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

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.

```c
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.

```c
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:

```c
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

x = x + 1;

<table>
<thead>
<tr>
<th>Raw Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=x+1;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature Generation</th>
<th>Feature Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior Knowledge</td>
<td>Activations/Predictions</td>
</tr>
<tr>
<td></td>
<td>bubblesort</td>
</tr>
<tr>
<td></td>
<td>quicksort</td>
</tr>
<tr>
<td></td>
<td>mergesort</td>
</tr>
<tr>
<td></td>
<td>radixsort</td>
</tr>
</tbody>
</table>
General Task: Representation Learning

x=x+1;

\[
z_i = \alpha + \frac{N + \alpha d}{i}
\]

\[
\begin{align*}
0.21 \\
1.01 \\
0.01 \\
0.05 \\
0.37
\end{align*}
\]
General Task: Representation Learning

$$x = x + 1;$$

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Feature Generation</th>
<th>Feature Representation</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Image]</td>
<td>$z_i = \alpha / N + \alpha d$</td>
<td>0.21 1.01 0.01 0.05 0.37</td>
<td>[Diagram]</td>
</tr>
</tbody>
</table>

- bubblesort
- quicksort
- mergesort
- radixsort
General Task: Representation Learning

\[ x = x + 1; \]

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Feature Generation</th>
<th>Feature Representation</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior Knowledge</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activations/Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.98</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.03</td>
</tr>
<tr>
<td>0.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bubble Sort</th>
<th>Quick Sort</th>
<th>Merge Sort</th>
<th>Radix Sort</th>
</tr>
</thead>
</table>

0.21
1.01
0.01
0.05
0.37
## General Task: Representation Learning

### Raw Data

<table>
<thead>
<tr>
<th>Feature Generation</th>
<th>Feature Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = x + 1$;</td>
<td>$z_i + \alpha$</td>
</tr>
<tr>
<td></td>
<td>$N + \alpha d$</td>
</tr>
<tr>
<td>0.21</td>
<td>1.01</td>
</tr>
<tr>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>0.05</td>
<td>0.37</td>
</tr>
</tbody>
</table>

### Prior Knowledge

- Training Data: 1, 0.5, 0, 0.1, 0, 0.0, 0, 0.2

### Model
General Task: Representation Learning

Raw Data → Feature Generation → Feature Representation → Model → Training Data

$x = x + 1$

$\frac{z_i + \alpha}{N + \alpha d}$

Prior Knowledge

0.21 1.01 0.01 0.05 0.37

1, 0.5 0, 0.1 0, 0.0 0, 0.2
History of Static Code Representation

Exact Representation

Static Rule Inference + Checking

Binary Feature Vectors, N-Grams

Constructed Features

Deep Learning Features

“Semantic Space” Vector Embeddings (code2vec)


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:

```
%5 = load float, float* %a1, align 4, !tbaa !1 ; comment
```

<table>
<thead>
<tr>
<th>Output Identifier</th>
<th>Instruction</th>
<th>Types</th>
<th>Input Identifier</th>
<th>Other Parameters</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LLVM Intermediate Representation

Source Code

\[
\begin{align*}
\text{double thres} &= 5.0; \\
\text{if} \ (x < \text{thres}) & \quad x = y \ast y; \\
\text{else} & \quad x = 2.0 \ast y; \\
x &= x + 1.0;
\end{align*}
\]

LLVM IR

\[
\begin{align*}
\%\text{cmp} &= \text{fcmp olt double } \%x, 5.0 \\
\text{br} & \ i1 \ %\text{cmp}, \text{ label } \%\text{LT}, \text{ label } \%\text{GE} \\
\%2 &= \text{fmul double } \%y, \%y \\
\%3 &= \text{fmul double } 2.0, \%y \\
\text{AFTER:} & \quad \%4 = \text{phi double } [\%2, \%\text{LT}], [\%3, \%\text{GE}] \\
\%5 &= \text{fadd double } \%4, 1.0
\end{align*}
\]
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 %11 %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* %5, align 4
    %17 = load i32, i32* %4, align 4
    %18 = sub nsw i32 %16, %17
    %19 = sub nsw i32 %18 to %19
    br %18 %19, label %21, label %60

; <label>:21:                                     ; preds = %14
    %22 = load i32, i32* %3, align 8
    %23 = load i32, i32* %2, align 4
    %24 = sext i32 %23 to 164
    %25 = getelementptr inbounds i32, i32* %22, 164 %24
    %26 = load i32, i32* %25, align 4
    %27 = load i32, i32* %26, align 4
    %28 = add nsw i32 %27, 1
    %29 = load i32, i32* %24, align 4
    %30 = sext i32 %29 to 164
    %31 = getelementptr inbounds i32, i32* %27, 164 %30
    %32 = load i32, i32* %31, align 4
    %33 = load i32, i32* %26, align 4
    br %21 %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 164
    %38 = getelementptr inbounds i32, i32* %35, 164 %37
    %39 = load i32, i32* %38, align 4
    store i32 %39, i32* %7, align 4
    %40 = load i32, i32* %8, align 4
    %41 = load i32, i32* %6, align 4
    %42 = add nsw i32 %41, 1
    %43 = load i32, i32* %42 to 164
    %44 = getelementptr inbounds i32, i32* %43, 164 %44
    %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 164
    %49 = getelementptr inbounds i32, i32* %48, 164 %49
    store i32 %49, i32* %5, 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 = load i32 %53 to 164
    %55 = load i32, i32* %51, 164 %54
    store i32 %55, i32* %55, align 4
    br label %56

; <label>:56:                                         ; preds = %34, %21
    %57 = load i32, i32* %6, align 4
    %58 = load i32, i32* %5, align 4
    %59 = add nsw i32 %58, 1
    store i32 %59, i32* %59, align 4
    br label %54

; <label>:60:                                         ; preds = %14
    %61 = load i32, i32* %6, align 4
    %62 = load i32, i32* %5, align 4
    %63 = add nsw i32 %62, 1
    store i32 %63, i32* %5, align 4
    br label %58

; <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

```plaintext
%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

```plaintext
%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

(1) Internal Data Dependencies
Contextual Flow Graph (XFG not CFG)

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

\[
\begin{align*}
\%\text{cmp} &= \text{fcmp olt} \text{ double} \%x, \ 5.0 \\
\text{br} \ i1 \ \%\text{cmp}, \ \text{label} \ %\text{LT}, \ \text{label} \ %\text{GE} \\
\text{LT}: \quad \%2 &= \text{fmul} \text{ double} \ %y, \ %y \\
\text{GE}: \quad \%3 &= \text{fmul} \text{ double} \ 2.0, \ %y \\
\text{AFTER}: \quad \%4 &= \text{phi} \text{ double} \ [\%2, %\text{LT}], \ [\%3, %\text{GE}] \\
\%5 &= \text{fadd} \text{ double} \ %4, \ 1.0
\end{align*}
\]

(2) Conditional Branch Dependencies
Contextual Flow Graph (XFG not CFG)

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

```plaintext
%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
```

(4) Control Flow Orphans
Statement Embeddings: Skipgram model

[Diagram of a neural network showing an input vector and a hidden layer with softmax classifier.]

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)

(a) LLVM IR

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

(b) inst2vec statements

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

\[
\begin{align*}
\text{store float } & \%250, \text{ float* } \%82, \text{ align 4, !tbaa } \!1 \\
\%10 = \text{fadd fast float } & \%9, 1.3 \\
\%8 = \text{load } & \%"\text{struct.aa}"*, \%"\text{struct.aa}"** \%2 \\
\end{align*}
\]
(a) LLVM IR

\[
\begin{align*}
\text{store float } & \%ID, \text{ float* } \%ID, \text{ align 4} \\
\%ID = \text{fadd fast float } & \%ID, <\text{FLOAT}> \\
\%ID = \text{load } & \{ \text{float, float } \}*, \{ \text{float, float } \}** \%ID \\
\end{align*}
\]
(b) inst2vec statements

• Discard rare statements (<300)
• Subsample frequent pairs\(^1\)

\[
P(w_i) = 1 - \frac{t}{\sqrt{f(w_i)}}
\]
\[
P(w_i) = \text{Discard Probability}
\]
\[
t = \text{hyperparameter } 10^{-5}
\]
\[
f(w_i) = w_i \text{ frequency}
\]

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

Table 1: `inst2vec` training dataset statistics

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

(a) Context size = 1

(b) Context size = 2

(c) Context size = 3

- \texttt{<d x int> operation} as \texttt{<xID> = and <8 x i32> <xID>, <xID>}

- \texttt{<d x struct/class> operation} as \texttt{store <2 x (i64, i64)> <xID>, <2 x (i64, i64)>* <xID>, align 8}

- \texttt{int operation} as \texttt{<xID> = add i16 <xID>, <INT>}

- \texttt{type conversion operation} as \texttt{<xID> = bitcast <4 x i32> <xID> to <16 x i8>}

- \texttt{floating point* operation} as \texttt{<xID> = icmp eq double* <xID>, null}

- \texttt{floating point operation} as \texttt{<xID> = getelementptr double, double* <xID>, i64 <xID>}

- \texttt{<d x floating point> operation} as \texttt{<xID> = call <4 x float> <xID>(float* <xID>)}

- \texttt{void function definition} as \texttt{define linkonce_odr void <xID>(<i32 (...)*>*) unnamed_addr}
## Evaluation: Clustering

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

Table 2: Analogy and test scores for inst2vec

<table>
<thead>
<tr>
<th>Context type</th>
<th>Context Size</th>
<th>Syntactic Analogies</th>
<th>Semantic Analogies</th>
<th>Semantic Distance Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Types</td>
<td>Options</td>
<td>Conversions</td>
</tr>
<tr>
<td>CFG</td>
<td>1</td>
<td>0 (0 %)</td>
<td>1 (1.89 %)</td>
<td>1 (0.07 %)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1 (0.18 %)</td>
<td>1 (1.89 %)</td>
<td>0 (0 %)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0 (0 %)</td>
<td>1 (1.89 %)</td>
<td>4 (0.27 %)</td>
</tr>
<tr>
<td>DFG</td>
<td>1</td>
<td>53 (9.46 %)</td>
<td>12 (22.64 %)</td>
<td>2 (0.13 %)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>71 (12.68 %)</td>
<td>12 (22.64 %)</td>
<td>12 (0.80 %)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>67 (22.32 %)</td>
<td>18 (33.96 %)</td>
<td>40 (2.65 %)</td>
</tr>
<tr>
<td>XFG</td>
<td>1</td>
<td>101 (18.04 %)</td>
<td>13 (24.53 %)</td>
<td>100 (6.63 %)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>226 (40.36 %)</td>
<td>45 (84.91 %)</td>
<td>134 (8.89 %)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>125 (22.32 %)</td>
<td>24 (45.28 %)</td>
<td>48 (3.18 %)</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Metric</th>
<th>Surface Features</th>
<th>RNN</th>
<th>TBCNN</th>
<th>inst2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RBF SVM + Bag-of-Trees)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Accuracy [%]</td>
<td>88.2</td>
<td>84.8</td>
<td>94.0</td>
<td>94.83</td>
</tr>
</tbody>
</table>
Evaluation: Compute Device Mapping

• Predict whether program will run faster on CPU or GPU
• OpenCL Code Dataset ([https://sites.google.com/site/treebasedcnn/](https://sites.google.com/site/treebasedcnn/))
• Use Data Input Size and Work Group Size (number of threads) as additional inputs
• Optionally incorporate immediate values (i.e., `%x` instead of `%ID`)
### Table 4: Heterogeneous device mapping results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Prediction Accuracy [%]</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPU</td>
<td>Grewe et al. [29]</td>
<td>DeepTune [18]</td>
<td>inst2vec</td>
<td>inst2vec-imm</td>
</tr>
<tr>
<td><strong>AMD Tahiti 7970</strong></td>
<td>41.18</td>
<td>73.38</td>
<td>83.68</td>
<td>82.79</td>
<td><strong>88.09</strong></td>
</tr>
<tr>
<td><strong>NVIDIA GTX 970</strong></td>
<td>56.91</td>
<td>72.94</td>
<td>80.29</td>
<td>82.06</td>
<td><strong>86.62</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPU</td>
</tr>
<tr>
<td><strong>AMD Tahiti 7970</strong></td>
<td>3.26</td>
</tr>
<tr>
<td><strong>NVIDIA GTX 970</strong></td>
<td>1.00</td>
</tr>
</tbody>
</table>
Evaluation: Compute Device Mapping

Immediate Value concatenation types:

- (a) concat_naive
- (b) concat_embed
- (c) extract_concat
Evaluation: Compute Device Mapping

Immediate Value concatenation results:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Prediction Accuracy [%]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ignore</td>
<td>concat naïve</td>
</tr>
<tr>
<td>AMD Tahiti 7970</td>
<td>82.79</td>
<td>88.09</td>
</tr>
<tr>
<td>NVIDIA GTX 970</td>
<td>82.06</td>
<td>86.62</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 5900</td>
<td>1.21</td>
<td>1.10</td>
<td>1.17</td>
<td>1.37</td>
<td>1.28</td>
</tr>
<tr>
<td>AMD Tahiti 7970</td>
<td>1.01</td>
<td>1.05</td>
<td>1.23</td>
<td>1.10</td>
<td>1.18</td>
</tr>
<tr>
<td>NVIDIA GTX 480</td>
<td>0.86</td>
<td>1.10</td>
<td>1.14</td>
<td>1.07</td>
<td>1.11</td>
</tr>
<tr>
<td>NVIDIA Tesla K20c</td>
<td>0.94</td>
<td>0.99</td>
<td>0.93</td>
<td>1.06</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Evaluation: Thread Coarsening

Immediate Value concatenation results:

<table>
<thead>
<tr>
<th>Computing Platform</th>
<th>ignore</th>
<th>concat_naïve</th>
<th>extract_concat</th>
<th>concat_embed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 5900</td>
<td>1.37</td>
<td>1.21</td>
<td>1.28</td>
<td>1.30</td>
</tr>
<tr>
<td>AMD Tahiti 7970</td>
<td>1.10</td>
<td>1.06</td>
<td><strong>1.18</strong></td>
<td>0.92</td>
</tr>
<tr>
<td>NVIDIA GTX 480</td>
<td>1.07</td>
<td>0.99</td>
<td><strong>1.11</strong></td>
<td>0.97</td>
</tr>
<tr>
<td>NVIDIA Tesla K20c</td>
<td><strong>1.06</strong></td>
<td>1.04</td>
<td>1.00</td>
<td>0.99</td>
</tr>
</tbody>
</table>
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?