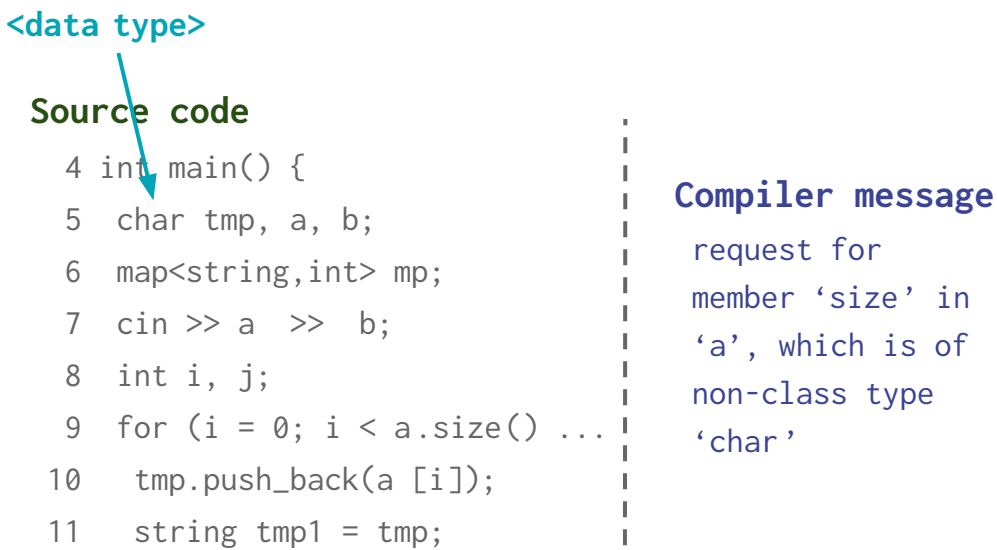
Compiler message

```
request for
member ‘size’ in
‘a’, which is of
non-class type
‘char’
```

1. Reasoning via program-feedback graph

How to construct graph?

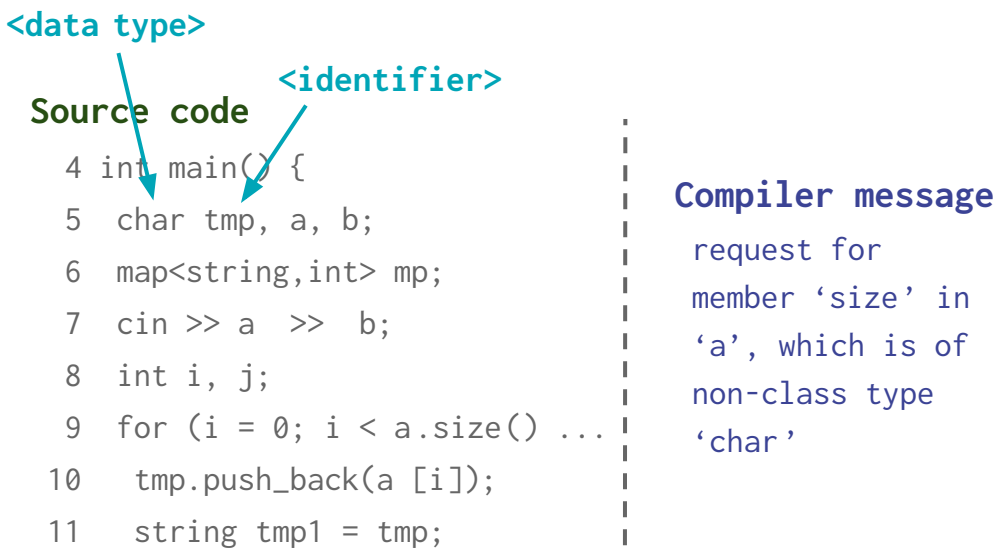
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

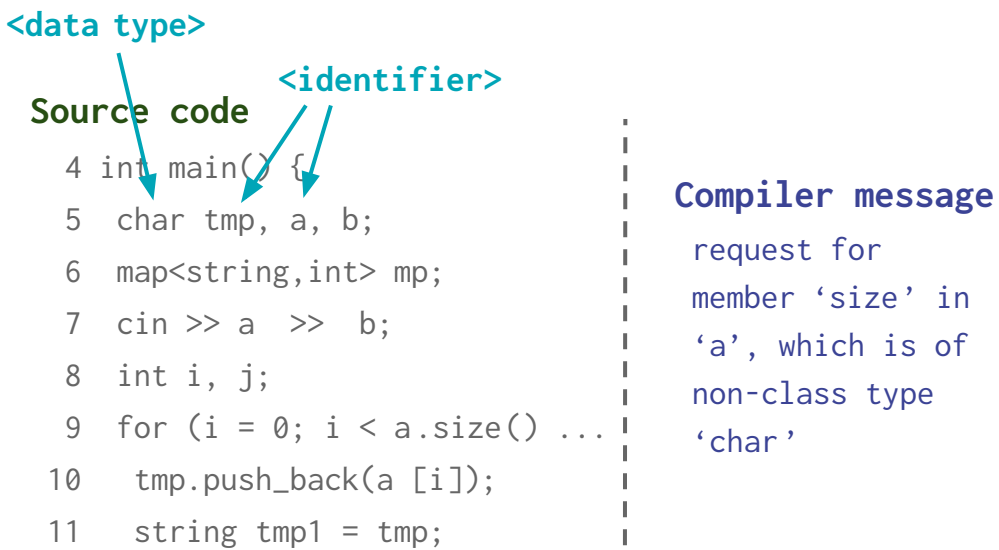
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

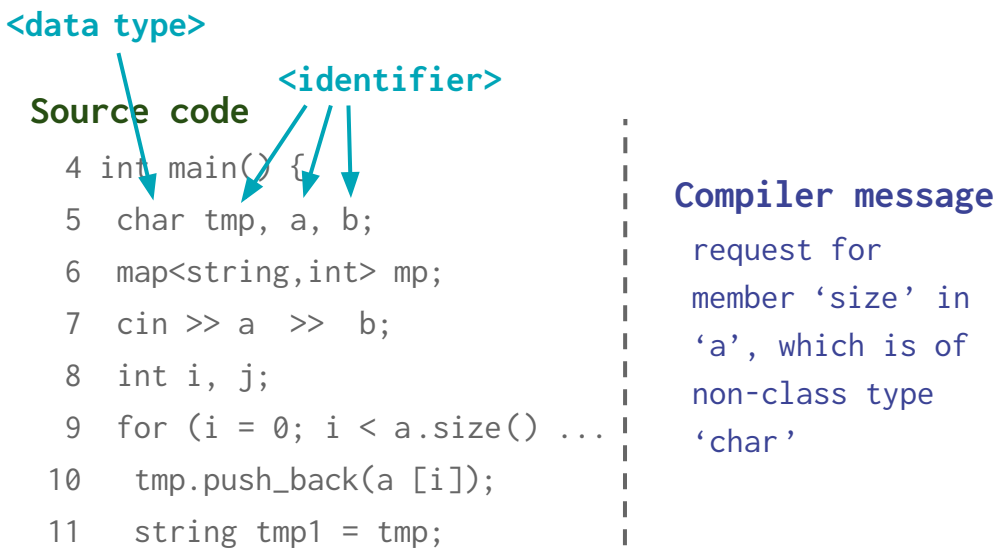
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

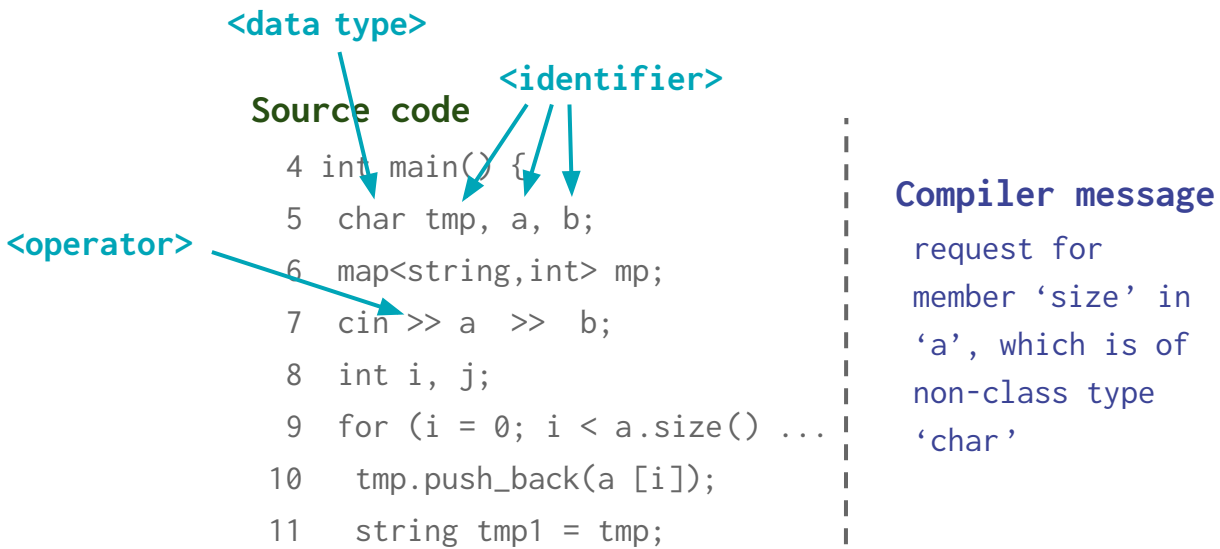
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

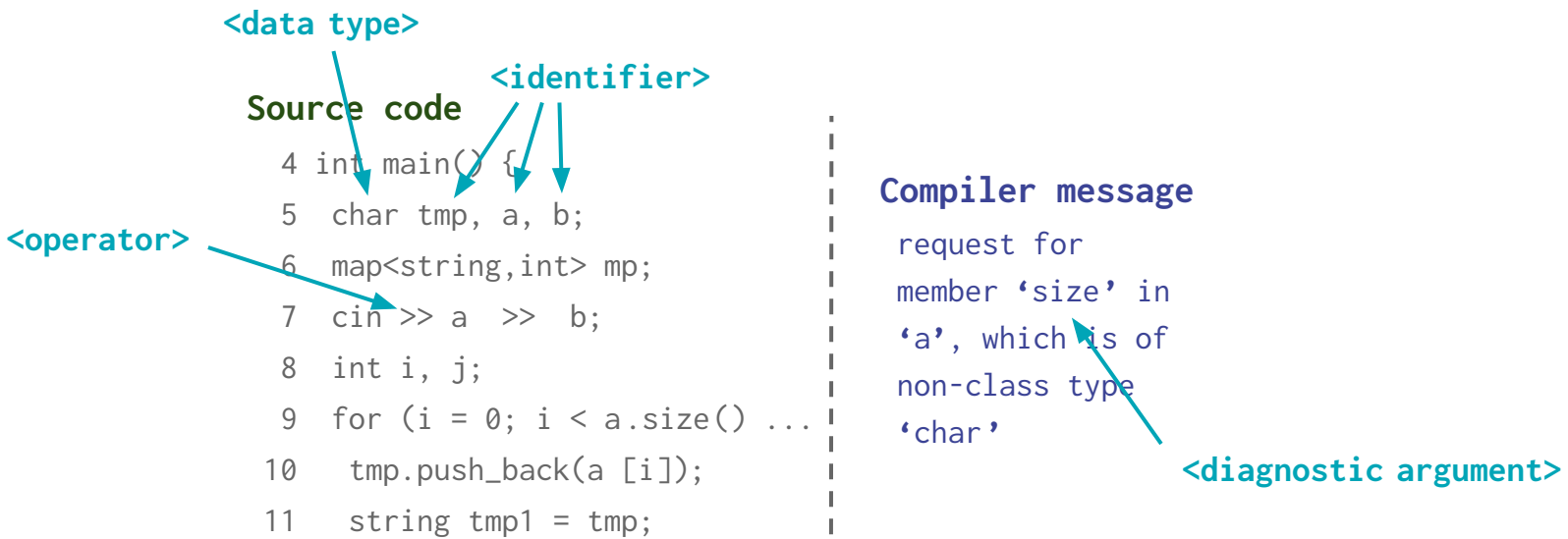
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

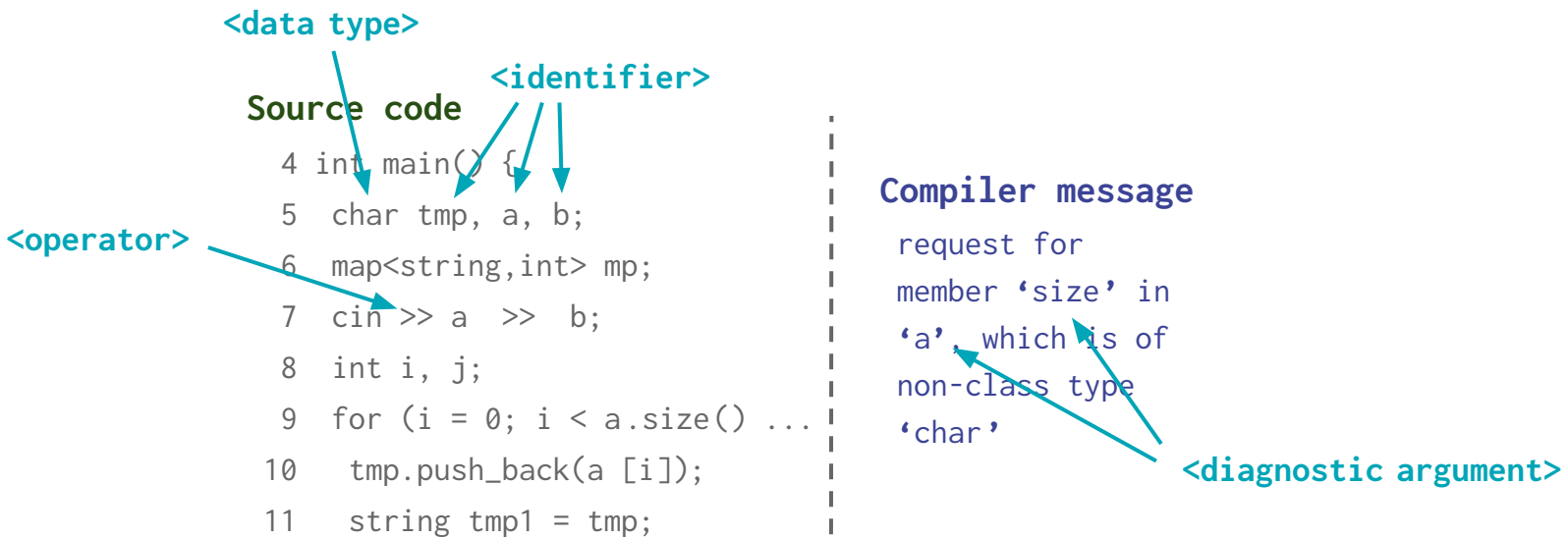
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

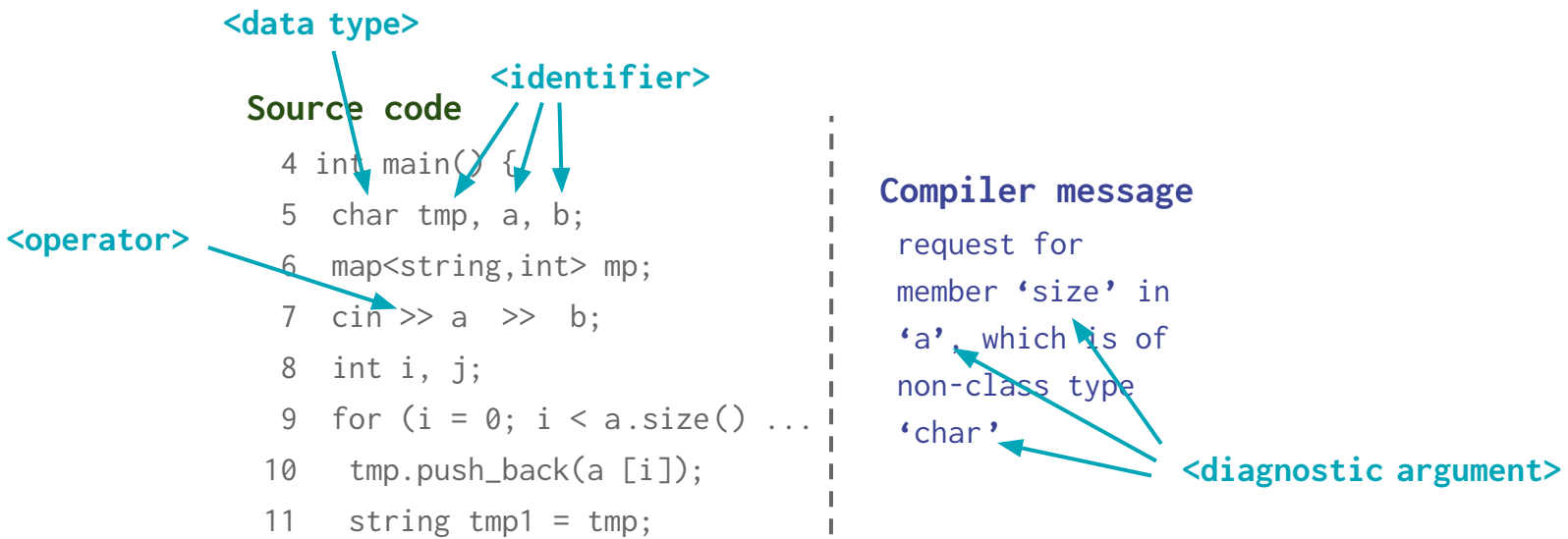
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

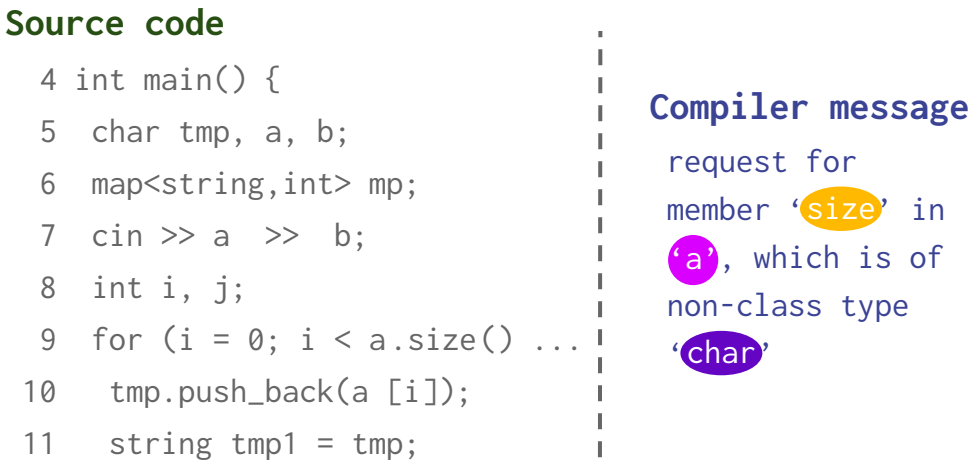
- Recognize token types



1. Reasoning via program-feedback graph

How to construct graph?

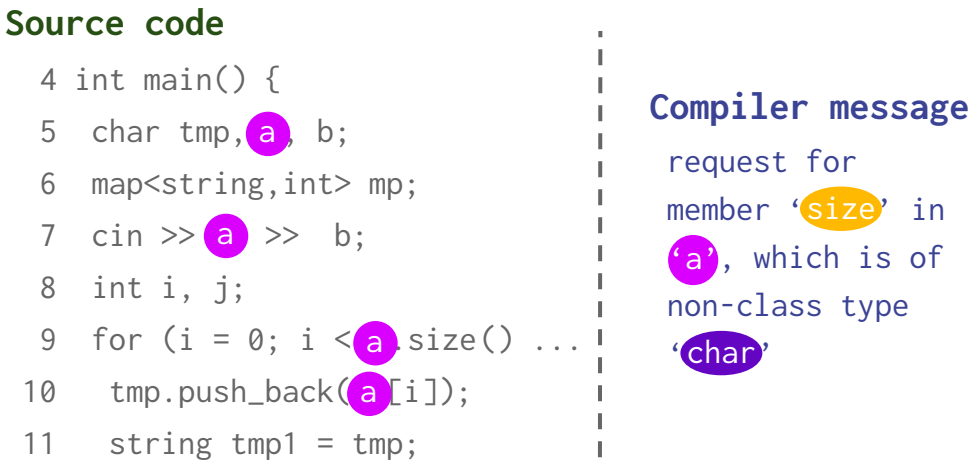
- Recognize token types
- **Nodes**: diagnostic arguments



1. Reasoning via program-feedback graph

How to construct graph?

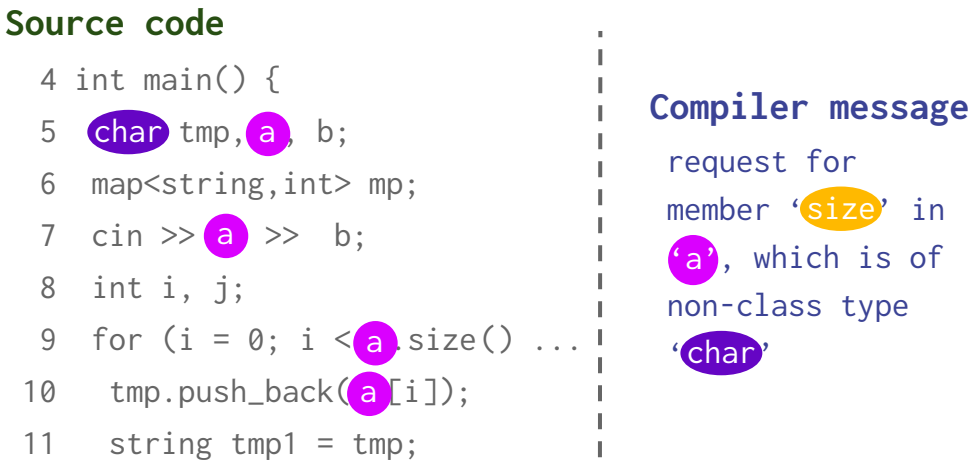
- Recognize token types
- **Nodes**: diagnostic arguments, their occurences



1. Reasoning via program-feedback graph

How to construct graph?

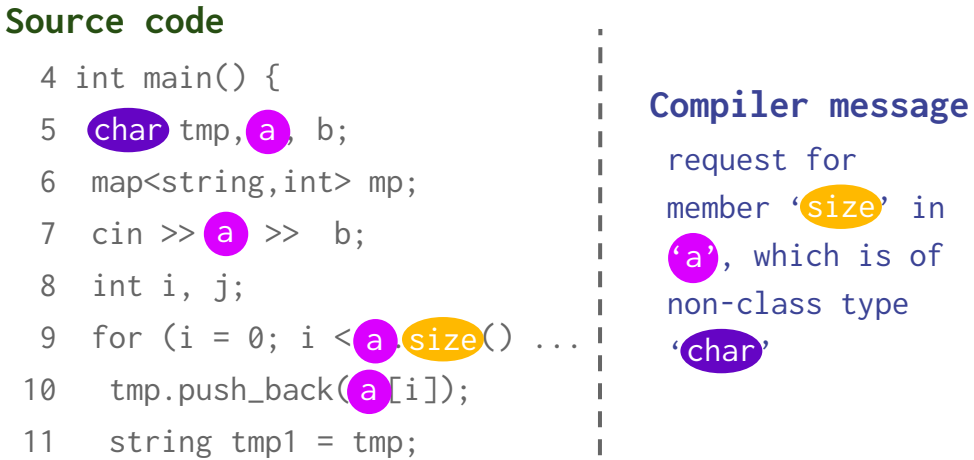
- Recognize token types
- **Nodes**: diagnostic arguments, their occurences



1. Reasoning via program-feedback graph

How to construct graph?

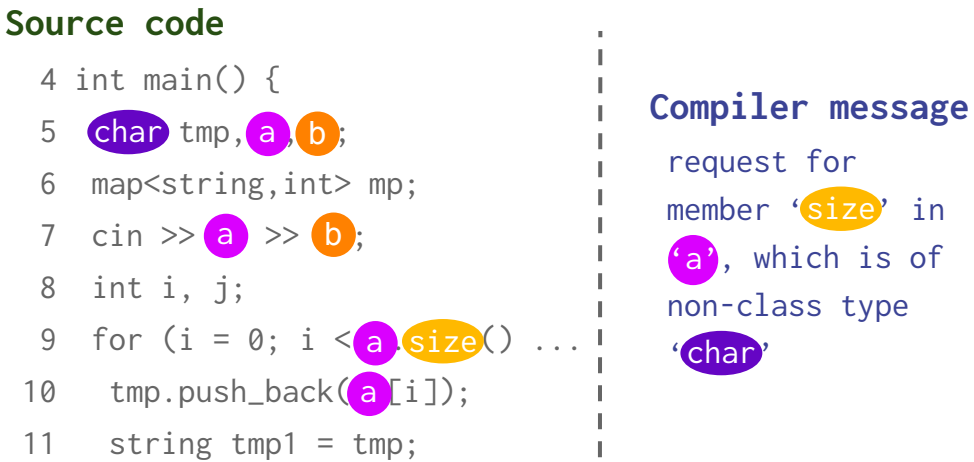
- Recognize token types
- **Nodes**: diagnostic arguments, their occurences



1. Reasoning via program-feedback graph

How to construct graph?

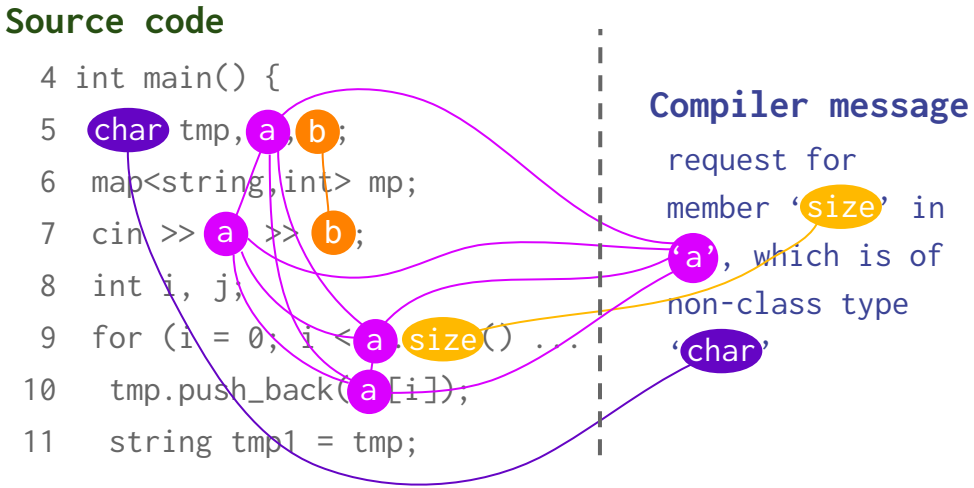
- Recognize token types
- **Nodes**: diagnostic arguments, their occurences, and all identifiers



1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types
- **Nodes**: diagnostic arguments, their occurences, and all identifiers
- **Edges**: connect identical tokens to capture semantic correspondence



1. Reasoning via program-feedback graph

Model

- Initial encoding
- Graph attention
- Recontextualization
- Decoding

1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   ‘size’ …
```

1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Line 1

Compiler message

```
9: request for member
   ‘size’ …
```

1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

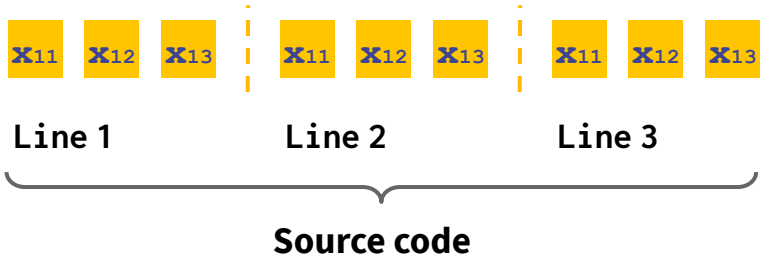
```
9: request for member
   ‘size’ …
```

1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```



Compiler message

```
9: request for member
   'size' …
```

1. Reasoning via program-feedback graph

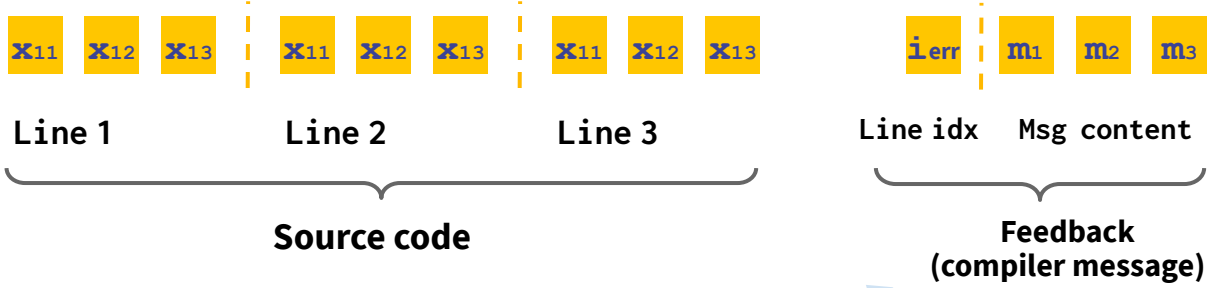
Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

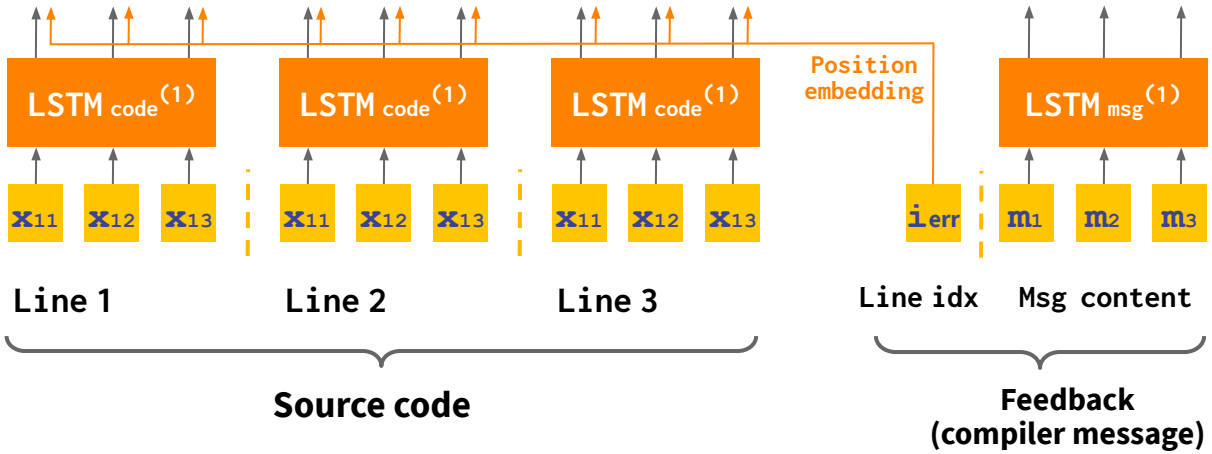
Model (Initial encoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

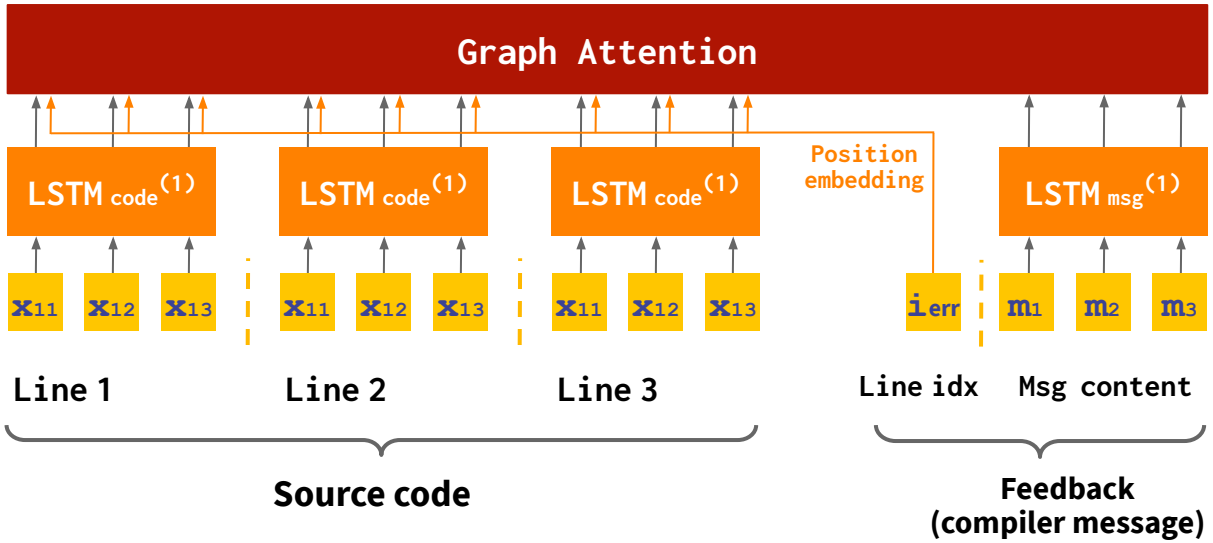
Model (Graph attention)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

Model (Graph attention)

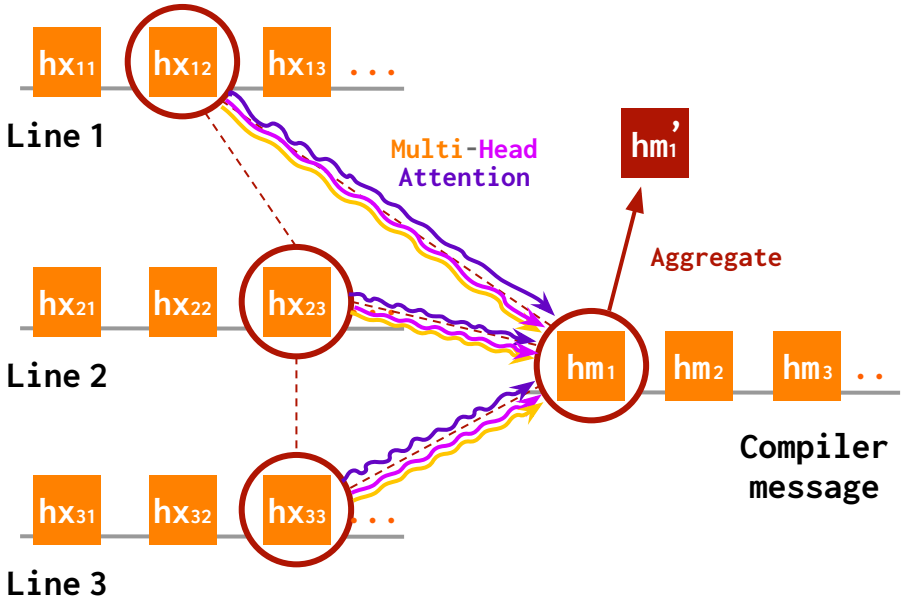
- Message passing across tokens with **long-range dependencies**

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



Program-Feedback Graph

1. Reasoning via program-feedback graph

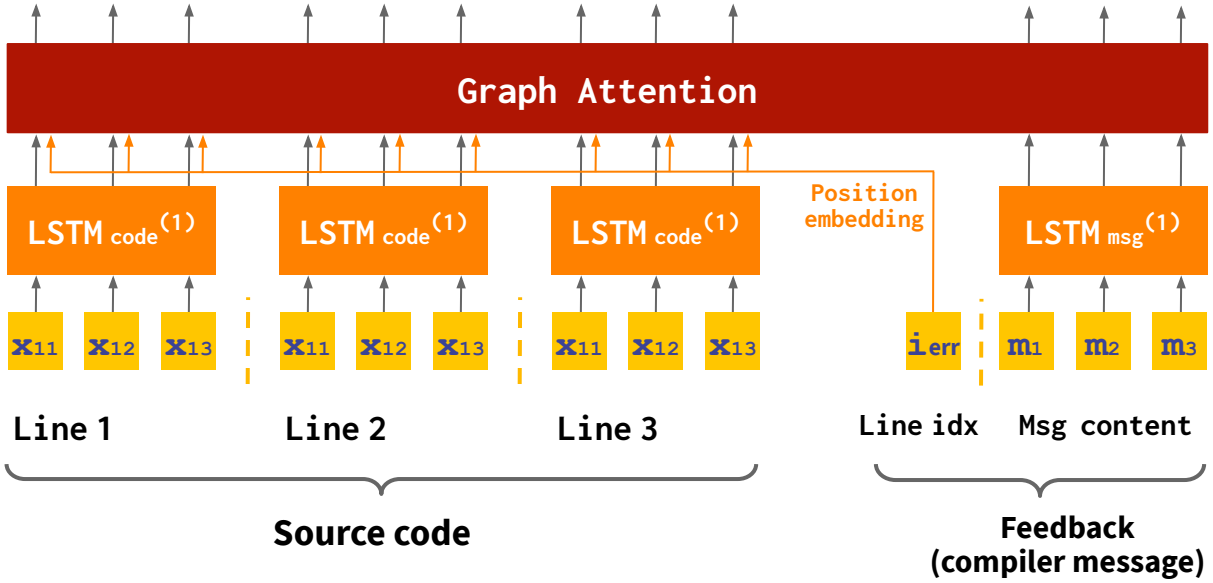
Model (Graph attention)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

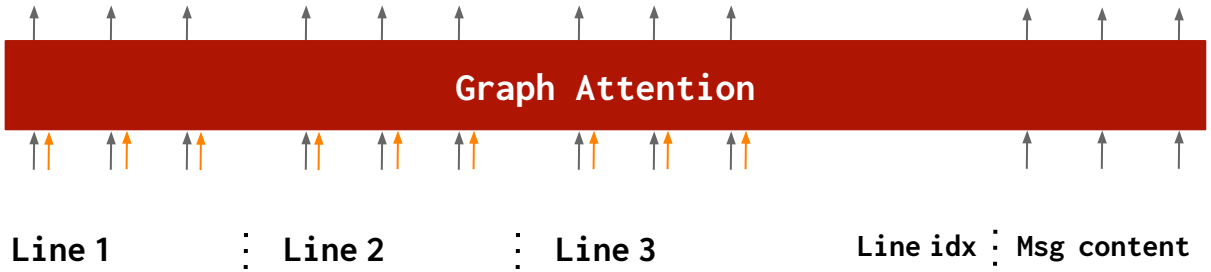
Model (Recontextualization)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   ‘size’ …
```



1. Reasoning via program-feedback graph

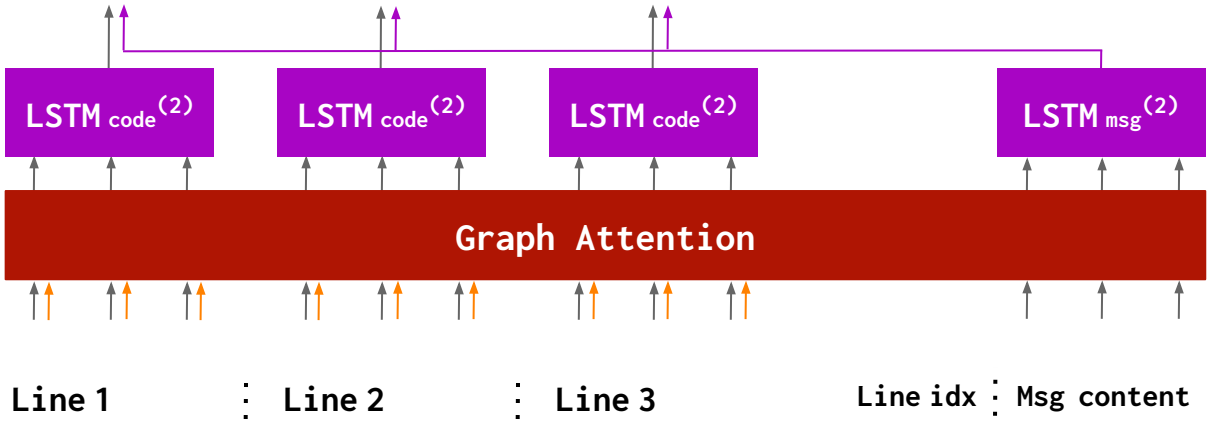
Model (Recontextualization)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

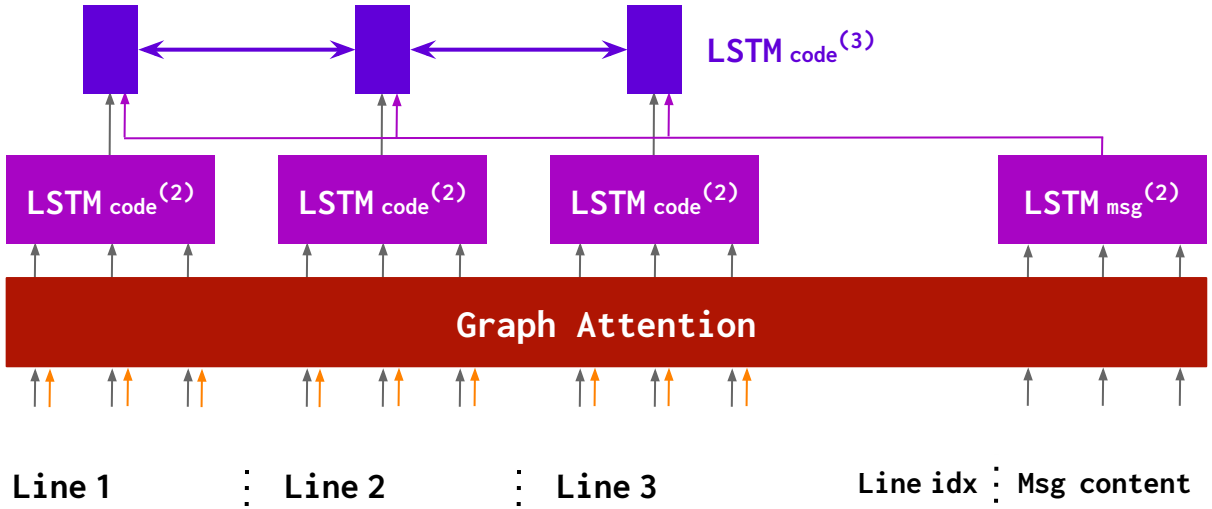
Model (Recontextualization)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

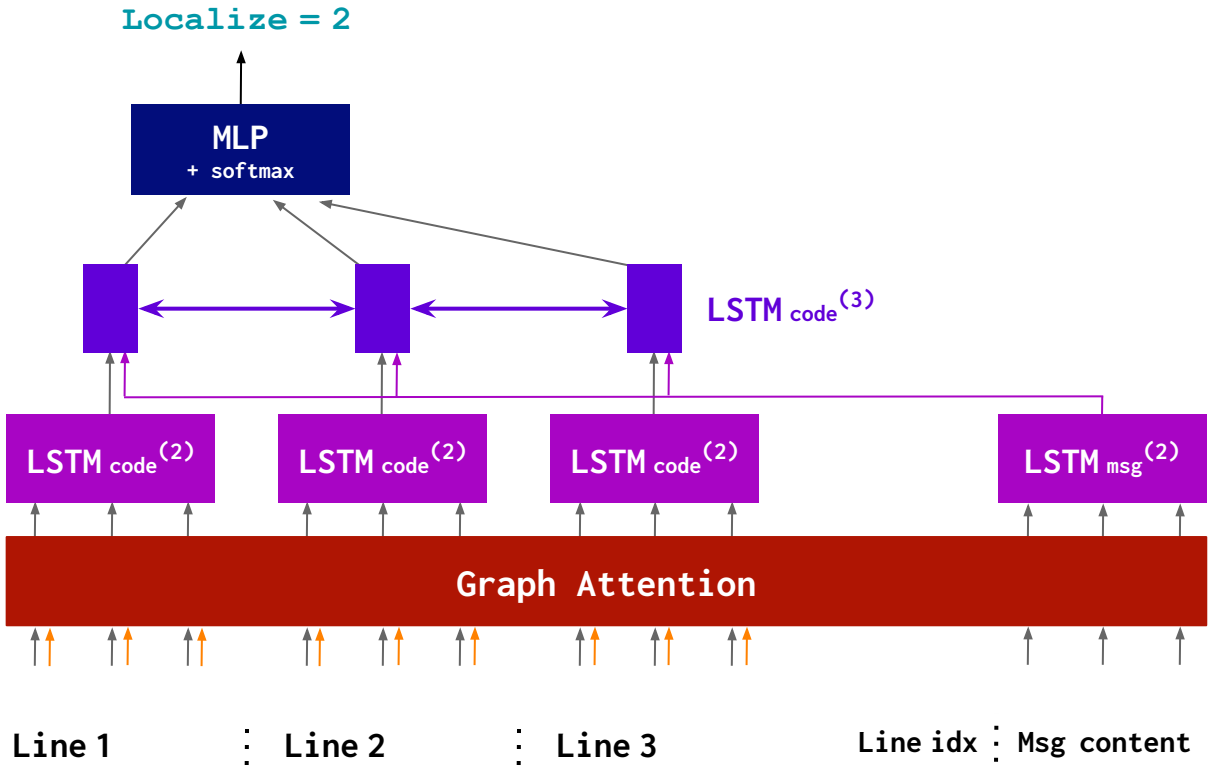
Model (Decoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

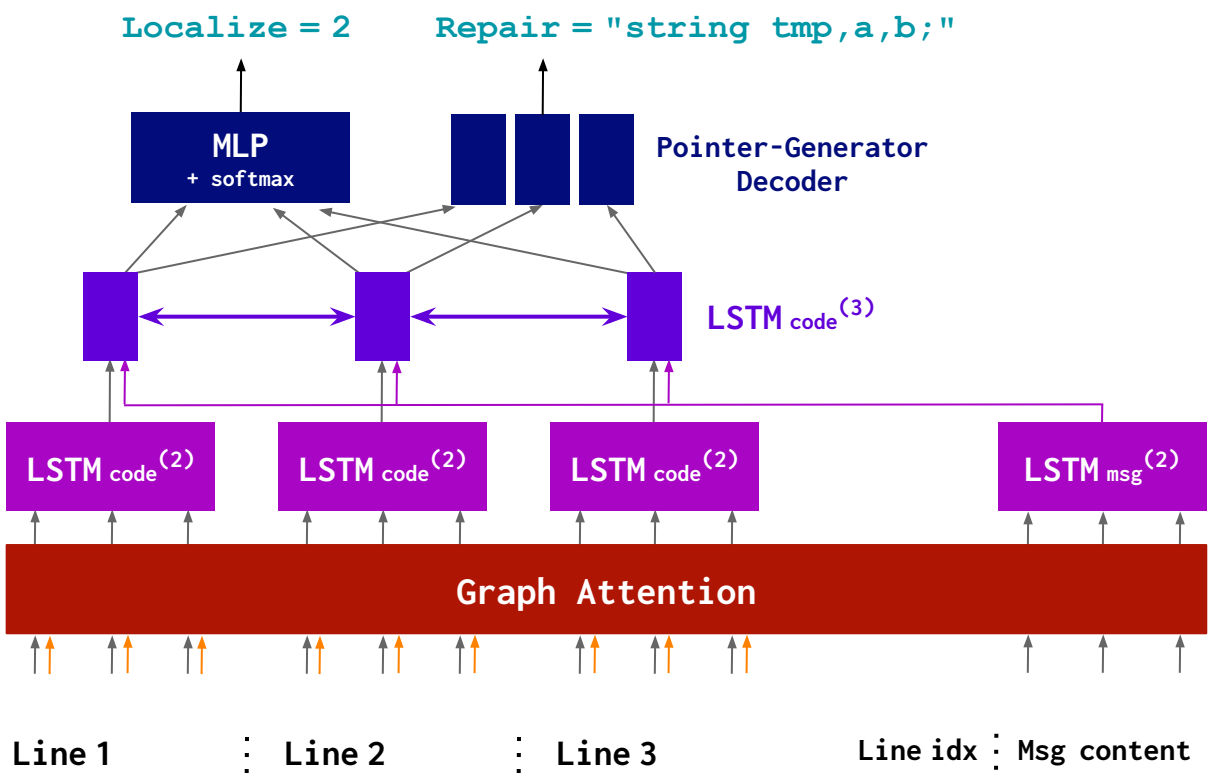
Model (Decoding)

Source code

```
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
```

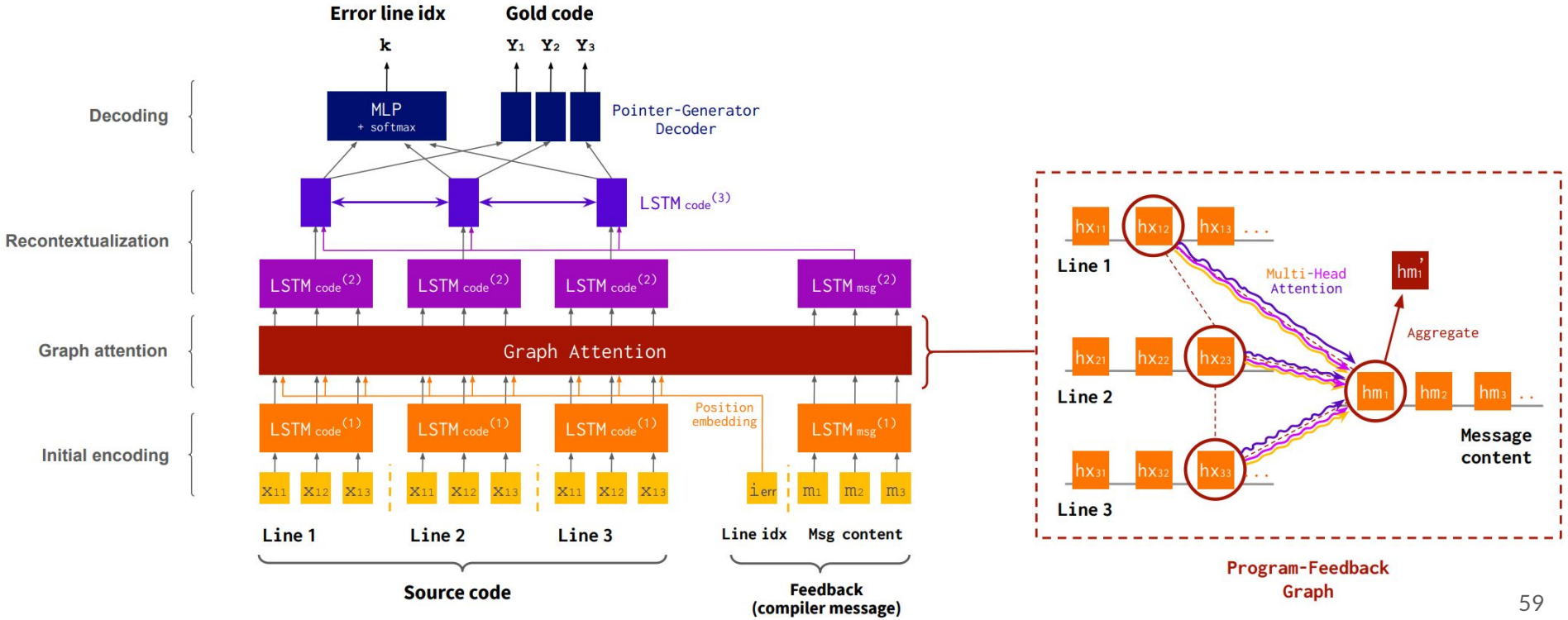
Compiler message

```
9: request for member
   'size' …
```



1. Reasoning via program-feedback graph

Model overview



2. Self-supervised learning

2. Self-supervised learning

Why?

- Labeled datasets of program repair are small (10-100K examples)
- Vast amount of **unlabeled programs** available online
- Can we leverage them to improve learning?

> 30M repos

>> 1M submissions

2. Self-supervised learning

Our idea (outline)

Step 1. Collect unlabeled, working programs **y**

Step 2. Design (randomized) program corruption procedure ***P***

Step 3. Corrupt and get diagnostic feedback (e.g. run compiler)

⇒ **Extra training data**: <broken code **x**, feedback ***f***, fixed code **y**>

Step 4. Use them for pre-training

2. Self-supervised learning

1. **Collect unlabeled programs**

- Our target tasks (DeepFix & SPoC) are in C/C++
- Collect 300K working C++ programs from `codeforces.com`

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`				
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`				
Identifier **undeclared**	`@@@ was not declared`				
Others	`'else' without a previous 'if'` `no matching function for call to...`				

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
		Experienced			
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`	9%			
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`	9%			
Identifier **undeclared**	`@@@ was not declared`	62%			
Others	`'else' without a previous 'if'` `no matching function for call to...`	20%			

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
		Experienced	Beginner		
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`	9%	48% • 37% • 11		
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`	9%	5%		
Identifier **undeclared**	`@@@ was not declared`	62%	33%		
Others	`'else' without a previous 'if'` `no matching function for call to...`	20%	14%		

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
		Experienced	Beginner	SPoC	
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`	9%	48% • 37% • 11	35% • 29% • 6%	
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`	9%	5%	18%	
Identifier **undeclared**	`@@@ was not declared`	62%	33%	31%	
Others	`'else' without a previous 'if'` `no matching function for call to...`	20%	14%	16%	

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
		Experienced	Beginner	SPoC	**Avg.**
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`	9%	48% • 37% • 11	35% • 29% • 6%	**30%** • **23%** • **7%**
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`	9%	5%	18%	**11%**
Identifier **undeclared**	`@@@ was not declared`	62%	33%	31%	**42%**
Others	`'else' without a previous 'if'` `no matching function for call to...`	20%	14%	16%	**17%**

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors (know your enemy)

Error type	Common compiler messages	Statistics			
		Experienced	Beginner	SPoC	**Avg.**
Expected ... • operator/punctuator • primary expression	`expected @@@ (e.g. expected ';' before...)` `missing @@@ (e.g. missing ")`	9%	48% • 37% • 11	35% • 29% • 6%	**30%** • **23%** • **7%**
Identifier **type**	`redeclaration/conflicting declaration` `invalid conversion from <type> to <type>`	9%	5%	18%	**11%**
Identifier **undeclared**	`@@@ was not declared`	62%	33%	31%	**42%**
Others	`'else' without a previous 'if'` `no matching function for call to...`	20%	14%	16%	**17%**

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]

2. Self-supervised learning

2. **How to design corruption procedure P?**
   - Look at common errors
   - Design perturbation modules M to cause those errors

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors
- Design perturbation modules *M* to cause those errors

Perturbation module	Example
Syntax `(delete/insert/replace operators .;(){}'"+, etc.)`	`return 0; }` → `return 0; } }`

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors
- Design perturbation modules *M* to cause those errors

Perturbation module	Example
Syntax (delete/insert/replace **operators** .;(){}'"+, etc.)	`return 0; }` → `return 0; } }`
ID-type (delete/insert/replace **type**)	`string tmp;` → `char tmp;`

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors
- Design perturbation modules *M* to cause those errors

Perturbation module	Example
Syntax (delete/insert/replace **operators** .;(){}'"+, etc.)	`return 0; }` → `return 0; } }`
ID-type (delete/insert/replace **type**)	`string tmp;` → `char tmp;`
ID-typo (delete/insert/replace **IDentifier**)	`int a, b=0, m;` → `int a, m;`

2. Self-supervised learning

2. How to design corruption procedure *P*?

- Look at common errors
- Design perturbation modules *M* to cause those errors

Perturbation module	Example
Syntax (delete/insert/replace **operators** .;(){}'"+, etc.)	`return 0; }` `→ return 0; } }`
ID-type (delete/insert/replace **type**)	`string tmp;` `→ char tmp;`
ID-typo (delete/insert/replace **IDentifier**)	`int a, b=0, m;` `→ int a, m;`
Keyword (delete/insert/replace **keyword/call**)	`if (n >= 0)` `→ while (n >= 0)`

2. Self-supervised learning

2. **Our corruption procedure *P***
   - Look at common errors
   - Design perturbation modules *M* to cause those errors
   - ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

2. Self-supervised learning

2. **Our corruption procedure *P***
   - Look at common errors
   - Design perturbation modules *M* to cause those errors
   - ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

     e.g. `ID-type`, `ID-typo`, `Syntax`.

2. Self-supervised learning

2. Our corruption procedure *P*

- Look at common errors
- Design perturbation modules *M* to cause those errors
- ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

  e.g. `ID-type`, `ID-typo`, `Syntax`.

Working code

```
 5  int i, n;
 6  string A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

2. Self-supervised learning

2. Our corruption procedure *P*

- Look at common errors
- Design perturbation modules *M* to cause those errors
- ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

  e.g. `ID-type`, `ID-typo`, `Syntax`.

Working code

```
 5  int i, n;
 6  string A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

Perturbed 1

```
 5  int i, n;
 6  char A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

2. Self-supervised learning

2. Our corruption procedure *P*

- Look at common errors
- Design perturbation modules *M* to cause those errors
- ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

  e.g. `ID-type`, `ID-typo`, `Syntax`.

Working code

```
 5  int i, n;
 6  string A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

Perturbed 1

```
 5  int i, n;
 6  char A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

Perturbed 2

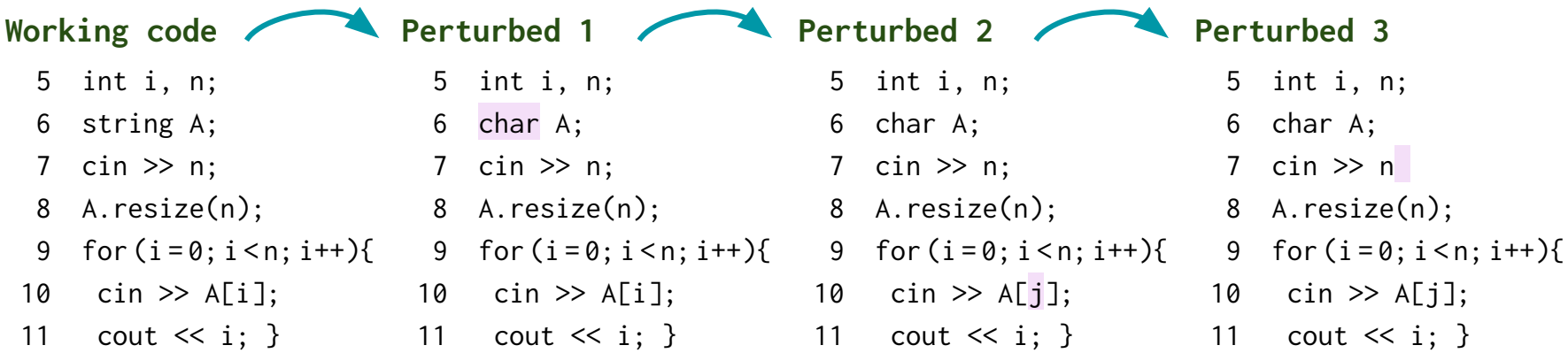
```
 5  int i, n;
 6  char A;
 7  cin >> n;
 8  A.resize(n);
 9  for(i=0;i<n;i++){
10   cin >> A[j];
11   cout << i; }
```

2. Self-supervised learning

2. Our corruption procedure *P*

- Look at common errors
- Design perturbation modules *M* to cause those errors
- ***P***: Sample 1-5 modules from *M*, and apply to program sequentially

  e.g. `ID-type`, `ID-typo`, `Syntax`.



2. Self-supervised learning

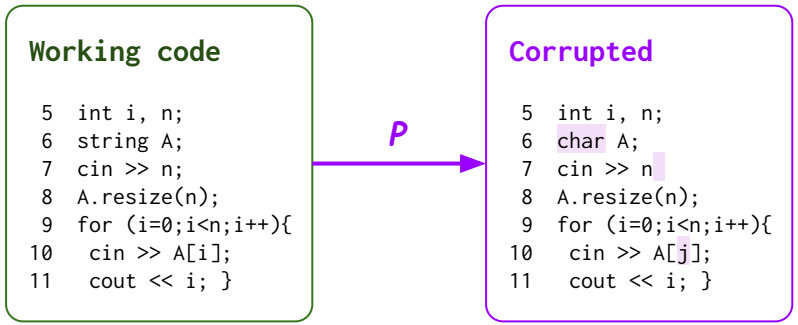
3. **Prepare pre-training data**
   - 300K working programs from **codeforces.com**

Working code

```
 5  int i, n;
 6  string A;
 7  cin >> n;
 8  A.resize(n);
 9  for (i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

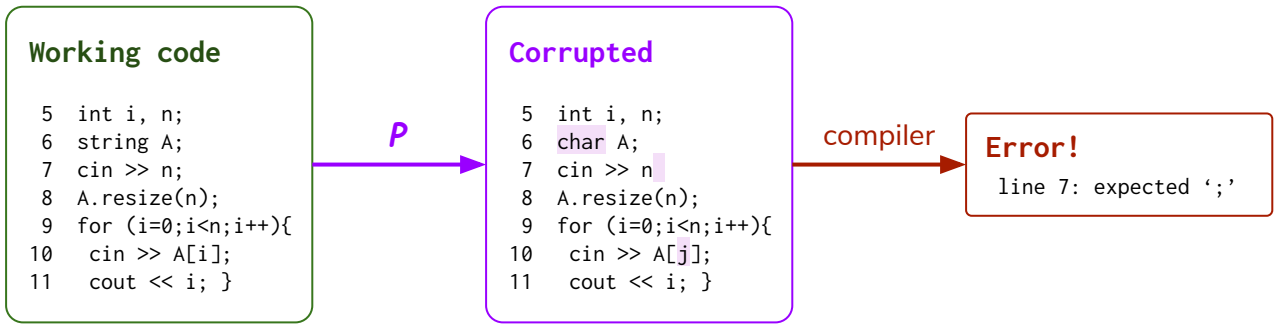
2. Self-supervised learning

3. **Prepare pre-training data**
   - 300K working programs from **codeforces.com**
   - For each program, create corrupted versions by applying ***P***



2. Self-supervised learning

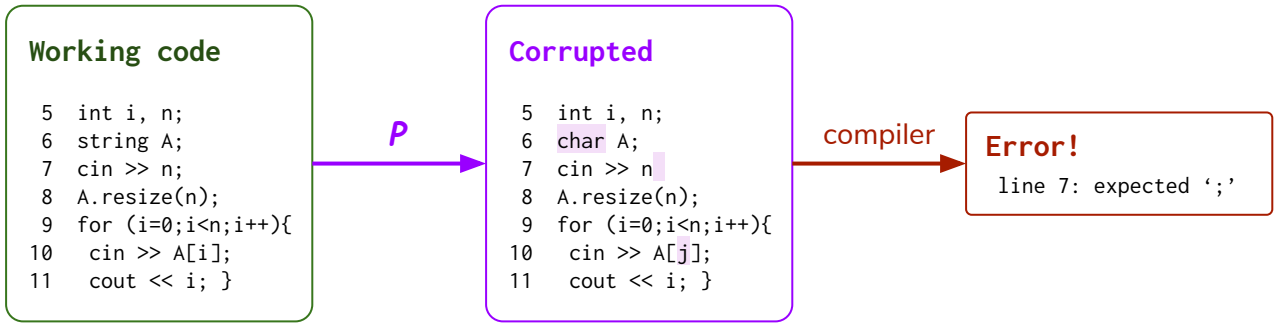
3. **Prepare pre-training data**
   - 300K working programs from **codeforces.com**
   - For each program, create corrupted versions by applying ***P***



2. Self-supervised learning

3. **Prepare pre-training data**
   - 300K working programs from **`codeforces.com`**
   - For each program, create corrupted versions by applying ***P***

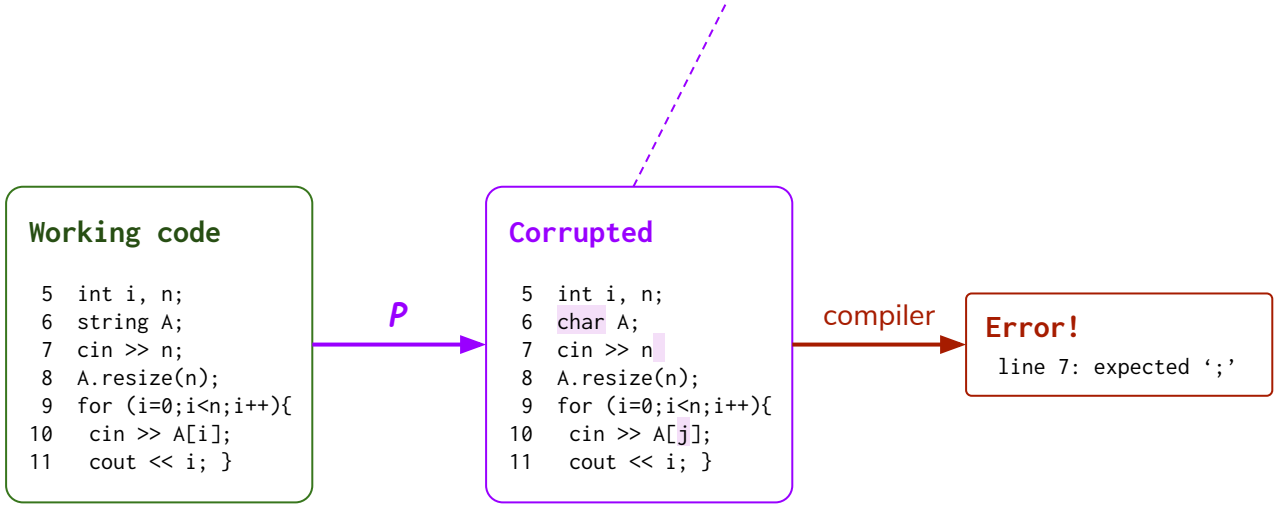
     ⇒ New program repair examples: <broken code, feedback, fixed code>



2. Self-supervised learning

3. **Prepare pre-training data**
   - 300K working programs from **`codeforces.com`**
   - For each program, create corrupted versions by applying ***P***

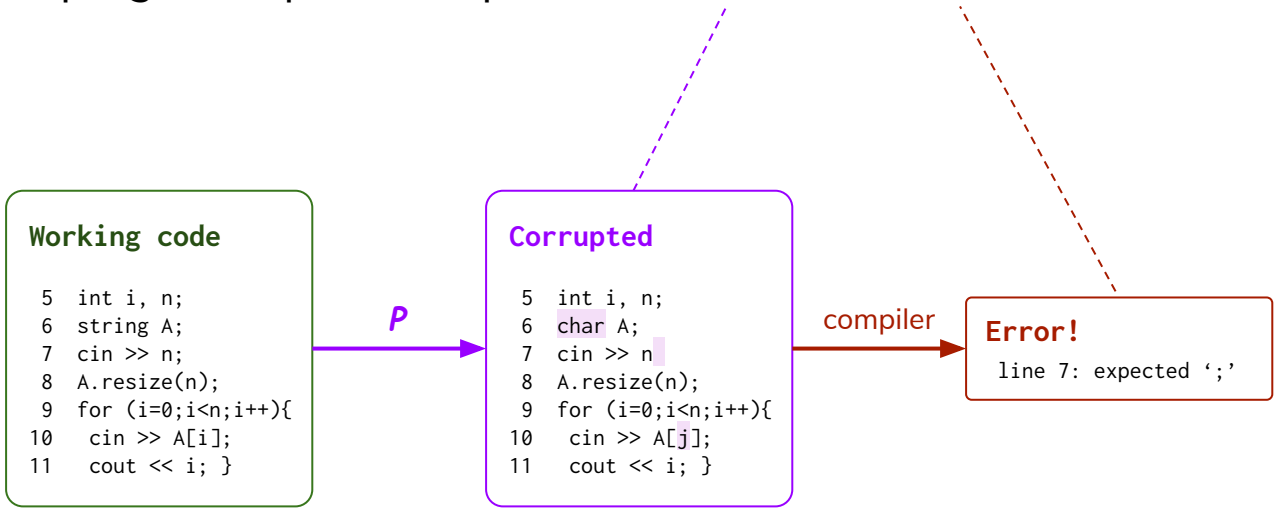
     ⇒ New program repair examples: <broken code, feedback, fixed code>



2. Self-supervised learning

3. **Prepare pre-training data**
   - 300K working programs from **codeforces.com**
   - For each program, create corrupted versions by applying ***P***

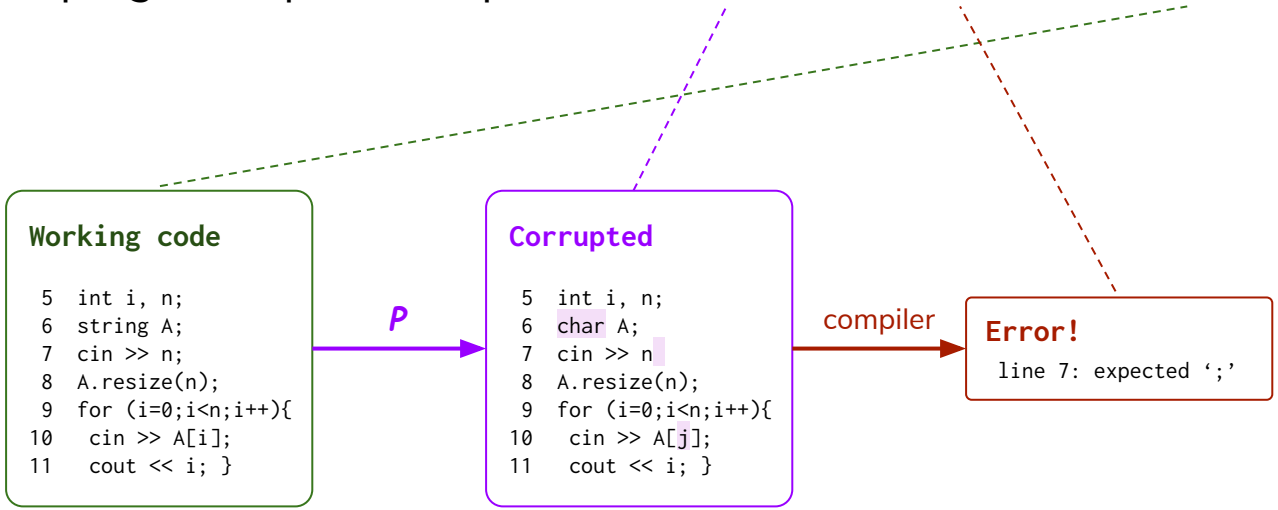
     ⇒ New program repair examples: <broken code, feedback, fixed code>



2. Self-supervised learning

3. **Prepare pre-training data**
   - 300K working programs from **codeforces.com**
   - For each program, create corrupted versions by applying ***P***

     ⇒ New program repair examples: <broken code, feedback, fixed code>



2. Self-supervised learning

What's interesting?

- Typically, pre-training **task** ≠ target **task** (e.g. masked LM v.s. QA)
- Here, **targeted** pre-training (pre-training **task** = target **task** = program repair)
  - More direct pre-training structure
  - **Data** distributions can be different between pre-training & target

Evaluation 1: DeepFix

Evaluation 1: DeepFix

Task

- Repair C programs
- May have **multiple error lines**
- Apply repair model iteratively (up to 5 times)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int pow(int a, int b);
4  int main(){
5   int n;
6   scanf("%d",&n);
7   int i, j;
8   for(i=1;i<=n;i++){
9    for(j=0;j<=n;j++){
10    if(j<i){
11     printf("%d ",pow(i,j));}}
12   printf("\n");}
13  return 0;}
```

[Gupta et al., 17]

Evaluation 1: DeepFix

Our model outputs

Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Evaluation 1: DeepFix

Our model outputs

Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

```
line 9: ‘i’ undeclared
```

Evaluation 1: DeepFix

Our model outputs

DrRepair

Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Attempt 1

```
 4 int main() {
 5  int n, i;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

```
line 9: ‘i’ undeclared
```

Evaluation 1: DeepFix

Our model outputs

DrRepair

Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

```
line 9: ‘i’ undeclared
```

Attempt 1

```
 4 int main() {
 5  int n, i;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

```
line 12: expected ‘;’ before
‘}’
```

Evaluation 1: DeepFix

Our model outputs

DrRepair

DrRepair

Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

```
line 9: ‘i’ undeclared
```

Attempt 1

```
 4 int main() {
 5  int n, i;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

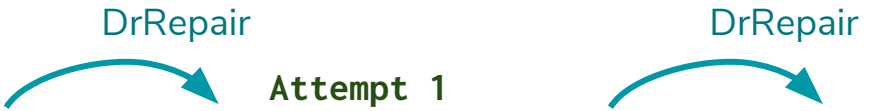
```
line 12: expected ‘;’ before
‘}’
```

Attempt 2

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 ; }
```

Evaluation 1: DeepFix

Our model outputs



Input code

```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

line 9: ‘i’ undeclared

Attempt 1

```
 4 int main() {
 5  int n, i;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

line 12: expected ‘;’ before ‘}’

Attempt 2

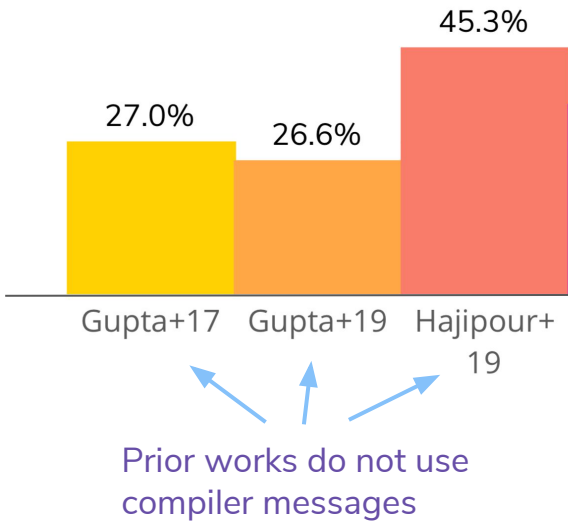
```
 4 int main() {
 5  int n;
 6  int * m[2];
 7  m[0] = malloc(n*sizeof(int));
 8  m[1] = malloc(n*sizeof(int));
 9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 ; }
```

Compiled!!

Evaluation 1: DeepFix

Results

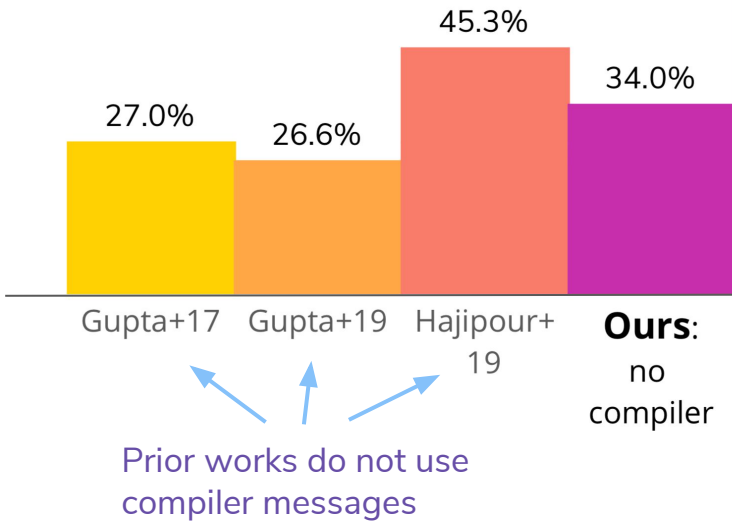
Test (full repair accuracy)



Evaluation 1: DeepFix

Results

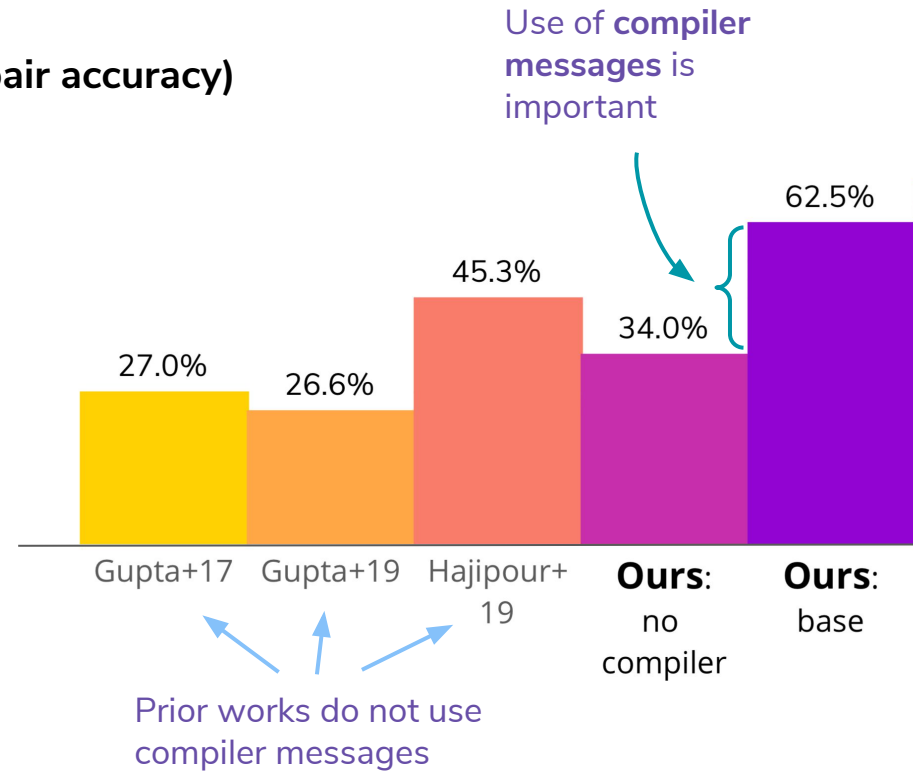
Test (full repair accuracy)



Evaluation 1: DeepFix

Results

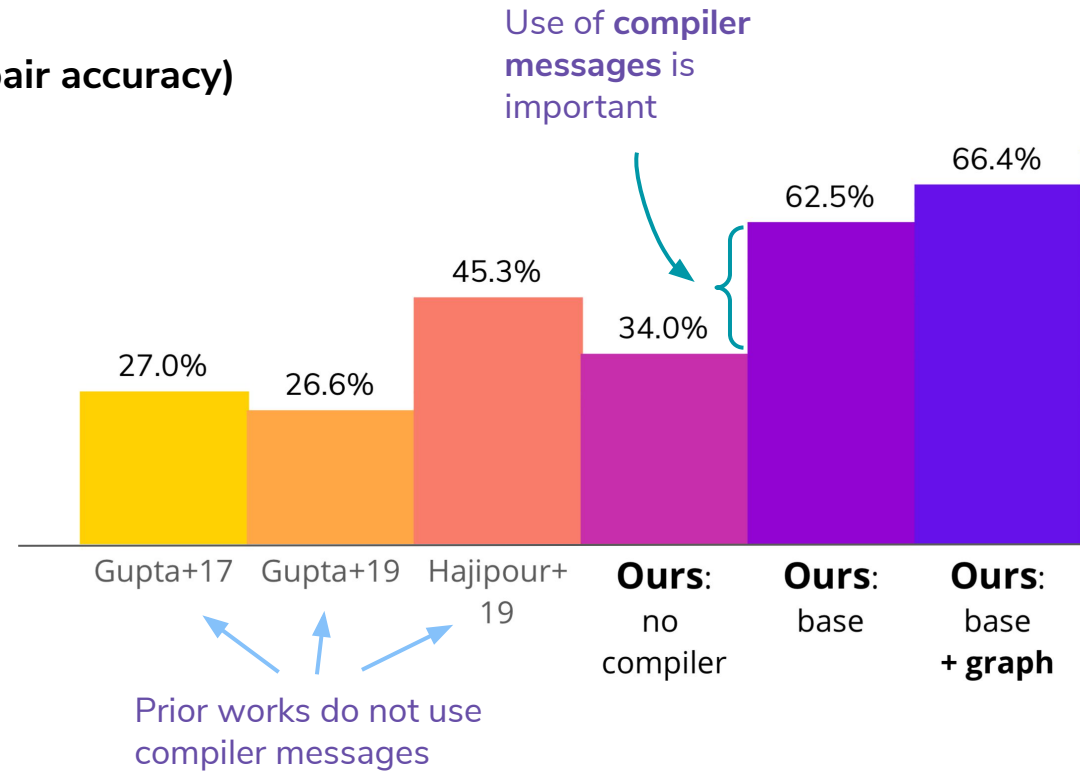
Test (full repair accuracy)



Evaluation 1: DeepFix

Results

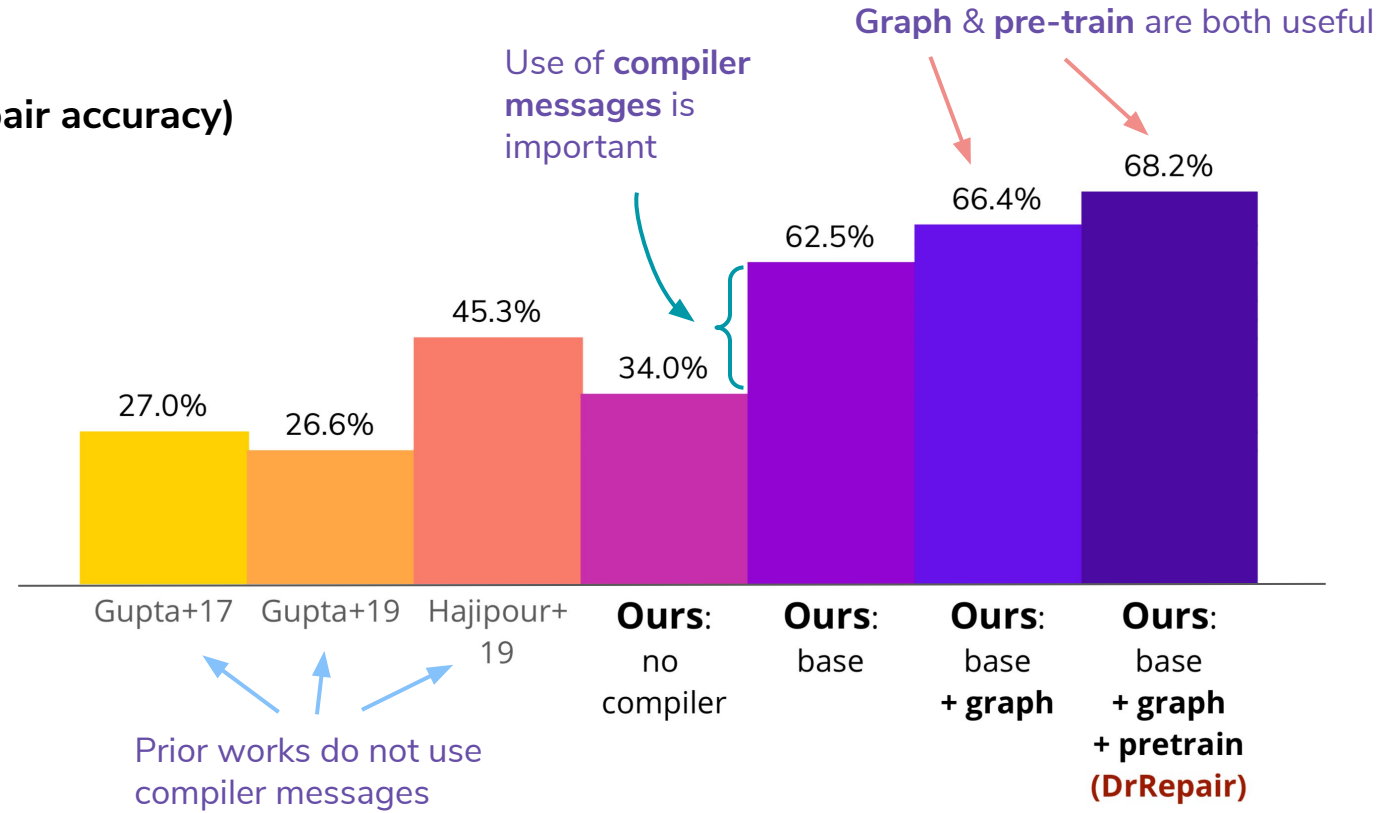
Test (full repair accuracy)



Evaluation 1: DeepFix

Results

Test (full repair accuracy)

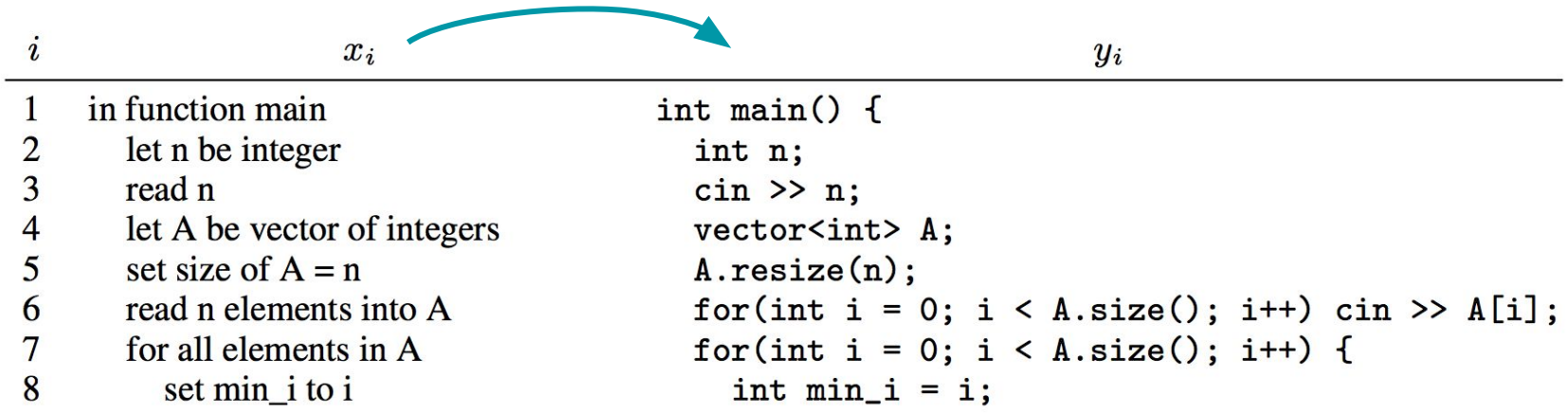


Evaluation 2: SPoC

Evaluation 2: SPoC

Task

- Translate pseudocode into C++ code (**program synthesis**)
- Line-level alignment



i	x_i	y_i
1	in function main	`int main() {`
2	let n be integer	`int n;`
3	read n	`cin >> n;`
4	let A be vector of integers	`vector<int> A;`
5	set size of A = n	`A.resize(n);`
6	read n elements into A	`for(int i = 0; i < A.size(); i++) cin >> A[i];`
7	for all elements in A	`for(int i = 0; i < A.size(); i++) {`
8	set min_i to i	`int min_i = i;`

[Kulal et al., 19]

Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search

Pseudocode

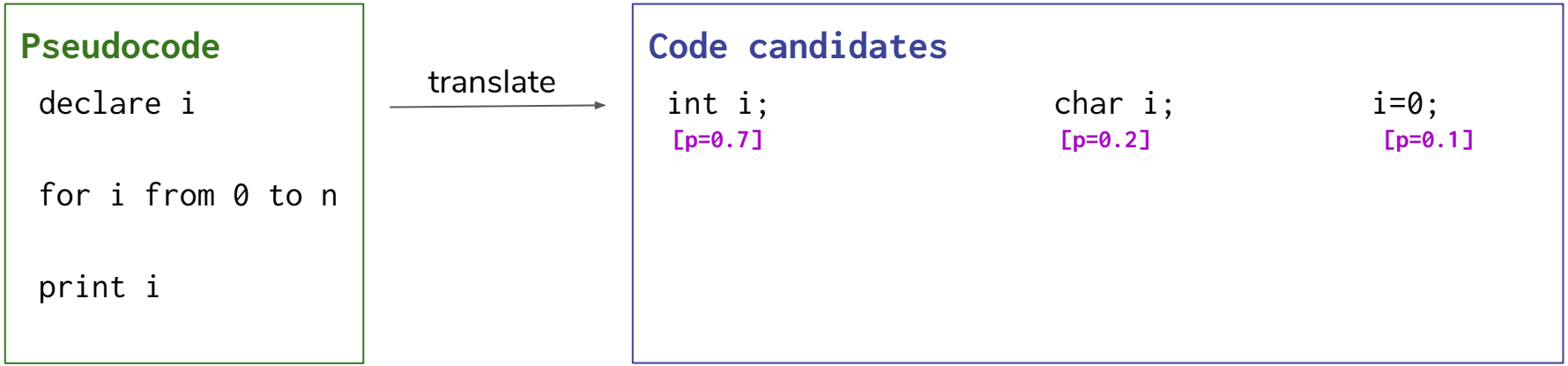
```
declare i

for i from 0 to n

print i
```

Code candidates

Evaluation 2: SPoC

Prior work (Kulal+19)

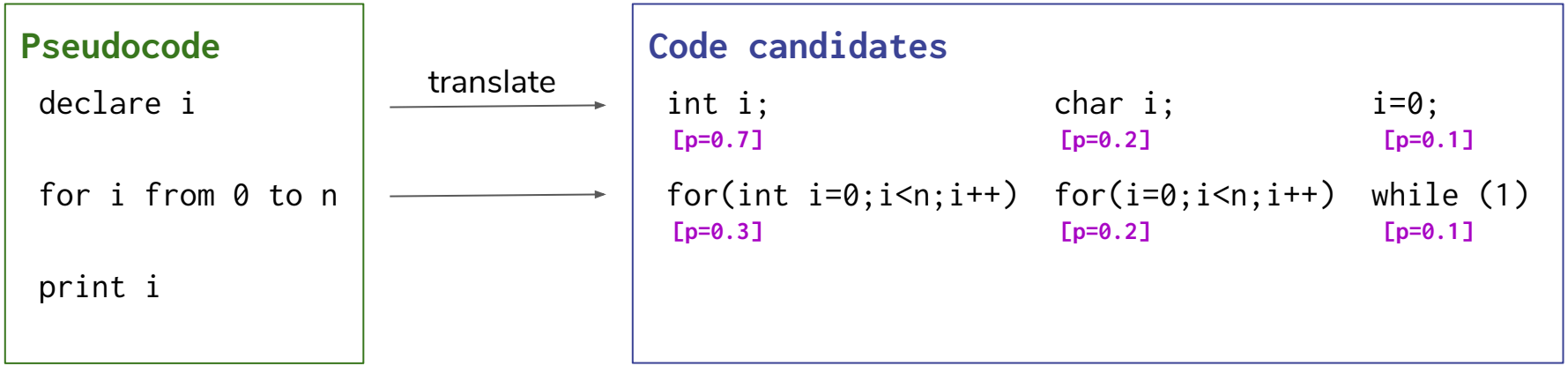
- Stitch line-level translations & search



Evaluation 2: SPoC

Prior work (Kulal+19)

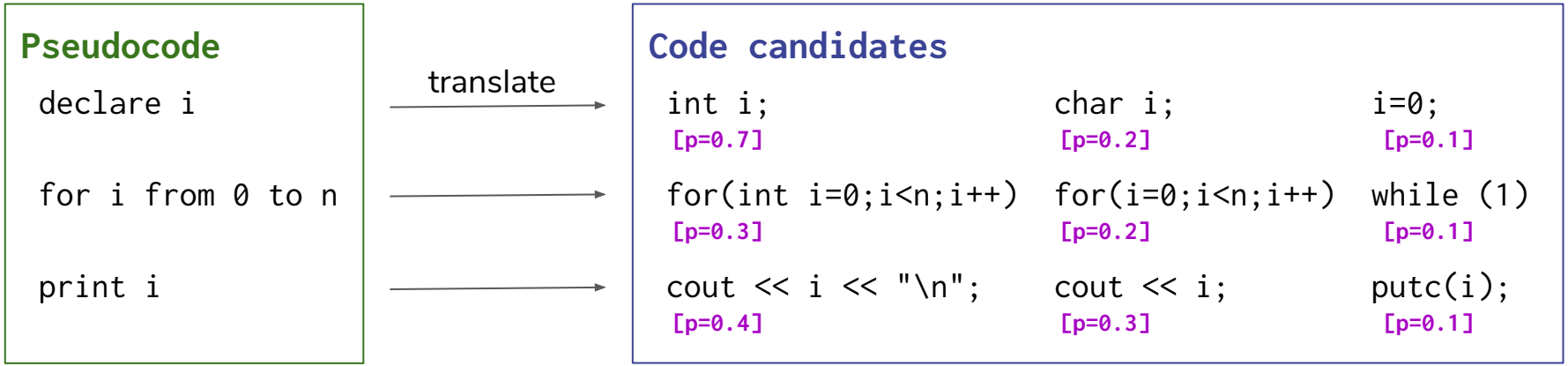
- Stitch line-level translations & search



Evaluation 2: SPoC

Prior work (Kulal+19)

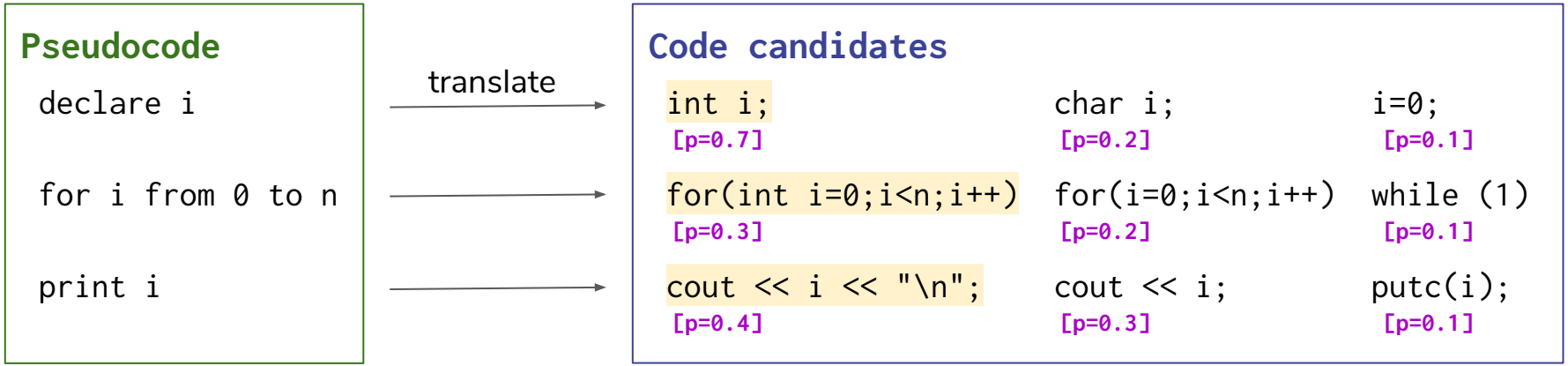
- Stitch line-level translations & search



Evaluation 2: SPoC

Prior work (Kulal+19)

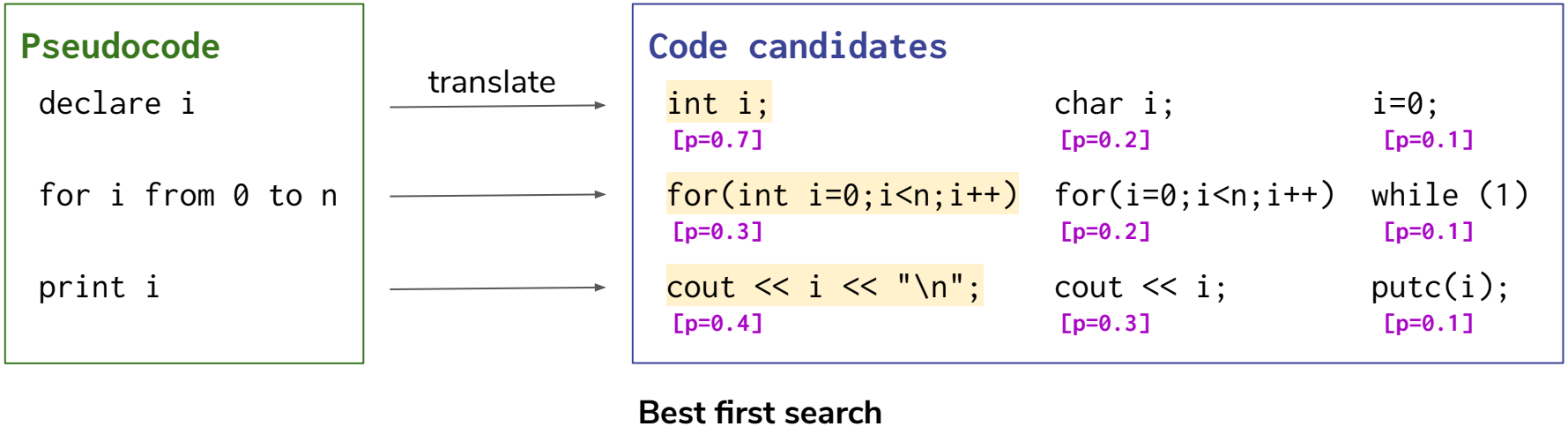
- Stitch line-level translations & search



Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search

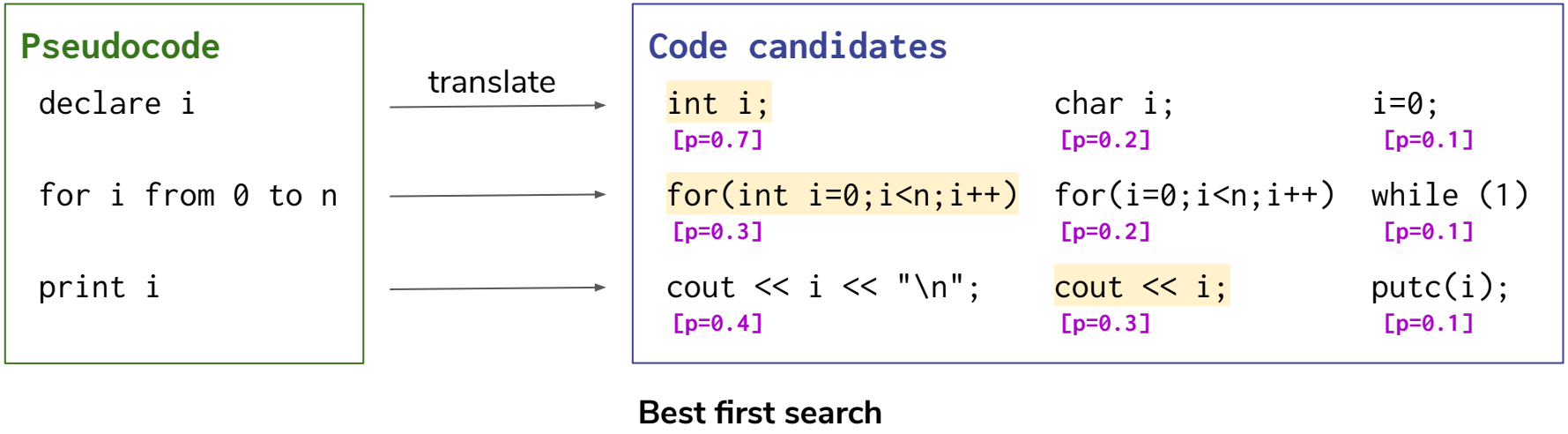


Best first search

Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search

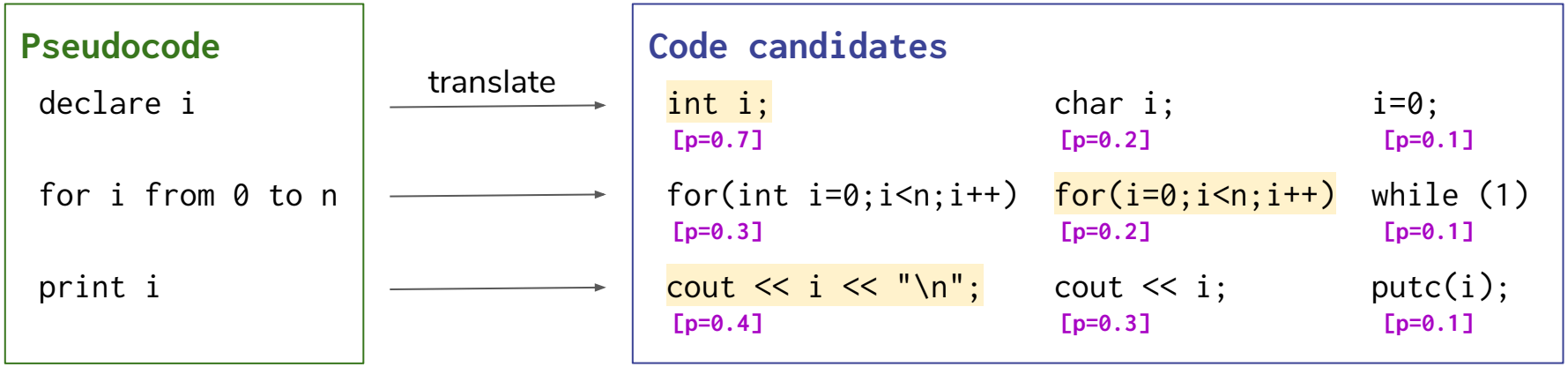


Best first search

Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search



Best first search ...

Evaluation 2: SPoC

Problem

- Line-level translation misses global context

Code candidates

`int i;` [p=0.7]	`char i;` [p=0.2]	`i=0;` [p=0.1]
`for(int i=0;i<n;i++)` [p=0.3]	`for(i=0;i<n;i++)` [p=0.2]	`while (1)` [p=0.1]
`cout << i << "\n";` [p=0.4]	`cout << i;` [p=0.3]	`putc(i);` [p=0.1]

Evaluation 2: SPoC

Problem

- Line-level translation misses global context

Code candidates

`int i;` [p=0.7]	`char i;` [p=0.2]	`i=0;` [p=0.1]
`for(int i=0;i<n;i++)` [p=0.3]	`for(i=0;i<n;i++)` [p=0.2]	`while (1)` [p=0.1]
`cout << i << "\n";` [p=0.4]	`cout << i;` [p=0.3]	`putc(i);` [p=0.1]

→ **Does not compile!**
(redeclaration of ‘i’)

Evaluation 2: SPoC

Problem

- Line-level translation misses global context

Code candidates

`int i;` [p=0.7]	`char i;` [p=0.2]	`i=0;` [p=0.1]
`for(int i=0;i<n;i++)` [p=0.3]	`for(i=0;i<n;i++)` [p=0.2]	`while (1)` [p=0.1]
`cout << i << "\n";` [p=0.4]	`cout << i;` [p=0.3]	`putc(i);` [p=0.1]

→ **Does not compile!**
(redeclaration of ‘i’)

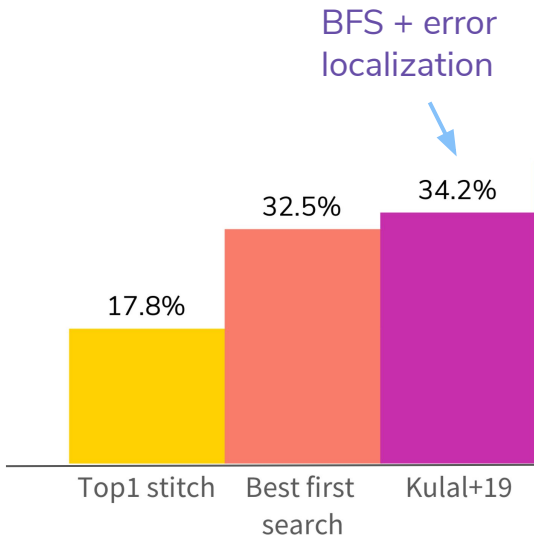
Our solution

- Apply repair model if current candidate program does not compile

Evaluation 2: SPoC

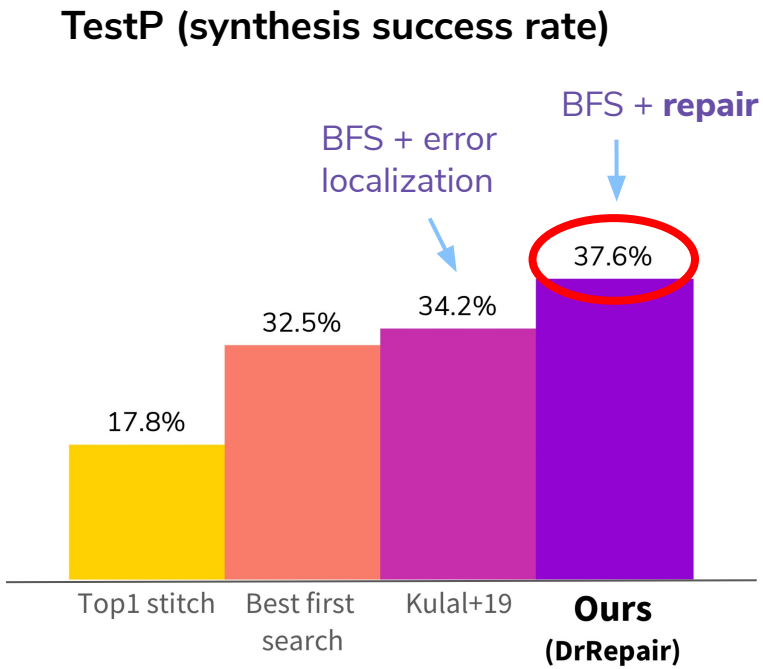
Results

TestP (synthesis success rate)



Evaluation 2: SPoC

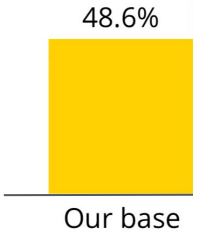
Results



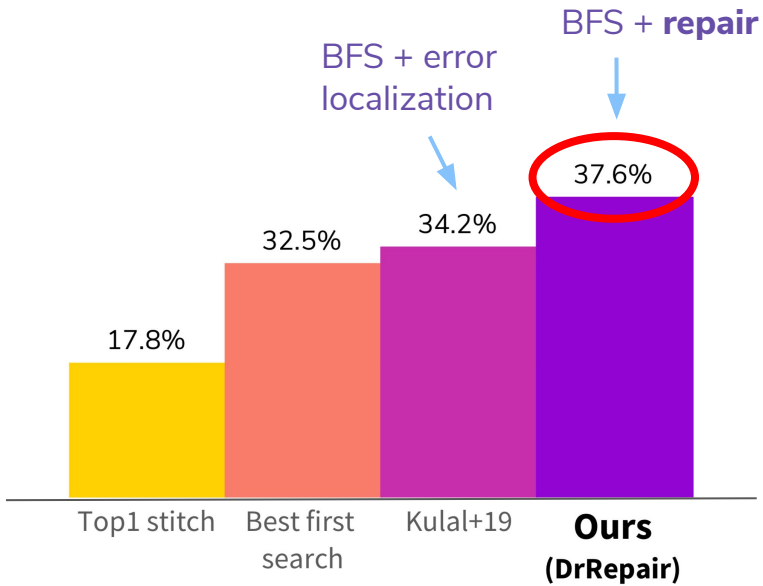
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation



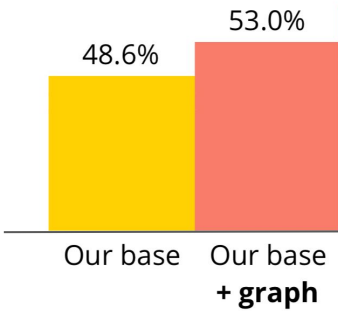
TestP (synthesis success rate)



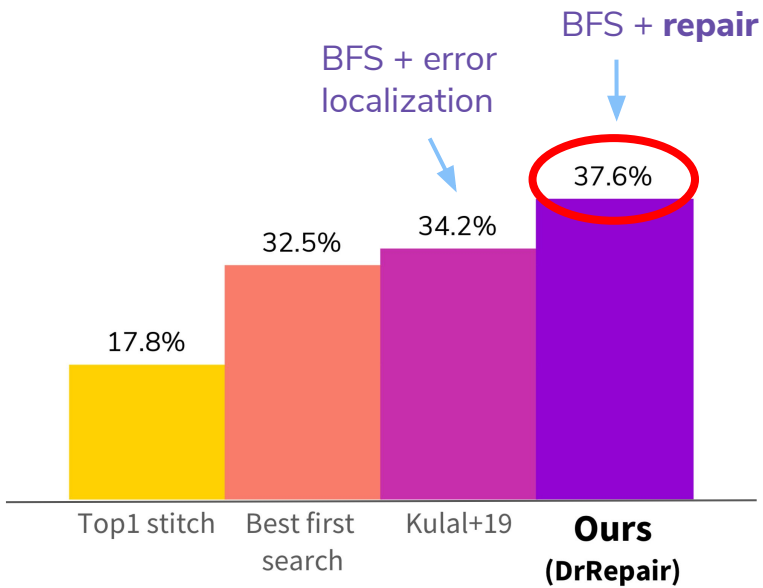
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation



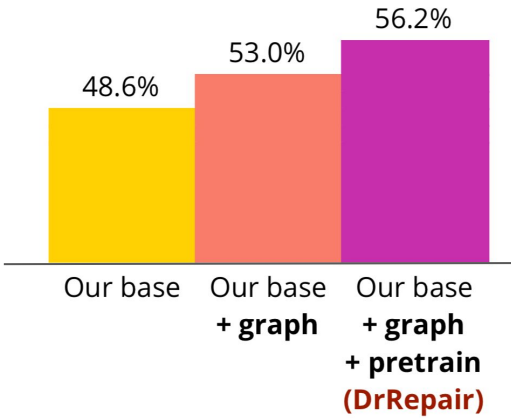
TestP (synthesis success rate)



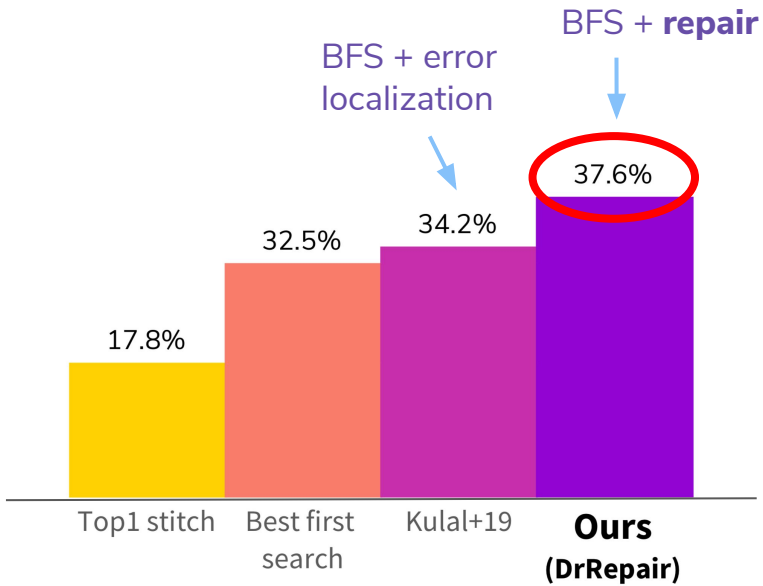
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation



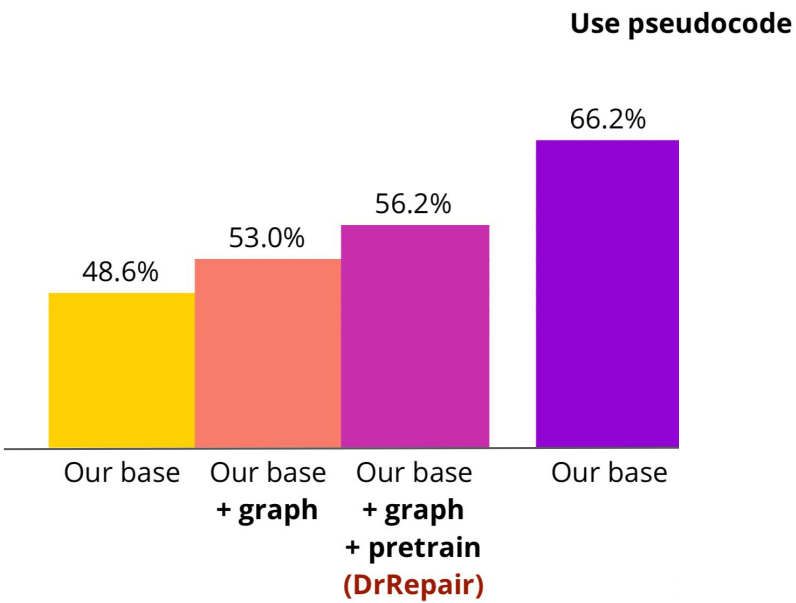
TestP (synthesis success rate)



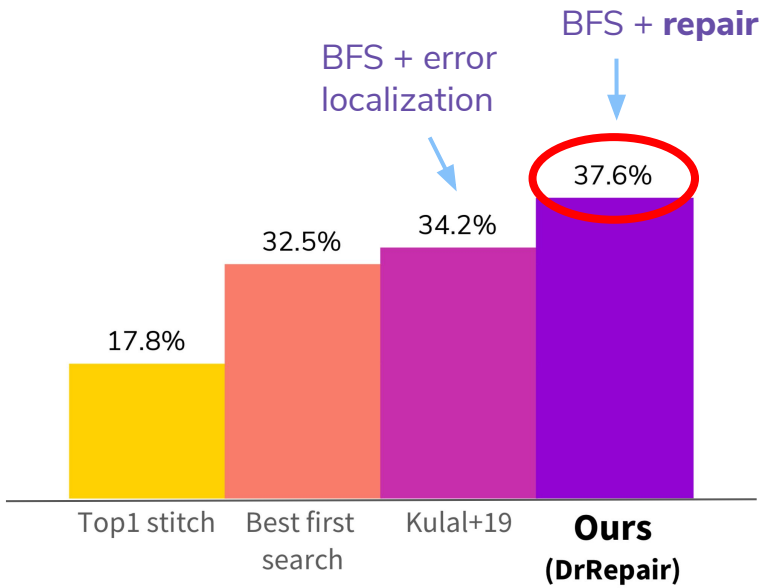
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation



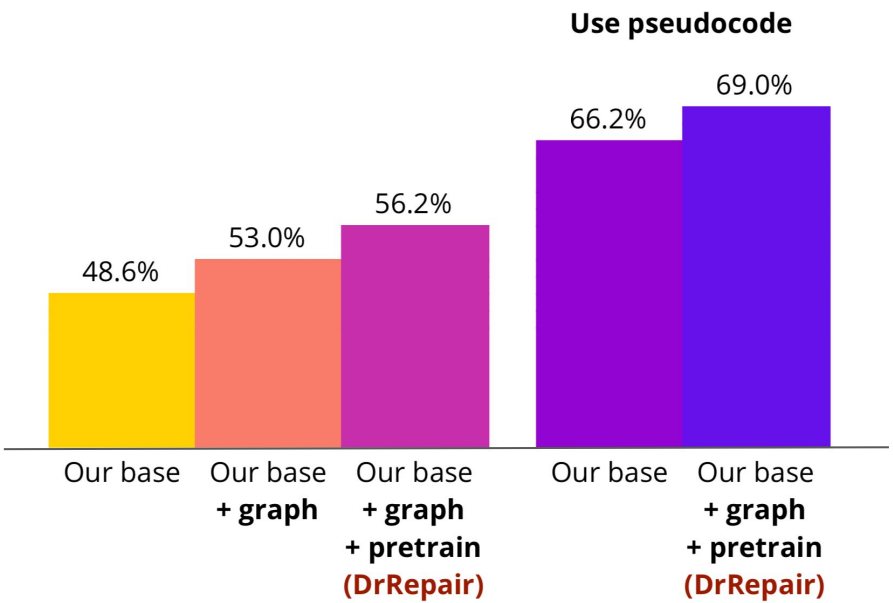
TestP (synthesis success rate)



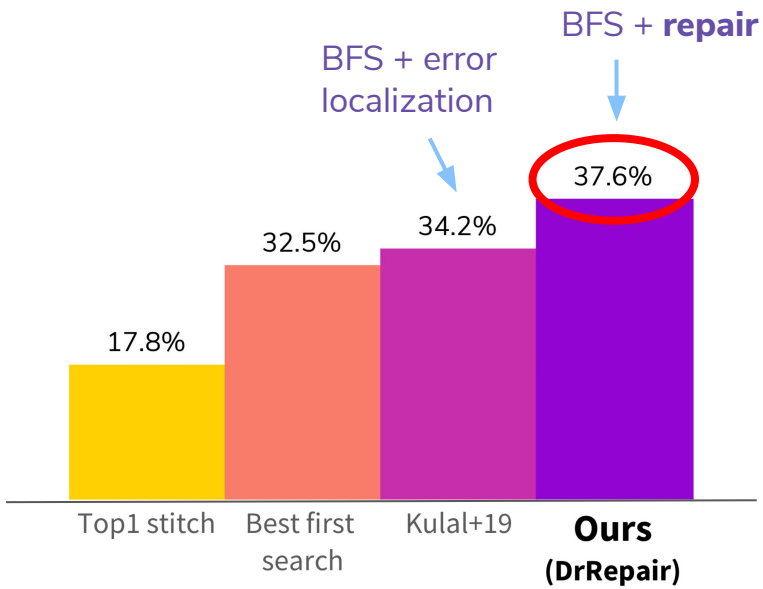
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation



TestP (synthesis success rate)



Analysis 1: When is **graph** useful?

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`‘@@@’ was not declared …`	35.2 %	50.2	58.9	65.0
`redeclaration of ‘@@@’`	8.9 %	40.7	43.0	49.1
`expected ‘@@@’ before ‘@@@’`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member ‘@@@’ in ‘@@@’, …`	2.9 %	37.9	56.9	48.4
`expected initializer before ‘@@@’`	2.1 %	48.8	50.1	93.0
`‘@@@’ without a previous ‘@@@’`	1.3 %	37.0	38.7	44.4

Analysis 1: When is **graph** useful?

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`'@@@' was not declared …`	35.2 %	50.2	**58.9**	65.0
`redeclaration of '@@@'`	8.9 %	40.7	**43.0**	49.1
`expected '@@@' before '@@@'`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member '@@@' in '@@@', …`	2.9 %	37.9	**56.9**	48.4
`expected initializer before '@@@'`	2.1 %	48.8	50.1	93.0
`'@@@' without a previous '@@@'`	1.3 %	37.0	38.7	44.4

Analysis 1: When is **graph** useful?

Need reasoning over multiple lines of code

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`‘@@@’ was not declared …`	35.2 %	50.2	**58.9**	65.0
`redeclaration of ‘@@@’`	8.9 %	40.7	**43.0**	49.1
`expected ‘@@@’ before ‘@@@’`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member ‘@@@’ in ‘@@@’, …`	2.9 %	37.9	**56.9**	48.4
`expected initializer before ‘@@@’`	2.1 %	48.8	50.1	93.0
`‘@@@’ without a previous ‘@@@’`	1.3 %	37.0	38.7	44.4

Recall this example

Broken program

```
1  #include <bits/stdc++.h>
2  #include <string>
3  using namespace std;
4  int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++){
10     tmp.push_back(a[i]);
11     string tmp1 = tmp;
12     for (j = 0; j < b.size(); j++){
13       tmp1.push_back(b[j]);
14       mp[tmp1] = 1;
15     }
16   } ...
```

Multi-hop!

Feedback

```
line 9:error: request for
member ‘size’ in ‘a’, which
is of non-class type ‘char’
```

Repair

1. Error localized **line 5**
2. Repair

```
  char tmp, a, b;
→ string tmp, a, b;
```

Analysis 1: When is **graph** useful?

Need reasoning over multiple lines of code

Graph captures **long-range connections** of tokens

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`‘@@@’ was not declared …`	35.2 %	50.2	**58.9**	65.0
`redeclaration of ‘@@@’`	8.9 %	40.7	**43.0**	49.1
`expected ‘@@@’ before ‘@@@’`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member ‘@@@’ in ‘@@@’, …`	2.9 %	37.9	**56.9**	48.4
`expected initializer before ‘@@@’`	2.1 %	48.8	50.1	93.0
`‘@@@’ without a previous ‘@@@’`	1.3 %	37.0	38.7	44.4

Analysis 2: When is **pre-training** useful?

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`‘@@@’ was not declared …`	35.2 %	50.2	**58.9**	65.0
`redeclaration of ‘@@@’`	8.9 %	40.7	**43.0**	49.1
`expected ‘@@@’ before ‘@@@’`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member ‘@@@’ in ‘@@@’, …`	2.9 %	37.9	**56.9**	48.4
`expected initializer before ‘@@@’`	2.1 %	48.8	50.1	**93.0**
`‘@@@’ without a previous ‘@@@’`	1.3 %	37.0	38.7	**44.4**

Analysis 2: When is **pre-training** useful?

Rare in train set

Extra examples in pre-training are helpful

Compiler message type	Frequency in train set (SPoC)	Repair acc. (SPoC dev)		
		base	+ graph	+ graph + pretrain
`‘@@@’ was not declared …`	35.2 %	50.2	**58.9**	65.0
`redeclaration of ‘@@@’`	8.9 %	40.7	**43.0**	49.1
`expected ‘@@@’ before ‘@@@’`	3.2 %	67.6	70.7	86.1
`expected primary-expression before …`	3.0 %	47.4	47.4	49.1
`request for member ‘@@@’ in ‘@@@’, …`	2.9 %	37.9	**56.9**	48.4
`expected initializer before ‘@@@’`	**2.1 %**	48.8	50.1	**93.0**
`‘@@@’ without a previous ‘@@@’`	**1.3 %**	37.0	38.7	**44.4**

Takeaways

New insights

- Use of **error messages** is crucial to learn program repair
- Program-feedback **graph** helps complex reasoning
- Unlabeled programs can be used for **targeted pre-training**

General framework

- **Learning to reason with feedback** - many applications, e.g.
  - Edit essays based on written feedback
  - Learn from user inputs in interactive dialogue
- We show that **graph-based reasoning** can be a good solution

Thanks!

Michihiro Yasunaga

Percy Liang

Thank you to John Hewitt, Mina Lee, Sumith Kulal, Pang Wei Koh, Robin Jia, Ananya Kumar, Ruiqi Zhong, and anonymous reviewers.

Funded in part by NSF CAREER Award, PECASE Award, and Amazon Research Award.

Code/Data: https://cs.stanford.edu/~myasu/