Similarity Metric Method for Binary Basic Blocks of Cross-Instruction Set Architecture

Xiaochuan Zhang

zhangxiaochuan@outlook.com

Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing, China

Content

• Background

Binary program similarity metric can be used in:

malware classification

vulnerability detection

authorship analysis

The similarity between basic blocks is the basis

• Background

Two step of basic block similarity metric

• Background

Type of methods

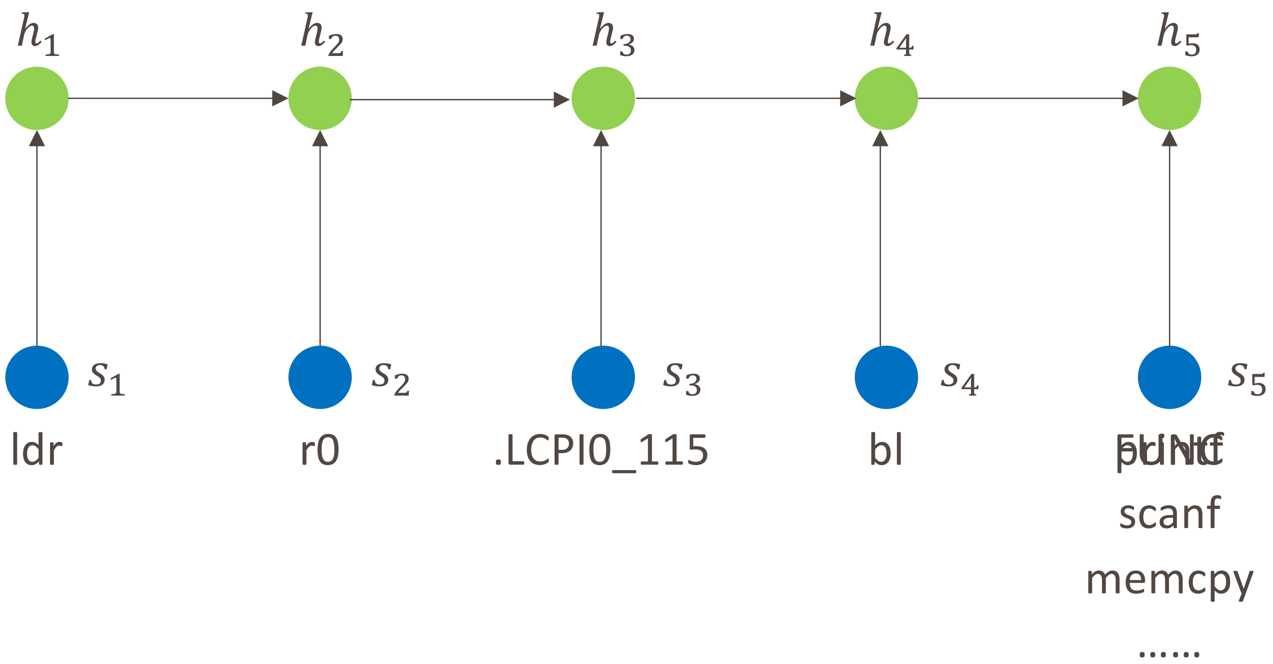
- basic block embedding
  - manually
    - each dimension corresponds to a manually selected static feature [1-3]
  - automatically
    - static word representation based methods [4-7]
    - INNEREYE-BB, an RNN based method [8]

[1] Qian Feng, et al. Scalable Graph-based Bug Search for Firmware Images. CCS 2016
[2] Xiaojun Xu,et al. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. CCS 2017
[3] Gang Zhao, Jeff Huang. DeepSim: deep learning code functional similarity. ESEC/SIGSOFT FSE 2018
[4] Yujia Li,et al.Graph Matching Networks for Learning the Similarity of Graph Structured Objects. ICML 2019
[5] Luca Massarelli, et al. SAFE: Self-Attentive Function Embeddings for Binary Similarity. DIMVA 2019
[6] Uri Alon, et al. code2vec: learning distributed representations of code. PACMPL 3(POPL) 2019
[7] Steven H. H. Ding, et al. Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization. S&P 2019
[8] Fei Zuo, et al. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs. NDSS 2019

• Background

INNEREYE-BB [1]

$$h_t = F(s_t, h_{t-1})$$

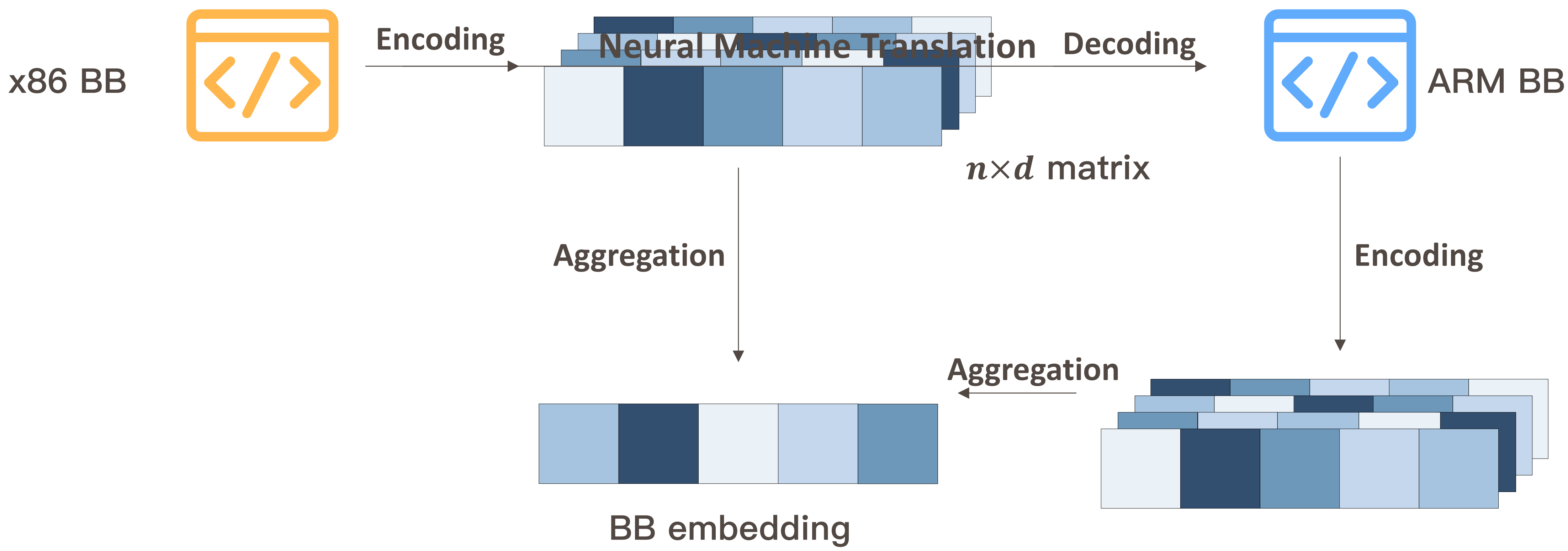


Token type	x86	ARM
basic block label	77.68%	61.23%
function name	3.71%	12.58%
others	18.60%	26.19%

[1] Fei Zuo, et al. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs. NDSS 2019

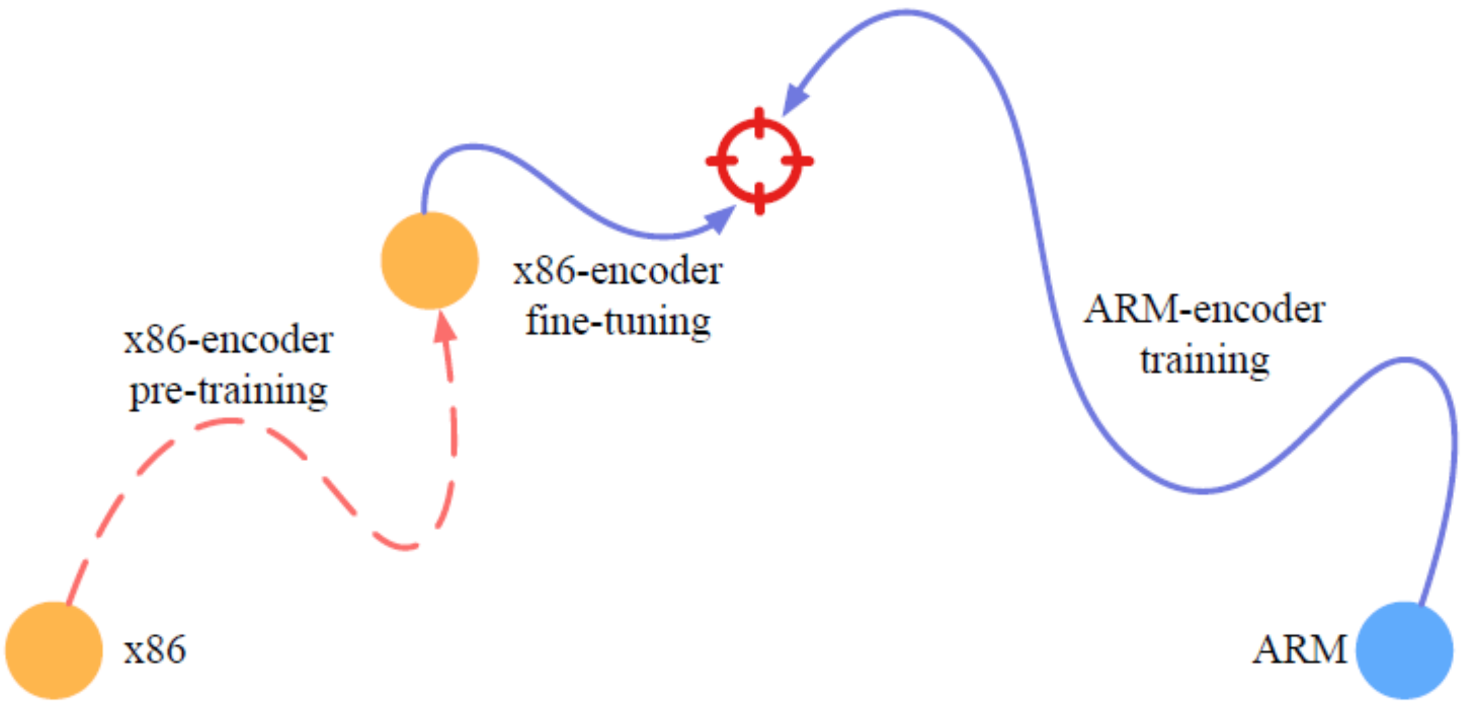
Methodology & Implementation

Idealized Solution (based on **PERFECT TRANSLATION** assumption)



Methodology & Implementation

Practical Solution



• Methodology & Implementation

x86-encoder pre-training

- data: x86-ARM basic block pairs
- NMT model: Transformer [1], other NMT models also work
- Optimization goal: minimize the translation loss

$$L = -\sum_{k=1}^{m} \sum_{j=1}^{|V_{ARM}|} \hat{y}_{kj} \log\left(y_{kj}\right)$$

[1] Ashish Vaswani, et al. Attention is All you Need. NIPS 2017

• Methodology & Implementation

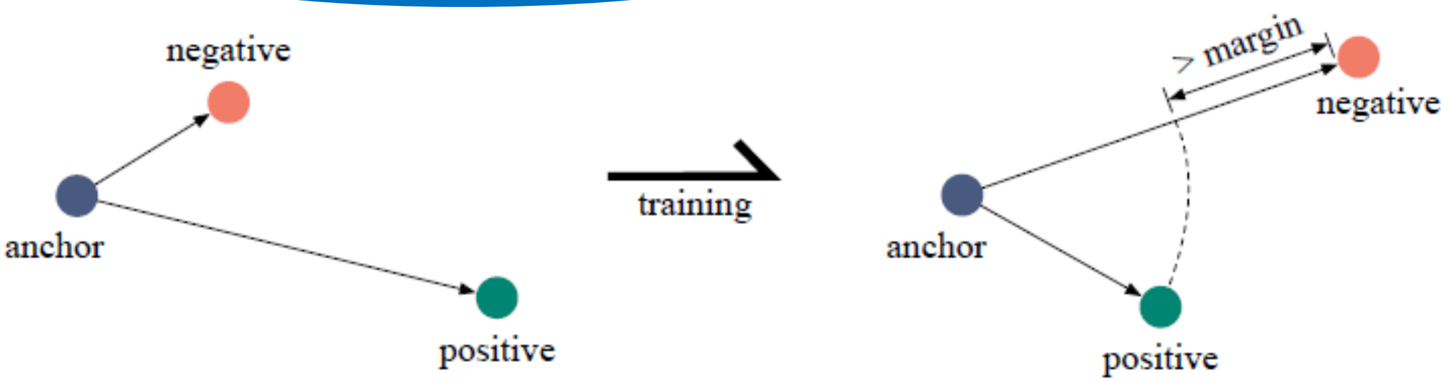
ARM-encoder training & x86-encoder fine-tuning

- data: basic block triplets, {anchor, positive, negative}
- Optimization goal: minimize the margin-based triplet loss

$$L = \max\{D(E_1, E_2) - D(E_1, E_3) + \gamma, 0\}$$

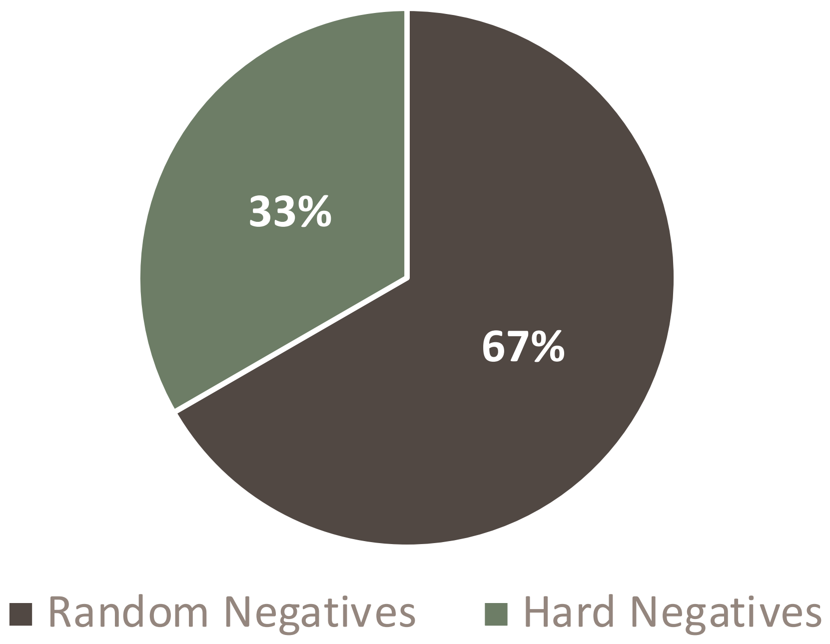
semantically equivalent basic block pair

anchor positive negative margin



• Methodology & Implementation

Mixed negative sampling

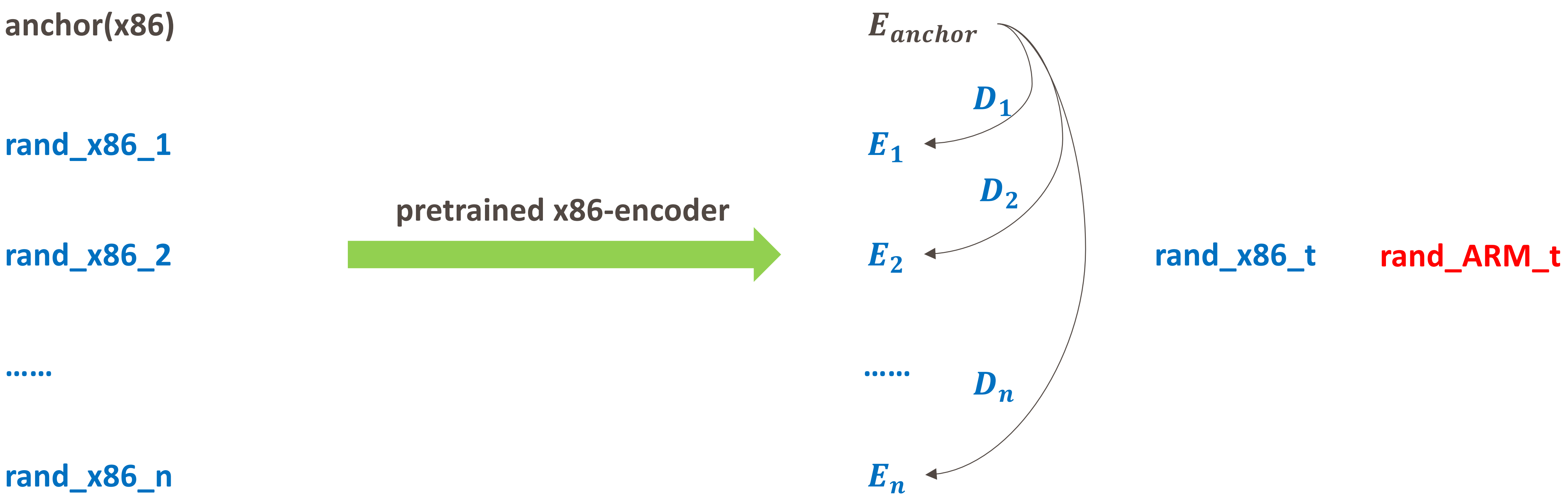


Hard Negatives:

Similar but not equivalent to anchor

• Methodology & Implementation

Hard negative sampling: if anchor is a x86 basic block



• Methodology & Implementation

Similarity Metric

Euclidean distance

$$\text{Sim}(B_1, B_2) = \exp(-\frac{\text{D}(E_1, E_2)}{d})$$

embedding dimension

• Experiment & Result

Setup

- prototype: MIRROR
  https://github.com/zhangxiaochuan/MIRROR
- Dataset: MISA, **1,122,171** semantically equivalent x86-ARM basic block pairs
  https://drive.google.com/file/d/1krJbsfu6EsLhF86QAUVxVRQjbkfWx7ZF/view

Project	Version	Description
Binutils	2.30	collection of binary tools
Coreutils	8.29	GNU core utilities
FFmpeg	n3.2.13	collection of multimedia process tools
OpenSSL	1.1.1b	security protocols and cryptographic library
Redis	5.0.5	key-value database

Experiment & Result

Comparison with Baseline

Model	x86-ARM			ARM-x86		
	P@1	P@3	P@10	P@1	P@3	P@10
INNEREYE-BB	51.0%	66.6%	77.2%	32.8%	54.8%	79.5%
MIRROR ($MISA_{Triplet\_Base}$)	64.0%	77.2%	85.7%	58.7%	73.8%	83.1%
MIRROR ($MISA_{Triplet\_Large}$)	**77.4%**	**88.7%**	**94.9%**	**74.2%**	**87.2%**	**94.1%**

* Higher is better

• Experiment & Result

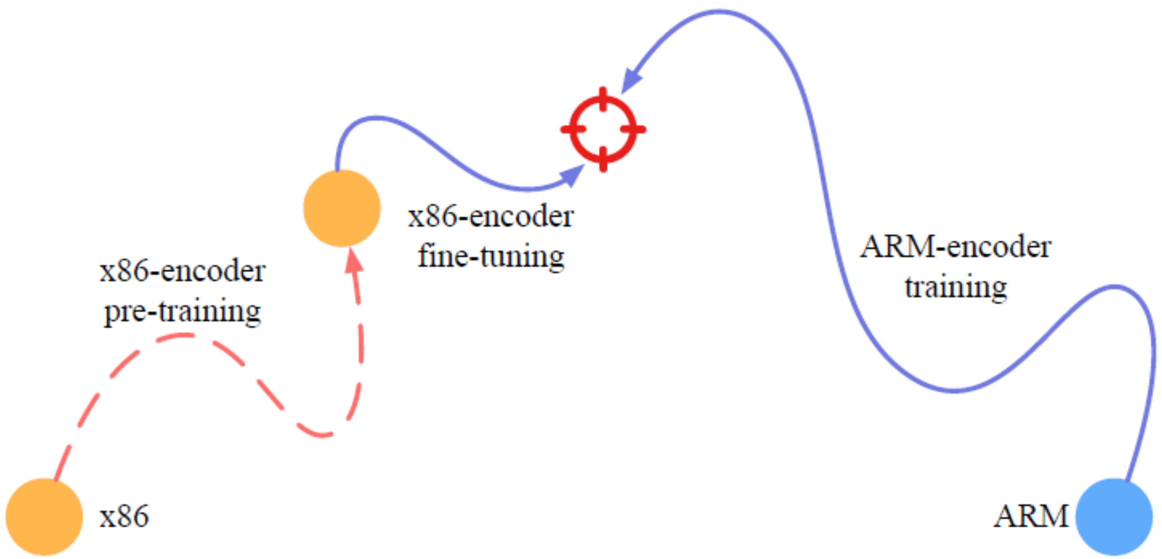
Evaluation of negative sampling methods

Negative Samples	x86-ARM			ARM-x86		
	P@1	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
None	49.6%	56.2%	66.4%	52.5%	62.6%	71.5%
Random only	62.2%	79.2%	89.5%	56.6%	76.1%	87.6%
Hard only	60.0%	74.6%	84.8%	52.7%	70.1%	80.0%
Mixed (ours)	**69.0%**	**83.8%**	**92.9%**	**67.0%**	**83.0%**	**91.5%**

* Higher is better

• Experiment & Result

Effectiveness of pre-training



The pre-training phase seems redundant?

• Experiment & Result

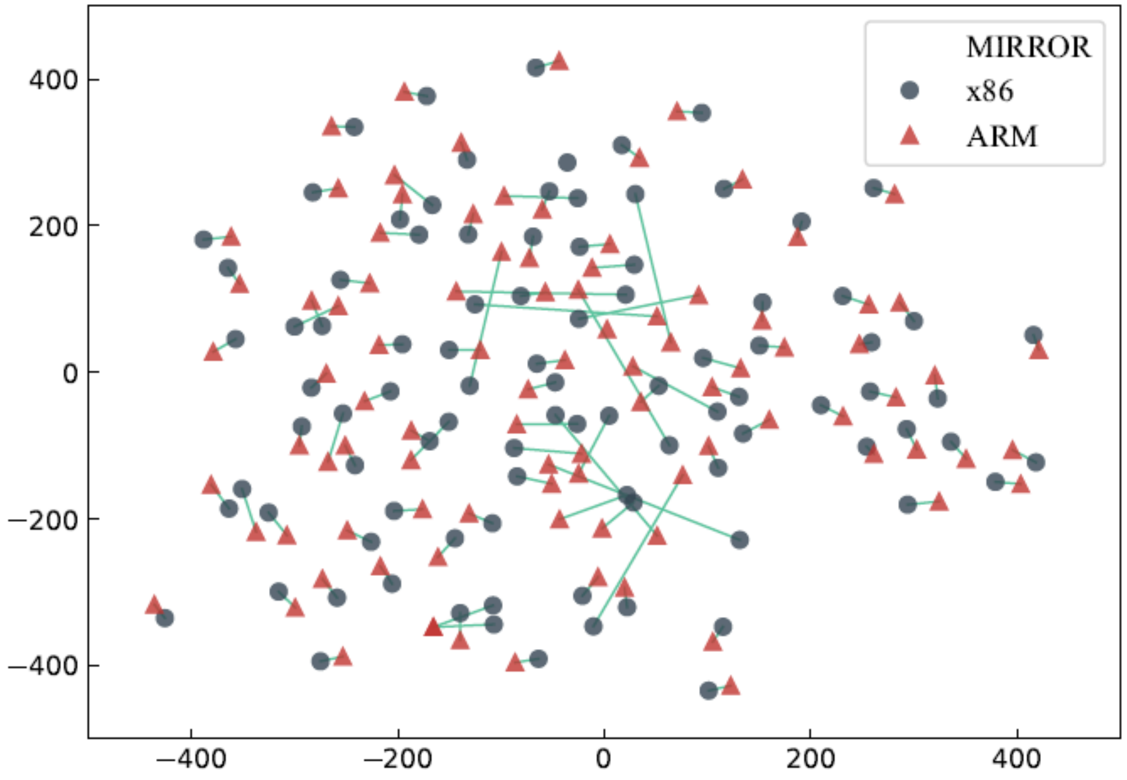
Effectiveness of pre-training

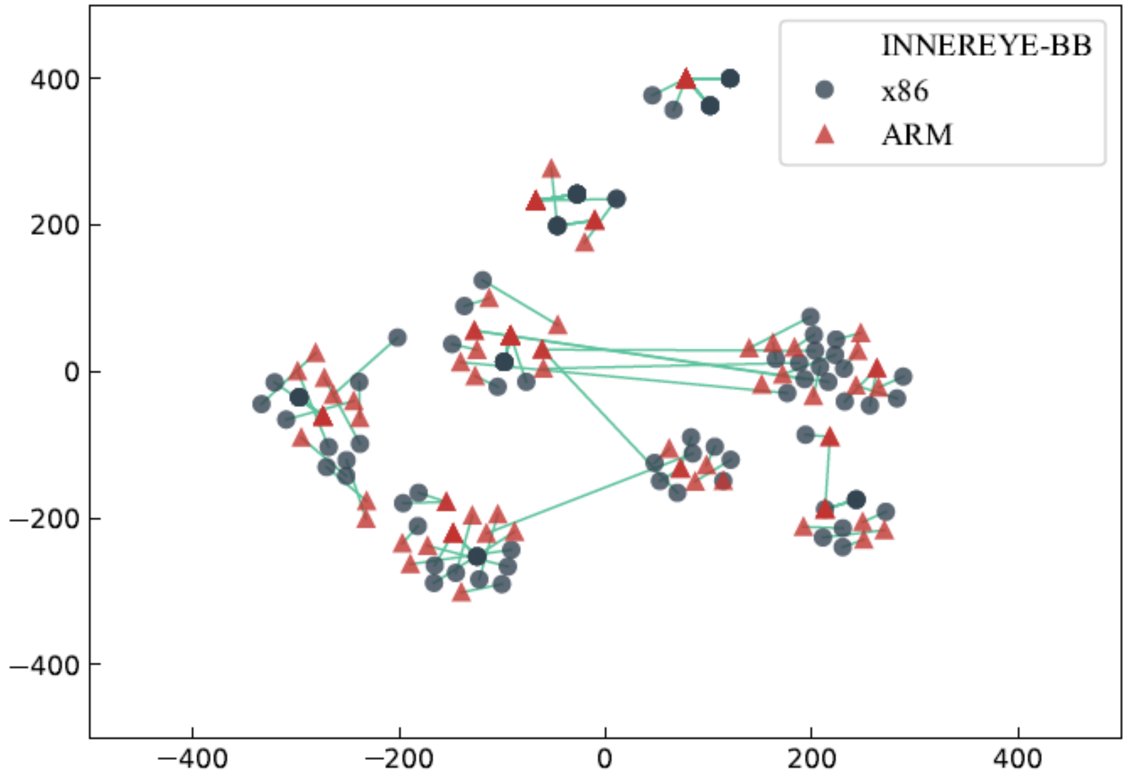
Setting		x86-ARM			ARM-x86		
Pre-train	**Negative**	**P@1**	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
False	Random	58.2%	76.3%	88.4%	53.9%	73.8%	85.7%
True	Random	**62.2%**	**79.2%**	**89.5%**	**56.6%**	**76.1%**	**87.6%**
False	Mixed	64.4%	79.4%	89.1%	61.0%	78.7%	87.7%
True	Mixed	**69.0%**	**83.8%**	**92.9%**	**67.0%**	**83.0%**	**91.5%**

* Higher is better

• Experiment & Result

Visualization





Thanks!

zhangxiaochuan@outlook.com