PalmTree: Learning an Assembly Language Model for Instruction Embedding

Xuezixiang Li, Yu Qu, Heng Yin

University of California, Riverside

yuq@ucr.edu

CCS 2021

Currently available at: https://arxiv.org/abs/2103.03809

Outline

1. **Introduction**
2. **Background**
3. **PalmTree: an Assembly Language Model for Instruction Embedding**
4. **Evaluations (Intrinsic and Extrinsic)**
5. **Conclusion**

Introduction

- Recently, Deep Learning has demonstrated its strengths in binary analysis tasks.
  - Function boundary identification
  - Binary code similarity detection
  - Function prototype inference
  - Value set analysis
  - Malware detection…

Introduction

- First design choice of using DL is: *What input should be fed into the neural network model*?
  - **Raw bytes**: Shin et al. (USENIX’15), αDiff (ASE’18), DeepVSA (USENIX’19), MalConv (AAAI Workshop’18)
  - **Manually-designed features**: Gemini (CCS’17), Instruction2Vec (ICONI’17)
  - **Instruction Embedding**: InnerEye (NDSS’19), EKLAVYA (USENIX’17), mainly used word2vec and PV-DM (doc2vec)

- Instruction Embedding is more attractive:
  - Avoids manually designing efforts
  - Higher-level features

Introduction

- However, existing schemas have **unsolved problems**:
  1. **Ignore the complex internal formats of instructions**

     Treat an instruction as a word: InnerEye (NDSS'19), EKLAVYA (USENIX'17)

     Only consider a simple format: Asm2Vec (S&P'19)
  2. **Use Control Flow to capture contextual information**

     Asm2Vec (S&P'19), InnerEye (NDSS'19), Order Matters (AAAI'20)

- **Pre-trained models** (**PTMs**).
  - **large-scale unlabeled corpora** and **self-supervised** training tasks
  - BERT, GPT, RoBERTa, ALBERT, Swin Transformer…
  - Assembly language is one of the programming languages – Naturalness [1]

[1] Allamanis, Miltiadis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. "A survey of machine learning for big code and naturalness." *ACM Computing Surveys (CSUR)* 51, no. 4 (2018): 1-37.

Introduction

- We propose a pre-trained assembly language model

PalmTree: **P**re-trained **A**ssembly **L**anguage **M**odel for Ins**TR**uction **E**mb**E**dding

- Based on BERT
- Newly designed training tasks
  - Complex internal formats
  - Instruction reordering introduced by compiler optimization
  - Long range data dependencies
- Results
  - Best performance in intrinsic evaluations
  - Significantly improves downstream models in **binary code similarity detection**, **function signatures recovery**, **value set analysis**

Background – Existing approaches

1. Raw-byte Encoding

- One-hot encoding: converts each byte into a 256-dimensional vector. MalConv (AAAI Workshop’18), DeepVSA (USENIX’19)
- *It does not provide any semantic level information.*

2. Manual Encoding of Disassembled Instructions

- First disassembles each instruction and then extract features
- Li et al. (ICML’19) only extracted opcode, then used one-hot encoding
- Instruction2Vec (ICONI’17): a manually defined encoding rule
- *Cannot capture high-level semantic information*.

Background – Existing approaches

3. Learning-based Encoding

- Word2vec: instruction – word, function – document
  - ➢Code similarity detection: SAFE (DIMVA'19), InnerEye (NDSS'19)
  - ➢Function prototype inference: EKLAVYA (USENIX'17)
- Doc2vec (PV-DM):
  - ➢Asm2Vec (S&P'19) – treat instruction as one opcode and two operands
- *Can carry higher-level semantic information.* ***However:***

Background – Challenges

1. Instructions are complex and diverse

A constant or a string symbol

Small Constant

CPU registers

Memory operand: **base**+**index***scale*+**displacement**

```
; 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
```

Conditional Jump takes **EFLAGS** as a implicit input

Background – Challenges

2. Instructions can be reordered

This **test** stores its result into **EFLAGS**. It is **moved by the compiler** so that such that it is further away from the **je** instruction at Line 14, which will use (load) the **EFLAGS** computed by the **test** —— Complier move the load away from its last store to avoid stalls in the instruction execution pipeline

```
1  ; prepare the third argument for function call
2  mov rdx, rbx
3  ; prepare the first argument for function call
4  mov rsi, rbp
5  ; prepare the second argument for function call
6  mov rdi, rax
7  ; call memcpy() function
8  call memcpy
9  ; test rbx register (this instruction is reordered)
10 test rbx, rbx
11 ; store the return value of memcpy() into rcx register
12 mov rcx, rax
13 ; conditional jump based on EFLAGS from test instruction
14 je  0x40adf0
```

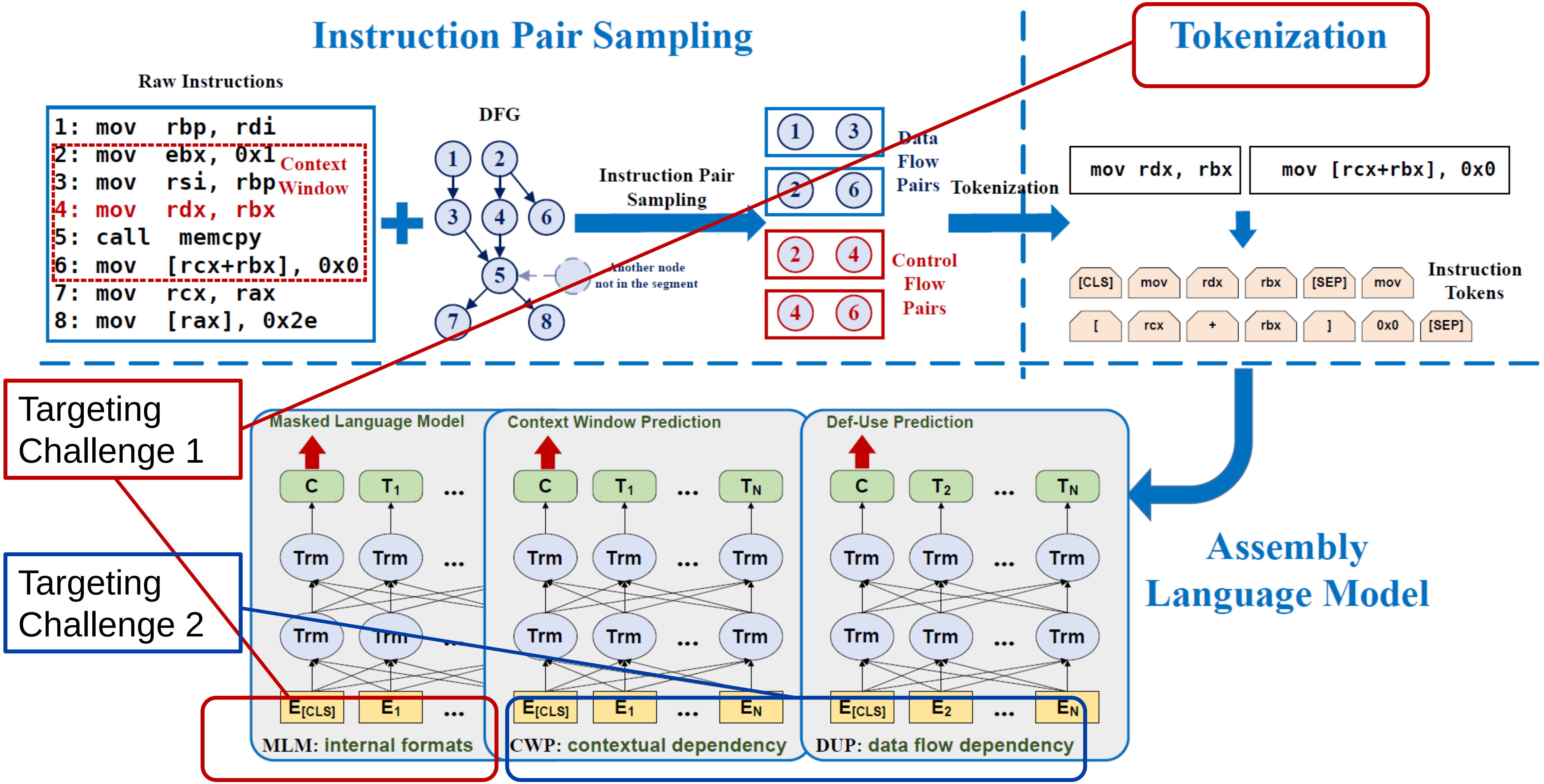
Data Dependency

Background – Summary

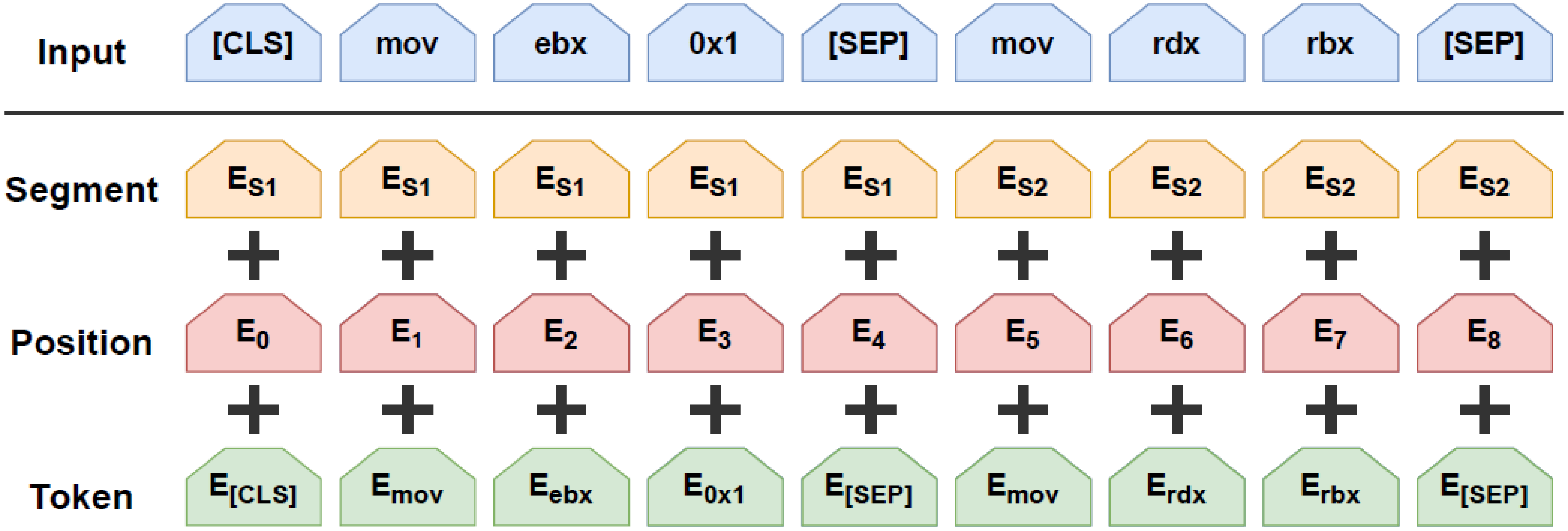
Table 1: Summary of Approaches

Name	Encoding	Internal Structure	Context	Disassembly Required
DeepVSA [14]	1-hot encoding on raw-bytes	no	no	no
Instruction2Vec [41]	manually designed	yes	no	yes
InnerEye [43]	word2vec	no	control flow	yes
Asm2Vec [10]	PV-DM	partial	control flow	yes
PalmTree (this work)	BERT	yes	control flow & data flow	yes

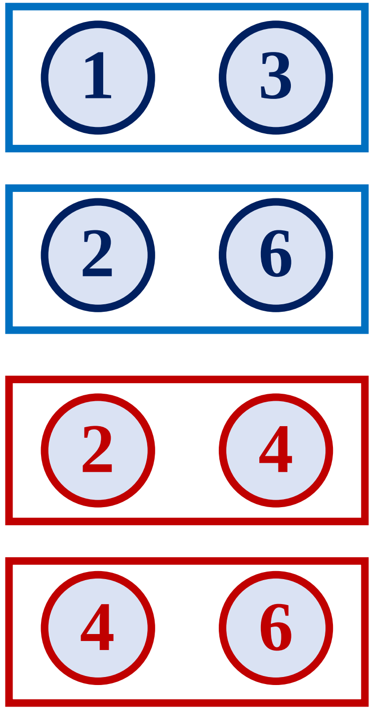
PalmTree: a pre-trained assembly language model



PalmTree: Input Representation



PalmTree Training task 1: Masked Language Model



Data Flow Pairs

Control Flow Pairs

Randomly select 15% of the tokens to replace. For the chosen tokens, 80% are masked by **[MASK]** (mask out tokens), 10% are replaced with another token (corrupted tokens)

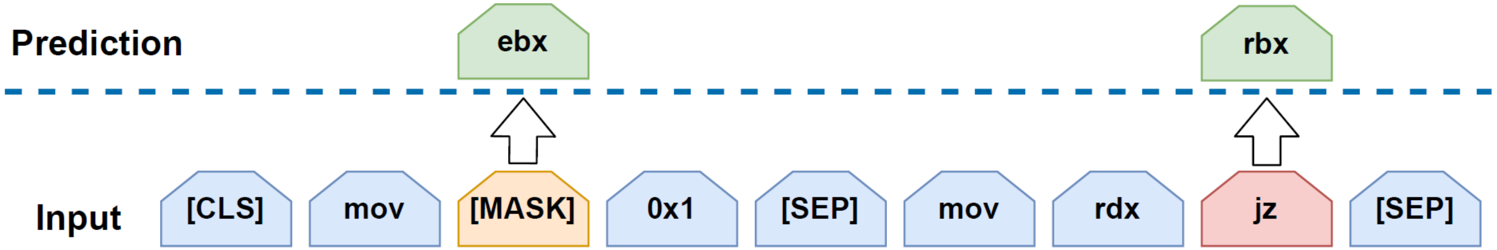


Figure 3: Masked Language Model (MLM)

PalmTree Training task 2: Context Window Prediction

```
1: mov   rbp, rdi
2: mov   ebx, 0x1
3: mov   rsi, rbp
4: mov   rdx, rbx
5: call  memcpy
6: mov   [rcx+rbx], 0x0
7: mov   rcx, rax
8: mov   [rax], 0x2e
```

Context Window

Given an instruction ***I*** and a candidate instruction I_{cand} as input, we train PalmTree to predict whether they are located in the context window (w=2)

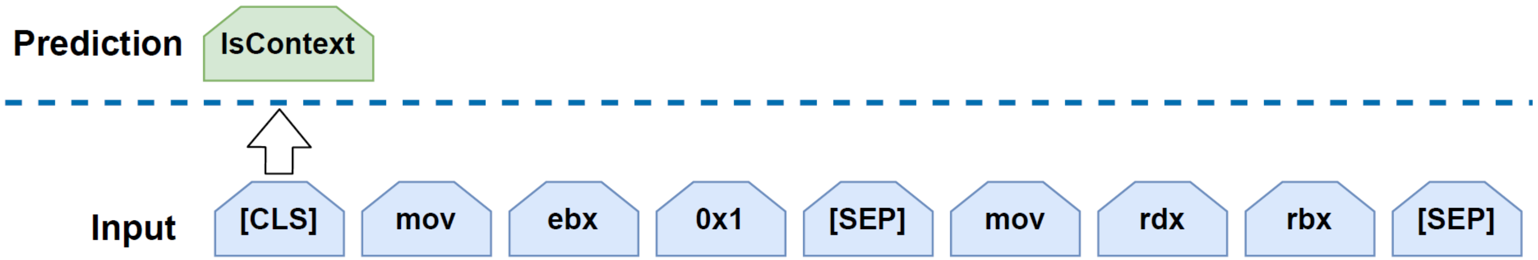
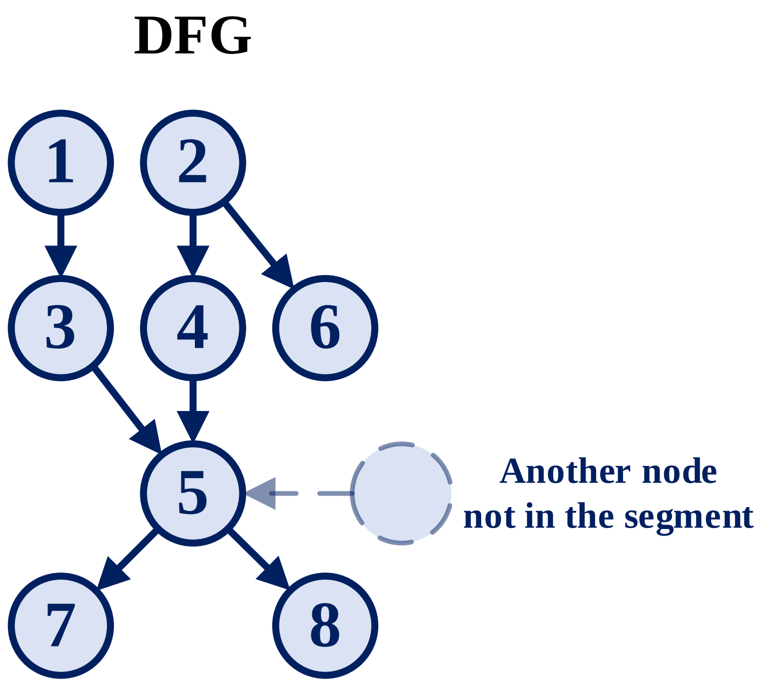


Figure 4: Context Window Prediction (CWP)

PalmTree Training task 3: Def-Use Prediction



Given an instruction pair I_1 and I_2 in DFG as input, we feed $I_1||I_2$ as a positive sample and $I_2||I_1$ as a negative sample, then let PalmTree predict whether this pair is swapped.

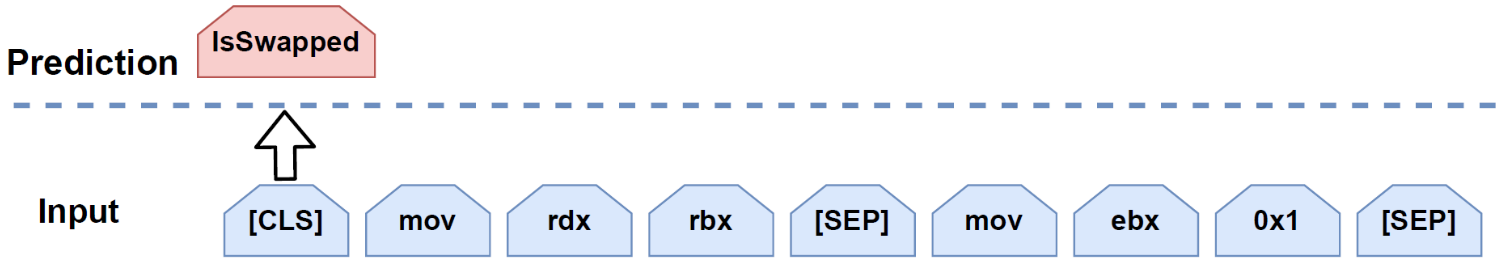


Figure 5: Def-Use Prediction (DUP)

Loss function and other designs

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

- We use **mean pooling** of the hidden states of *the second last layer* as the instruction embedding.
- There are two ways of deploying PalmTree:
  1. **Instruction embedding generation**

     PalmTree is used as an off-the-shelf assembly language model.
  2. **Fine-tuning**

     Provide extra benefits when enough resources and budget are available.

Evaluation – Configurations and Intrinsic Evaluation

- **Hyper-parameters**: #Layers=12, Head=8, Hidden_dimension=128
- **Three configurations**:
  - **PALMTREE-M**: PALMTREE trained with MLM only
  - **PALMTREE-MC**: PALMTREE trained with MLM and CWP
  - **PALMTREE**: PALMTREE trained with MLM, CWP, and DUP
- **Intrinsic evaluation results**:

Table 2: Intrinsic Evaluation Results, Avg. denotes the average of accuracy scores, and Stdev. denotes the standard deviation

Model	opcode outlier		operand outlier		basicblock sim search
	Avg.	Stdev.	Avg.	Stdev.	AUC
Instruction2Vec	0.863	0.0529	0.860	0.0363	0.871
word2vec	0.269	0.0863	0.256	0.0874	0.842
Asm2Vec	0.865	0.0426	0.542	0.0238	0.894
PALMTREE-M	0.855	0.0333	0.785	0.0656	0.910
PALMTREE-MC	0.870	0.0449	0.808	0.0435	0.913
PALMTREE	**0.871**	0.0440	**0.944**	0.0343	**0.922**

Extrinsic Evaluation: Gemini

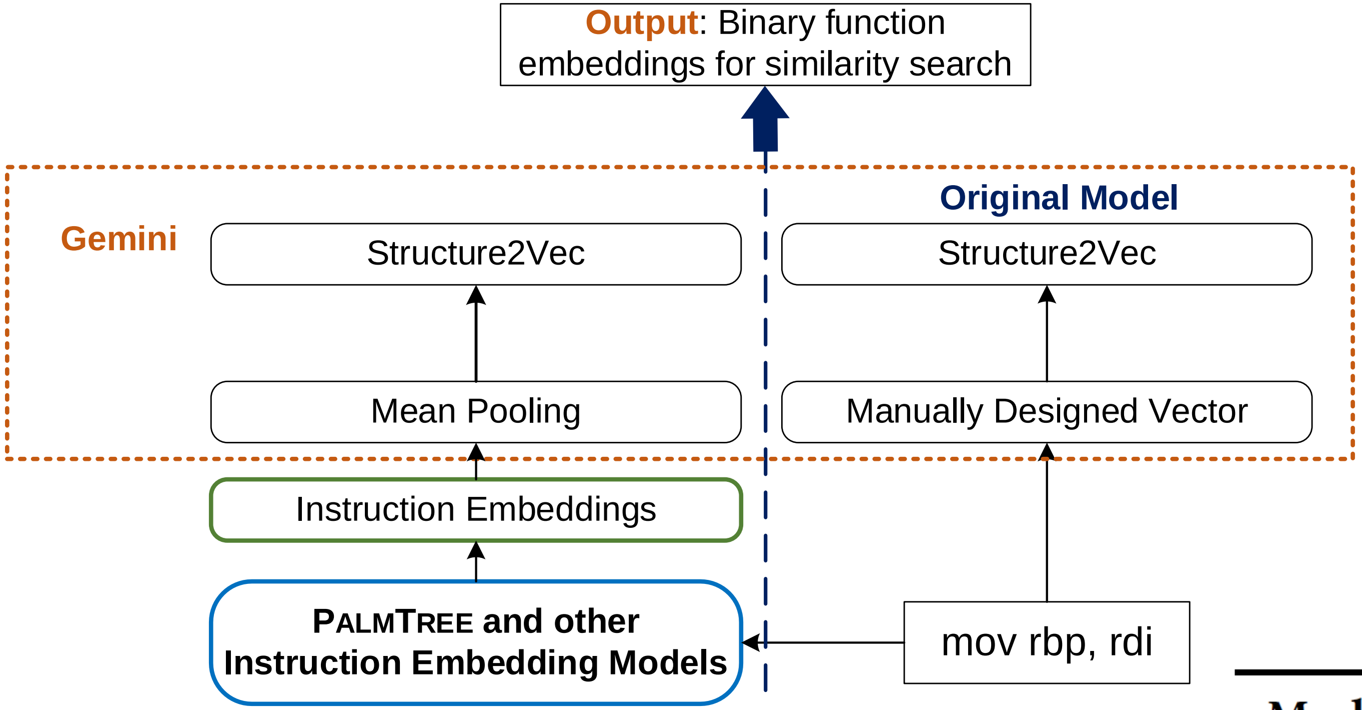


Table 3: Attributes of Basic Blocks in Gemini [40]

Type	Attribute name
Block-level attributes	String Constants Numeric Constants No. of Transfer Instructions No. of Calls No. of Instructions No. of Arithmetic Instructions
Inter-block attributes	No. of offspring Betweenness

Table 4: AUC values of Gemini

Model	AUC	Model	AUC
one-hot	0.745	Gemini	0.866
Instruction2Vec	0.738	PALMTREE-M	0.864
word2vec	0.826	PALMTREE-MC	0.866
Asm2Vec	0.823	PALMTREE	**0.921**

Extrinsic Evaluation: EKLAVYA

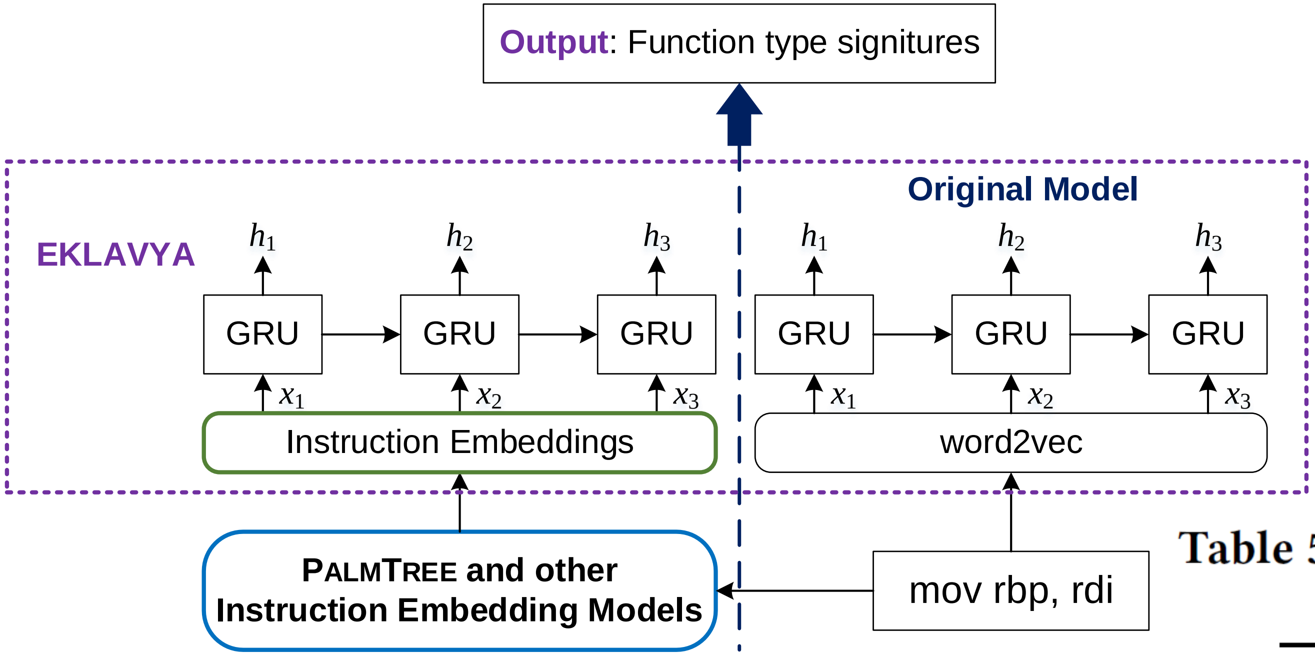
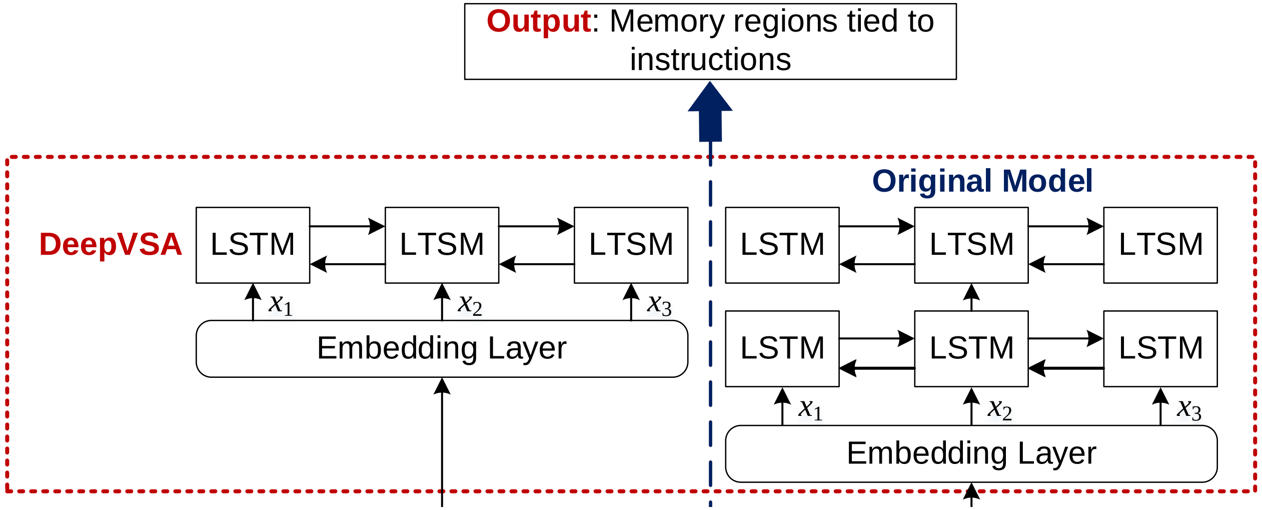


Table 5: Accuracy and Standard Deviation of EKLAVYA

Model	Accuracy	Standard Deviation
one-hot	0.309	0.0338
Instruction2Vec	0.311	0.0407
word2vec	0.856	0.0884
Asm2Vec	0.904	0.0686
PALMTREE-M	0.929	0.0554
PALMTREE-MC	0.943	0.0476
PALMTREE	**0.946**	0.0475

Extrinsic Evaluation: DeepVSA



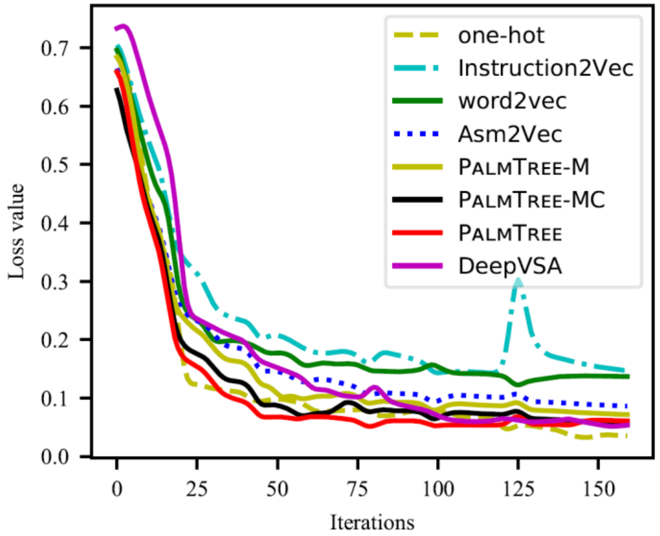


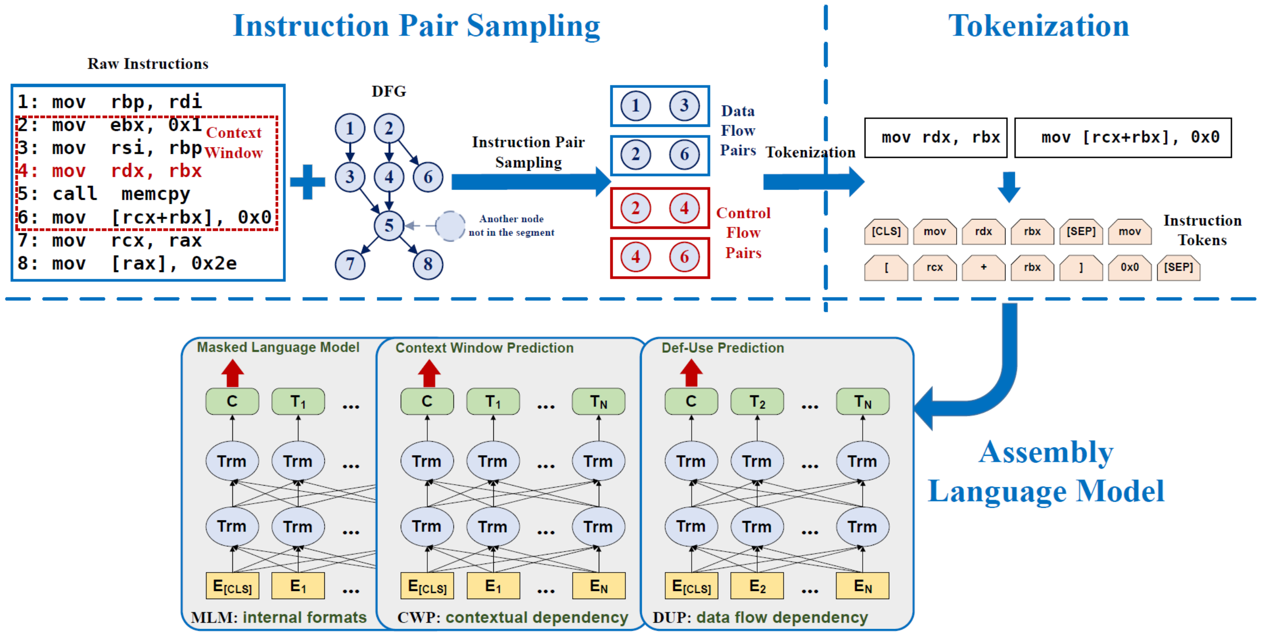
Figure 10: Loss value of DeepVSA during training

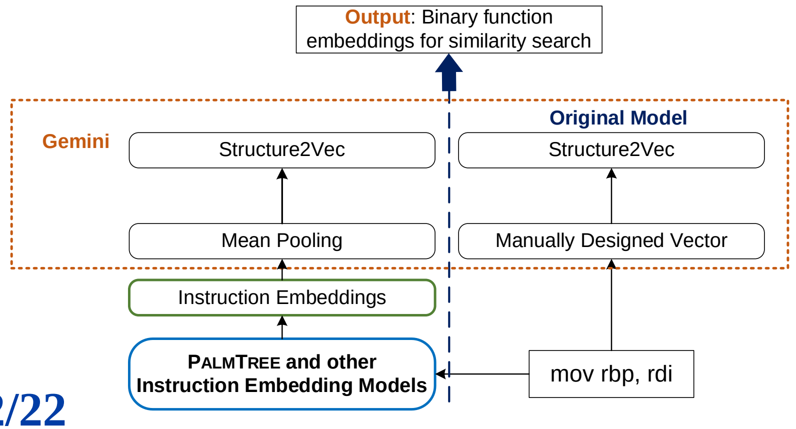
Table 6: Results of DeepVSA

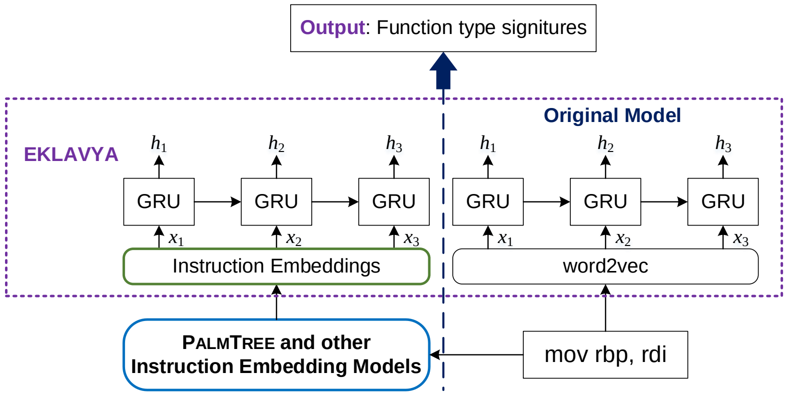
Embeddings	Global			Heap			Stack			Other		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
one-hot	0.453	0.670	0.540	0.507	**0.716**	0.594	0.959	0.866	0.910	0.953	0.965	0.959
Instruction2Vec	0.595	0.726	0.654	0.512	0.633	0.566	0.932	0.898	0.914	0.948	0.946	0.947
word2vec	0.147	0.535	0.230	0.435	0.595	0.503	0.802	0.420	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

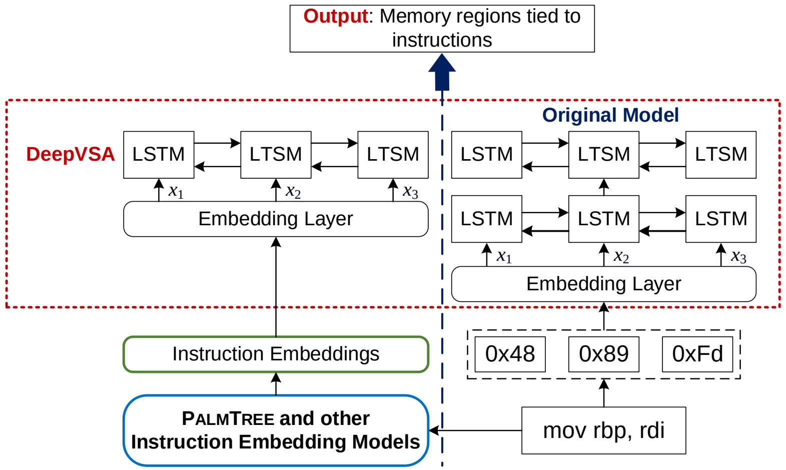
Conclusion

```
; prepare the third argument for function call
mov rdx, rbx
; prepare the first argument for function call
mov rsi, rbp
; prepare the second argument for function call
mov rdi, rax
; call memcpy() function
call memcpy
; test rbx register (this instruction is reordered)
test rbx, rbx
; store the return value of memcpy() into rcx register
mov rcx, rax
; conditional jump based on EFLAGS from test instruction
je  0x40adf0
```









Thank You!

We will open source our pre-trained model and source code.

Yu Qu, yuq@ucr.edu,
https://sites.google.com/view/yuqu