Neural Code Comprehension: A Learnable Representation of Code Semantics

Tal Ben-Nun
ETH Zurich
Zurich 8092, Switzerland
talbn@inf.ethz.ch

Alice Shoshana Jakobovits
ETH Zurich
Zurich 8092, Switzerland
alicej@student.ethz.ch

Torsten Hoefler
ETH Zurich
Zurich 8092, Switzerland
htor@inf.ethz.ch

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Example Task: Algorithm Classification

Try to identify what algorithm the following code implements.

Example Task: Algorithm Classification

Try to identify what algorithm the following code implements.

```
void a(int*b,int c){int d,e,f;for(d=0;d<c-1;d++)for(e=0;e<c-d-
1;e++)if(*(b+e)>*(b+e+1)){f=*(b+e);*(b+e)=*(b+e+1);*(b+e+1)=f;}}
```

Example Task: Algorithm Classification

Try to identify what algorithm the following code implements.

```
void a(int*b,int c){int d,e,f;for(d=0;d<c-1;d++)for(e=0;e<c-d-
1;e++)if(*(b+e)>*(b+e+1)){f=*(b+e);*(b+e)=*(b+e+1);*(b+e+1)=f;}}
```

This is what code looks like to an untrained model.

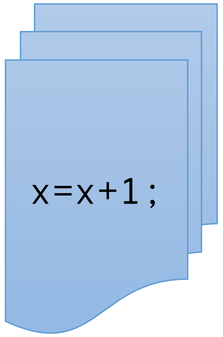
Example Task: Algorithm Classification

Try to identify what algorithm the following code implements.

Same code with semantically meaningful tokens & syntax:

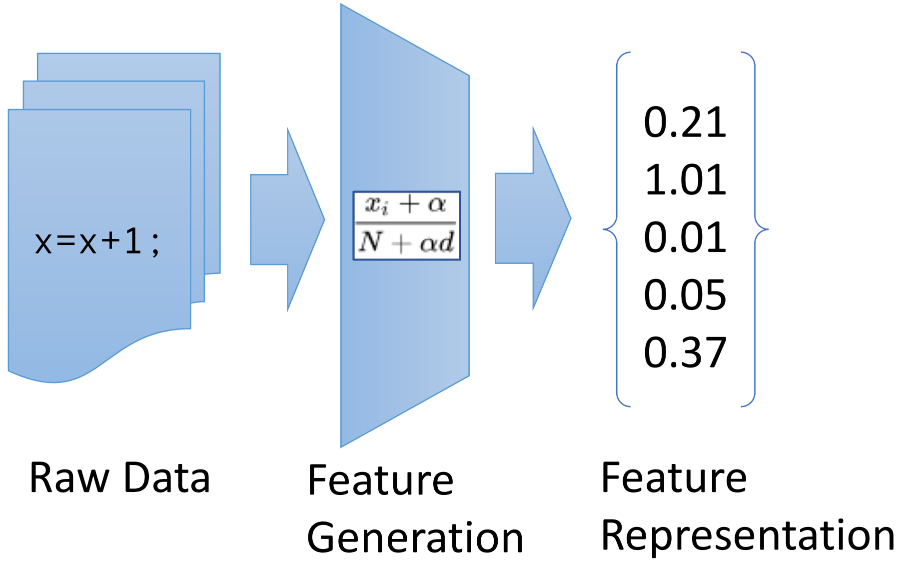
```
void bubbleSort(int arr[], int n) {
   int i, j, tmp;
   for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1]) {
           tmp = arr[j];
           arr[j] = arr[j+1];
           arr[j+1] = tmp;
         }
}
```

General Task: Representation Learning

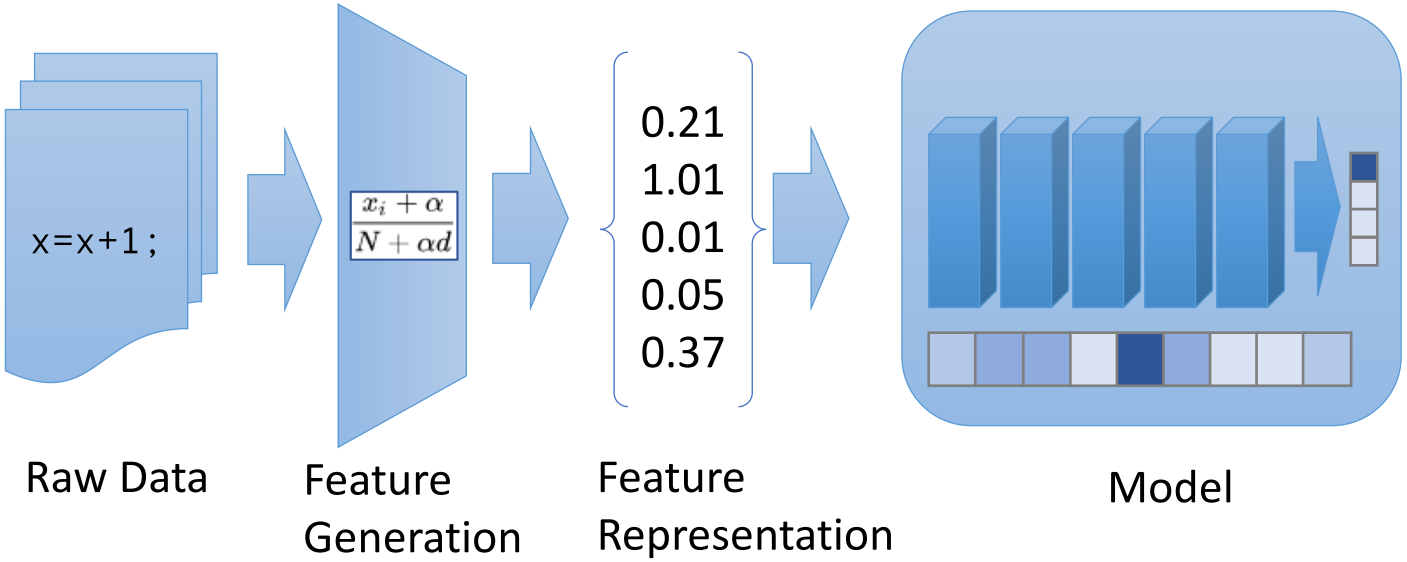


Raw Data

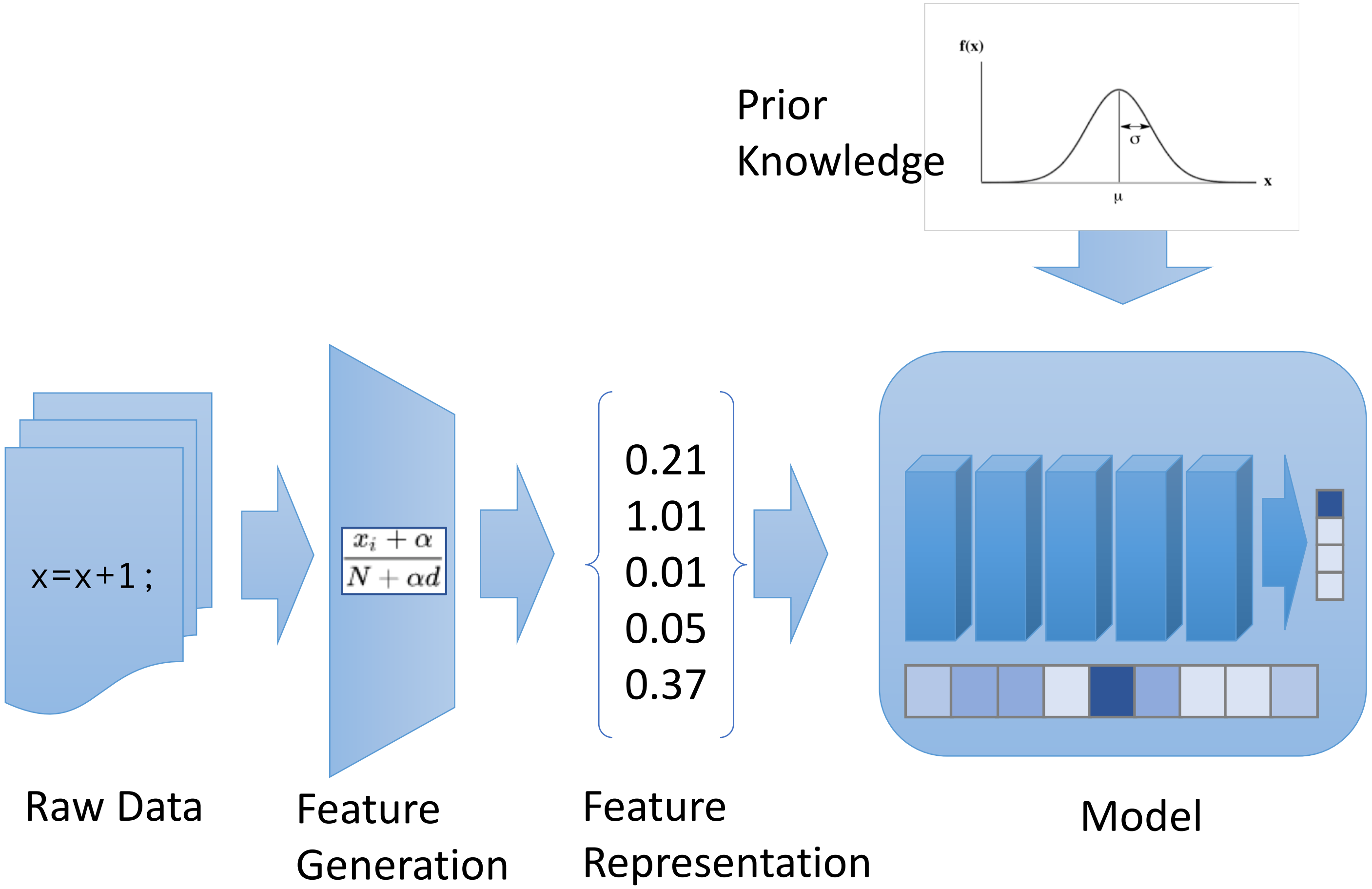
General Task: Representation Learning



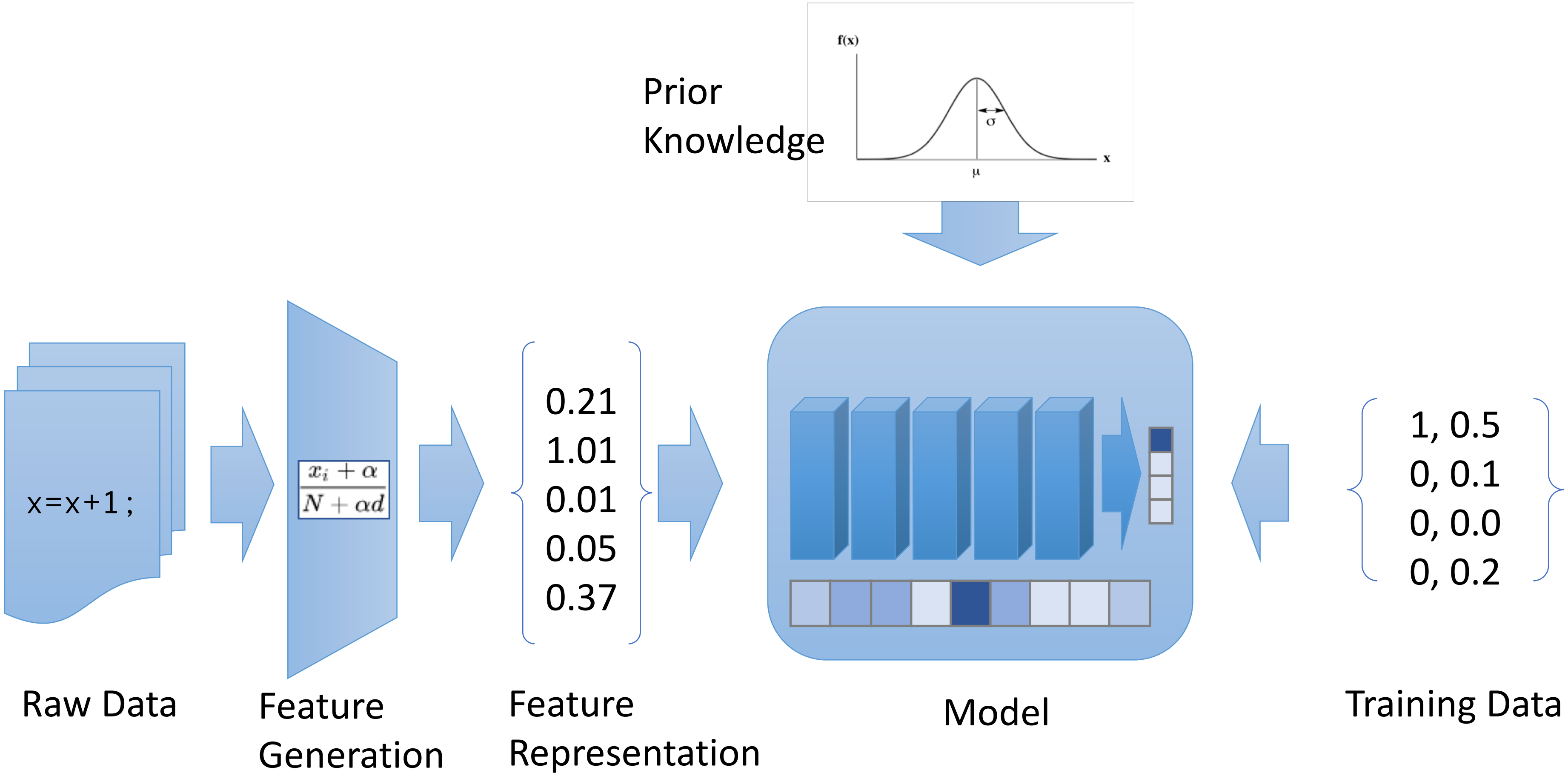
General Task: Representation Learning



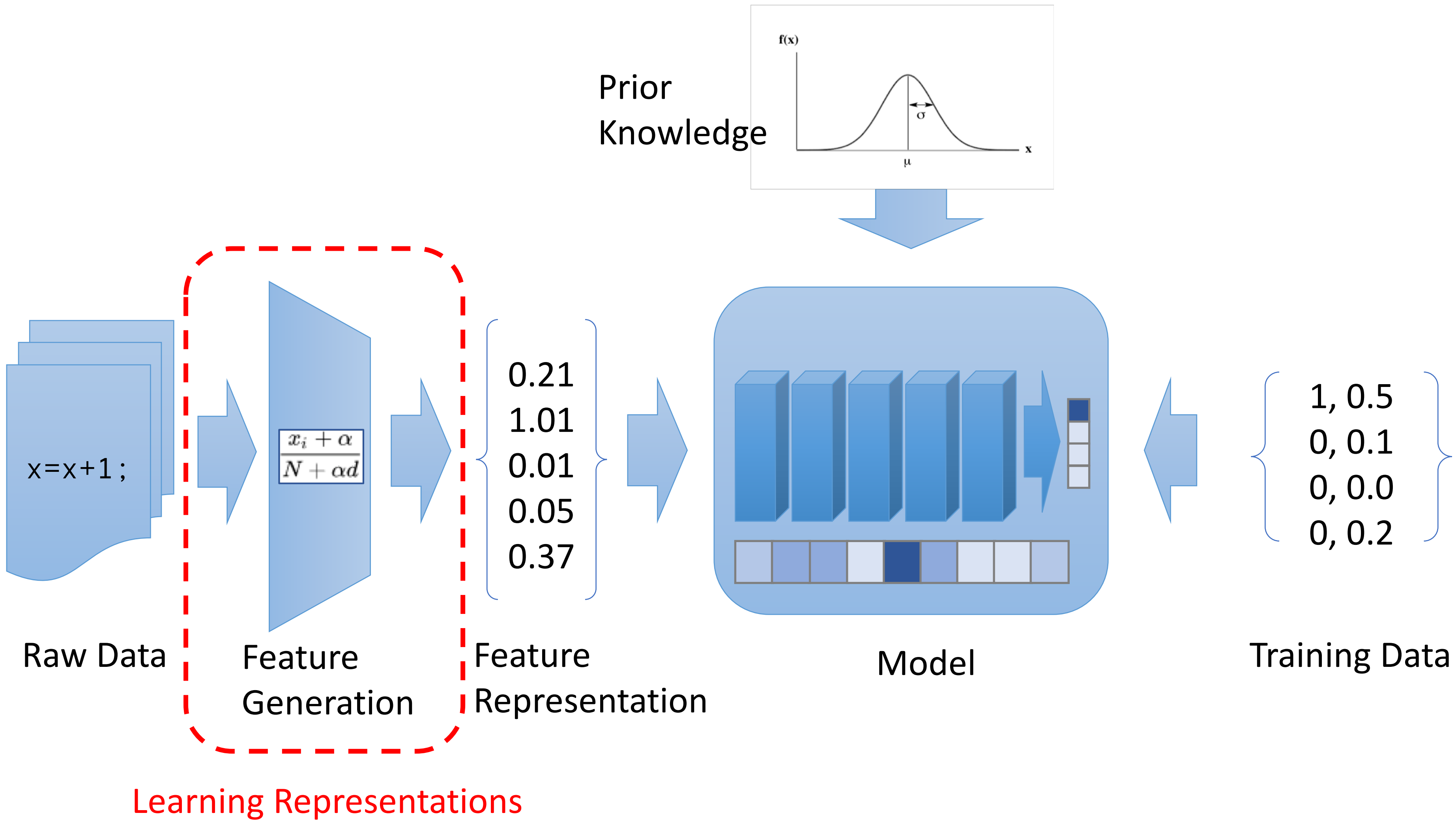
General Task: Representation Learning



General Task: Representation Learning



General Task: Representation Learning



History of Static Code Representation

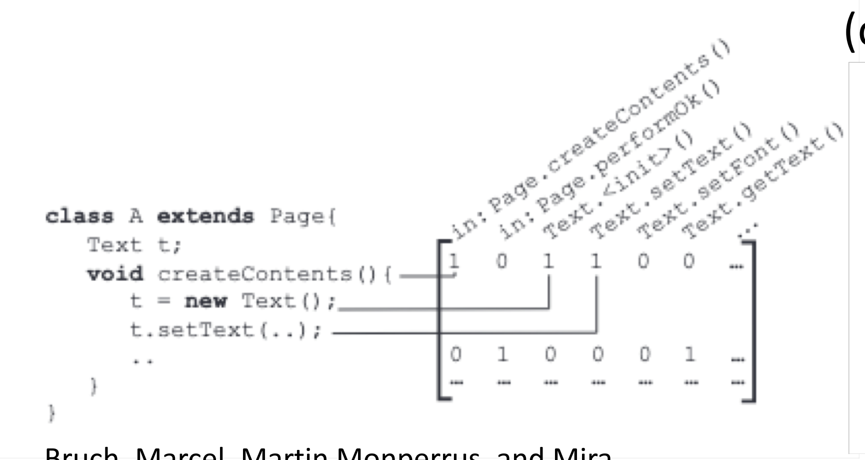
Exact Representation → Constructed Features → Deep Learning Features

Static Rule Inference + Checking

```
void foo() {
    lock(l);       // Enter critical section
    a = a + b;     // MAY: a,b protected by l
    unlock(l);     // Exit critical section
    b = b + 1;     // MUST: b not protected by l
}
```

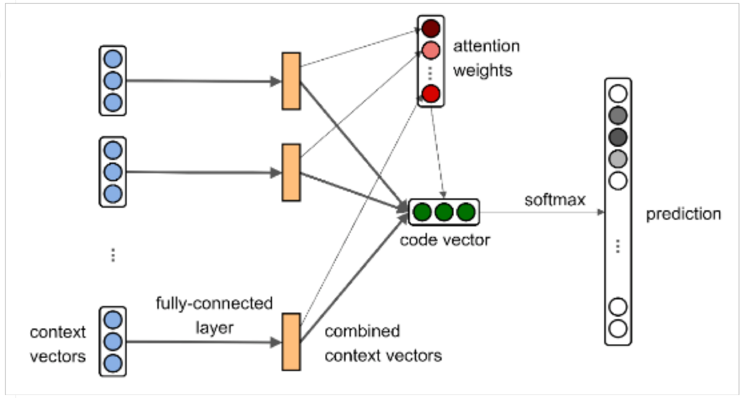
Engler, Dawson, et al. "Bugs as deviant behavior: A general approach to inferring errors in systems code." *ACM SIGOPS Operating Systems Review*. Vol. 35. No. 5. ACM, 2001.

Binary Feature Vectors, N-Grams



Bruch, Marcel, Martin Monperrus, and Mira Mezini. "Learning from examples to improve code completion systems." *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009.

“Semantic Space” Vector Embeddings (code2vec)



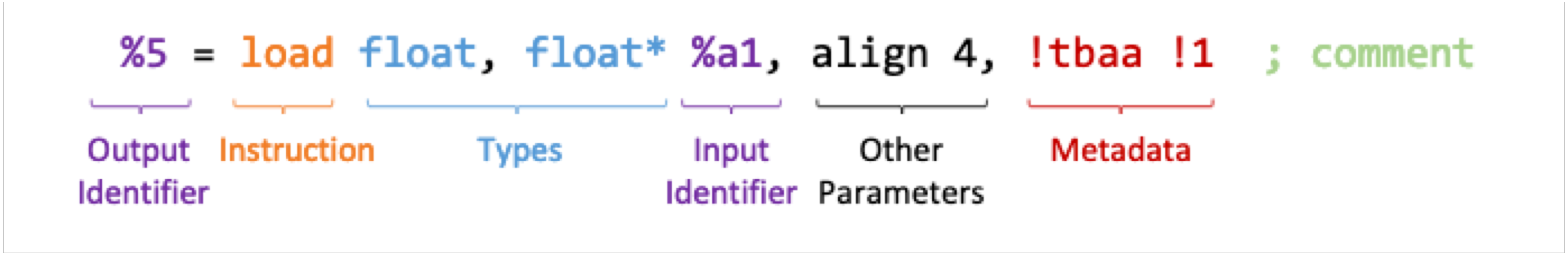
Alon, Uri, et al. "code2vec: Learning distributed representations of code." *Proceedings of the ACM on Programming Languages* 3.POPL (2019): 40.

Formal Code Comprehension Task

- Generally based on linguistic Distributional Hypothesis: **Statements** that occur in the same **contexts** tend to have **similar semantics**
- **Statements**: LLVM, each operation is unique & represents single action, Static Single Assignment (SSA) makes analysis easier
- **Context**: Statements that have either *Control Flow Dependencies* or *Data Dependencies*
- **Similarity**: Based on Alterations to System State

LLVM Intermediate Representation

Single statement:



LLVM Intermediate Representation

```
double thres = 5.0;
if (x < thres)
    x = y * y;
else
    x = 2.0 * y;
x += 1.0;
```

Source Code

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```

LLVM IR

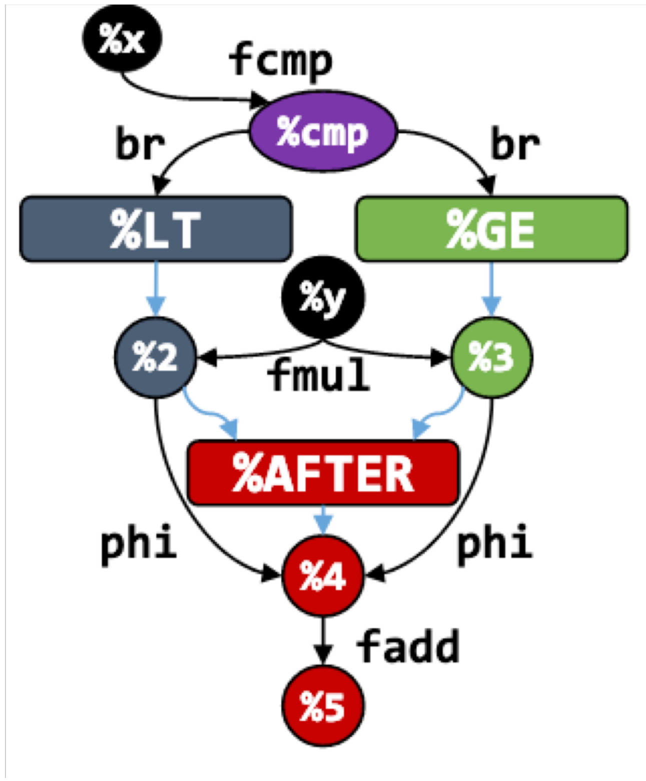
LLVM Intermediate Representation

```
; Function Attrs: noinline nounwind optnone ssp uwtable
define void @bubbleSort(i32*, i32) #0 {
  %3 = alloca i32*, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  store i32* %0, i32** %3, align 8
  store i32 %1, i32* %4, align 4
  store i32 0, i32* %5, align 4
  br label %8

; <label>:8:                                      ; preds = %61, %2
  %9 = load i32, i32* %5, align 4
  %10 = load i32, i32* %4, align 4
  %11 = sub nsw i32 %10, 1
  %12 = icmp slt i32 %9, %11
  br i1 %12, label %13, label %64

; <label>:13:                                     ; preds = %8
  store i32 0, i32* %6, align 4
  br label %14

; <label>:14:                                     ; preds = %57, %13
  %15 = load i32, i32* %6, align 4
  %16 = load i32, i32* %4, align 4
  %17 = load i32, i32* %5, align 4
  %18 = sub nsw i32 %16, %17
  %19 = sub nsw i32 %18, 1
  %20 = icmp slt i32 %15, %19
  br i1 %20, label %21, label %60

; <label>:21:                                     ; preds = %14
  %22 = load i32*, i32** %3, align 8
  %23 = load i32, i32* %6, align 4
  %24 = sext i32 %23 to i64
  %25 = getelementptr inbounds i32, i32* %22, i64 %24
  %26 = load i32, i32* %25, align 4
  %27 = load i32*, i32** %3, align 8
  %28 = load i32, i32* %6, align 4
  %29 = add nsw i32 %28, 1
  %30 = sext i32 %29 to i64
  %31 = getelementptr inbounds i32, i32* %27, i64 %30
  %32 = load i32, i32* %31, align 4
  %33 = icmp sgt i32 %26, %32
  br i1 %33, label %34, label %56

; <label>:34:                                     ; preds = %21
  %35 = load i32*, i32** %3, align 8
  %36 = load i32, i32* %6, align 4
  %37 = sext i32 %36 to i64
  %38 = getelementptr inbounds i32, i32* %35, i64 %37
  %39 = load i32, i32* %38, align 4
  store i32 %39, i32* %7, align 4
  %40 = load i32*, i32** %3, align 8
  %41 = load i32, i32* %6, align 4
  %42 = add nsw i32 %41, 1
  %43 = sext i32 %42 to i64
  %44 = getelementptr inbounds i32, i32* %40, i64 %43
  %45 = load i32, i32* %44, align 4
  %46 = load i32*, i32** %3, align 8
  %47 = load i32, i32* %6, align 4
  %48 = sext i32 %47 to i64
  %49 = getelementptr inbounds i32, i32* %46, i64 %48
  store i32 %45, i32* %49, align 4
  %50 = load i32, i32* %7, align 4
  %51 = load i32*, i32** %3, align 8
  %52 = load i32, i32* %6, align 4
  %53 = add nsw i32 %52, 1
  %54 = sext i32 %53 to i64
  %55 = getelementptr inbounds i32, i32* %51, i64 %54
  store i32 %50, i32* %55, align 4
  br label %56

; <label>:56:                                     ; preds = %34, %21
  br label %57

; <label>:57:                                     ; preds = %56
  %58 = load i32, i32* %6, align 4
  %59 = add nsw i32 %58, 1
  store i32 %59, i32* %6, align 4
  br label %14

; <label>:60:                                     ; preds = %14
  br label %61

; <label>:61:                                     ; preds = %60
  %62 = load i32, i32* %5, align 4
  %63 = add nsw i32 %62, 1
  store i32 %63, i32* %5, align 4
  br label %8

; <label>:64:                                     ; preds = %8
  ret void
}
```

Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```



Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

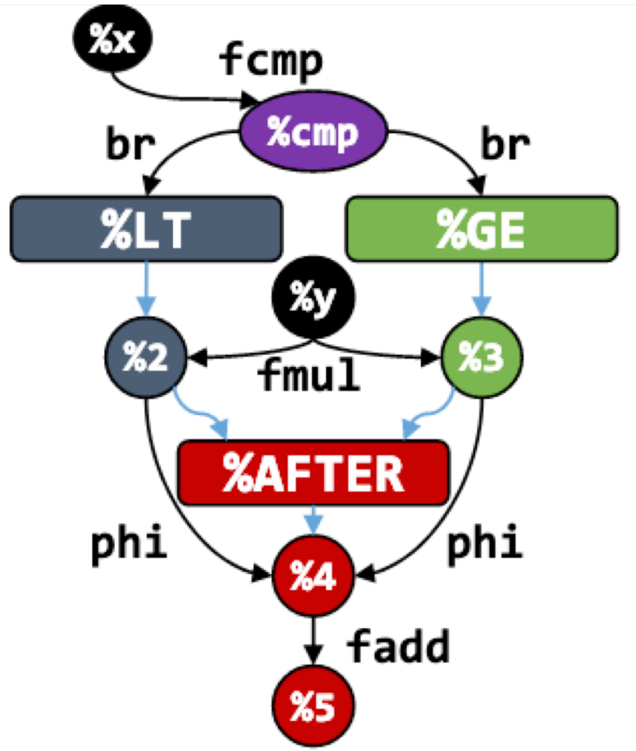
Construction in 2 Passes O(n):

1. First Pass store all function names and return statements
2. Second pass construct graph as follows:
    1. Direct data dependencies connected within basic block
    2. Conditional Branches create data dependencies to labels
    3. Merge Operations connect data dependencies and also connect through label
    4. Identifiers without parent connected to root function or label

Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

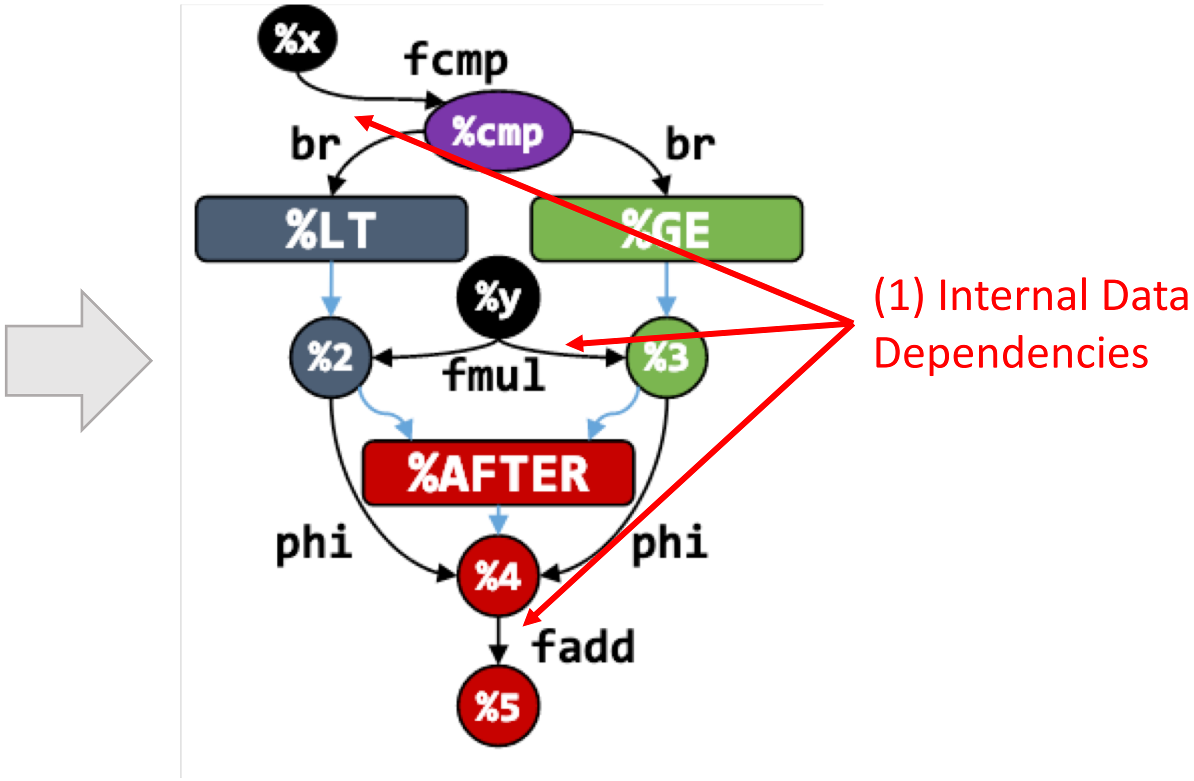
```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```



Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

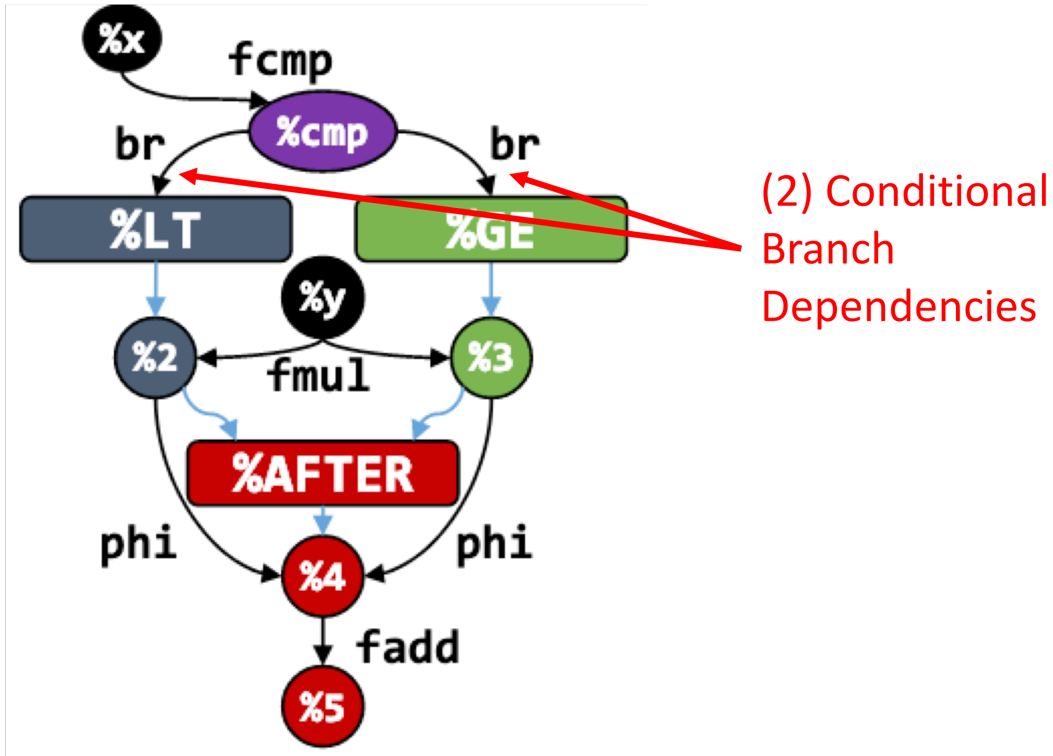
```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```



Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

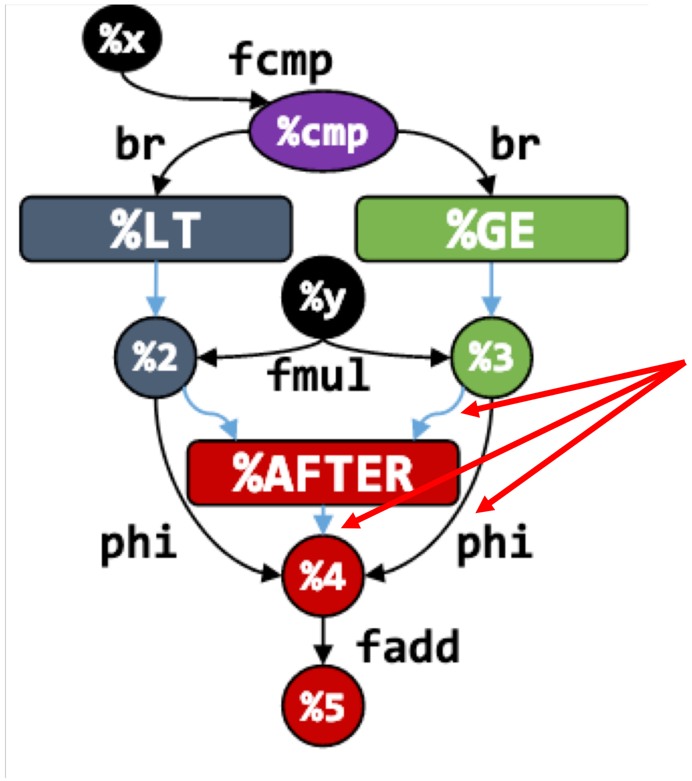
```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```



Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```

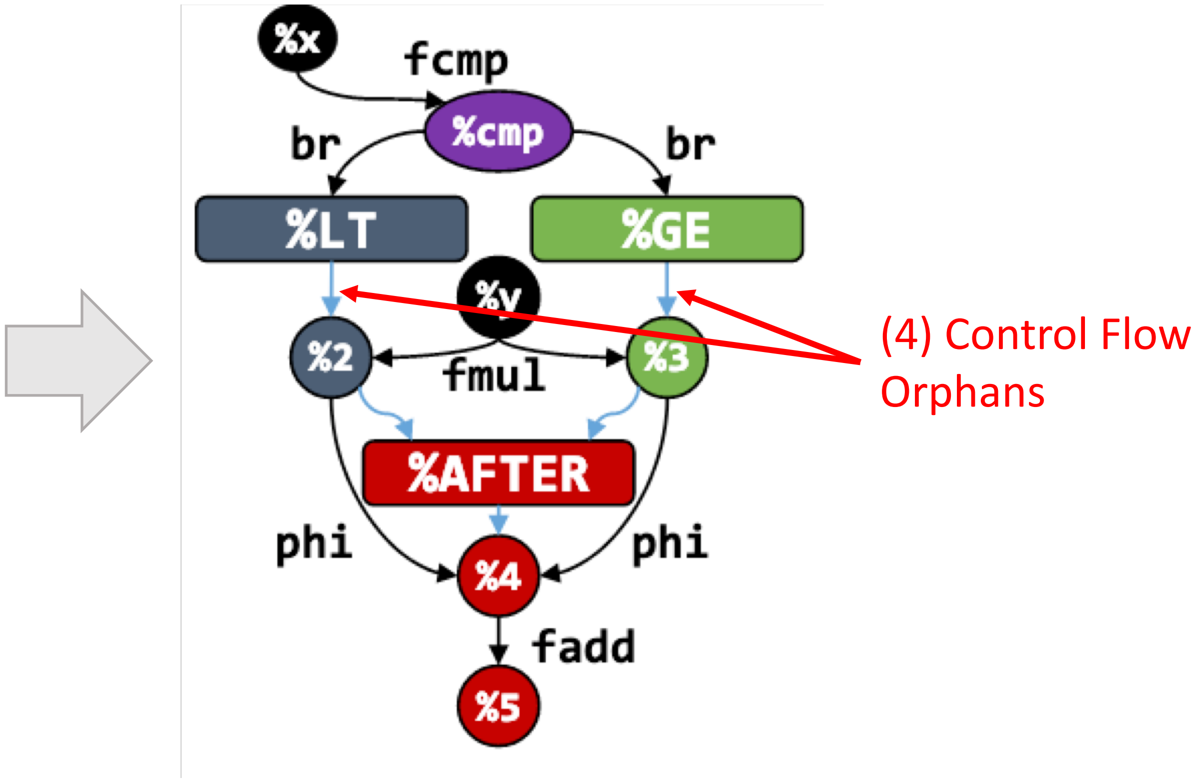


(3) Merge Op Dependencies

Contextual Flow Graph (XFG not CFG)

- **Nodes**: variables or labels (functions or basic blocks)
- **Edges**: Data Dependence or Execution Dependence

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```



Statement Embeddings: Skipgram model

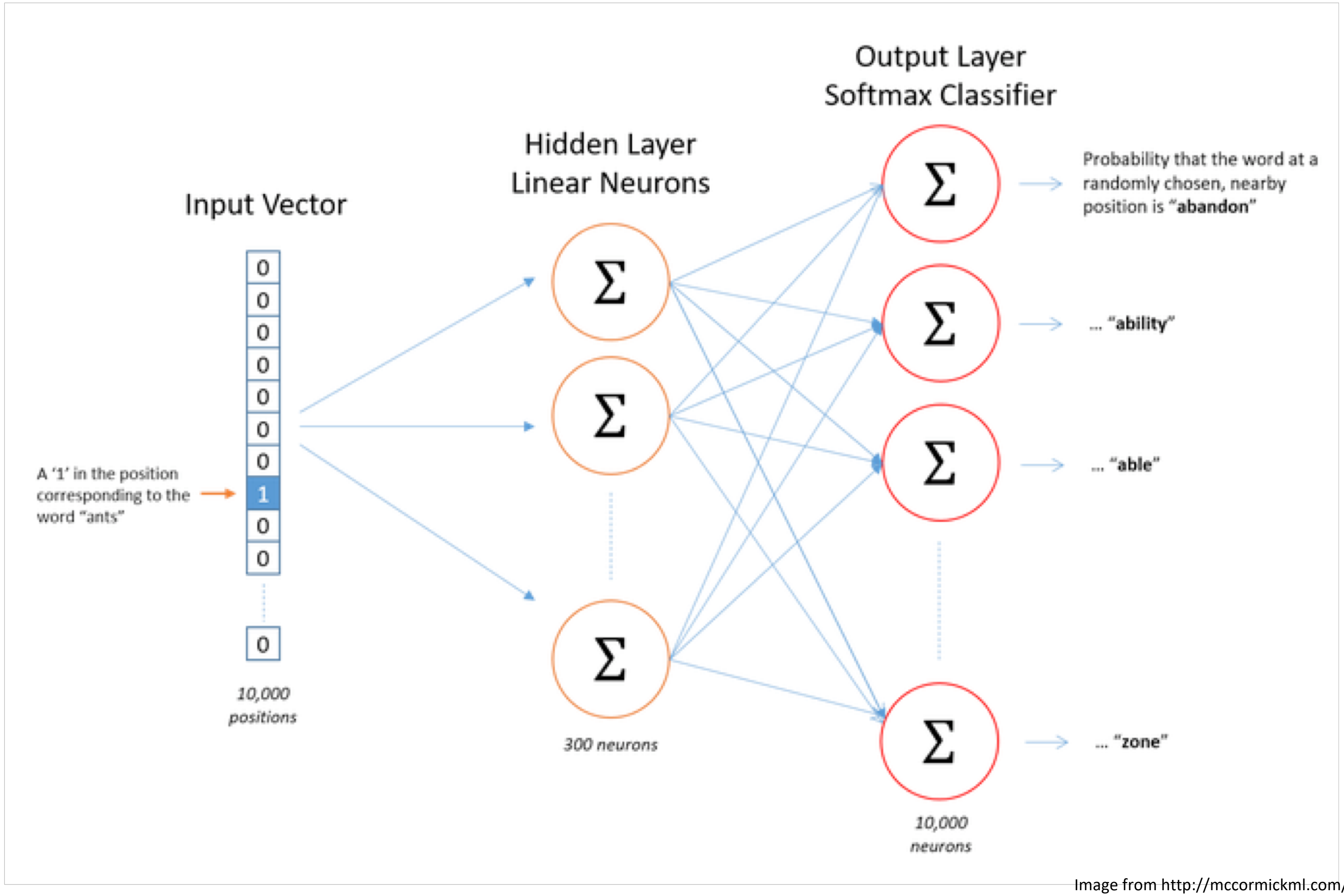


Image from http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Statement Embeddings: Skipgram model

Example: context size = 2

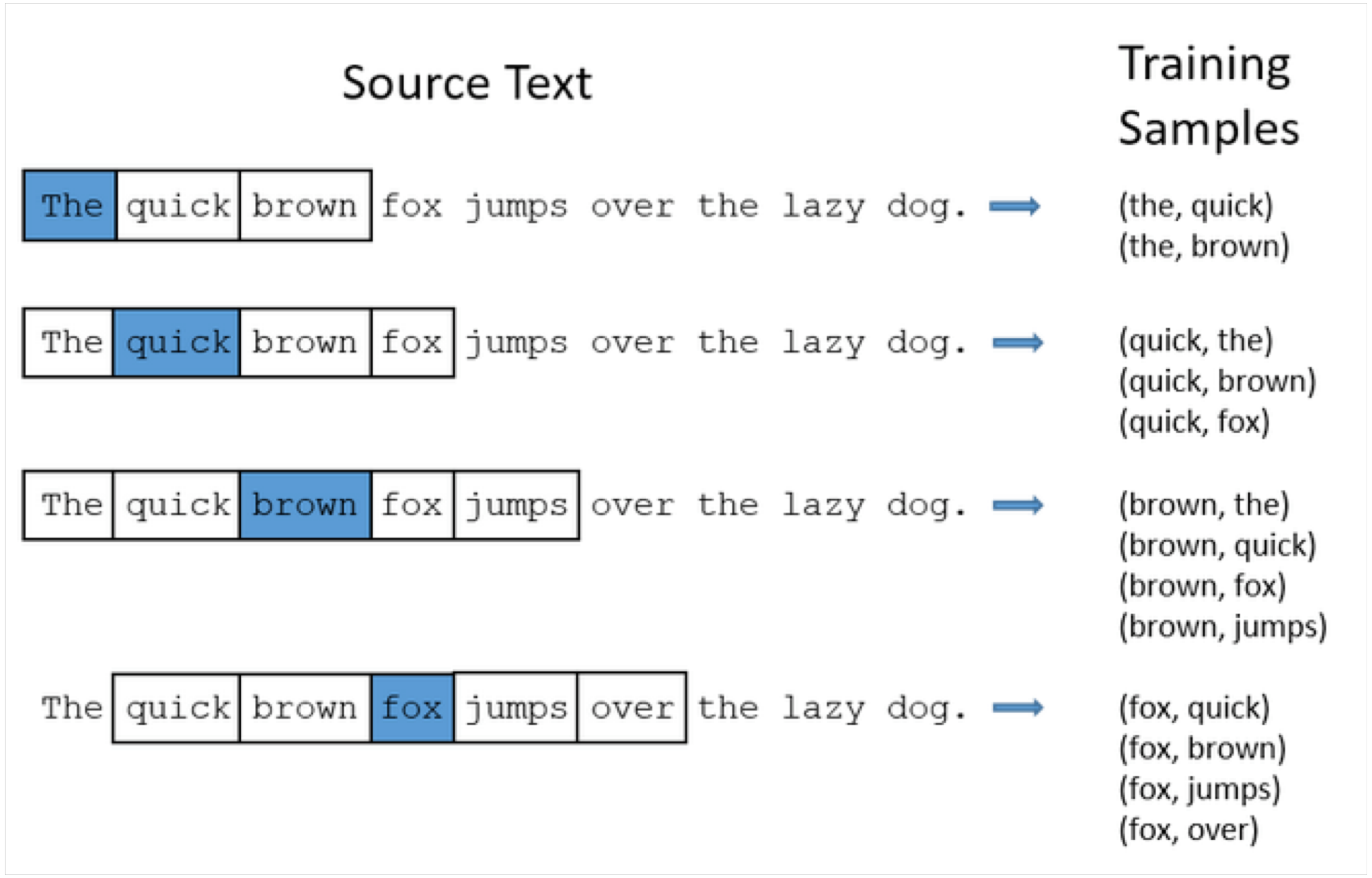
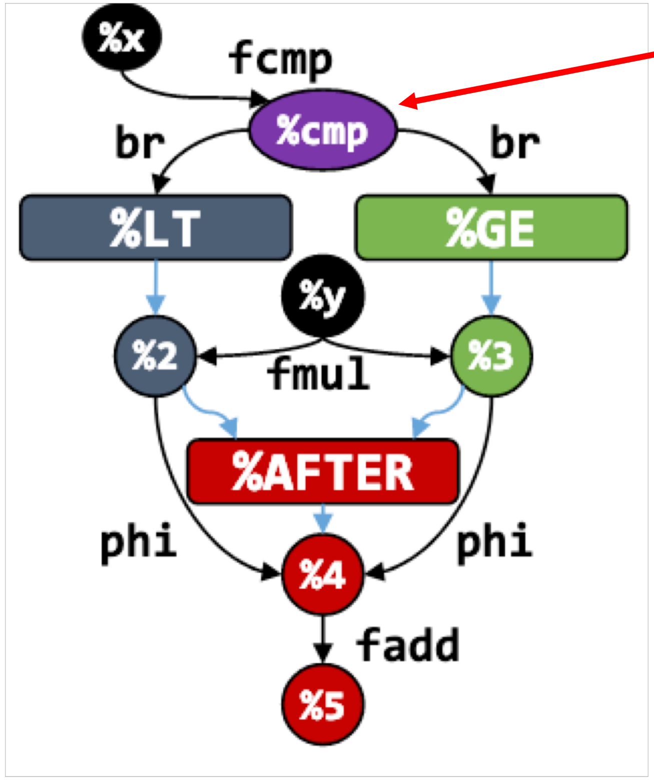


Image from http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Statement Embeddings: Skipgram model

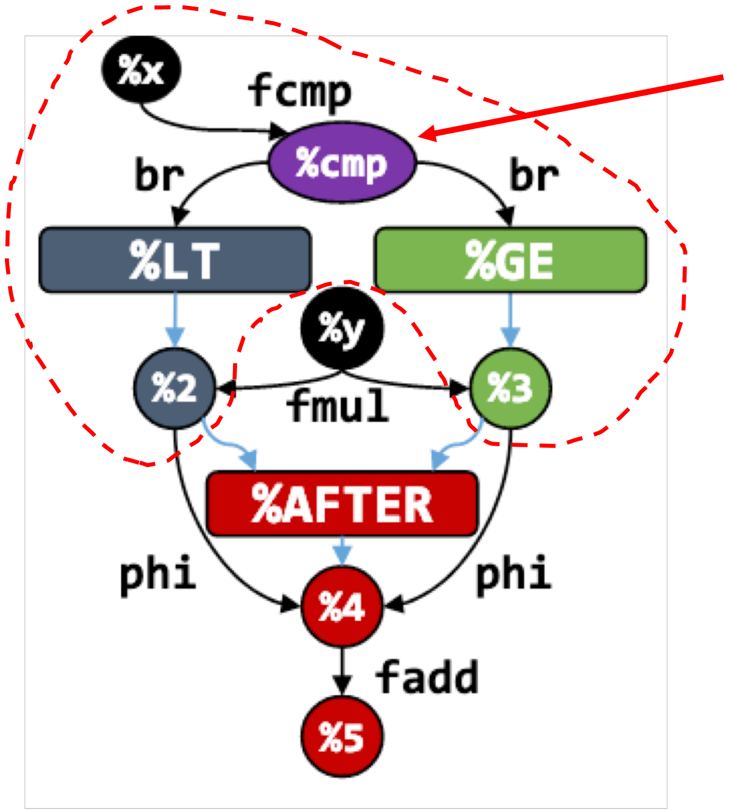
Example: context size = 2



Training Pairs for %cmp:

Statement Embeddings: Skipgram model

Example: context size = 2



Training Pairs for %cmp:

(%cmp, %x)
(%cmp, %LT)
(%cmp, %RT)
(%cmp, %2)
(%cmp, %3)

Data Preparation

- Preprocess put in generic ids and types:
  - Id -> %ID
  - Float literal -> FLOAT (same for ints)

```
store float %250, float* %82, align 4, !tbaa !1
%10 = fadd fast float %9, 1.3
%8 = load %"struct.aaa"*, %"struct.aaa"** %2
```

(a) LLVM IR

```
store float %ID, float* %ID, align 4
%ID = fadd fast float %ID, <FLOAT>
%ID = load { float, float }*, { float, float }** %ID
```

(b) inst2vec statements

Data Preparation

- Preprocess put in generic ids and types:
  - Id -> %ID
  - Float literal -> FLOAT (same for ints)

```
store float %250, float* %82, align 4, !tbaa !1
%10 = fadd fast float %9, 1.3
%8 = load %"struct.aaa"*, %"struct.aaa"** %2
```

(a) LLVM IR

```
store float %ID, float* %ID, align 4
%ID = fadd fast float %ID, <FLOAT>
%ID = load { float, float }*, { float, float }** %ID
```

(b) inst2vec statements

- Discard rare statements (<300)

Data Preparation

- Preprocess put in generic ids and types:
  - Id -> %ID
  - Float literal -> FLOAT (same for ints)

```
store float %250, float* %82, align 4, !tbaa !1
%10 = fadd fast float %9, 1.3
%8 = load %"struct.aaa"*, %"struct.aaa"** %2
```

(a) LLVM IR

```
store float %ID, float* %ID, align 4
%ID = fadd fast float %ID, <FLOAT>
%ID = load { float, float }*, { float, float }** %ID
```

(b) inst2vec statements

- Discard rare statements (<300)
- Subsample frequent pairs[1]

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

$P(w_i)$ = Discard Probability
t = hyperparameter 10^{-5}
$f(w_i)$= w_i frequency

1. Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.

Embedding Model

- Embedding Dimension = 200
- Implemented in Tensorflow
- Train for 5 epochs over given dataset
- Adam optimizer default params

Embedding Model

- Embedding Dimension = 200
- Implemented in Tensorflow
- Train for 5 epochs over given dataset
- Adam optimizer default params

Batch Size? Time & resources to train?

Embedding Data

Table 1: inst2vec training dataset statistics

Discipline	Dataset	Files	LLVM IR Lines	Vocabulary Size	XFG Stmt. Pairs
Machine Learning	Tensorflow [1]	2,492	16,943,893	220,554	260,250,973
High-Performance Computing	AMD APP SDK [9]	123	1,304,669	4,146	45,081,359
	BLAS [22]	300	280,782	566	283,856
Benchmarks	NAS [57]	268	572,521	1,793	1,701,968
	Parboil [59]	151	118,575	2,175	151,916
	PolybenchGPU [27]	40	33,601	577	40,975
	Rodinia [14]	92	103,296	3,861	266,354
	SHOC [21]	112	399,287	3,381	12,096,508
Scientific Computing	COSMO [11]	161	152,127	2,344	2,338,153
Operating Systems	Linux kernel [42]	1,988	2,544,245	136,545	5,271,179
Computer Vision	OpenCV [36]	442	1,908,683	39,920	10,313,451
	NVIDIA samples [17]	60	43,563	2,467	74,915
Synthetic	Synthetic	17,801	26,045,547	113,763	303,054,685
Total (Combined)	—	24,030	50,450,789	8,565	640,926,292

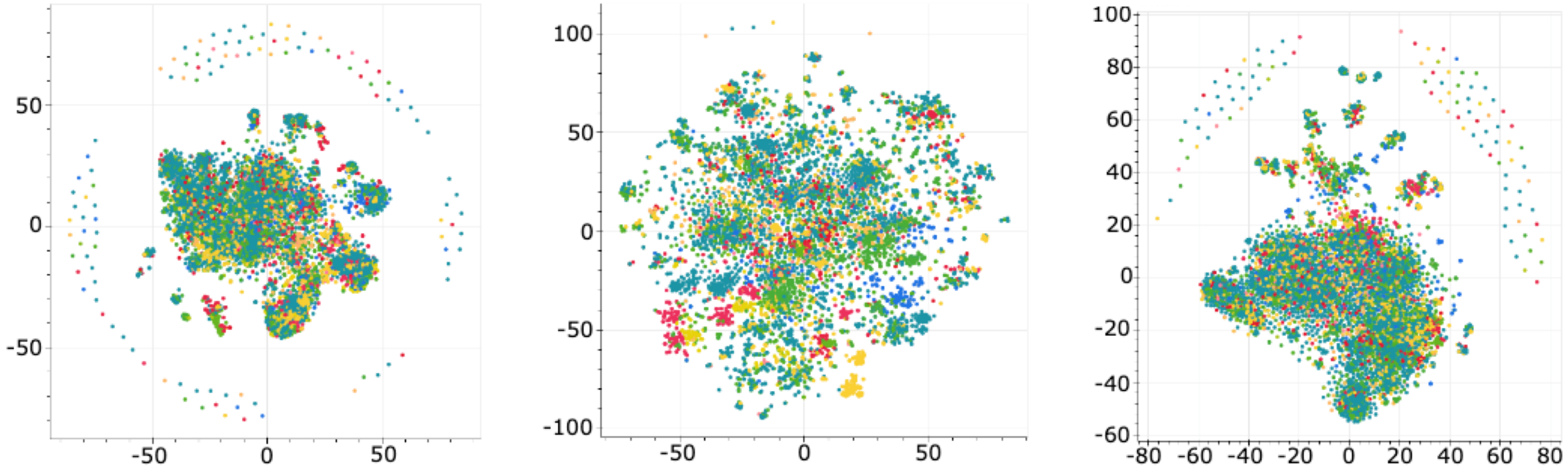
Embedding Data

Table 1: `inst2vec` training dataset statistics

Discipline	Dataset	Files	LLVM IR Lines	Vocabulary Size	XFG Stmt. Pairs
Machine Learning	Tensorflow [1]	2,492	16,943,893	220,554	260,250,973
High-Performance Computing	AMD APP SDK [9]	123	1,304,669	4,146	45,081,359
	BLAS [22]	300	280,782	566	283,856
Benchmarks	NAS [57]	268	572,521	1,793	1,701,968
	Parboil [59]	151	118,575	2,175	151,916
	PolybenchGPU [27]	40	33,601	577	40,975
	Rodinia [14]	92	103,296	3,861	266,354
	SHOC [21]	112	399,287	3,381	12,096,508
Scientific Computing	COSMO [11]	161	152,127	2,344	2,338,153
Operating Systems	Linux kernel [42]	1,988	2,544,245	136,545	5,271,179
Computer Vision	OpenCV [36]	442	1,908,683	39,920	10,313,451
	NVIDIA samples [17]	60	43,563	2,467	74,915
Synthetic	Synthetic	17,801	26,045,547	113,763	303,054,685
Total (Combined)	—	24,030	50,450,789	8,565	640,926,292

Half Synthetic?

Evaluation: Clustering



(a) Context size = 1 (b) Context size = 2 (c) Context size = 3

<d x int> operation	<%ID> = and <8 x i32> <%ID>, <%ID>
<d x struct/class*> operation	store <2 x { i64, i64 }*> <%ID>, <2 x { i64, i64 }*>* <%ID>, align 8
int operation	<%ID> = add i16 <%ID>, <INT>
type conversion operation	<%ID> = bitcast <4 x i32> <%ID> to <16 x i8>
floating point* operation	<%ID> = icmp eq double* <%ID>, null
floating point operation	<%ID> = getelementptr double, double* <%ID>, i64 <%ID>
<d x floating point> operation	<%ID> = call <4 x float> <@ID>(float* <%ID>)
void function definition	define linkonce_odr void <@ID>({ i32 (...)** }*) unnamed_addr

Evaluation: Clustering

Color	Statement Category	Example
	<d x int>* operation	<%ID> = load <2 x i64>*, <2 x i64>** <%ID>, align 8
	<d x int> operation	<%ID> = and <8 x i32> <%ID>, <%ID>
	<d x struct/class*> operation	store <2 x { i64, i64 }*> <%ID>, <2 x { i64, i64 }*>* <%ID>, align 8
	struct/class* operation	<%ID> = phi { float, float }* [<%ID>, <%ID>], [<%ID>, <%ID>]
	struct/class operation	<%ID> = alloca { i32, i32 }, align 4
	int** operation	<%ID> = phi i8** [<%ID>, <%ID>], [<%ID>, <%ID>]
	int* operation	<%ID> = load i8*, i8** <%ID>, align 8
	int operation	<%ID> = add i16 <%ID>, <INT>
	type conversion operation	<%ID> = bitcast <4 x i32> <%ID> to <16 x i8>
	global variable definition	<@ID> = global i32 <INT>, align 4
	<d x int*> operation	<%ID> = phi <4 x i8*> [<%ID>, <%ID>], [<%ID>, <%ID>]
	load function pointer	<%ID> = load { i32 (...)** }*, { i32 (...)** }** <%ID>, align 8
	store function pointer	store void ()* <@ID>, void ()** <%ID>, align 8
	floating point** operation	<%ID> = phi float** [<%ID>, <%ID>], [<%ID>, <%ID>]
	floating point* operation	<%ID> = icmp eq double* <%ID>, null
	floating point operation	<%ID> = getelementptr double, double* <%ID>, i64 <%ID>
	call void	tail call void <@ID>(i64 <INT>)
	other/misc.	cleanup; unreachable
	[d x [d x type]] operation	<%ID> = getelementptr inbounds [8 x [256 x i32]], [8 x [256 x i32]]*
	[d x struct/class] operation	<%ID> = alloca [5 x { i8*, i64 }], align 8
	[d x int] operation	<%ID> = alloca [100 x i8], align 16
	[d x floating point] operation	<%ID> = getelementptr inbounds [1024 x double], [1024 x double]*
	<d x floating point>* operation	<%ID> = alloca <8 x float>*, align 8
	<d x floating point> operation	<%ID> = call <4 x float> <@ID>(float* <%ID>)
	void function definition	define linkonce_odr void <@ID>({ i32 (...)** }*) unnamed_addr
	invoke void	invoke void <@ID>(i8* <%ID>) to label <%ID> unwind label <%ID>

Evaluation: 4 Experiments

- Analogies
- Algorithm Classification
- Compute Device Mapping
- Thread Coarsening

Evaluation: Analogies

- Generate test analogies from LLVM IR Syntax
- Analogy types:
  - Same op different types
  - Adding options to different ops
  - Type conversions
  - Data Structures

```
%ID = add i64 %ID, %ID    :   %ID = fadd float %ID, %ID;
%ID = sub i64 %ID, %ID    :?  %ID = fsub float %ID, %ID
```

```
%ID = extractvalue { double, double } %ID, 0   :   %ID = extractelement <2 x double> %ID, <TYP> 0;
%ID = extractvalue { double, double } %ID, 1   :?  %ID = extractelement <2 x double> %ID, <TYP> 1
```

Evaluation: Analogies

Table 2: Analogy and test scores for `inst2vec`

Context type	Context Size	Syntactic Analogies		Semantic Analogies		Semantic Distance Test
		Types	Options	Conversions	Data Structures	
CFG	1	0 (0 %)	1 (1.89 %)	1 (0.07 %)	0 (0 %)	51.59 %
	2	1 (0.18 %)	1 (1.89 %)	0 (0 %)	0 (0 %)	50.47 %
	3	0 (0 %)	1 (1.89 %)	4 (0.27 %)	0 (0 %)	53.79 %
DFG	1	53 (9.46 %)	12 (22.64 %)	2 (0.13 %)	4 (50.00 %)	56.79 %
	2	71 (12.68 %)	12 (22.64 %)	12 (0.80 %)	3 (37.50 %)	57.44 %
	3	67 (22.32 %)	18 (33.96 %)	40 (2.65 %)	4 (50.00 %)	60.38 %
XFG	1	101 (18.04 %)	13 (24.53 %)	100 (6.63 %)	3 (37.50 %)	60.98 %
	2	**226 (40.36 %)**	**45 (84.91 %)**	**134 (8.89 %)**	**7 (87.50 %)**	**79.12 %**
	3	125 (22.32 %)	24 (45.28 %)	48 (3.18 %)	**7 (87.50 %)**	62.56 %

Evaluation: Classification Model

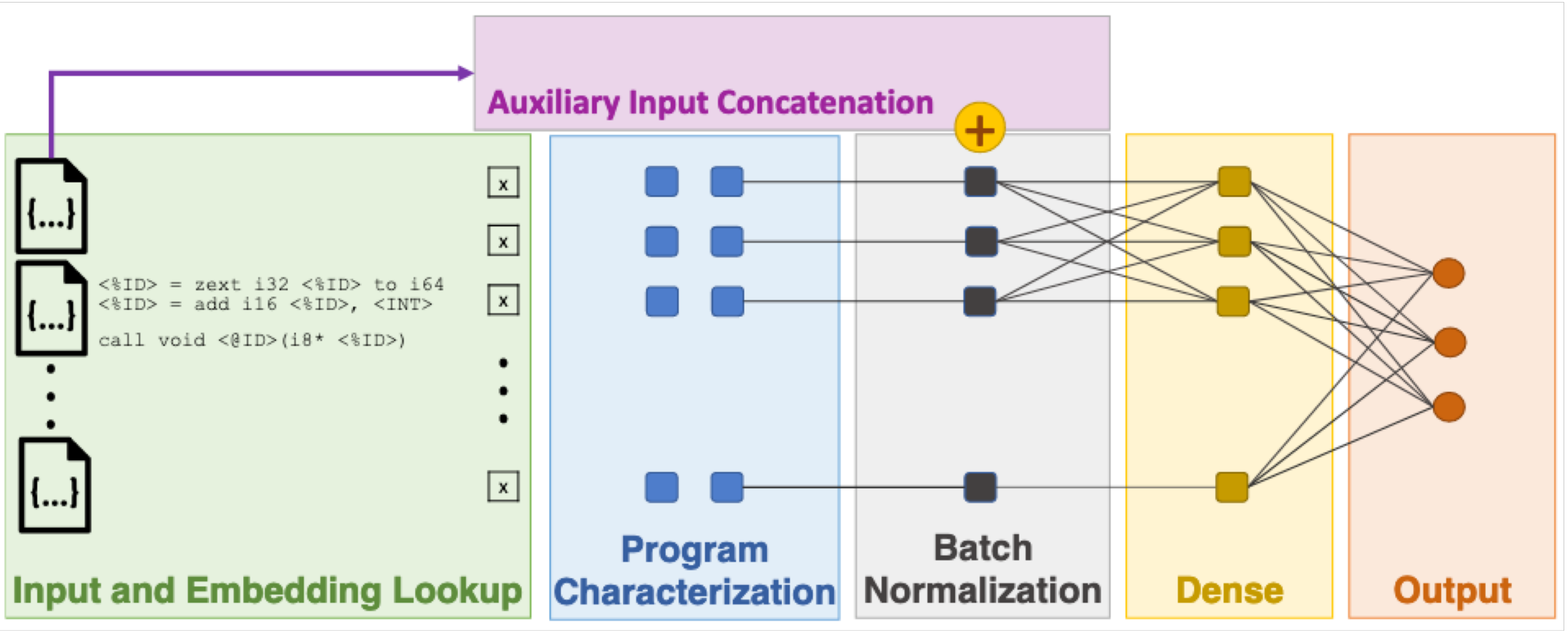
- 2 Stacked LSTMs with 200 units
- Batchnorm
- Dense Layer with 32 Units
- Output Softmax Classifier with Crossentropy Loss

Evaluation: Classification Model

- 2 Stacked LSTMs with 200 units
- Batchnorm
- Dense Layer with 32 Units
- Output Softmax Classifier with Crossentropy Loss

Could another model do better?

Evaluation: Classification Model



Evaluation: Algorithm Classification

- Given Program, predict what algorithm it implements (identical input/output)
- POJ 104 Dataset: 104 algorithm classes written by 500 people
  - https://github.com/ChrisCummins/paper-end2end-dl
  - Compare with Tree based CNNs (previous best)
- Use precomputed inst2vec embedding (not trained on POJ 104)

Table 3: Algorithm classification test accuracy

Metric	Surface Features [49] (RBF SVM + Bag-of-Trees)	RNN [49]	TBCNN [49]	inst2vec
Test Accuracy [%]	88.2	84.8	94.0	**94.83**

Evaluation: Compute Device Mapping

- Predict whether program will run faster on CPU or GPU
- OpenCL Code Dataset (https://sites.google.com/site/treebasedcnn/)
- Use Data Input Size and Work Group Size (number threads) as additional inputs
- Optionally incorporate immediate values (ie %x instead of %ID)

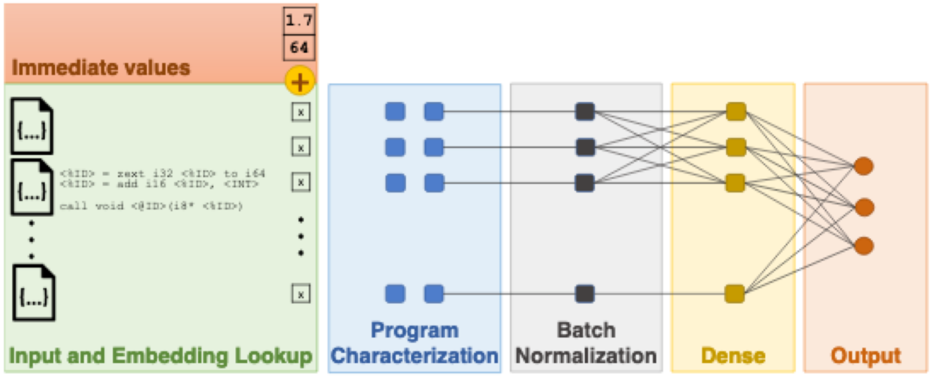
Evaluation: Compute Device Mapping

Table 4: Heterogeneous device mapping results

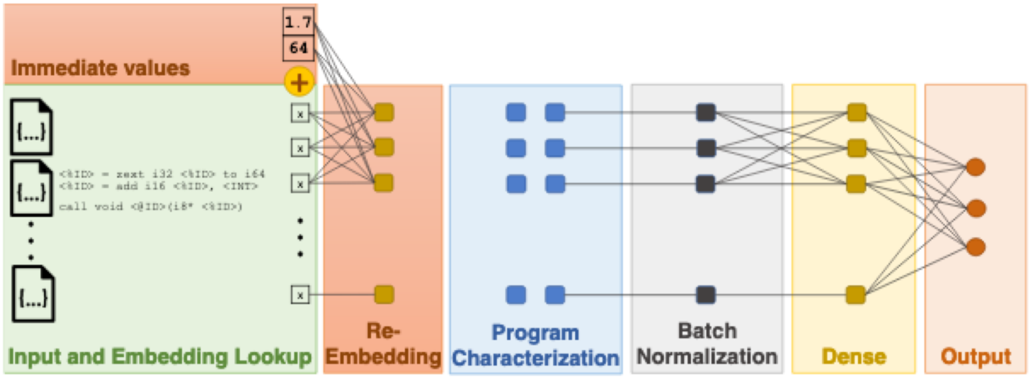
Architecture	Prediction Accuracy [%]				
	GPU	Grewe et al. [29]	DeepTune [18]	inst2vec	inst2vec-imm
AMD Tahiti 7970	41.18	73.38	83.68	82.79	**88.09**
NVIDIA GTX 970	56.91	72.94	80.29	82.06	**86.62**
	Speedup				
	GPU	Grewe et al.	DeepTune	inst2vec	inst2vec-imm
AMD Tahiti 7970	3.26	2.91	3.34	3.42	**3.47**
NVIDIA GTX 970	1.00	1.26	1.41	1.42	**1.44**

Evaluation: Compute Device Mapping

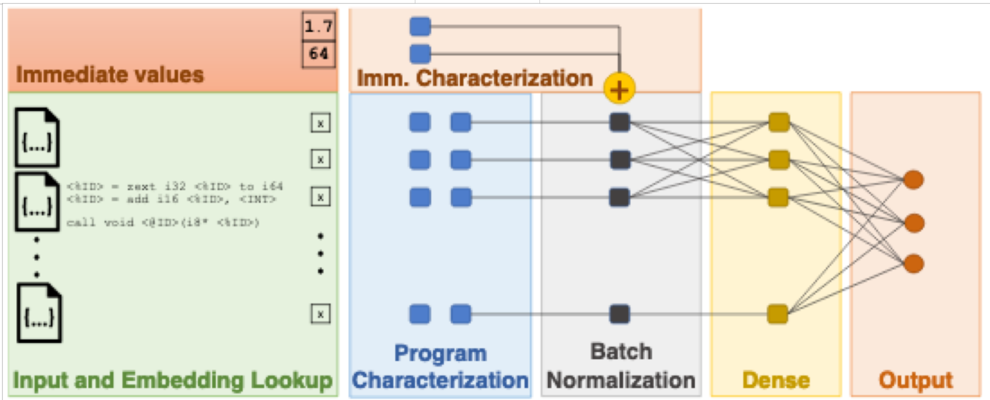
Immediate Value concatenation types:



(a) concat_naïve



(b) concat_embed



(c) extract_concat

Evaluation: Compute Device Mapping

Immediate Value concatenation results:

Architecture	Prediction Accuracy [%]				Speedup			
	ignore	concat naïve	extract concat	concat emb	ignore	concat naïve	extract concat	concat emb
AMD Tahiti 7970	82.79	**88.09**	76.18	72.06	3.42	**3.47**	3.36	2.76
NVIDIA GTX 970	82.06	**86.62**	79.71	72.50	1.42	**1.44**	1.40	1.32

Evaluation: Thread Coarsening

- Predict optimal thread coarsening factor = reduce number of GPU threads on OpenCL program
- Options are 1, 2, 4, 8, 16, 32
- Explain poorer performance with small dataset (17 programs) vs 680 per platform for device mapping

Table 5: Speedups achieved by coarsening threads

Computing Platform	Magni et al. [46]	DeepTune [18]	DeepTune-TL [18]	inst2vec	inst2vec-imm
AMD Radeon HD 5900	1.21	1.10	1.17	**1.37**	1.28
AMD Tahiti 7970	1.01	1.05	**1.23**	1.10	1.18
NVIDIA GTX 480	0.86	1.10	**1.14**	1.07	1.11
NVIDIA Tesla K20c	0.94	0.99	0.93	**1.06**	1.00

Evaluation: Thread Coarsening

Immediate Value concatenation results:

Computing Platform	Speedup			
	ignore	concat_naïve	extract_concat	concat_embed
AMD Radeon HD 5900	**1.37**	1.21	1.28	1.30
AMD Tahiti 7970	1.10	1.06	**1.18**	0.92
NVIDIA GTX 480	1.07	0.99	**1.11**	0.97
NVIDIA Tesla K20c	**1.06**	1.04	1.00	0.99

Related Work

- Token Sequences -> Embeddings -> LSTMs directly on source code
  - Model context based on Lexigraphic Locality, Dataflow, Control Flow, ASTs, Paths in ASTs
- Alternate Models: Conditional Random Fields
- XFG similar to Program Dependence Graph and Sea of Nodes but more flexible since not used by compiler backend

Discussion Questions

- Are you convinced by this paper?
  - Do XFG Skipgram Embeddings make sense?
  - Are the evaluations fair?

Discussion Questions

- Are you convinced by this paper?
  - Do XFG Skipgram Embeddings make sense?
  - Are the evaluations fair?
- How could the method be improved?

Discussion Questions

- Are you convinced by this paper?
  - Do XFG Skipgram Embeddings make sense?
  - Are the evaluations fair?
- How could the method be improved?
  - Use attention models, allow embedding to train in conjunction with model.
  - Try graph based models on XFG (Graph embeddings)

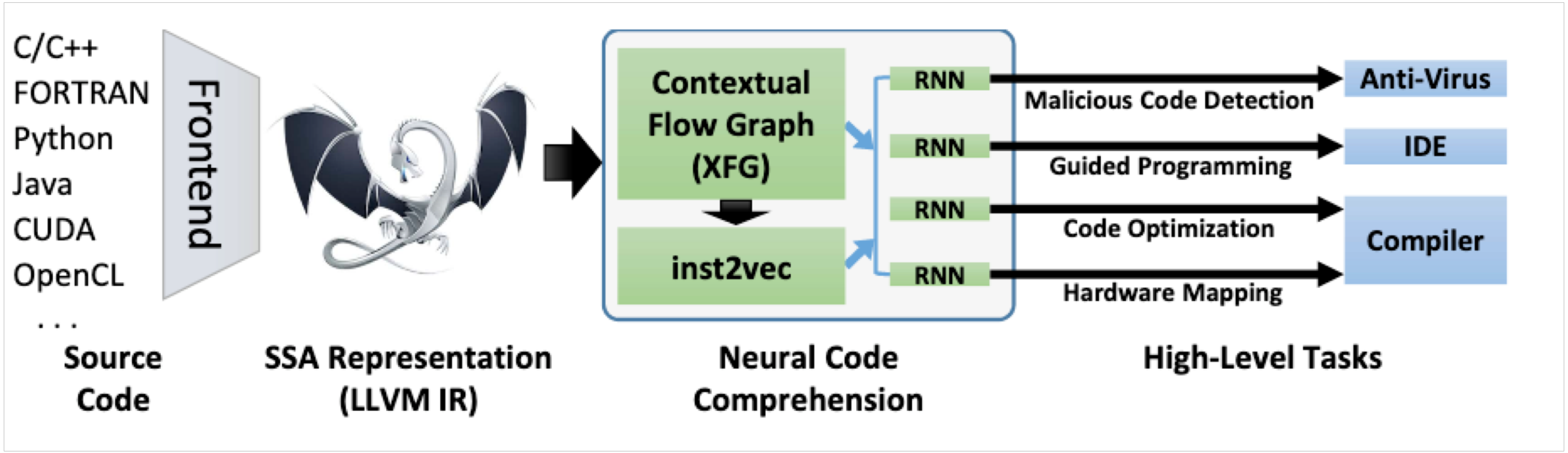
Discussion Questions

- Are you convinced by this paper?
    - Do XFG Skipgram Embeddings make sense?
    - Are the evaluations fair?
- How could the method be improved?
    - Use attention models, allow embedding to train in conjunction with model.
    - Try graph based models on XFG (Graph embeddings)
- What are other applications XFG embeddings could be used for?

Discussion Questions

- Are you convinced by this paper?
  - Do XFG Skipgram Embeddings make sense?
  - Are the evaluations fair?
- How could the method be improved?
  - Use attention models, allow embedding to train in conjunction with model.
  - Try graph based models on XFG (Graph embeddings)
- What are other applications XFG embeddings could be used for?
  - Code similarity
  - Predict internal properties like loop invariants
  - Code modeling (predict next symbol when typing)

Questions?


C/C++
FORTRAN
Python
Java
CUDA
OpenCL
. . .
Frontend
Contextual Flow Graph (XFG)
inst2vec
RNN
RNN
RNN
RNN
Malicious Code Detection
Guided Programming
Code Optimization
Hardware Mapping
Anti-Virus
IDE
Compiler
Source Code
SSA Representation (LLVM IR)
Neural Code Comprehension
High-Level Tasks