Transforming Generic Coder LLMs to Effective Binary Code Embedding Models for Similarity Detection

NeurIPS 2025

Litao Li, Leo Song, Steven Ding, Benjamin C. M. Fung, Philippe Charland
(Queen’s University, McGill University, Mission Critical Cyber Security Section, Defence R&D Canada)

Motivation

Why binary code matters

- Needed when source code is unavailable
- Works across compilers and platforms, even without source code

Why it is hard

- Harder than source code: **sparse information, limited syntax**
- LLMs train on trillions of tokens, but the **proportion of binary code is small**

Motivation

Gap in prior work

- From-scratch models stay small and **fail to generalize across compiler settings**
- Some LLM-based methods rely on closed models or very large backbones

Goal

- Uptrain a **small generic coder LLM** into a binary matching expert
- Open and lightweight, robust to **cross-optimization, cross-architecture, cross-obfuscation**

EBM: A Four-Stage Uptraining Framework

1. **Data Augmentation:** Clean assembly and add structure & language tokens
2. **Binary Translation Continual Training:** Translation-style next-token training across settings
3. **LLM2Vec:** Causal decoder to embedding model via bidirectional attention + MNTP
4. **Cumulative GTE Loss:** Enhanced contrastive learning for limited resources

Stage 1: Data Augmentation

Reduce noise

- Assembly comes from a disassembler (IDA Pro)
- Addresses, strings, bytes replaced by special tokens `addr`, `str`, `byte`
- Reduces noise and context length after tokenization

Structure awareness

- Flatten basic blocks and instruction streams into one sentence
- Add a special `BLK` token between basic blocks
- Removes need for hierarchical architecture and layered attention

Translation awareness

- Add language tokens for optimization, compiler, architecture, and obfuscation
- Available during training only, not inference

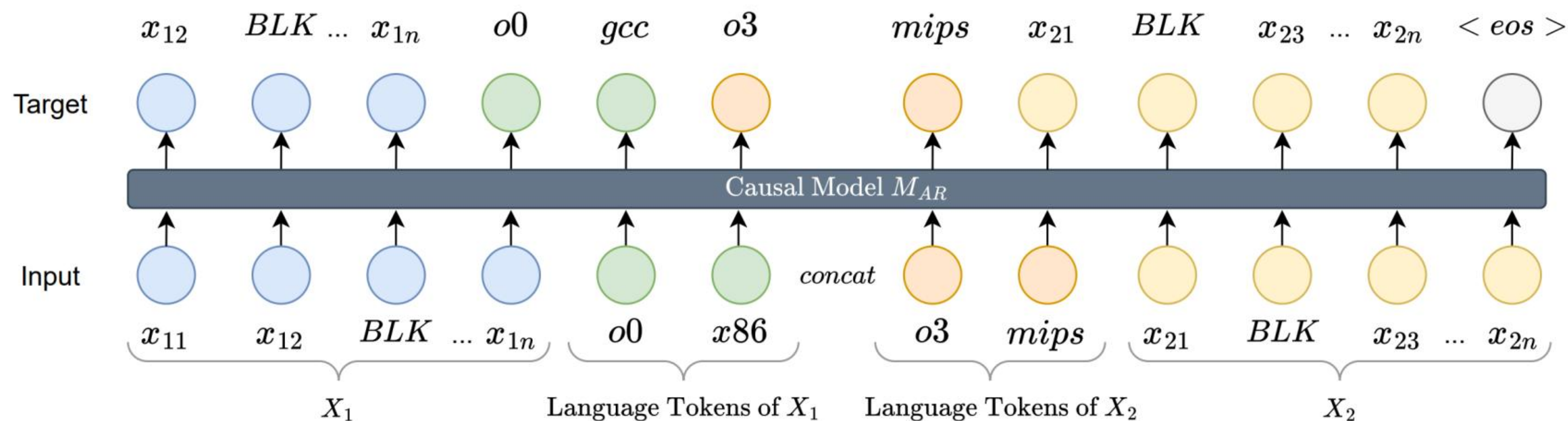
Stage 2: Binary Translation Continual Training

Uptrain assembly with an autoregressive setup

- Coder models include assembly, but understand it less than source languages
- Cheap and effective: **predict the next tokens**

Treat it as translation

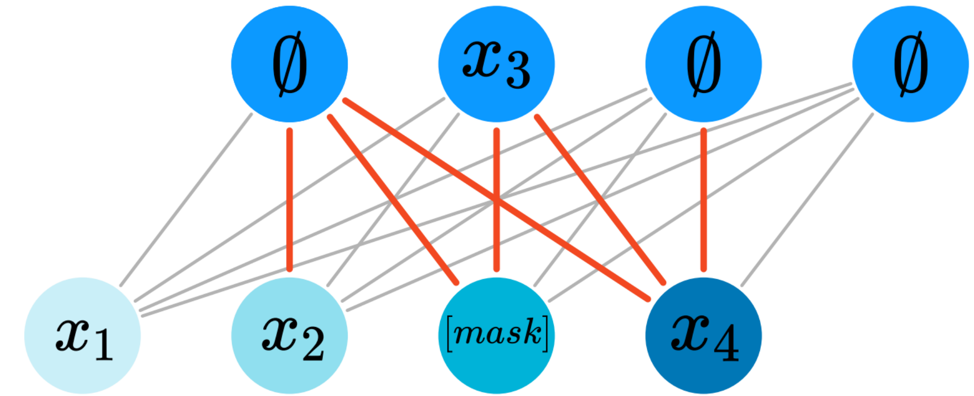
- Different compilers produce different syntaxes, like translation in **natural language**
- Learn the transition from X1 to X2 through autoregressive training
- Language token placed between the pair acts as a **transition** (training only)



Stages 3 & 4: Embedding + Loss

Stage 3: LLM2Vec

- **Bidirectional attention**: reads the full sequence, like BERT
- Causal decoder to embedder via **masked next-token prediction (MNTP)**
- Reuses pretrained weights; beats BERT-from-scratch



Stage 4: Cumulative GTE loss (cGTE)

- Extends InfoNCE: adds **query–query, key–key, key–query** contrasts
- Accumulates embeddings across batches before backpropagation
- Creates more contrastive pairs under small batch size

Experimental Setup

- **Dataset 1: C libraries (pool 1,000)**
  - **Training:** BusyBox, Coreutils, Curl, ImageMagick, PuTTY, SQLite
  - **Evaluation:** GMP, LibTomCrypt, OpenSSL
  - **Optimization:** O0, O1, O2, O3
  - **Compiler:** GCC, Clang
  - **Architecture:** x86/x64, Arm, PowerPC, MIPS

- **Dataset 2: BinaryCorp-3M (pool 10,000)**
  - ArchLinux binaries, cross-optimization only

- **Backbone:** Qwen2.5-Coder-0.5B

- **Baselines:** SAFE, PalmTree, Asm2Vec (non-LLM) / jTrans, CLAP, CodeT5+, CodeGemma (LLM)

- **Metrics:** MRR, Recall@1

Dataset 1: Cross-Architecture

- EBM 0.5B outperforms larger coder LLMs by a large margin

Models	MRR				Recall@1			
	Arm, x64	PowerPC, x64	MIPS, x64	Avg.	Arm, x64	PowerPC, x64	MIPS, x64	Avg.
SAFE	0.239	0.187	0.196	0.208	0.063	0.063	0.063	0.063
PalmTree	0.037	0.036	0.018	0.031	0.031	0.013	0.007	0.017
Asm2Vec	0.242	0.293	0.417	0.317	0.085	0.113	0.231	0.143
OrderMatters	0.007	0.007	0.007	0.007	0.002	0.000	0.001	0.001
GraphCodeBERT (125M)	0.067	0.269	0.495	0.277	0.037	0.204	0.419	0.220
CodeT5+ (110M)	0.056	0.303	0.462	0.274	0.035	0.227	0.392	0.218
Qwen2.5-Emb (1.5B)	0.039	0.059	0.409	0.169	0.031	0.035	0.331	0.132
Qwen2.5-Coder (1.5B)	0.256	0.481	0.548	0.428	0.179	0.380	0.442	0.334
CodeGemma (2B)	0.293	0.581	0.548	0.474	0.208	0.479	0.432	0.373
EBM (0.5B)	**0.783**	**0.792**	**0.859**	**0.811**	**0.675**	**0.703**	**0.784**	**0.721**

Table 2: Evaluation on cross-architecture settings (Arm, x86-64, PowerPC, and MIPS) with a pool size of 1,000.

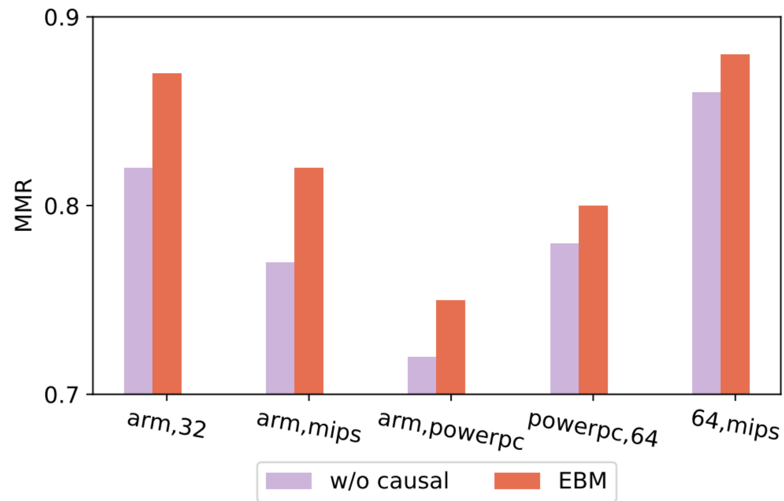
Dataset 2: BinaryCorp-3M

- EBM remains competitive on a larger 10K-pool benchmark

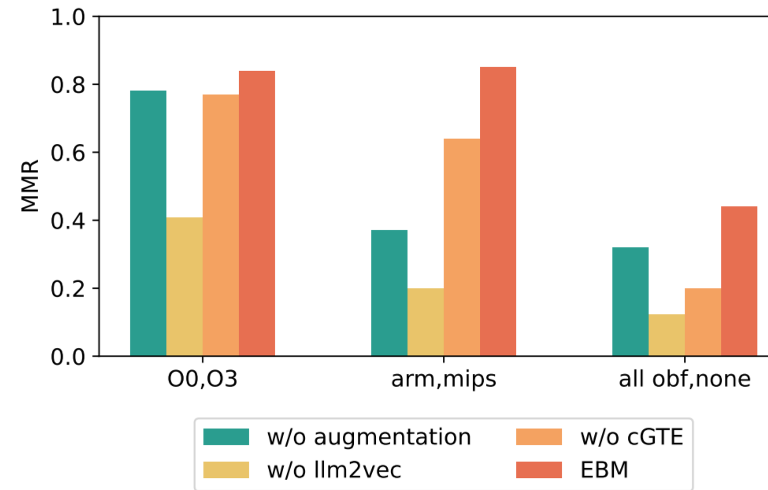
Models	MRR						Recall@1					
	O0,03	O1,O3	O2,O3	O0,Os	O2,Os	**Avg.**	O0,O3	O1,O3	O2,O3	O0,Os	O2,Os	**Avg.**
Gemini	0.037	0.161	0.416	0.049	0.195	0.172	0.024	0.122	0.367	0.030	0.151	0.139
SAFE	0.127	0.345	0.643	0.147	0.377	0.328	0.068	0.247	0.575	0.079	0.283	0.250
OrderMatters	0.062	0.319	0.600	0.075	0.233	0.258	0.040	0.248	0.535	0.040	0.158	0.204
Asm2Vec	0.072	0.449	0.669	0.083	0.510	0.357	0.046	0.367	0.589	0.052	0.426	0.296
PalmTree	0.130	0.403	0.677	0.152	0.496	0.372	0.080	0.326	0.609	0.097	0.420	0.306
jTrans (Zero Shot)	0.137	0.490	0.693	0.182	0.513	0.403	0.088	0.412	0.622	0.122	0.430	0.335
jTrans (Finetune)	0.475	0.663	0.731	0.539	0.664	0.614	0.376	0.580	0.661	0.443	0.585	0.529
CLAP	0.764	0.903	0.941	**0.813**	0.877	0.860	0.719	0.875	0.920	0.774	0.847	0.827
EBM	**0.779**	**0.911**	**0.955**	0.808	**0.909**	**0.872**	**0.725**	**0.882**	**0.942**	**0.808**	**0.889**	**0.849**

Table 4: Evaluation on the BinaryCorp-3M dataset with pool size=10,000

Ablation Study



(a) Causal training ablation study



(b) Data augmentation, LLM2Vec, and cGTE ablation.

Figure 3: (3a) shows the MRR for two versions of EBM, with and without causal training. Cross-architecture retrieval is particularly sensitive to causal training, as it enables better translation between different languages. In (3b), EBM significantly improves the MRR compared to ablated models. LLM2Vec contributes the most to the increase.

Conclusion

Contributions

- 4-process uptraining framework (EBM) for binary similarity
- 0.5B model beats zero-shot LLMs & SOTA, no GPT / CFG needed

My thoughts

- Retrieval-only evaluation
    - No direct validation on malware, vulnerability, or patch detection
- Limited real-world coverage
    - Unclear generalization to C++, packed binaries, commercial binaries, malware, and larger LLMs

Thank You