PalmTree: Learning an Assembly Language Model for Instruction Embedding

Xuezixiang Li, Qu Yu, Heng Yin

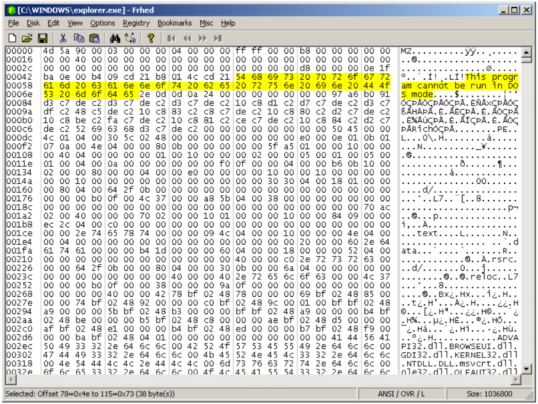
2021 Conference on Computer and Communications Security

Overview

- Deep learning has been successful in a variety of binary analysis tasks
- Important aspect of these algorithms is how to represent an instruction as a vector
- PalmTree is a pretrained BERT analogue that learns high-quality instruction embeddings that can be used for downstream tasks

How to Feed a Neural Network

- Three options:
  1. raw-byte encoding,
  2. manual encoding of disassembled instructions, or
  3. learning-based encoding



Raw-byte Encoding

- One-hot encoding of bytes into a 256 dimensional vector

$$0x01 \rightarrow \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}$$

$$0xFF \rightarrow \begin{bmatrix} 0 & 0 & \dots & 1 \end{bmatrix}$$

- Simple, efficient, but captures no semantic information about instructions

Manual Encoding of Disassembled Instructions

- Binary is disassembled, different components are identified and assigned one-hot vectors, which are then concatenated

$$
\begin{aligned}
\text{mul eax, ebx} \quad &\rightarrow \quad \vec{\text{opcode}} \oplus \vec{\text{registers}} \oplus \vec{\text{addresses}} \\
&\rightarrow \quad \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}^T \oplus \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T
\end{aligned}
$$

- More semantic meaning than raw-byte, but could be more

Learning-based Encoding

- Model instructions as words and functions as documents
- Use word2vec model to learn word associations

$$\begin{aligned} \text{mul eax, ebx} &\rightarrow \vec{v_1} \\ \text{div ecx, edx} &\rightarrow \vec{v_2} \\ \text{jz} &\rightarrow \vec{v_3} \end{aligned}$$

$$\text{distance}(\vec{v_1}, \vec{v_2}) \ll \text{distance}(\vec{v_1}, \vec{v_3})$$

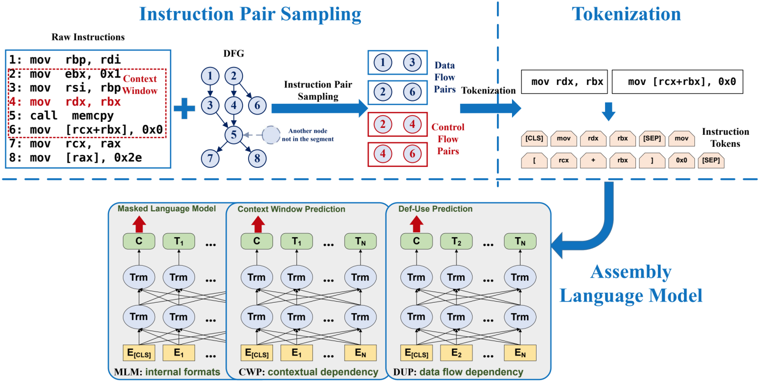
- Results in semantically similar units having close vector representations, i.e., captures the most semantic meaning!

Challenges in Learning-based Encoding

- Complex and diverse instruction formats
- Noisy instruction context

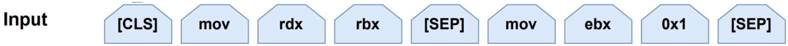
```
; memory operand with complex expression
mov [ebp+eax*4-0x2c], edx
; three explicit operands, eflags as implicit operand
imul [edx], ebx, 100
; prefix, two implicit memory operands
rep movsb
; eflags as implicit input
jne 0x403a98
```

PalmTree: Pre-trained Assembly Language Model for InsTRuction EmbEdding



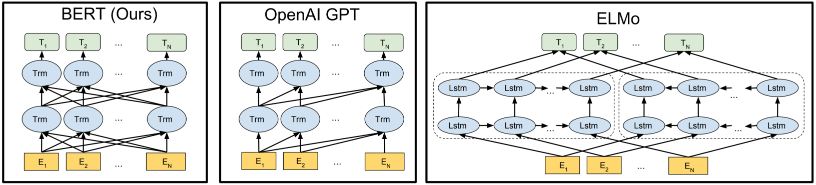
Instruction Pair Sampling & Tokenization

- Sample instruction pairs for downstream training tasks:
  1. two instructions have a *def-use relation* if their is any dependency between them
  2. two instructions have a *control-flow relation* if one instruction follows the other within a certain window size
- Fine-grained instruction tokenization
  1. strings → '[STR]'
  2. addresses → '[ADDR]'
- Example:
  1. 'mov ebx, 0x1'
  2. 'mov rdx, rbx'



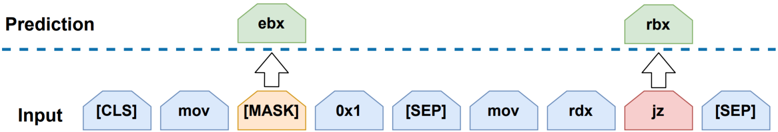
Assembly Language Model: Architecture

- PalmTree uses BERT (Bidirectional Encoder Representations from Transformers) architecture



Pretraining Task I: MLM

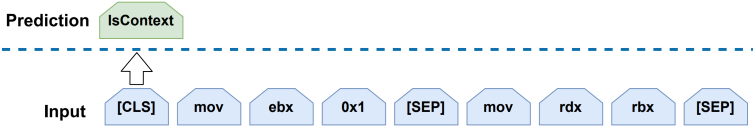
- Masked Language Modeling (MLM) masks/corrupts a token in a sequence and tasks the model to predict the masked token
- Example:
  1. 'mov ebx, 0x1'
  2. 'mov rdx, rbx'



$$\mathcal{L}_{\mathrm{MLM}} = -\sum_{t_i \in m(I)} log\, p(\hat{t}|I)$$

Pretraining Task II: CWP

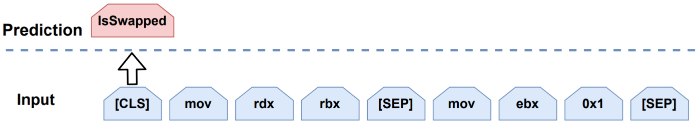
- Context Window Prediction (CWP) predicts whether or not two instructions occur within a fixed-context window of size w
- Example:
  1. 'mov ebx, 0x1'
  2. 'mov rdx, rbx'



$$\mathcal{L}_{\mathrm{CWP}} = -\sum_{I \in D} log\, p(\hat{y} | I, I_{cand})$$

Pretraining Task III: DUP

- Def Use Prediction (DUP) predicts whether or not a pair of instructions have been swapped
- Example:
  1. 'mov ebx, 0x1'
  2. 'mov rdx, rbx'



$$\mathcal{L}_{\mathrm{DUP}} = -\sum_{I \in D} log\, p(\hat{y} | I_1, I_2)$$

PalmTree Summary

- Multi-task pretraining objectives give PalmTree a strong understanding of the structure, relationships, and data dependencies of assembly instructions

$$\mathcal{L}_{\text{PalmTree}} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{CWP}} + \mathcal{L}_{\text{DUP}}$$

- PalmTree can be frozen an used to generate instruction embeddings or can be fine-tuned directly for downstream tasks

Experiment

- Intrinsic Evaluation: compares different embeddings for binary analysis subtasks
- Extrinsic Evaluation: compares different embeddings with state-of-the-art models for downstream binary analysis tasks
- PalmTree variants:
  1. PalmTree-M: only MLM
  2. PalmTree-MC: MLM & CWP
  3. PalmTree: MLM, CWP, & DUP

Intrinsic I: Outlier Evaluation

- Identify the "outlier" instruction from a set of instructions

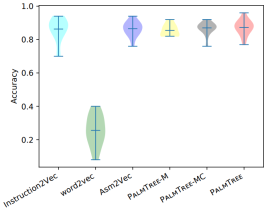


Figure 6: Accuracy of Opcode Outlier Detection

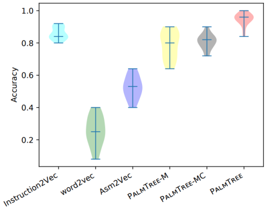


Figure 7: Accuracy of Operands Outlier Detection

Intrinsic II: Basic Block Search

- Find semantically equivalent blocks of x86 code

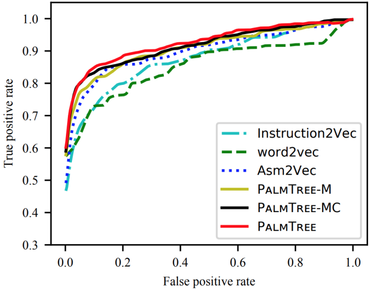


Figure 8: ROC curves for Basic Block Search

Extrinsic I: Binary Code Similarity Detection

- Determine if binary code is similar without access to source code

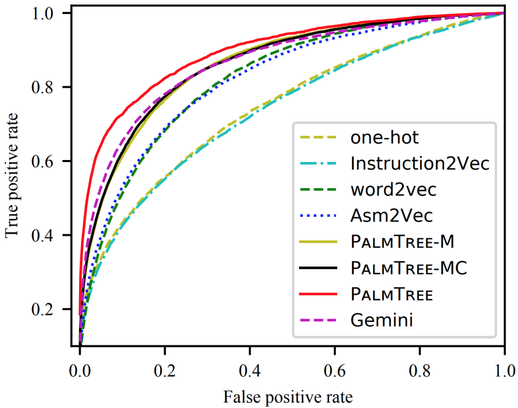


Figure 10: ROC curves of Gemini

Extrinsic II: Function Type Signature Analysis

- Determine the types of function parameters and return values

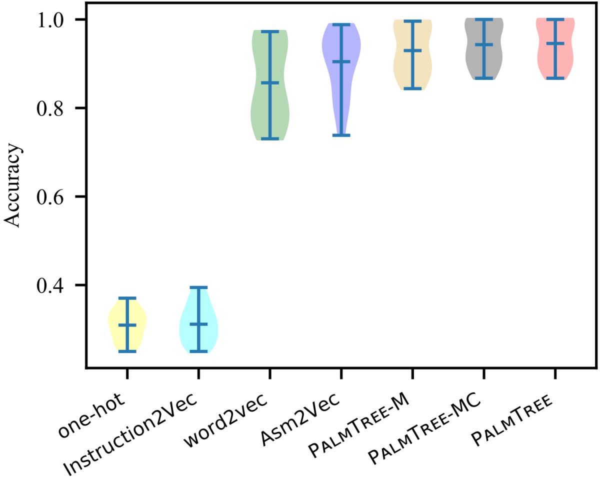


Figure 12: Accuracy of EKLAVYA

Extrinsic III: Value Set Analysis

- Determine if binary code contains a memory leak

Table 6: Results of DeepVSA

Embeddings	Global			Heap			Stack			Other		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
one-hot	0.453	0.670	0.540	0.507	**0.716**	0.594	0.959	0.866	0.910	0.953	0.965	0.959
Instruction2Vec	0.595	0.726	0.654	0.512	0.633	0.566	0.932	0.898	0.914	0.948	0.946	0.947
word2vec	0.147	0.535	0.230	0.435	0.595	0.503	0.802	0.420	0.776	0.889	0.863	0.876
Asm2Vec	0.482	0.557	0.517	0.410	0.320	0.359	0.928	0.894	0.911	0.933	0.964	0.948
DeepVSA	**0.961**	0.738	0.835	0.589	0.580	0.584	0.974	0.917	0.944	0.943	0.976	0.959
PalmTree-M	0.845	0.732	0.784	0.572	0.625	0.597	0.963	0.909	0.935	0.956	0.969	0.962
PalmTree-MC	0.910	0.755	0.825	**0.758**	0.675	0.714	0.965	0.897	0.929	0.958	**0.988**	**0.972**
PalmTree	0.912	**0.805**	**0.855**	0.755	0.678	**0.714**	**0.974**	**0.929**	**0.950**	**0.959**	0.983	0.971

Runtime Efficiency

- PalmTree takes significantly longer to train and produces embeddings at a slower rate

embedding size	encoding time	throughput (#ins/sec)
Instruction2vec	6.684	150,538
word2vec	0.421	2,386,881
Asm2Vec	17.250	58,328
PalmTree-64	41.682	24,138
PalmTree-128	70.202	14,332
PalmTree-256	135.233	7,440
PalmTree-512	253.355	3,971

Conclusions & Future Work

- PalmTree is a task-agnostic pretrained large language model for x86 instructions that can be used to train high performing binary analysis models
- Directions for future work include cross-architecture language modeling and improving how PalmTree handles long-range dependencies within code