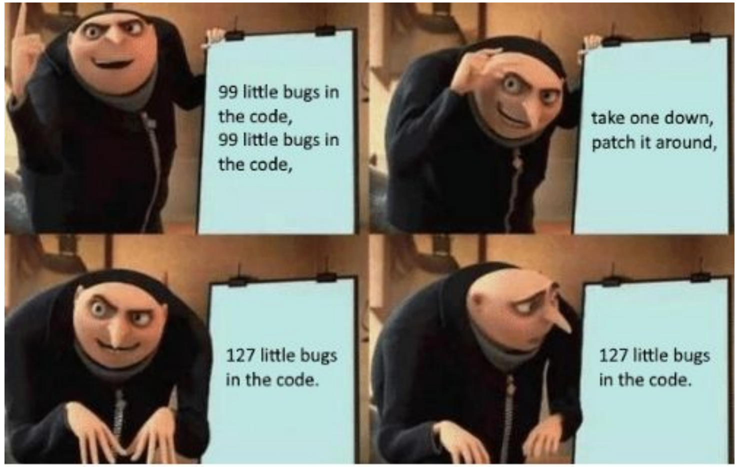
HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS

Elizabeth Dinella*, Hanjun Dai*, Ziyang Li, Mayur Naik, Le song, Ke Wang

Presenter: Liunian Li, Ruochen Wang

Motivation

- Sheer size and complexity of modern codebases are difficult to debug
- Automated debugging tools to the rescue


99 little bugs in the code,
99 little bugs in the code,
take one down, patch it around,
127 little bugs in the code.
127 little bugs in the code.

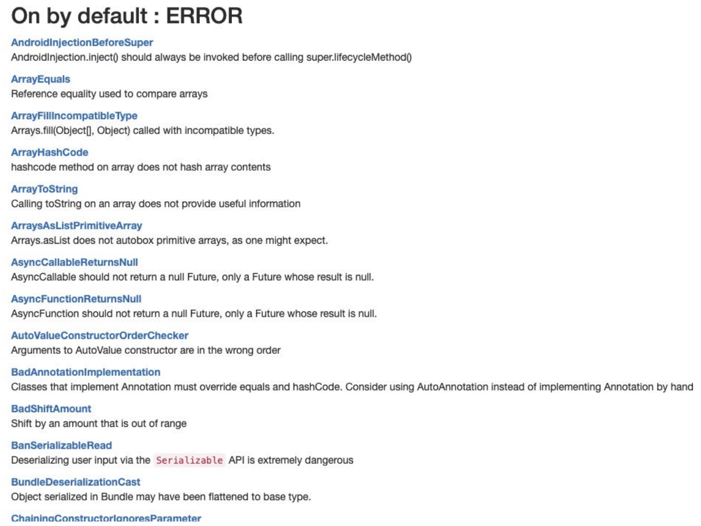
99 little bugs in the code

Existing Methods

- Rule-based
- Data-driven

Existing Methods

- Rule-based - Example
  - ErrorProne (part of Google Tricorder Analysis Tool)
    - Consists a list of bug patterns, and these will be used to match against a query code
    - Find bug patterns based on AST matching



On by default : ERROR

AndroidInjectionBeforeSuper
AndroidInjection.inject() should always be invoked before calling super.lifecycleMethod()

ArrayEquals
Reference equality used to compare arrays

ArrayFillIncompatibleType
Arrays.fill(Object[], Object) called with incompatible types.

ArrayHashCode
hashcode method on array does not hash array contents

ArrayToString
Calling toString on an array does not provide useful information

ArraysAsListPrimitiveArray
Arrays.asList does not autobox primitive arrays, as one might expect.

AsyncCallableReturnsNull
AsyncCallable should not return a null Future, only a Future whose result is null.

AsyncFunctionReturnsNull
AsyncFunction should not return a null Future, only a Future whose result is null.

AutoValueConstructorOrderChecker
Arguments to AutoValue constructor are in the wrong order

BadAnnotationImplementation
Classes that implement Annotation must override equals and hashCode. Consider using AutoAnnotation instead of implementing Annotation by hand

BadShiftAmount
Shift by an amount that is out of range

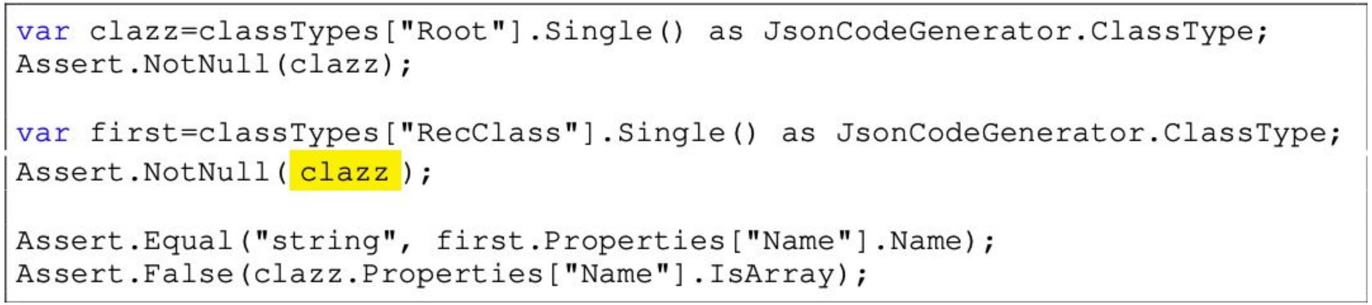
BanSerializableRead
Deserializing user input via the `Serializable` API is extremely dangerous

BundleDeserializationCast
Object serialized in Bundle may have been flattened to base type.

ChainingConstructorIgnoresParameter

Existing Methods

- Data-driven (Example)
  - Allamanis et al., LEARNING TO REPRESENT PROGRAMS WITH GRAPHS
  - Idea:
    - Construct a graph of the code snippet based on AST (Abstract Syntax Tree)
    - Utilize the learned node embedding for downstream tasks
  - Tasks:
    - VarNaming: predict the name of a variable given its usage
    - VarMisuse: infer which variable should be used for a given location

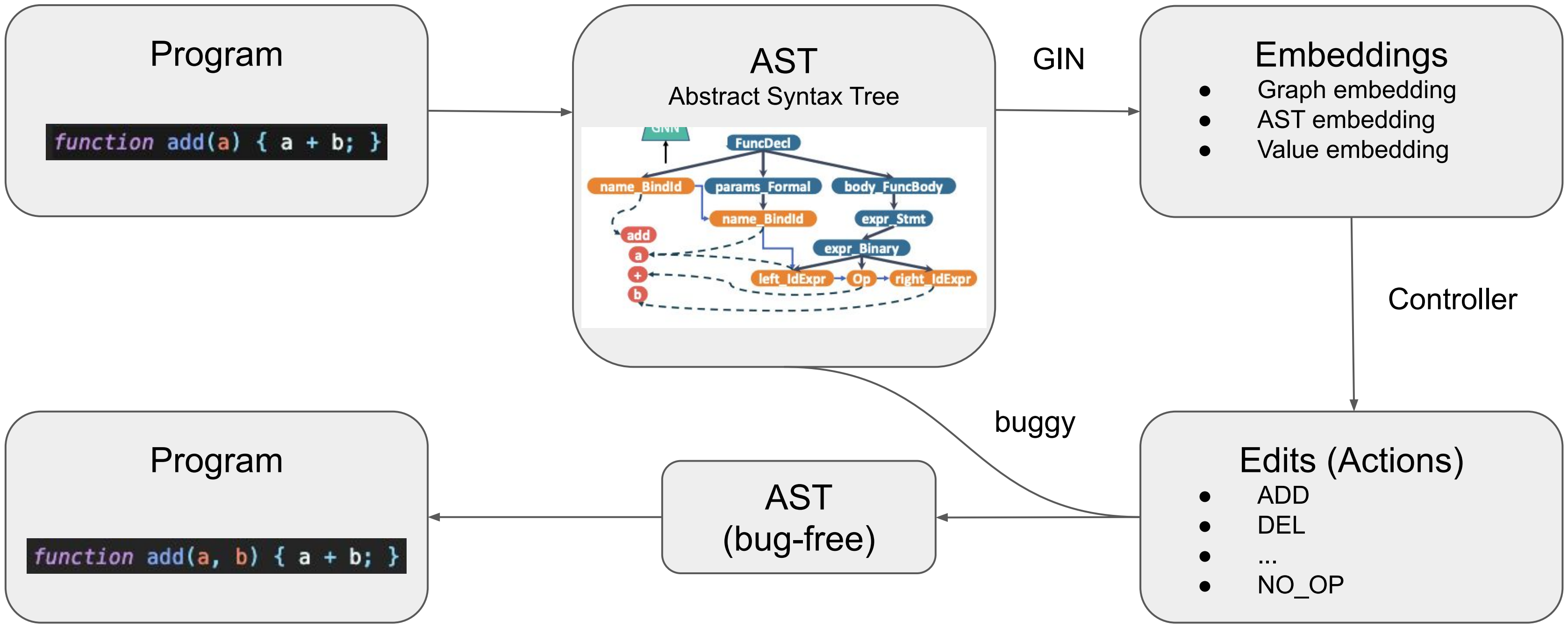


```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( clazz );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

This paper

- Propose a learning-based for detecting and fixing bugs in Javascript (JS)
  - Fixing JS bugs is challenging due to anti-human syntactic designs
- Hypothesis: A code snippet is buggy if it deviates from common practices. (pattern matching -> learning-based)
- Contribution:
  - Can handle complex bugs: adding/removing statements from a program
  - End-to-End in the sense that:
    - propose bug locations
    - propose fixes
    - implement fixes
    - Until the program is bug-free

Pipeline Overview



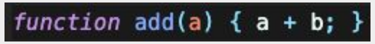
```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

Method Part I: Embedding

Method - Embedding

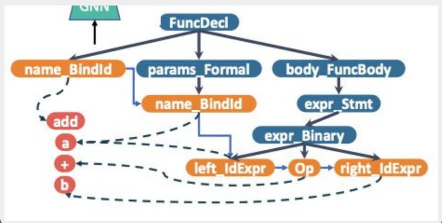
Program



```
function add(a) { a + b; }
```

AST

Abstract Syntax Tree



GIN

Embeddings

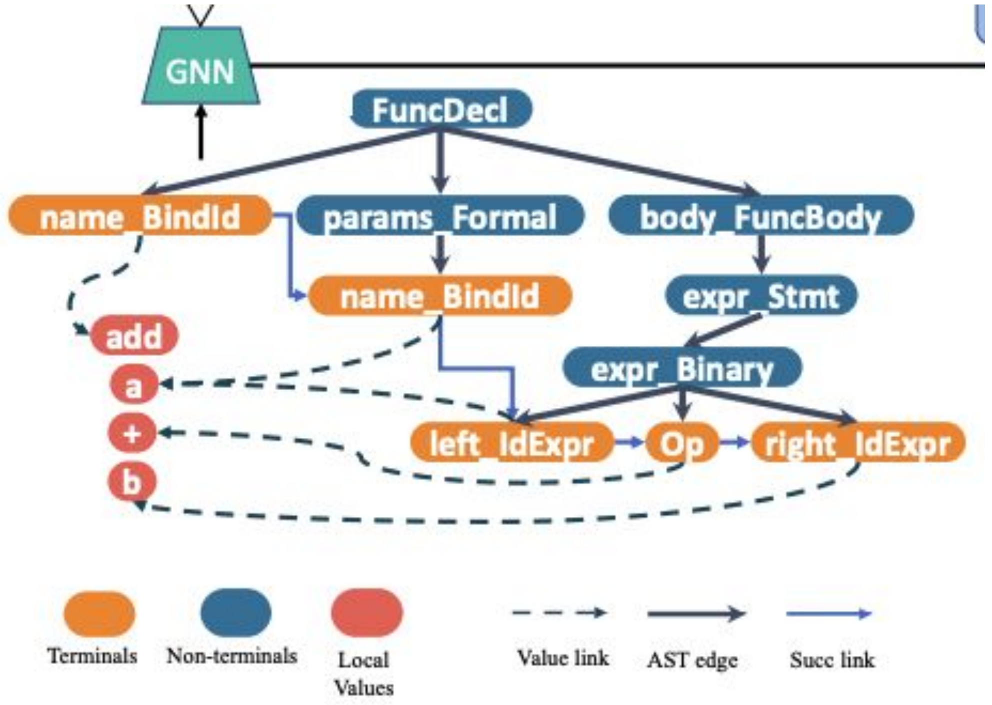
- Graph embedding
- AST embedding
- Value embedding

Method - Embedding

Program

```
function add(a) { a + b; }
```

- Abstract Syntax Tree (AST):
  - Nodes: motifs of a program
    - Non-terminal nodes
    - Terminal nodes
    - Local values
  - Edges:
    - AST edge
    - Succ link (SuccToken)
    - Value link



Method - Embedding

- GIN (graph isomorphism network)
- GIN vs regular GNN: GIN uses sum() as aggregation function
  - They argue that sum aggregation is better than mean and max aggregation in terms of distinguishing graph structure.
  - Proved to be as powerful as WL (Weisfeiler-Lehman) test.

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum\nolimits_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

Method - Embedding

- Update rule:
  - Extend GIN to Multi-graph setting (multiple types of edges)
    - AST edge
    - Succ link (SuccToken)
    - Value link

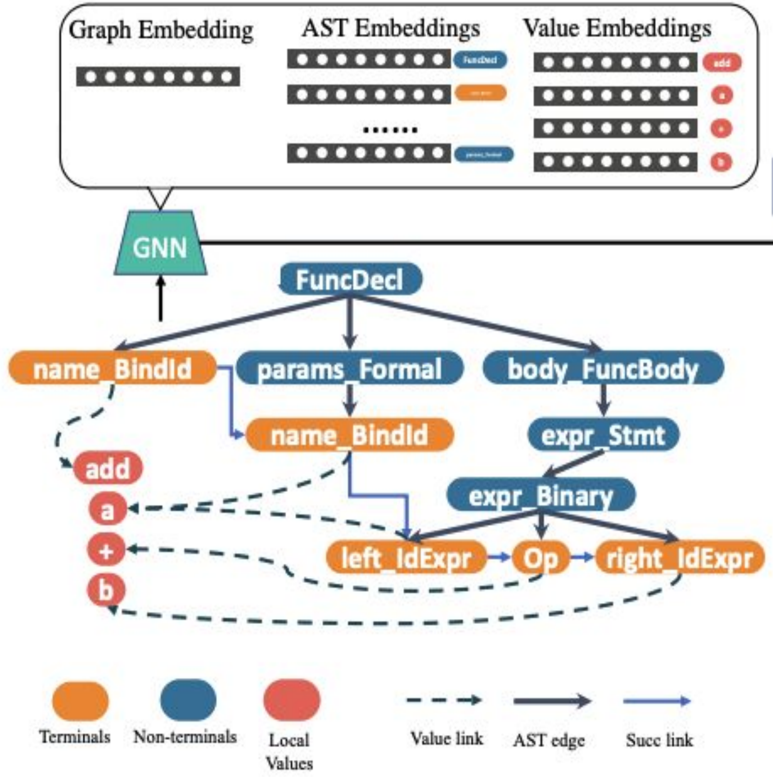
$$h_v^{(l+1),k} = \sigma(\textstyle\sum_{u \in \mathcal{N}^k(v)} \mathbf{W}_1^{l,k} h_u^{(l)}), \forall k \in \{1, 2, \ldots, K\}$$

$$h_v^{(l+1)} = \sigma(\mathbf{W}_2^l [h_v^{(l+1),1}, h_v^{(l+1),2}, \ldots, h_v^{(l+1),K}] + h_v^{(l)})$$

$$\vec{g} = AVG_l(MAXPOOL_v(h_v^l))$$

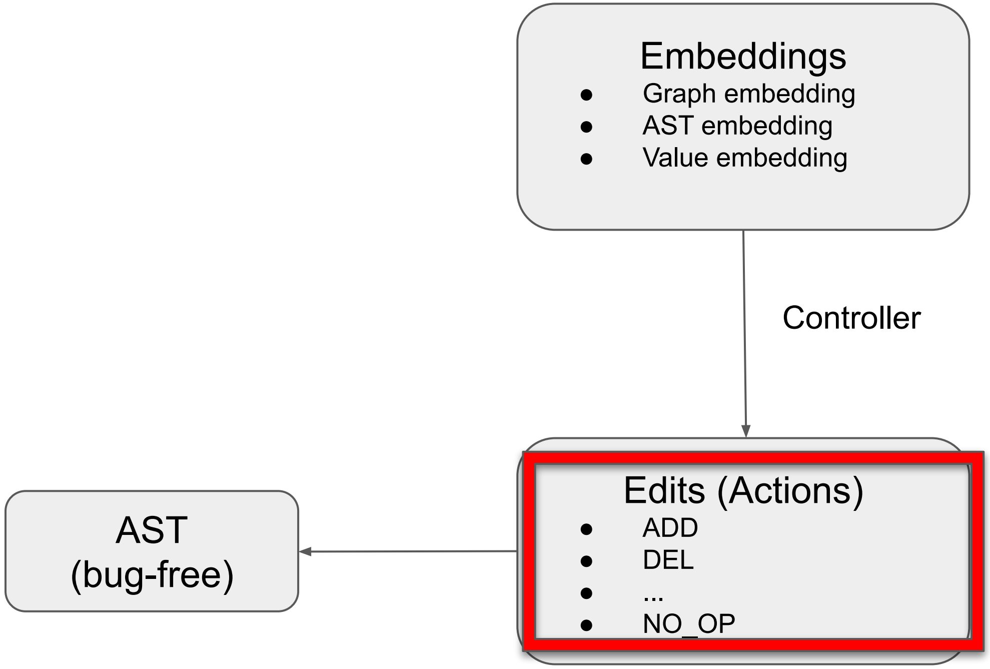
Method - Embedding

- GIN produces three types of embeddings
    - Graph Embedding (g)
    - AST Embedding
    - Value Embedding
- Those embeddings will be used to determine how to edit the code

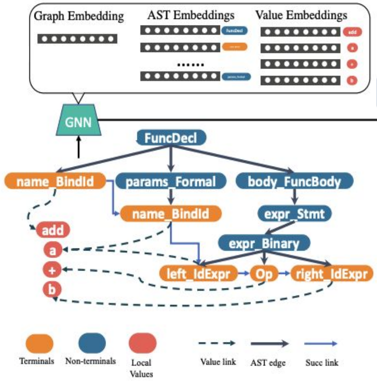


Method Part II: Controller

Pipeline Overview



Method - Controller - Edit Actions



- Five in total
  - ADD: add a new node to the graph
  - DEL: delete a node from graph
  - REP_VAL: replace the value of a terminal node
  - REP_TYPE (more like NAME): replace the <u>naming</u> of a <u>non</u>-terminal node (error in paper)
  - NO_OP: no operation needed, program fixed
- These actions can be constructed using “Edit Primitives”
  - Location
  - Value
  - Type

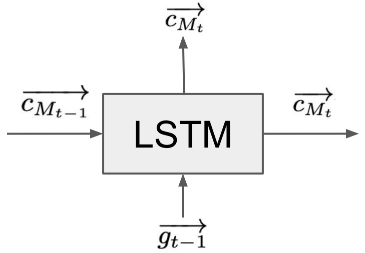
Method - Controller - context embedding (query)

- Context embedding “c”
  - encodes graph info and edit history
  - provide information for how to edit the graph
- Two types:
  - Macro: $\overrightarrow{c_{M_t}}' = \mathrm{LSTM}(\overrightarrow{g_{t-1}} | \overrightarrow{c_{M_{t-1}}})$ $\longrightarrow$
  - Determine the location, and edit action

$$\overrightarrow{c_{m_t}} = \mathrm{LSTM}(\overrightarrow{e_t} | \mathrm{LSTM}(\overrightarrow{v_t} | \overrightarrow{c_{M_t}}))$$

  - Micro:
  - Used for executing the edit action

edit history

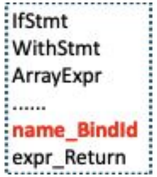


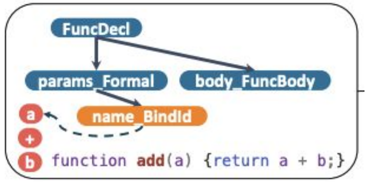
$\overrightarrow{c_{M_t}}$
$\overrightarrow{c_{M_{t-1}}}$
LSTM
$\overrightarrow{c_{M_t}}$
$\overrightarrow{g_{t-1}}$

CMT: typo in the update rule of Micro: missing “prime” sign

Method - Controller - Edit Primitives

- Query: context embedding c
- Three primitives that can be used to construct high level edits
  - Location
    - Determine the location (node) of bug
    - $loc(\vec{c}, g) = \arg\max_{v \in V} \vec{v}^{\top} \vec{c}$
  - Value
    - Assign a value to a terminal node
    - Possible values = local values (cur file) + global values (common for a language)
    - $val(\vec{c}, g) = \text{argmax}_{t \in D_{val} \cup V_{val}} \vec{t}^{\top} \vec{c}.$
  - Type (naming)
    - Determine the type of the non-terminal nodes *CMT: error in the main text: {Type} determines the type for all nodes*
    - Classification
    - e.g.


IfStmt
WithStmt
ArrayExpr
......
name_BindId
expr_Return

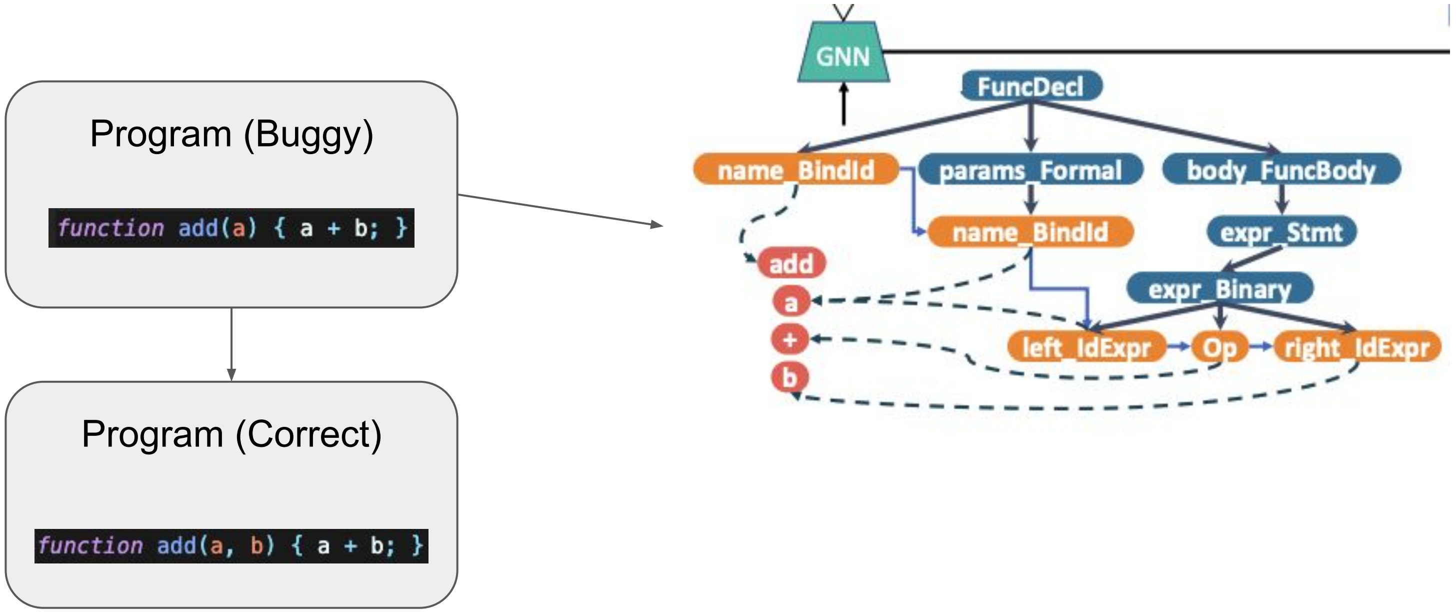


Method - Controller - Example

- Let’s use a concrete example to understand the entire pipeline
- Consider the previous buggy program “add”

Method - Controller - Edit Action

- Consider the previous buggy program “add” (simplify next)

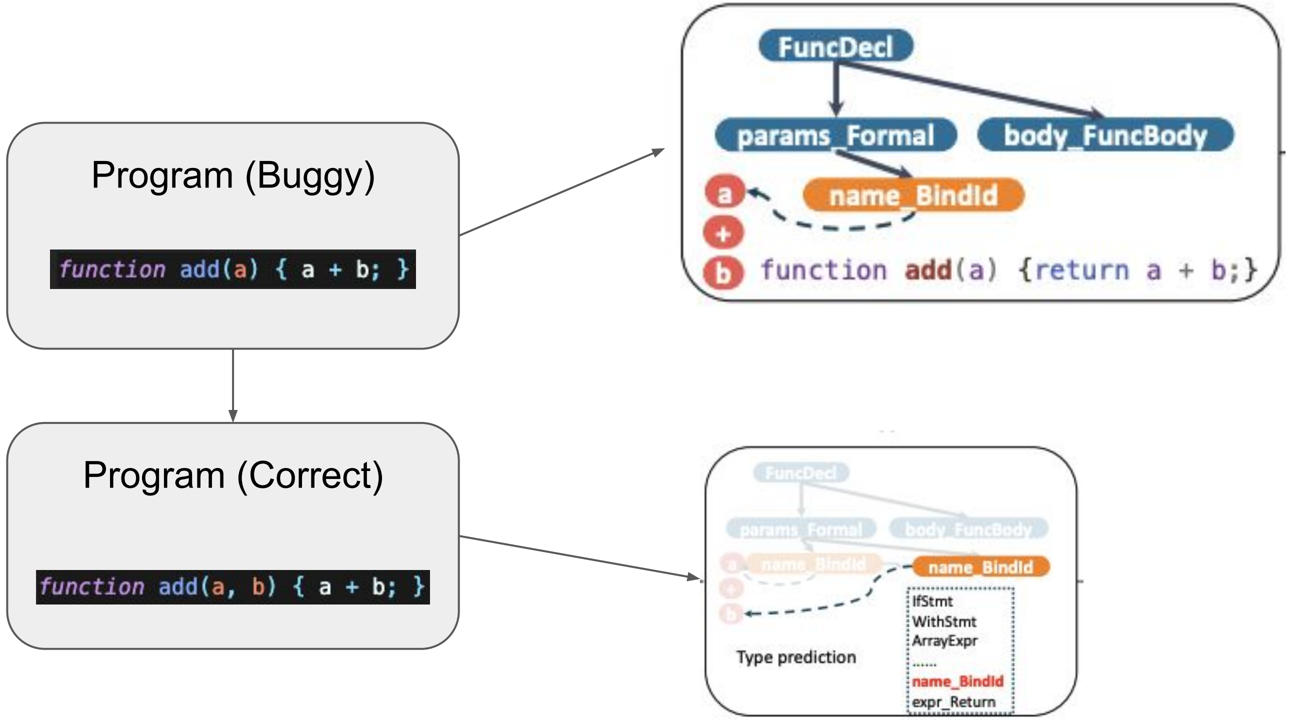


```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

Method - Controller - Edit Action

- Consider the previous buggy program “add”



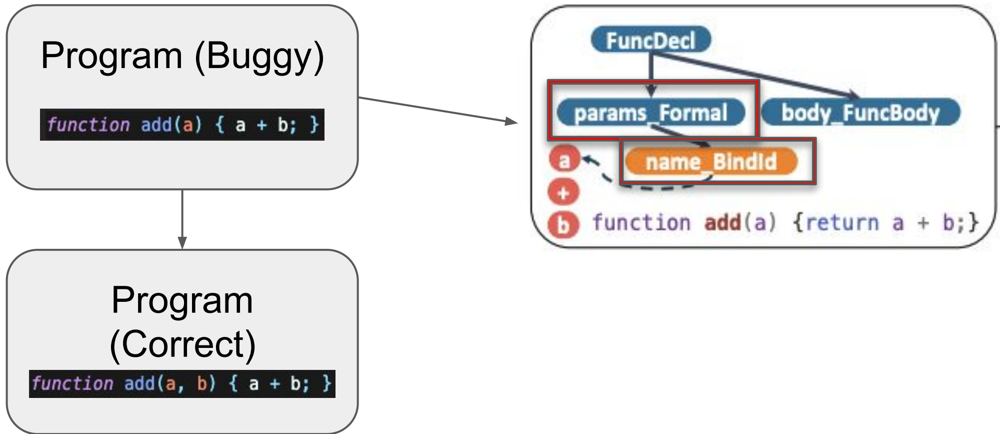
```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

```
function add(a) {return a + b;}
```

Method - Controller - Edit Action

- Step 1: locate the buggy node (location primitives)
  - Use macro-context embedding as query
  - locate the <u>parent node</u> of the bug $loc(\vec{c}, g) = \arg\max_{v \in V} \vec{v}^\top \vec{c}$
  - locate the <u>sibling node</u> of the bug (so that we can add SuccToken link)
    - Not mentioned in the paper
    - Our guess: also use the above equation, but this time search over child nodes



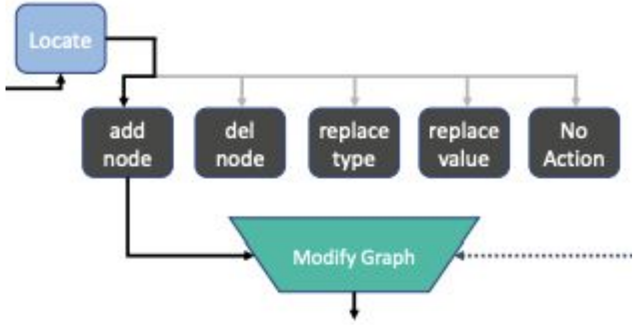
```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

```
function add(a) {return a + b;}
```

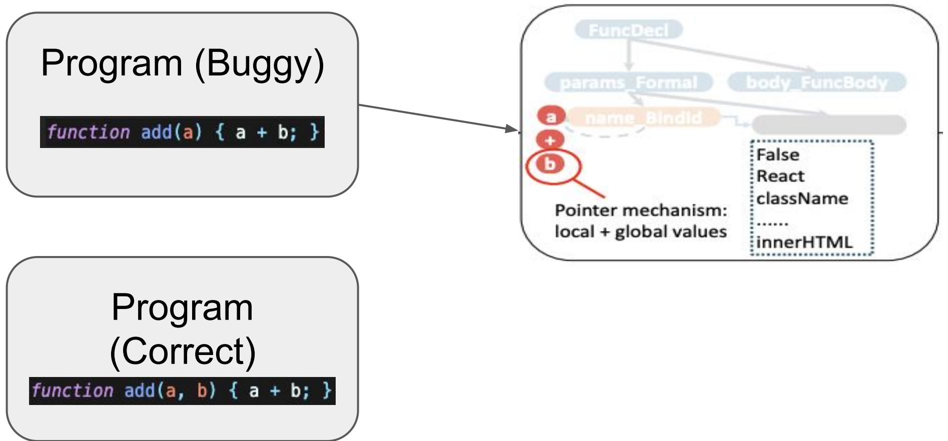
Method - Controller - Edit Action

- Step 2: Determine the edit action (“ADD” here) (e_t)
  - Use macro-context embedding as input
  - Classification (input: node embedding)
  - Update micro-context embedding
    - $\overrightarrow{c_{m_t}} = \mathrm{LSTM}(\overrightarrow{e_t} | \mathrm{LSTM}(\overrightarrow{v_t} | \overrightarrow{c_{M_t}}))$
    - $\vec{c_{m1}} = \mathrm{LSTM}(\overrightarrow{v_{sibling}} | \vec{c_m})$



Method - Controller - Edit Action

- Step 3: Assign a value to the newly added node (Value)
  - $val(\vec{c}, g) = \text{argmax}_{t \in D_{val} \cup V_{val}} \vec{t}^\top \vec{c}.$
  - Update micro-context embedding
    - $\vec{c_{m2}} = \text{LSTM}(val(\vec{c_{m1}}, g) | \vec{c_{m1}})$

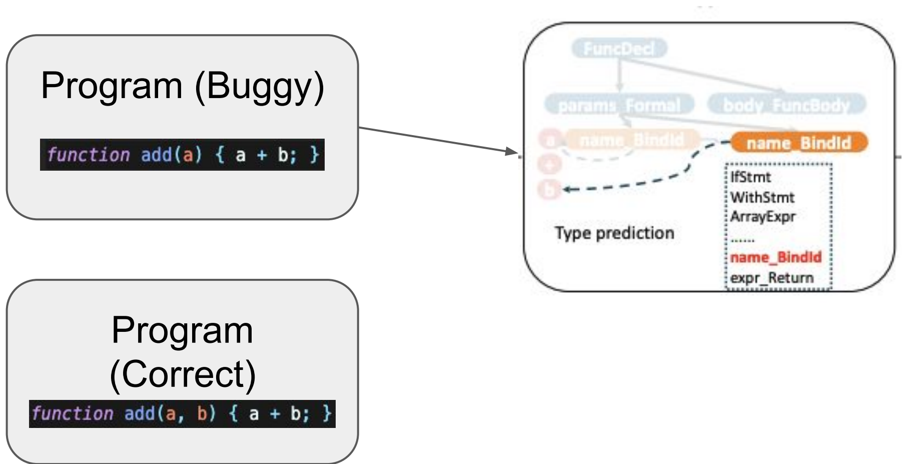


```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

Method - Controller - Edit Action

- Step 4: Assign a type to the newly added node (Type)
  - Classification (input: micro-context and graph embedding)
  - Update micro-context embedding
    - $\vec{c_{m3}} = \text{LSTM}(type(\vec{c_{m2}}, g) | \vec{c_{m2}})$
    - $\vec{c}_{ADD} = \vec{c_{m3}}$

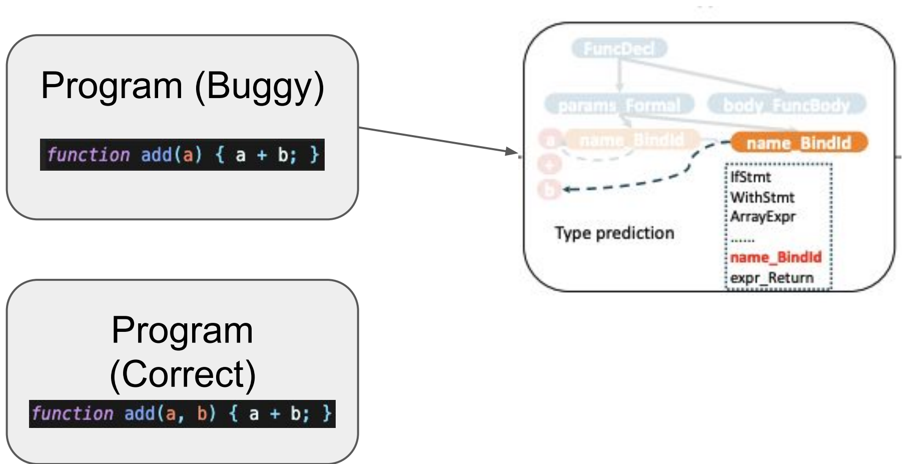


```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

Method - Controller - Edit Action

- Step 5: Complete the links for the newly added node
  - Connect Parent with newly added node via AST Link
  - Connect newly added node with assigned value via ValueLink
  - Connect sibling node with newly added node via SuccToken (Link)



```
function add(a) { a + b; }
```

```
function add(a, b) { a + b; }
```

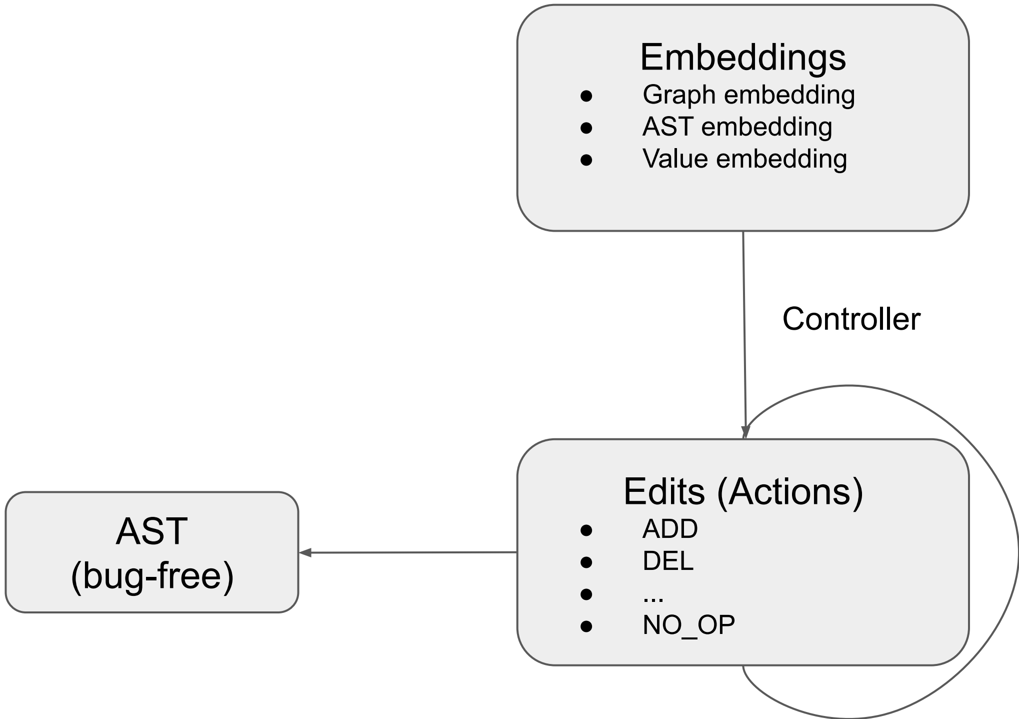
Method - Controller - Edit Action

- Step 6: Update macro contents
    - Graph embeddings g (refit the edited graph with GIN)
    - Macro-context embeddings (paper does not discuss how)

$$\overrightarrow{c_{M_t}} \longleftarrow \vec{c}_{ADD}$$

Method - Controller - Edit Action

- Repeat Step 1 - 6 until “NO_OP” is selected as the Edit Action.



Method Part III: Learning & Inference

Method - Learning

- Dataset: pairs of buggy code and fixed code $\mathcal{D} = \{(g_{bug}^{(i)}, g_{fix}^{(i)})\}_{i=1}^{|\mathcal{D}|}$
- Objective:

$$\max_\theta \mathbb{E}_{(g_{bug}, g_{fix}) \sim \mathcal{D}} p(g_{fix} | g_{bug}; \theta)$$

$$p(g_{fix} | g_{bug}; \theta) = p(g_1 | g_{bug}; \theta) p(g_2 | g_1; \theta) \ldots p(g_{fix} | g_{T-1}; \theta)$$

We can obtain supervision for the factorized graph transformation sequences:

Parse the source code using the SHIFT AST format, and utilize a JSON diff toolbox to compile the code differences into a sequence of AST edits

(Training is similar to how we train a forward language model)

Method - Inference

- Goal: $\arg\max_{g_{fix}} p(g_{fix} | g_{bug}; \theta)$
- Problem: combinational search space
- Solution: beam search

Method - Inference - Beam Search

- Maintain a pool of **B** partially fixed programs, which starts with simply the single buggy program
- For every program in the pool, we propose B locations, B operators, or B primitives, depending on the current state of the program
- Rank solutions based on the joint log-likelihood and keep top B partially fixed programs

Method - Inference - Beam Search

- Maintain a pool of **B** partially fixed programs, which starts with simply the single buggy program
- For every program in the pool, we propose B locations, B operators, or B primitives, depending on the current state of the program
- Rank solutions based on the joint log-likelihood and keep top B partially fixed programs

Experiments

Dataset

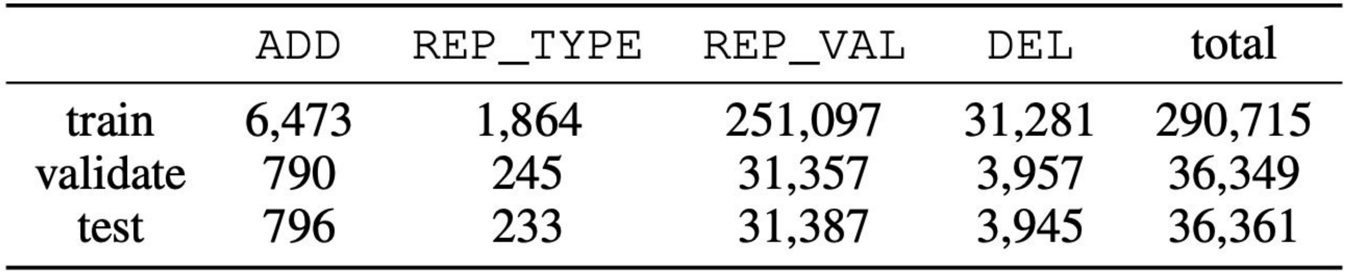
- Automatically mined from Github
- How to determine if a commit is a bug fix or not?

  Heuristic: a commit with a smaller number of AST differences is more likely to be a bug fix

- The program before a bug-fix commit is the buggy program and after the bug-fix commit is the correct program

Dataset

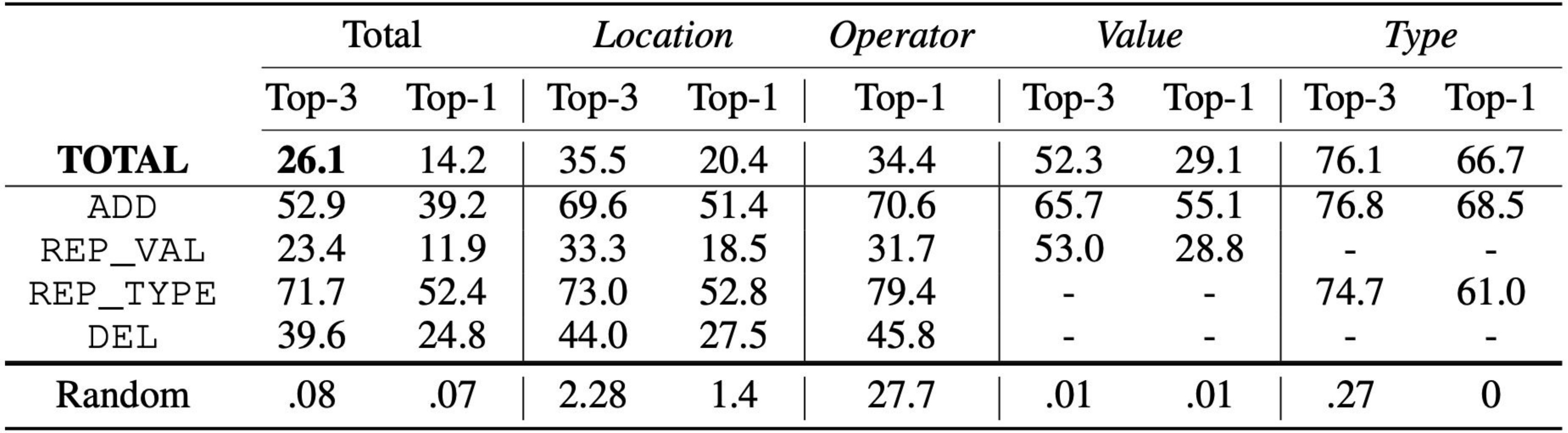
- Three datasets:
- OneDiff, ZeroOneDiff, ZeroOneTwoDiff



	ADD	REP_TYPE	REP_VAL	DEL	total
train	6,473	1,864	251,097	31,281	290,715
validate	790	245	31,357	3,957	36,349
test	796	233	31,387	3,945	36,361

Table 1: Statistic of `OneDiff` dataset. See appendix for more information of other dataset.

Evaluation

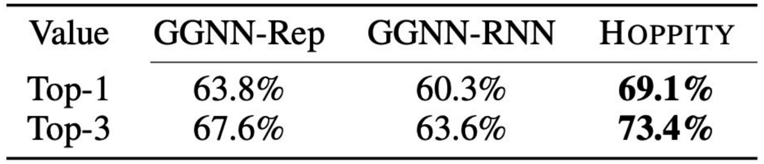


	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
TOTAL	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
`ADD`	52.9	39.2	69.6	51.4	70.6	65.7	55.1	76.8	68.5
`REP_VAL`	23.4	11.9	33.3	18.5	31.7	53.0	28.8	-	-
`REP_TYPE`	71.7	52.4	73.0	52.8	79.4	-	-	74.7	61.0
`DEL`	39.6	24.8	44.0	27.5	45.8	-	-	-	-
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 2: Evaluation of model on the `OneDiff` dataset: accuracy (%).

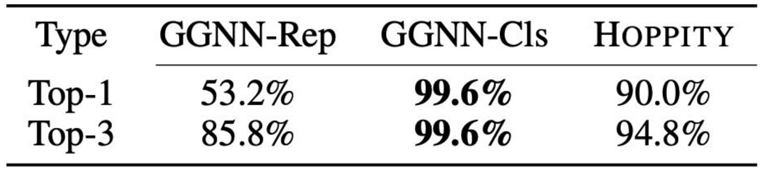
- A model will be penalized in Value and Type if it predict the Operator wrong
- Random baseline shows the large search space of the problem

Evaluation - Compare with GGNN



Value	GGNN-Rep	GGNN-RNN	HOPPITY
Top-1	63.8%	60.3%	**69.1%**
Top-3	67.6%	63.6%	**73.4%**

Table 4: `REP_VAL` accuracies with location+op.

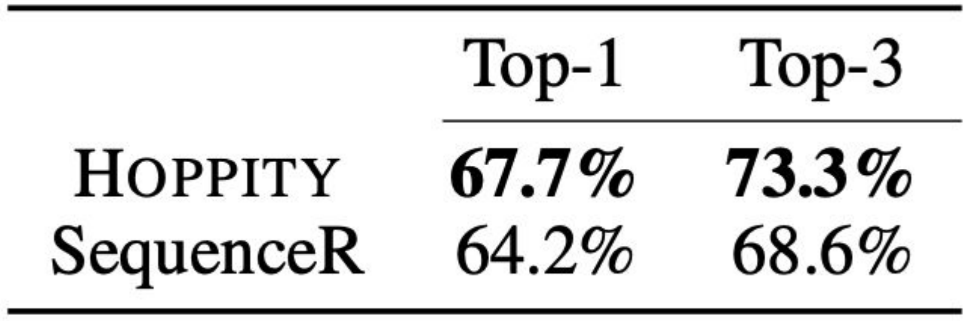


Type	GGNN-Rep	GGNN-Cls	HOPPITY
Top-1	53.2%	**99.6%**	90.0%
Top-3	85.8%	**99.6%**	94.8%

Table 3: `REP_TYPE` accuracies with location+op.

- GGNN-Rep: from Allamanis et al. 2018
- GGNN-Cls: muti-class classification with target node and graph embedding
- GGNN-RNN: predict the value as a character-level language model

Evaluation - Compare with SequenceR

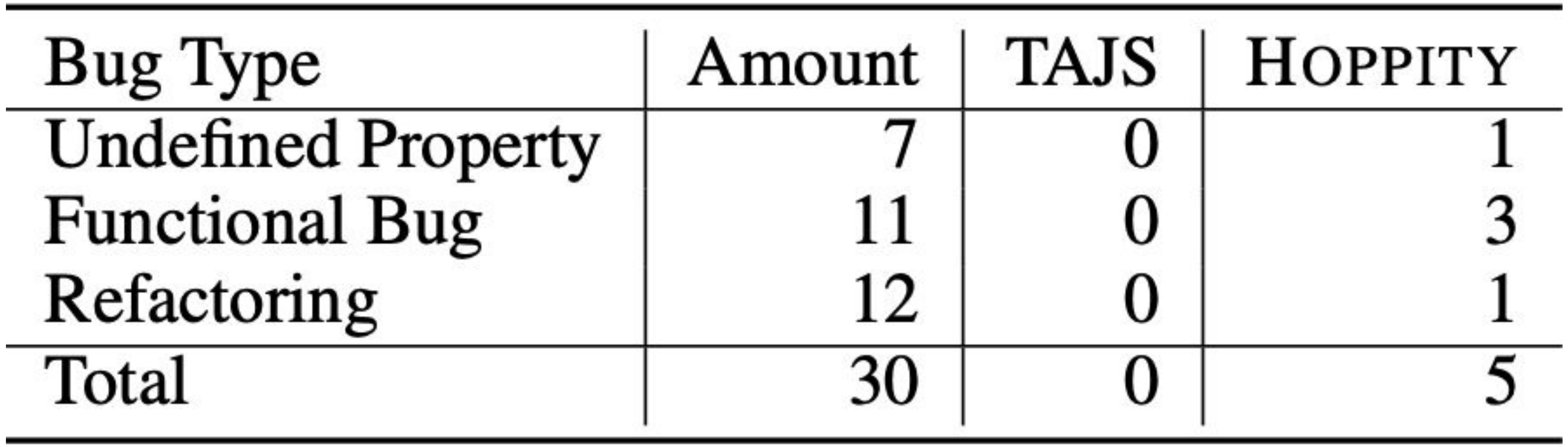


	Top-1	Top-3
HOPPITY	**67.7%**	**73.3%**
SequenceR	64.2%	68.6%

Table 5: Overall OneDiff accuracy with location.

- Sequence R: predict a sequence of tokens given a buggy line (similar to machine translation)

Evaluation - Compare with TAJS



Bug Type	Amount	TAJS	HOPPITY
Undefined Property	7	0	1
Functional Bug	11	0	3
Refactoring	12	0	1
Total	30	0	5

- Static analysis tool
- Only support part of the evaluation set; Need manual setup for its control options;

Related Work - Static analysis for bug detection

Advantage

- Targets a broad range of programming errors
- Not only localize bugs but also fixes them
- Significantly higher signal-to-noise ratio

Potential drawback: longer run time; scalability to long code sequence?

Related Work - Learning-based bug detection

Allamanis et al. (2018): use GNN to predict correct variable name given a buggy location

Vasic et al. (2019): pointer network on top of RNN

DeepBugs: only supports three classes of bugs

SequenceR: seq2seq bug fixing

Getafix: hierarchical clustering to sort fix patterns

All need the bug location as input.

Related Work - Graph learning and optimization

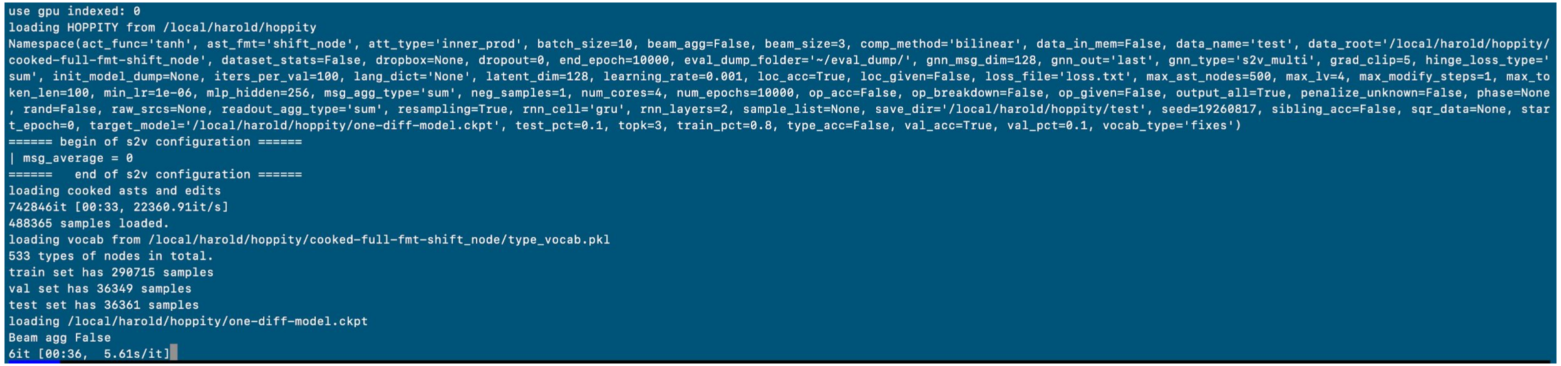
The local value table and pointer mechanism inspired by prior work

Related to auto-regressive graph modeling

Some work model graph modification in latent space but lack fine-grained control (graph-to-graph transformation)

Demo

Reproduce results



```
use gpu indexed: 0
loading HOPPITY from /local/harold/hoppity
Namespace(act_func='tanh', ast_fmt='shift_node', att_type='inner_prod', batch_size=10, beam_agg=False, beam_size=3, comp_method='bilinear', data_in_mem=False, data_name='test', data_root='/local/harold/hoppity/
cooked-full-fmt-shift_node', dataset_stats=False, dropbox=None, dropout=0, end_epoch=10000, eval_dump_folder='~/eval_dump/', gnn_msg_dim=128, gnn_out='last', gnn_type='s2v_multi', grad_clip=5, hinge_loss_type='
sum', init_model_dump=None, iters_per_val=100, lang_dict='None', latent_dim=128, learning_rate=0.001, loc_acc=True, loc_given=False, loss_file='loss.txt', max_ast_nodes=500, max_lv=4, max_modify_steps=1, max_to
ken_len=100, min_lr=1e-06, mlp_hidden=256, msg_agg_type='sum', neg_samples=1, num_cores=4, num_epochs=10000, op_acc=False, op_breakdown=False, op_given=False, output_all=True, penalize_unknown=False, phase=None
, rand=False, raw_srcs=None, readout_agg_type='sum', resampling=True, rnn_cell='gru', rnn_layers=2, sample_list=None, save_dir='/local/harold/hoppity/test', seed=19260817, sibling_acc=False, sqr_data=None, star
t_epoch=0, target_model='/local/harold/hoppity/one-diff-model.ckpt', test_pct=0.1, topk=3, train_pct=0.8, type_acc=False, val_acc=True, val_pct=0.1, vocab_type='fixes')
====== begin of s2v configuration ======
| msg_average = 0
======   end of s2v configuration ======
loading cooked asts and edits
742846it [00:33, 22360.91it/s]
488365 samples loaded.
loading vocab from /local/harold/hoppity/cooked-full-fmt-shift_node/type_vocab.pkl
533 types of nodes in total.
train set has 290715 samples
val set has 36349 samples
test set has 36361 samples
loading /local/harold/hoppity/one-diff-model.ckpt
Beam agg False
6it [00:36,  5.61s/it]
```

Online Demo

https://hoppity.cis.upenn.edu/demo

Conclusion

End-to-end learning-based approach to detect and fix bugs in Javascript

Correctly predicts 9490 out of 36361 code changes in real programs on Github

Future work

Bugs spanning multiple files

Deploy in IDE

Other languages