A Transformer-based Function Symbol Name Inference Model from an Assembly Language for Binary Reversing

HyunJin Kim, JinYeong Bak, Kyunghyun Cho, Hyungjoon Koo

AsiaCCS ’23

khyunjin1993@skku.edu

2023-11-08

Binary Reverse Engineering

- Gain the semantics from machine code

Disassembly **Decompilation**

```
41 00 41 00 00 00 42 00
00 00 44 00 44 00 00 00
46 00 00 00 47 00 47 00
49 00 49 00 00 00 4a 00
00 00 4c 00 4c 00 00 00
```

Binary code

```
0x8D70  endbr64
0x8D74  test     rdi, rdi
0x8D77  jz       short locret_8DB0
0x8D79  push     rbp
0x8D7A  push     rbx
```

Assembly code

```
#include <stdio.h>
int main(void) {
  printf("Hello World!\n");
  return(0);
}
```

C-like code

- Assembly instructions are made up of an opcode and operand(s)
- Opcode specifies an operation to perform
- Operand specifies data or memory location

Missing Pieces

- Decompiler allows reversing engineers to infer code semantics by converting binary code to source in a C-syntax-like format
- Most high-level information is lost during the compilation
- It is infeasible to recover certain high-level information (e.g., variable name, **function name**, variable type, # of parameters)

```c
void FUN_00108d70(void *param_1) {
    void *pvVar1;
    if (param_1 == (void *)0x0) {
        return;
    }
```

Lost information

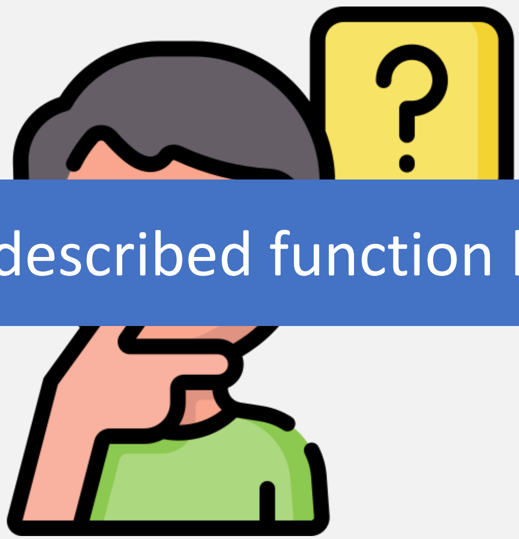
Binary Reversing

Difficult to comprehend a machine language (assembly)
→ A function often conveys a meaningful chunk with a name

Binary

Assembly Language (Machine Language)

```
...
0x8D70   endbr64
0x8D74   test     rdi, rdi
0x8D77   jz       short locret_8DB0
0x8D79   push     rbp
0x8DA5   add      rsp, 8
0x8DA9   pop      rbx
0x8DAA   pop      rbp
0x8DAB   retn
...
```

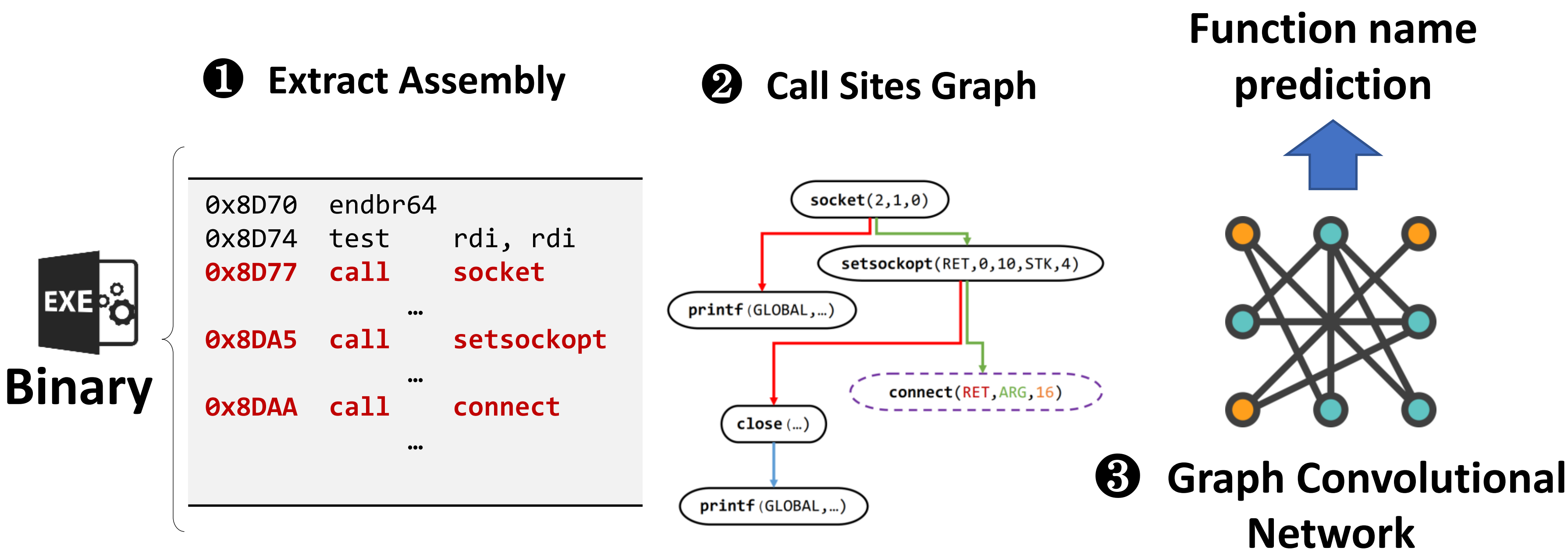


A well-developed program maintains a well-described function label!

Existing Work

Many researchers tried to analyze function names in a binary file

- Neural Reverse Engineering of Stripped Binaries using Augmented Control Flow Graphs (NERO) [David et al., OOPSLA 2020]



Goal

A series of function names allows reversers to **gain a quick overview** of a binary if they could be accurately inferred

Assembly code
mov_eax_dword_ptr_[rdi], test_eax, ...
endbr64, push_r12, mov_r10_rsi, ...
lea_rdi_[rip+0xc@@3f@@ 9], lea_r ...

Function name
cleanup_report_delgroup
parse_sec
log_get_errorcount

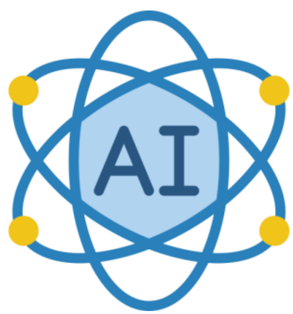
A Problem in NLP?

- Input and output are similar in terms of language
- The problem can be viewed as a **language translation task**

Input (One Language)

endbr64, test_rdi_rdi, jz
short_locret_8DB0, ..., retn

안녕하세요
(annyeonghaseyo)



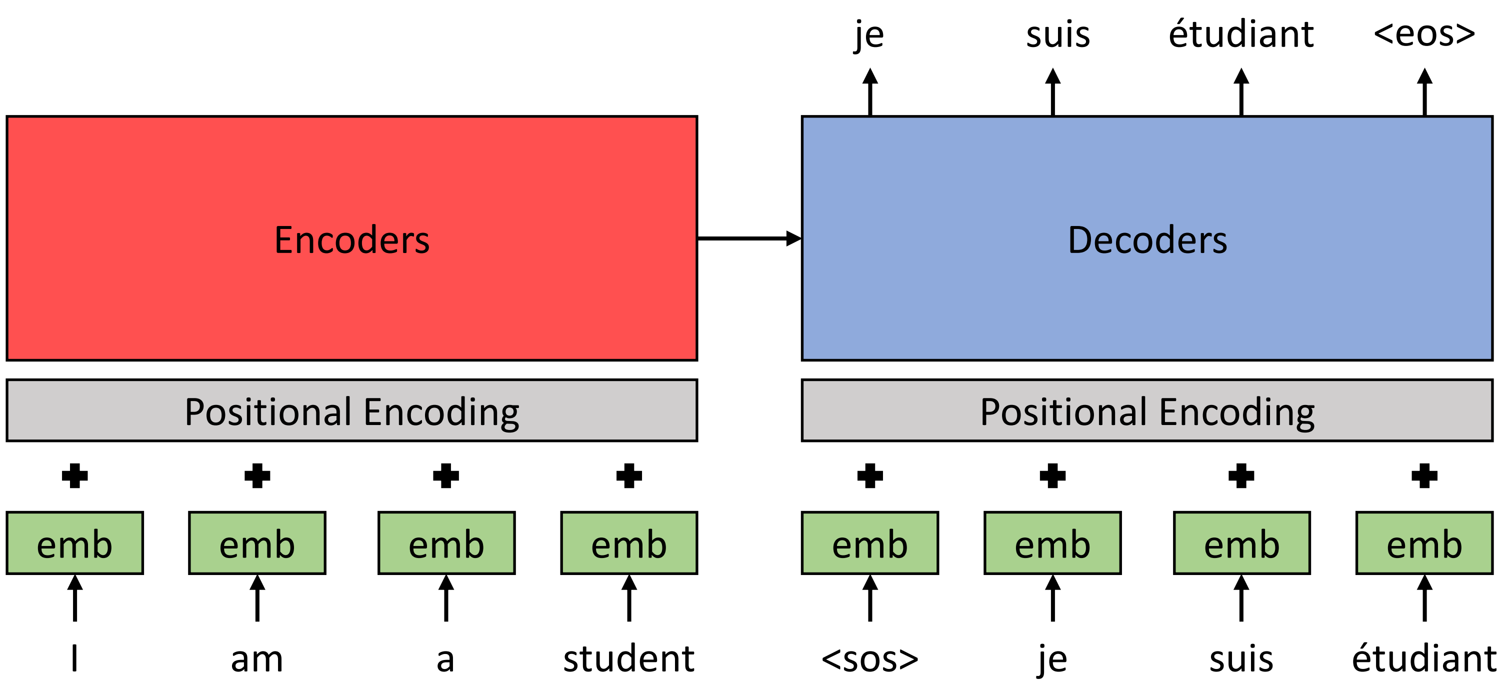
Output (Another Language)

ipc_sem_free_info

Hello

Transformer Language Model

- **Encoder** understands the meaning of input words (English)
- **Decoder** generates a sequence of words as output (French)



Motivation

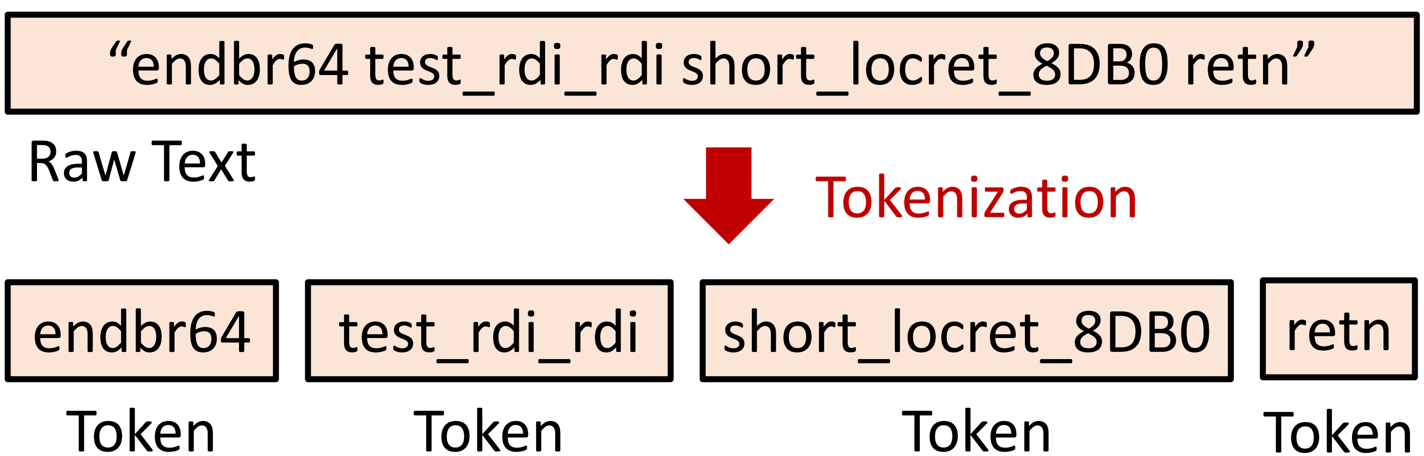
- Transformer: one of the breaking-through architectures in NLP
- Widely adopted to translate between two languages
- Naive Transformer structure struggles to learn the underlying semantics of machine instruction
- Explore optimal ways of feeding data considering the structure of a language model

➢ **Design a model AsmDepictor tailored to a function name prediction task**

Assembly Tokenization

Importance of a tokenization method

- Tokenization is splitting the raw text into small chunks of words or sentences, called tokens
- Tokenization determines how words are split, which affects the size of the vocabulary



Assembly Tokenization (Instruction)

Challenges in instruction tokenization (e.g., sub_rsp_0x50)

- Numerous *sparse* vocabularies (e.g., Address, Immediate)
- 400 million tokens with 4 bytes of immediate value

All possible Opcode and Operand combinations

- Diverse outcomes in vocabulary combinations
- **Generates a tremendous of vocabularies (compared to NLP)**
- High computational cost with larger datasets

Tokenization Methods	Large Dataset (approx. 400,000 functions)	
	Vocabulary Size	Parameter Size
Instruction	3,253,394	1,012,857,350

Assembly Tokenization (Byte Pair Encoding)

BPE tokenization for assembly instructions

Big Code!= Big Vocabulary: Open-Vocabulary Models for Source Code [ICSE, 2020]

- BPE calculates a sub-word frequency
- Widely adopted in the field of NLP (Natural Language Processing)

➢ **Significant reduction with a model parameter size**

sub_rsp_0x50 ➡ sub_rsp_0x5, 0

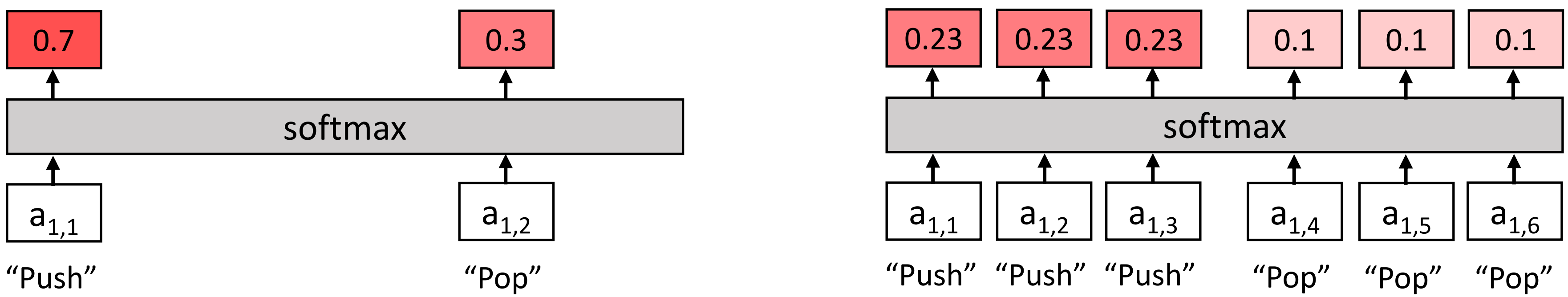
Tokenization Methods	Large Dataset (approx. 400,000 functions)	
	Vocabulary Size	Parameter Size
Instruction	3,253,394	1,012,857,350
Byte Pair Encoding	**9,923**	**40,004,102**

Problem of Attention for Our Purpose

Problem

- Frequent appearance of a machine instruction may convey important information (e.g., opcodes)

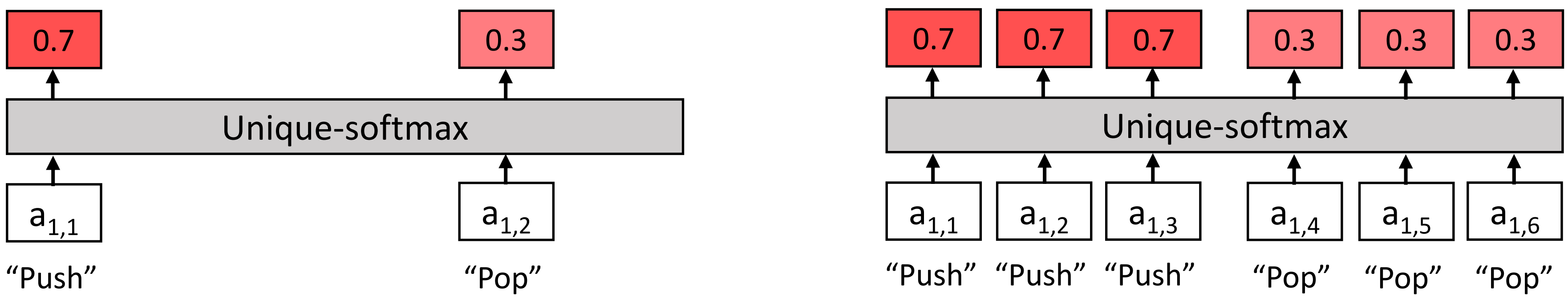
- Softmax reduces the collective output of duplicated words that might carry significant information



Our Approach (1): Unique-softmax

Unique-softmax

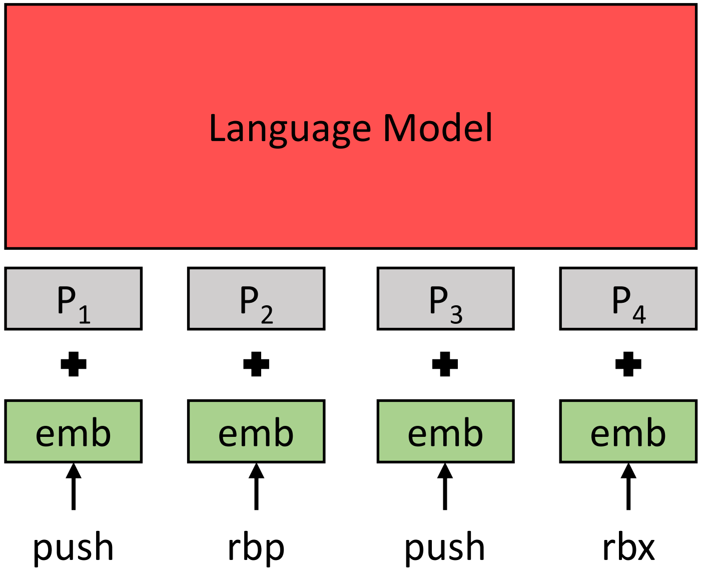
- Enables to calculate probability within a **unique set** of word
- Retain the collective output of duplicates
  (+ 3.17% on F1-score)



Problem of Positional Encoding

Problem

- Assigns a pre-defined value to give positional information
- They **cannot learn the representation of the position**
- Position information only added at the first layer



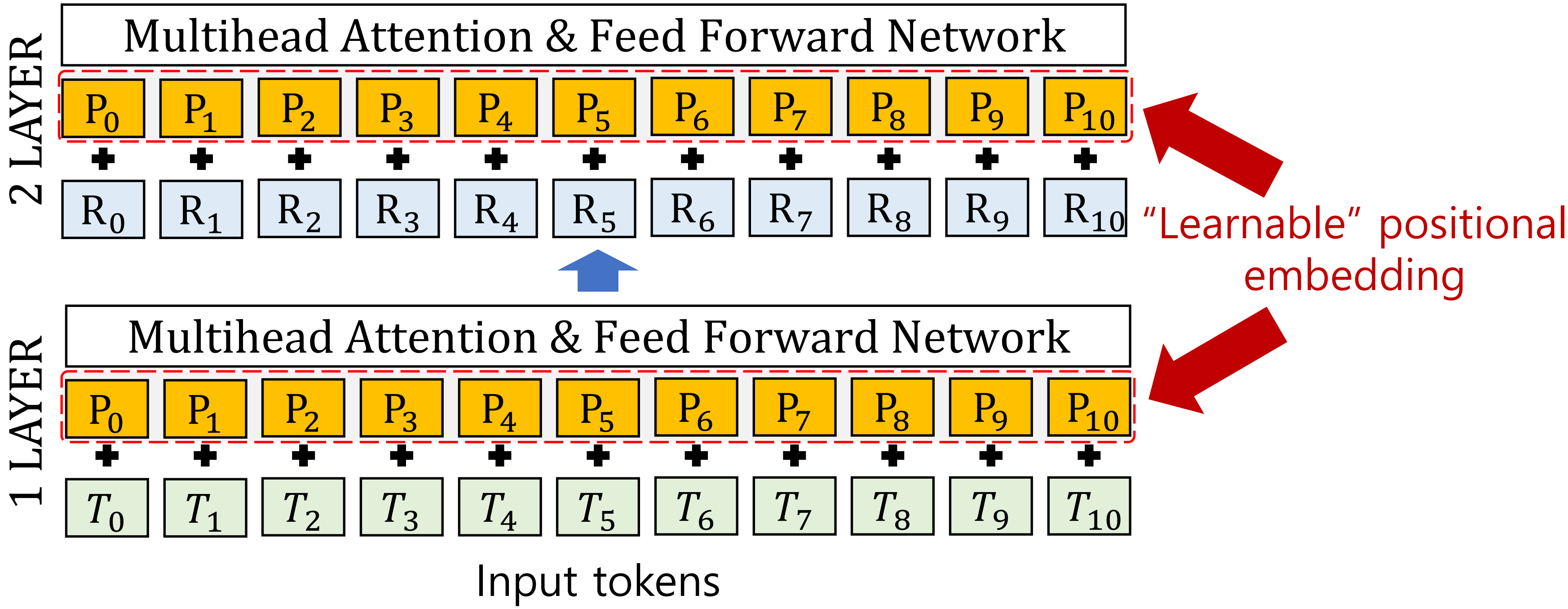
$$P(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{dmodel}}}\right)$$

$$P(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{dmodel}}}\right)$$

i : dimension of the positional encoding vector
pos : position of the token

Our Approach (2): Per-layer Positional Embedding

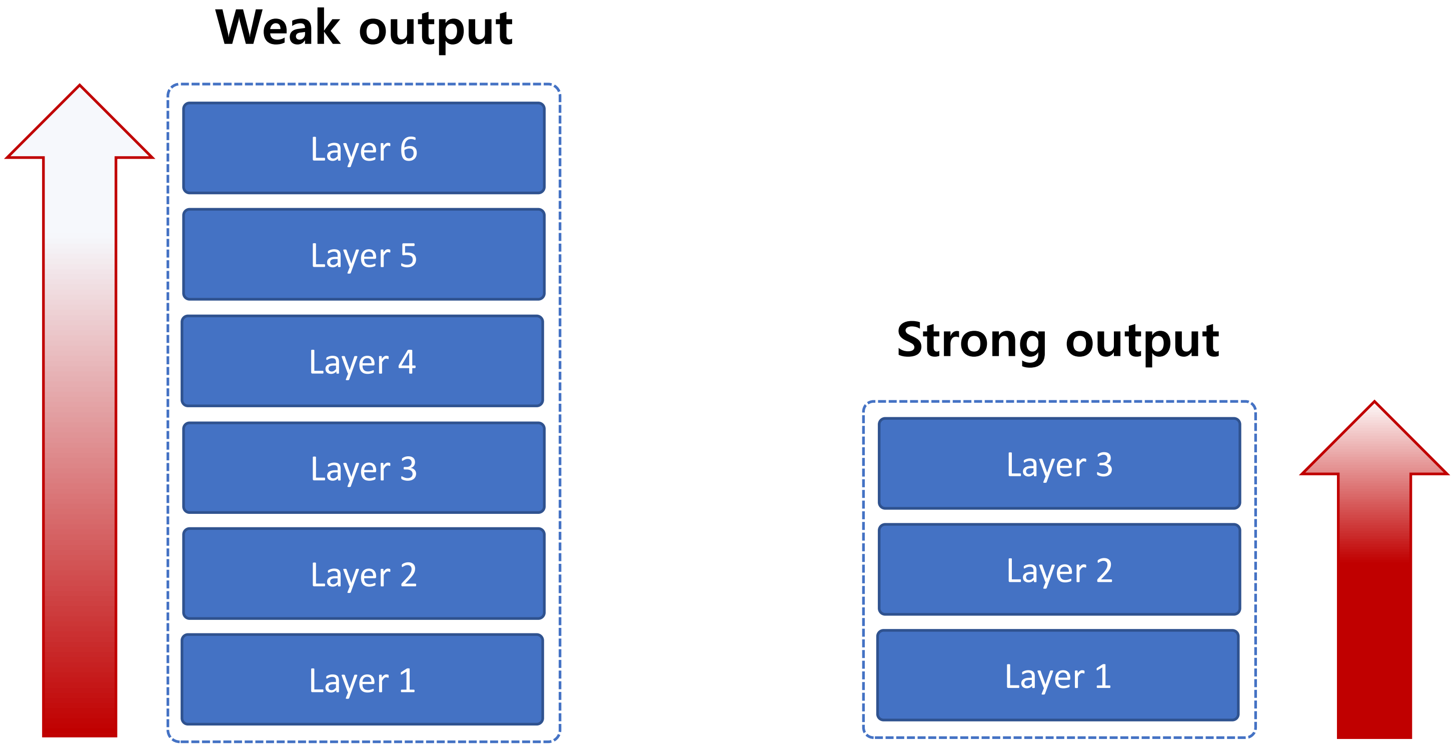
Per-Layer Positional Embedding
- Learns the positional representation
- Feed the order of machine instructions to every layer (+17.93% on F1-score)



Our Approach (3): Layer Reduction

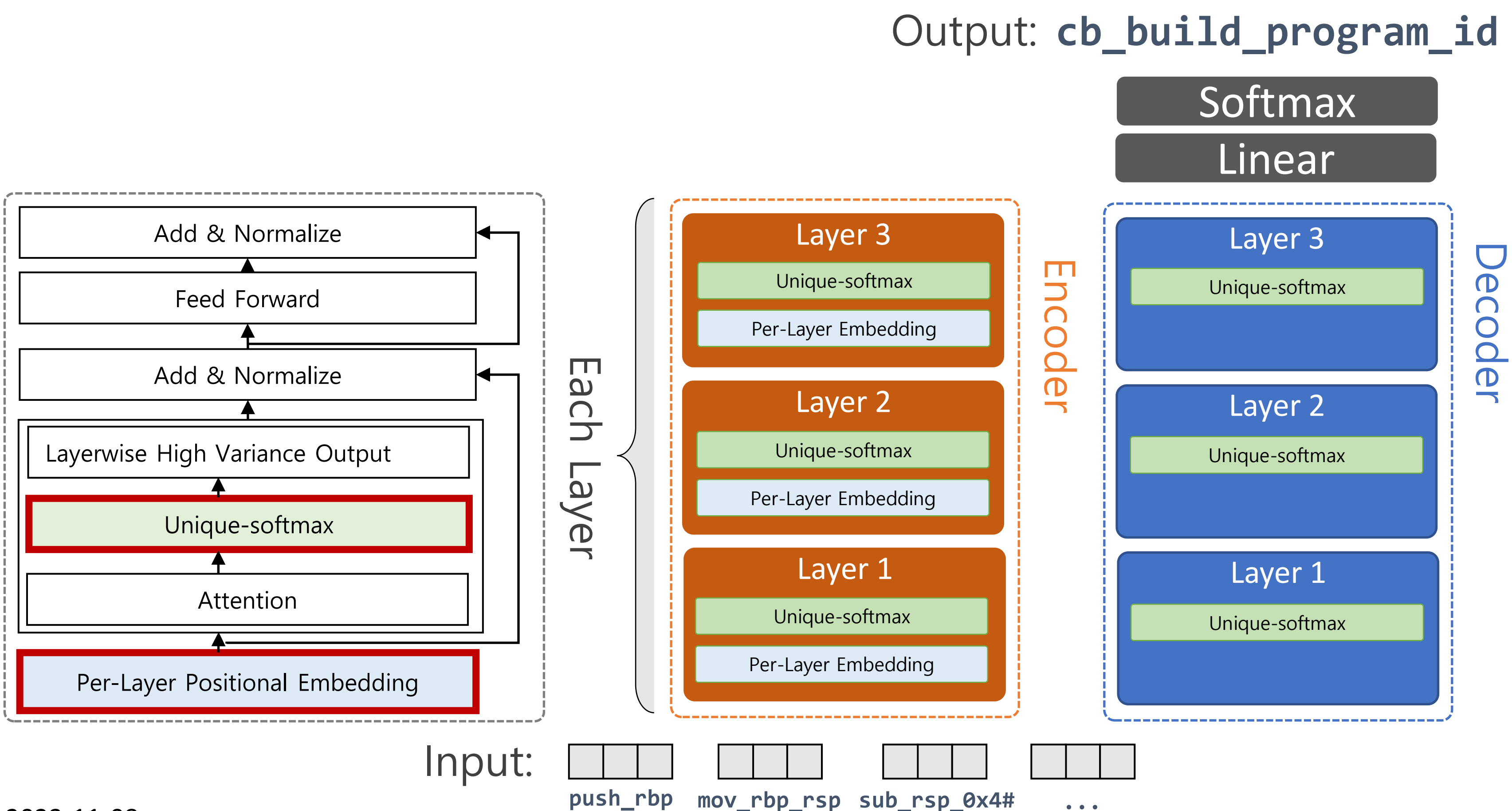
Layer Reduction

- Reducing the layer prevents values diminishing at the upper layers (+5.64% on F1-score)



AsmDepictor: Putting It All Together

Unique-softmax + Per-Layer Positional Embedding + Layer reduction



Experiment Settings

Data

- Small dataset: 67,000 functions from NERO
- Large dataset: 400,000 functions from common packages from popular Ubuntu Linux distributions + Small dataset

Metric

- F1-score: Token matching
- Rouge-l: Popular metric for longest common subsequences

Model

- NERO: Neural reverse engineering of stripped binaries using augmented control flow graphs [David et al., OOPSLA 2020]
- Debin: Predicting Debug Information in Stripped Binaries [He et al., CCS 2018]

Performance Comparison

- AsmDepictor surpasses existing SOTA models

Trained with **Small** Dataset

Model	Precision	Recall	F1	Rouge-l
Debin	5.73	5.66	5.66	5.87
NERO	12.35	12.36	12.35	14.07
AsmDepictor	57.13	57.17	**57.14**	**58.68**

+44%

- Performance improvement with a large dataset

Model	Precision	Recall	F1	Rouge-l
AsmDepictor (Small)	57.13	57.17	**57.14**	**58.68**
AsmDepictor (Large)	71.52	71.53	**71.52**	**73.75**

+15%

Demonstrative Examples

- Example of predictions
- Red indicates correctly predicted tokens

Debugging Symbol	AsmDepictor (Large Dataset)	AsmDepictor (Small Dataset)	NERO	Debin
cb_build program_id	**cb_build program_id**	**cb_build program**	options_menu	cint_remove
close_stdout	**close_stdout**	**close_stdout**	**close_stdout**	**close_stdout**
write_file	**write_file**	**write_file**	process	to_rgip
xcalloc	**xcalloc**	**xcalloc**	**xcalloc**	alloc_common

Conclusion

Wrap-up

- Our work introduces AsmDepictor, an effective prediction framework for a function name from machine instruction
- AsmDepictor surpasses existing state-of-the-art models, with F1 scores up to four times higher

- Future work: Integration function name inference model with a large language model such as LLaMA

- For more details of code & dataset, please visit our GitHub repository https://github.com/agwaBom/AsmDepictor

Thank you