Enhancing MLGO Inlining with IR2Vec Embeddings

S. VenkataKeerthy

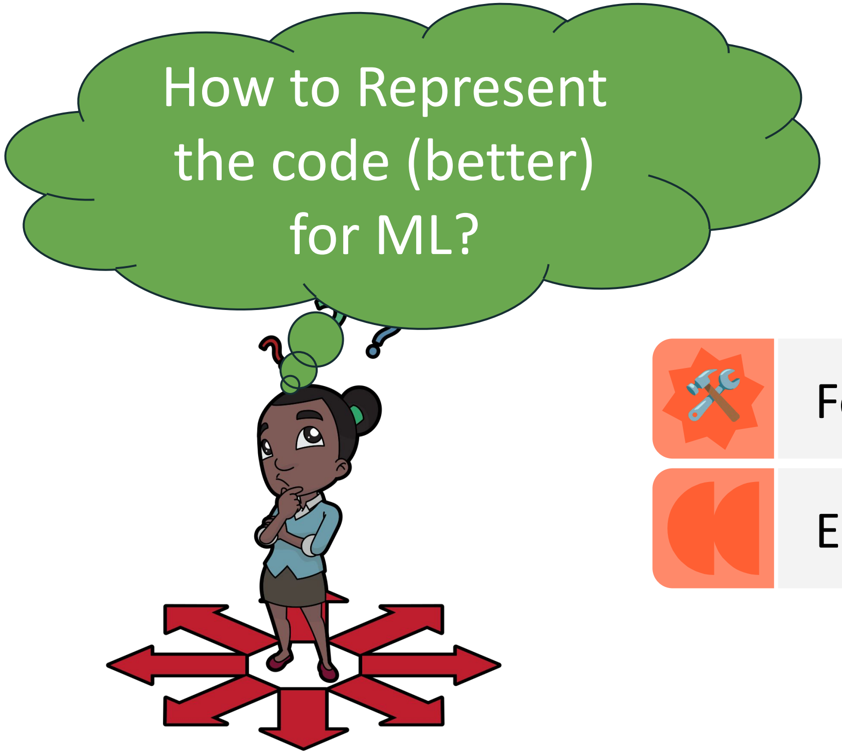
IIT Hyderabad, Google

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

US LLVM Developers’ Meeting

28th October 2025

Program Representations



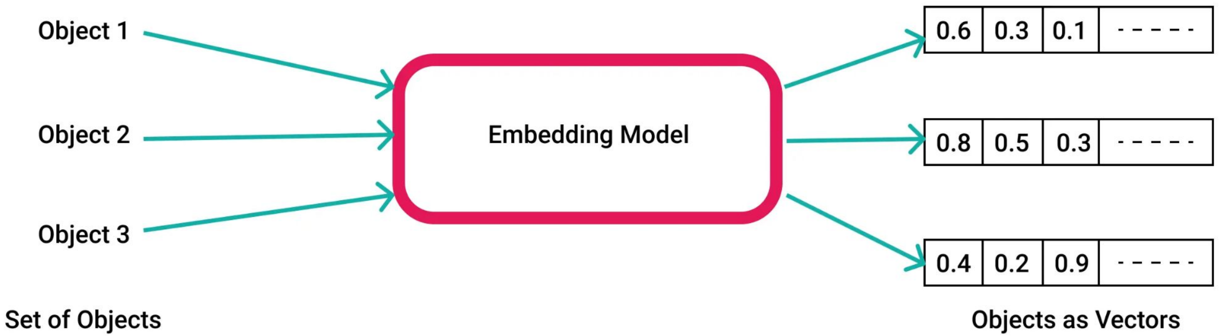
- Feature Engineering
- Embeddings

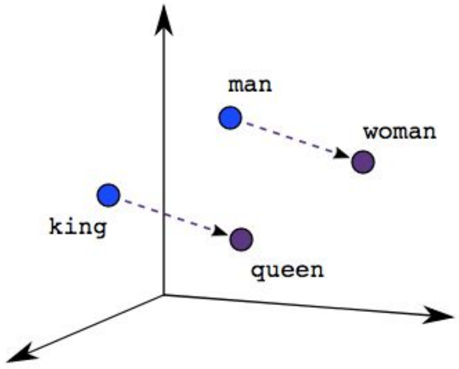
Features Vs. Embeddings

Features capture what we think is important; embeddings discover what is important.

🛠️ **Features**	🔮 **Embeddings**
Handcrafted	Learned automatically
Task-specific	General-purpose, reusable
Shallow statistics (counts, depths, CFG metrics)	Deep semantics

Embeddings: Brief Background

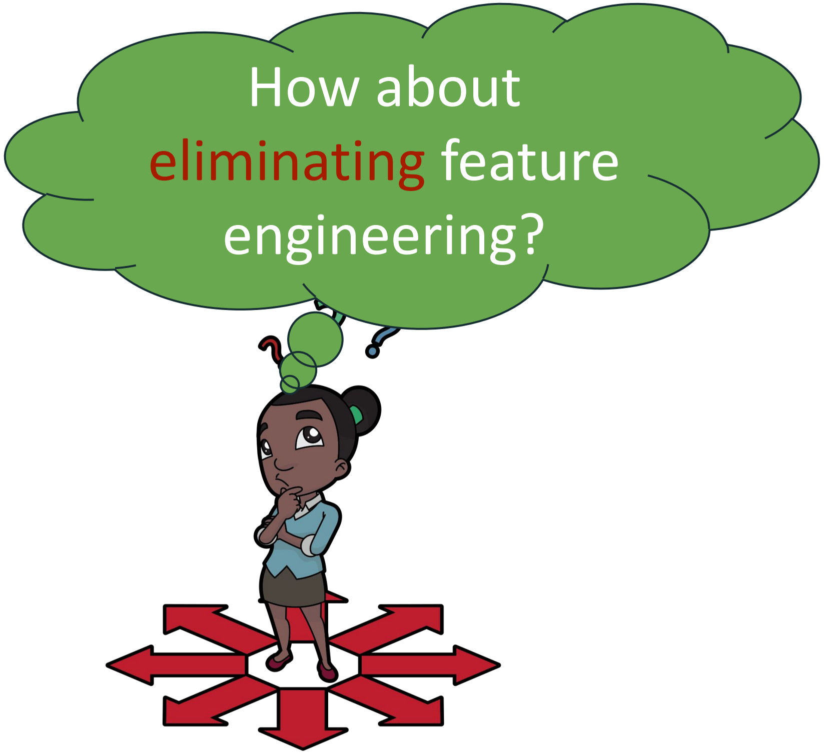




Male-Female

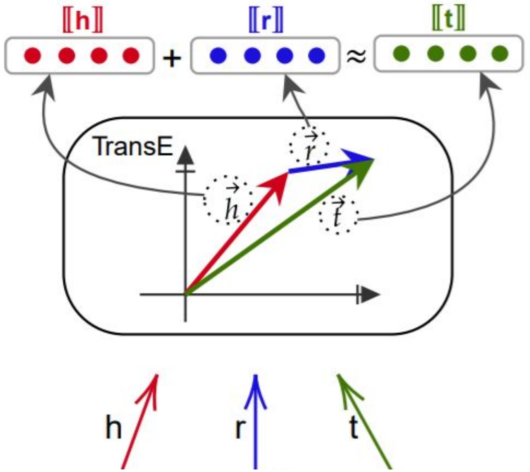
https://medium.com/data-science/a-guide-to-word-embeddings-8a23817ab60f

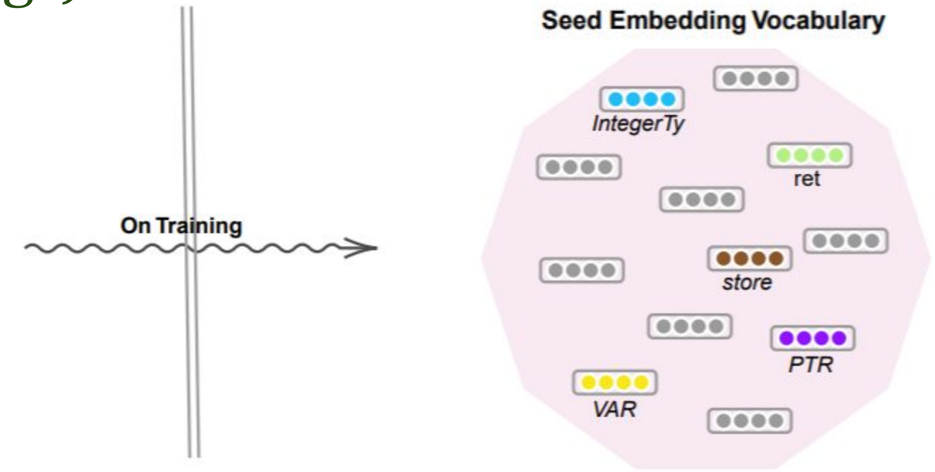
Program Representations



IR2Vec
LLVM IR Based *Language & Machine Independent* *Program Analysis based approach*
Distributed Encodings *semantic meaning is ‘distributed’ across components of the vector*
Agglomerative / Bottom-Up Approach *Doesn’t need complex ML models* *Independent of Applications*

IR2Vec (Symbolic Encodings)



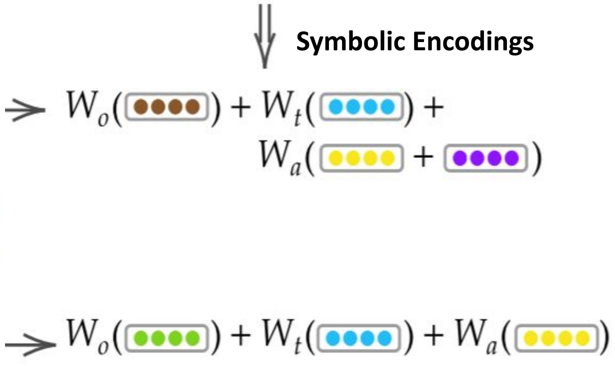


⟨ store, "TypeOf", IntegerTy ⟩
⟨ store, "NextInst", store ⟩
⟨ store, "Arg_1", VAR ⟩
⟨ store, "Arg_2", PTR ⟩

. . .

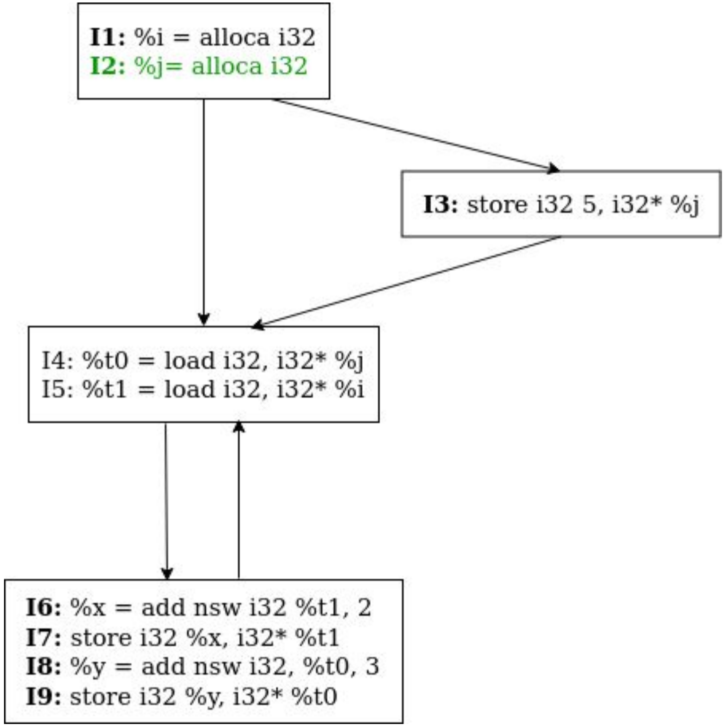
⟨ ret, "TypeOf", IntegerTy ⟩
⟨ ret, "Arg_1", VAR ⟩

```
%a.addr = alloca i32, align 4
%b.addr = alloca i32, align 4
store i32 %a, i32* %a.addr, align 4
store i32 %b, i32* %b.addr, align 4
%0 = load i32, i32* %a.addr, align 4
%1 = load i32, i32* %b.addr, align 4
%add = add nsw i32 %0, %1
ret i32 %add
```



Flow-Aware Encodings: Symbolic + Flow Information

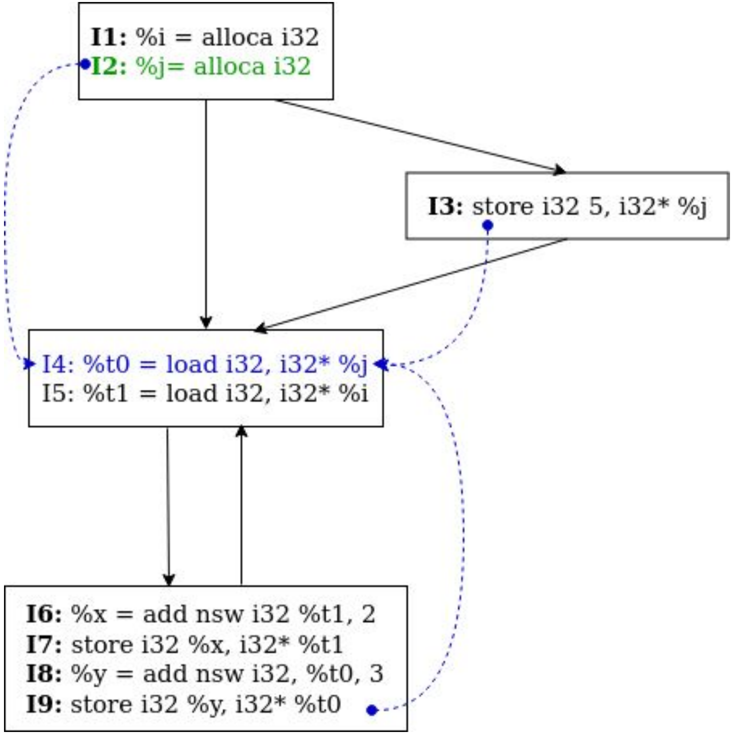
Improving Symbolic Encodings with Flow Information



$$\boxed{[\![\mathbf{I}_2]\!] = W_o([\![\mathbf{alloca}]\!]) + W_t([\![\mathbf{IntegerTy}]\!])}$$

Flow-Aware Encodings: Symbolic + Flow Information

Improving Symbolic Encodings with Flow Information

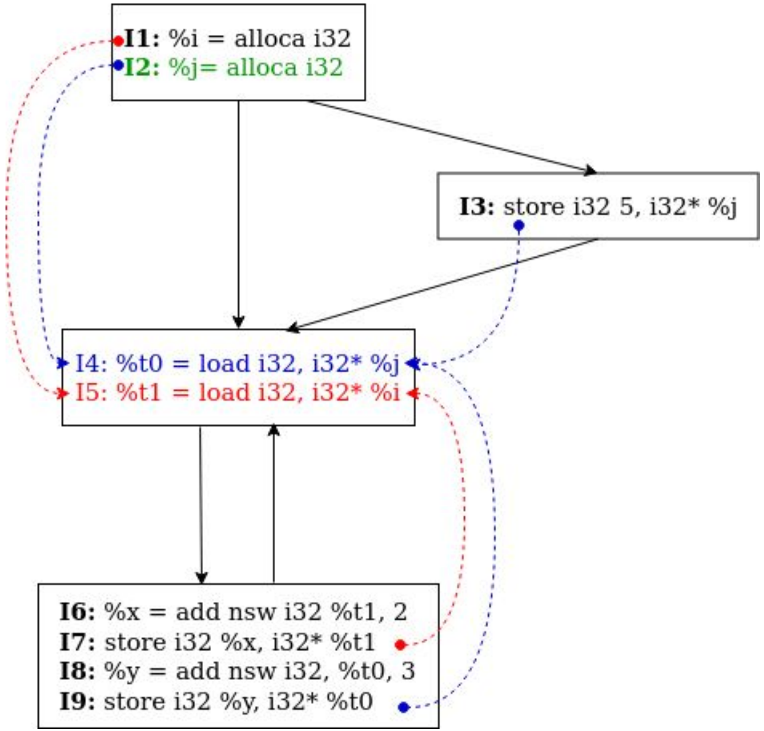


$$[\![I_2]\!] = W_o([\![\mathrm{alloca}]\!]) + W_t([\![\mathrm{IntegerTy}]\!])$$

$$[\![I_4]\!] = W_o([\![\mathrm{load}]\!]) + W_t([\![\mathrm{PointerTy}]\!]) + W_a([\![I_2]\!] + [\![I_3]\!] + [\![I_9]\!])$$

Flow-Aware Encodings: Symbolic + Flow Information

Improving Symbolic Encodings with Flow Information



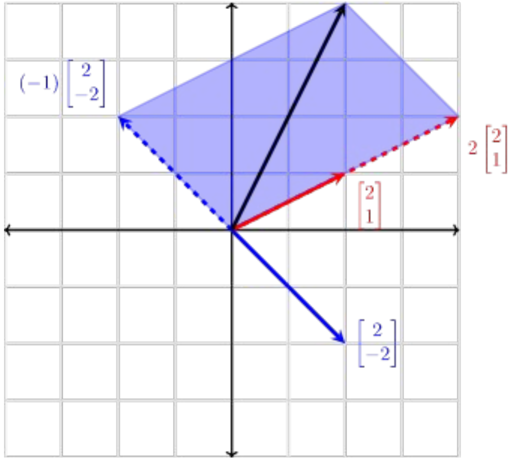
$$\llbracket \mathrm{I}_2 \rrbracket = W_o(\llbracket \mathbf{alloca} \rrbracket) + W_t(\llbracket \mathbf{IntegerTy} \rrbracket)$$

$$\llbracket \mathrm{I}_4 \rrbracket = W_o(\llbracket \mathbf{load} \rrbracket) + W_t(\llbracket \mathbf{PointerTy} \rrbracket) + W_a(\llbracket \mathrm{I}_2 \rrbracket + \llbracket \mathrm{I}_3 \rrbracket + \llbracket \mathrm{I}_9 \rrbracket)$$

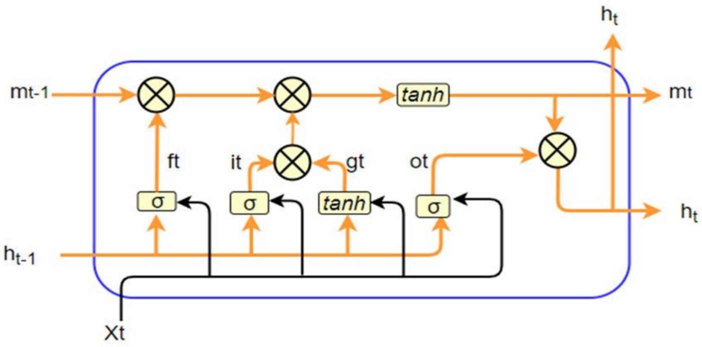
$$\llbracket \mathrm{I}_5 \rrbracket = W_o(\llbracket \mathbf{load} \rrbracket) + W_t(\llbracket \mathbf{PointerTy} \rrbracket) + W_a(\llbracket \mathrm{I}_1 \rrbracket + \llbracket \mathrm{I}_7 \rrbracket)$$

Code Vectors: Beyond Instruction Representation

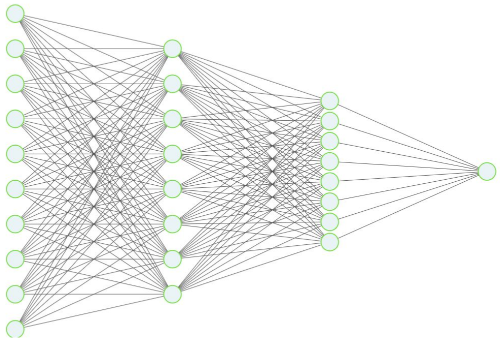
Aggregators to effectively compose Instruction vectors



Linear Combination



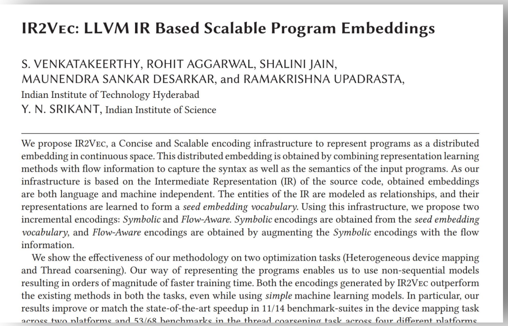
Sequential models



Non-Sequential models

Extensions to IR2Vec

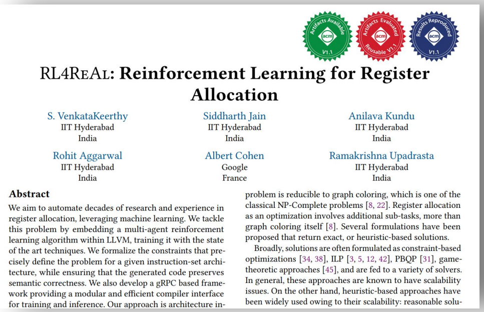
IR2Vec	MIR2Vec	VexIR2Vec
• Machine Independent • From Source Code	• Machine Specific • From Source Code	• Machine Specific • From Binary


IR2Vec: LLVM IR Based Scalable Program Embeddings

S. VENKATAKEERTHY, ROHIT AGGARWAL, SHALINI JAIN, MAUNENDRA SANKAR DESARKAR, and RAMAKRISHNA UPADRASTA, Indian Institute of Technology Hyderabad
Y. N. SRIKANT, Indian Institute of Science

We propose IR2Vec, a Concise and Scalable encoding infrastructure to represent programs as a distributed embedding in continuous space. This distributed embedding is obtained by combining representation learning methods with flow information to capture the syntax as well as the semantics of the input programs. As our infrastructure is based on the Intermediate Representation (IR) of the source code, obtained embeddings are both language and machine independent. The entities of the IR are modeled as relationships, and their representations are learned to form a *seed embedding vocabulary*. Using this infrastructure, we propose two incremental encodings: *Symbolic* and *Flow-Aware*. *Symbolic* encodings are obtained from the *seed embedding vocabulary*, and *Flow-Aware* encodings are obtained by augmenting the *Symbolic* encodings with the flow information.

We show the effectiveness of our methodology on two optimization tasks (Heterogeneous device mapping and Thread coarsening). Our way of representing the programs enables us to use non-sequential models resulting in orders of magnitude of faster training time. Both the encodings generated by IR2Vec outperform the existing methods in both the tasks, even while using *simple* machine learning models. In particular, our results improve or match the state-of-the-art speedup in 11/14 benchmark-suites in the device mapping task


RL4ReAl: Reinforcement Learning for Register Allocation

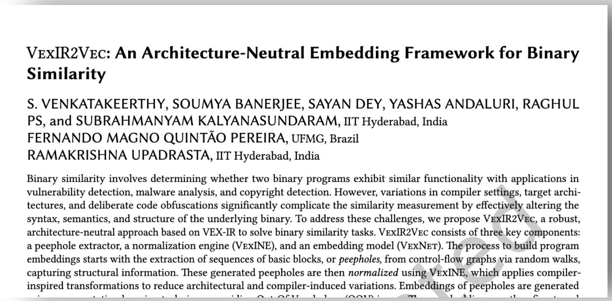
S. VenkataKeerthy, IIT Hyderabad, India
Siddharth Jain, IIT Hyderabad, India
Anilava Kundu, IIT Hyderabad, India
Rohit Aggarwal, IIT Hyderabad, India
Albert Cohen, Google, France
Ramakrishna Upadrasta, IIT Hyderabad, India

Abstract

We aim to automate decades of research and experience in register allocation, leveraging machine learning. We tackle this problem by embedding a multi-agent reinforcement learning algorithm within LLVM, training it with the state of the art techniques. We formalize the constraints that precisely define the problem for a given instruction-set architecture, while ensuring that the generated code preserves semantic correctness. We also develop a gRPC based framework providing a modular and efficient compiler interface for training and inference. Our approach is architecture in-

problem is reducible to graph coloring, which is one of the classical NP-Complete problems [8, 22]. Register allocation as an optimization involves additional sub-tasks, more than graph coloring itself [8]. Several formulations have been proposed that return exact, or heuristic-based solutions.

Broadly, solutions are often formulated as constraint-based optimizations [34, 38], ILP [3, 5, 12, 42], PBQP [31], game-theoretic approaches [45], and are fed to a variety of solvers. In general, these approaches are known to have scalability issues. On the other hand, heuristic-based approaches have been widely used owing to their scalability: reasonable solu-

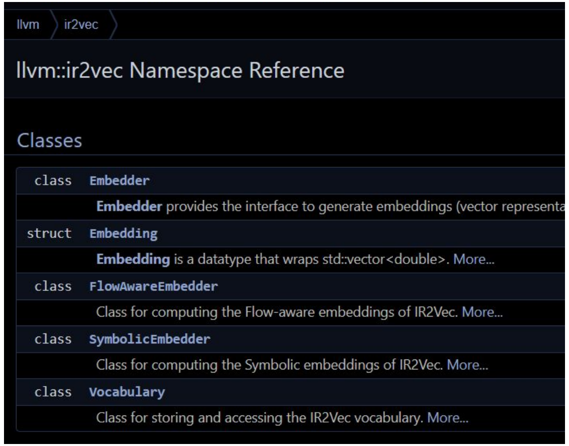

VexIR2Vec: An Architecture-Neutral Embedding Framework for Binary Similarity

S. VENKATAKEERTHY, SOUMYA BANERJEE, SAYAN DEY, YASHAS ANDALURI, RAGHUL PS, and SUBRAHMANYAM KALYANASUNDARAM, IIT Hyderabad, India
FERNANDO MAGNO QUINTÃO PEREIRA, UFMG, Brazil
RAMAKRISHNA UPADRASTA, IIT Hyderabad, India

Binary similarity involves determining whether two binary programs exhibit similar functionality with applications in vulnerability detection, malware analysis, and copyright detection. However, variations in compiler settings, target architectures, and deliberate code obfuscations significantly complicate the similarity measurement by effectively altering the syntax, semantics, and structure of the underlying binary. To address these challenges, we propose VexIR2Vec, a robust, architecture-neutral approach based on VEX-IR to solve binary similarity tasks. VexIR2Vec consists of three key components: a peephole extractor, a normalization engine (VexINE), and an embedding model (VexNet). The process to build program embeddings starts with the extraction of sequences of basic blocks, or *peepholes*, from control-flow graphs via random walks, capturing structural information. These generated peepholes are then *normalized* using VexINE, which applies compiler-inspired transformations to reduce architectural and compiler-induced variations. Embeddings of peepholes are generated

IR2Vec in LLVM

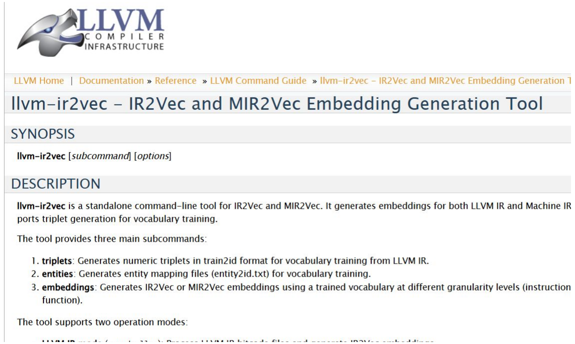
- RFC and Discussions*
- LLVM Upstreaming
  - IR2Vec integrated under `llvm/Analysis`
  - MIR2Vec in `llvm/CodeGen`
  - `llvm-ir2vec` standalone tool in `llvm/tools`
    - Supports embedding and vocab generation
- Upcoming
  - Python library, installable via pip
- Source code (Research) - https://github.com/IITH-Compilers/IR2Vec


llvm › ir2vec

llvm::ir2vec Namespace Reference

Classes

class	Embedder
	Embedder provides the interface to generate embeddings (vector representa
struct	Embedding
	Embedding is a datatype that wraps std::vector<double>. More...
class	FlowAwareEmbedder
	Class for computing the Flow-aware embeddings of IR2Vec. More...
class	SymbolicEmbedder
	Class for computing the Symbolic embeddings of IR2Vec. More...
class	Vocabulary
	Class for storing and accessing the IR2Vec vocabulary. More...


LLVM COMPILER INFRASTRUCTURE

LLVM Home | Documentation » Reference » LLVM Command Guide » llvm-ir2vec – IR2Vec and MIR2Vec Embedding Generation T

llvm-ir2vec – IR2Vec and MIR2Vec Embedding Generation Tool

SYNOPSIS

llvm-ir2vec [*subcommand*] [*options*]

DESCRIPTION

llvm-ir2vec is a standalone command-line tool for IR2Vec and MIR2Vec. It generates embeddings for both LLVM IR and Machine IR ports triplet generation for vocabulary training.

The tool provides three main subcommands:

1. **triplets**: Generates numeric triplets in train2id format for vocabulary training from LLVM IR.
2. **entities**: Generates entity mapping files (entity2id.txt) for vocabulary training.
3. **embeddings**: Generates IR2Vec or MIR2Vec embeddings using a trained vocabulary at different granularity levels (instruction function).

The tool supports two operation modes:

* https://discourse.llvm.org/t/rfc-enhancing-mlgo-inlining-with-ir2vec-embeddings

IR2Vec in LLVM

- Vocabulary Analyses
  - `IR2VecVocabAnalysis`, `MIR2VecVocabLegacyAnalysis`
- Embedder classes
  - `ir2vec::Embedder`, `mir2vec::MIREmbedder`
- Embeddings class
  - Wrapper around `std::vector<double>`

Using IR2Vec

- **Programmatically in LLVM Passes**
  - Query vocabulary via `IR2VecVocabAnalysis` / `MIR2VecVocabLegacyAnalysis`
  - Create an Embedder and extract embeddings at function or block or instruction level.

```
auto &VocabRes = MAM.getResult<IR2VecVocabAnalysis>(M);

if (!VocabRes.isValid()) { … }

const ir2vec::Vocab &Vocabulary = VocabRes.getVocabulary();

auto Emb = ir2vec::Embedder::create(IR2VecKind::Symbolic, F, Vocabulary);

Embedding FVec = Emb->getFunctionVector();
```

Using IR2Vec

- **Via standalone `llvm-ir2vec` Tool**

  Generates embeddings from .bc or .ll files.

  ```
  llvm-ir2vec embeddings --ir2vec-vocab-path=vocab.json input.bc -o embeddings.txt
  ```

- **Vocabulary Training**
  - `llvm-ir2vec` also helps in generating data for training vocabulary

    ```
    llvm-ir2vec triplets input.bc -o train2id.txt
    ```

  - Planning to automate the vocab generation
    - see `llvm/utils/mlgo-utils/IR2Vec/generateTriplets.py`

ML-Driven Compiler Optimizations

IR2Vec

[VenkataKeerthy, et al, TACO'20]

+

+

ML-Compiler-Bridge

[VenkataKeerthy, Jain, et al, CC'24]

ML-Driven Optimizations

RL-Loop Distribution

[Jain, VenkataKeerthy, et al, LLVM HPC'22]

Phase Ordering
(POSET-RL)

[Jain, VenkataKeerthy, et al, ISPASS'22]

Register Allocation
(RL4ReAl)

[VenkataKeerthy, Jain, et al, CC'23]

In LLVM

Function Inlining

Register Eviction in RegallocGreedy

*Planned - MIR2Vec application

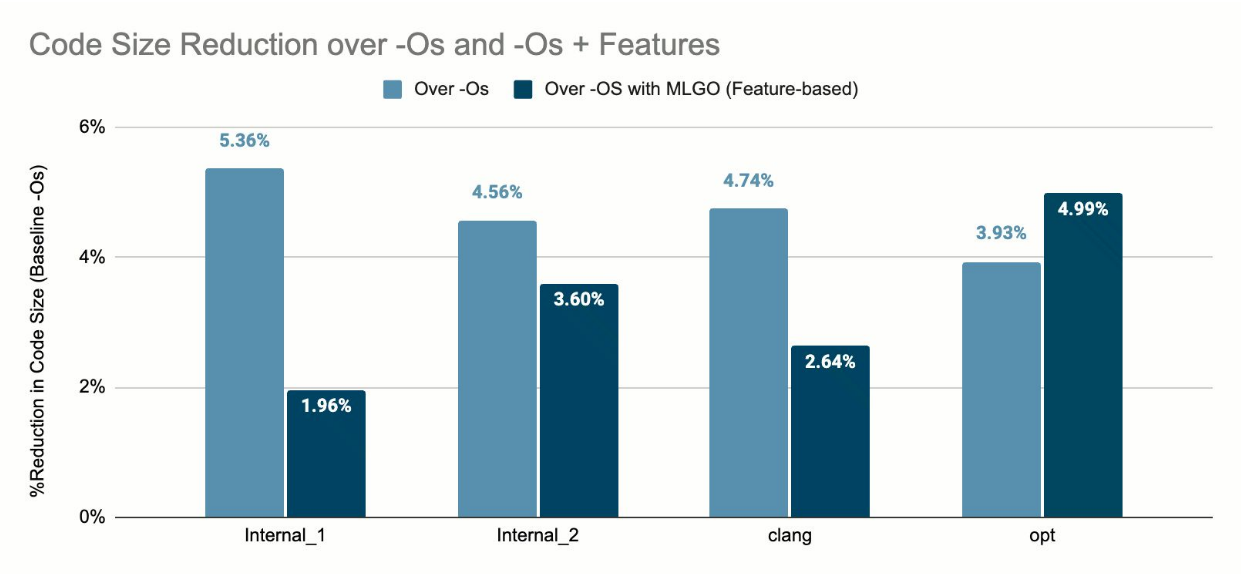
MLInlineAdvisor

- Feature-based, Uses 32 hand-picked features
- Optimization for size
- PPO, ES, Imitation Learning based approaches
- C++ related code available in LLVM
  - Python based training infra – https://github.com/google/ml-compiler-opt

IR2Vec with MLInlineAdvisor

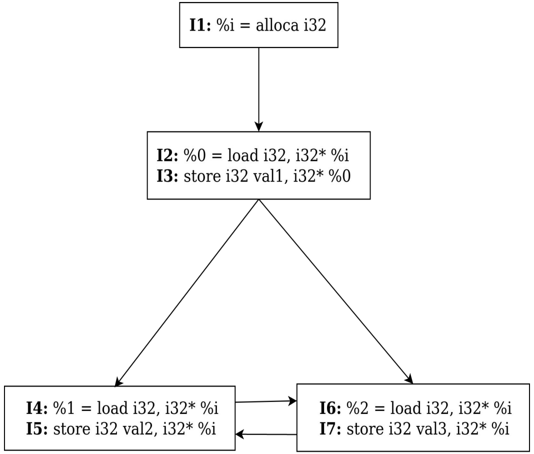
Concatenating embeddings with features

Trained models from scratch on internal datacenter binaries, ~50K modules, PPO policy, ~20M steps



Open Challenges & the road ahead

Flow-Aware Embeddings: Cyclic dependencies!



$$[\![\mathbf{I}_5]\!] = W_o([\![\mathbf{store}]\!]) + W_t([\![\mathbf{IntegerTy}]\!]) + W_a([\![\mathbf{VAR}]\!]) + W_a([\![\mathbf{I}_3]\!] + [\![\mathbf{I}_7]\!])$$

$$[\![\mathbf{I}_7]\!] = W_o([\![\mathbf{store}]\!]) + W_t([\![\mathbf{IntegerTy}]\!]) + W_a([\![\mathbf{VAR}]\!]) + W_a([\![\mathbf{I}_3]\!] + [\![\mathbf{I}_5]\!])$$

$$[\![\mathbf{I}_5]\!] = k_1 + W_a * [\![\mathbf{I}_7]\!]$$
$$[\![\mathbf{I}_7]\!] = k_2 + W_a * [\![\mathbf{I}_5]\!]$$

Linear Solver for Flow-Aware Embeddings / Cyclic Dependencies

Design trade-offs

- Simple linear solver - Handwritten (where - `llvm/utils`?) or `eigen` like libraries?
- Iterative solution

Encoding More Information

- Memory dependences
  - `Memdep` analysis or `MemorySSA` can make Flow-Aware embeddings more elegant
  - But they are highly conservative!
    - Embeddings can tolerate False-negatives unlike optimizations
    - Specialize them?
    - Biasing based on memtrace profiling?
- Profile Information

MIR Vocabulary

- Vocabulary uses regex to group MIR opcodes (target specific)
  - Eg: "ADD32rr" -> "ADD"
  - This is mainly a learning quality enhancer
- x86 → 6.8K opcodes after grouping
  - In comparison, IR2Vec vocab has only ~100 entities in total
- Need for better "canonicalization", beyond regex
  - x86 has a systematic approach (uses TableGen)
    - OpPrefix - PD, PS, XS, XD, etc.
    - Width - 32, 64, etc.
    - OpForm - rm, rr, ri, etc.
    - …
      - These prefixes and suffixes are attached with the "generic" opcodes
  - But such an approach is not generalizable across architectures

Automating Vocab Generation

- Buildbots to generate dataset (triplets) using the latest compiler
  - On LLVM codebase
  - Lightweight (<10 mins with 64 vCPUs)
  - Can also serve as an integration test
- Training Job on the generated data
  - Uses the last generated dataset to train the vocabulary
  - Can be less-frequent

In Summary...

- **IR2Vec**: learned, scalable, architecture-independent program embeddings
  - Upstreamed, addressed performance related issues, ready to use
  - Aims to reduce and remove efforts towards feature engineering
- Demonstrates potential in both research and real-world production environments

Next Steps

- Replace features, instead of concatenation
- Using MIR2Vec for eviction decisions in Regalloc Greedy
- Automating vocabulary training pipelines
- Improving Flow-Aware infrastructure
- …

Thank You!

S. VenkataKeerthy
https://svkeerthy.github.io

Acknowledgements

Mircea Trofin, Albert Cohen, Ramakrishna Upadrasta, Other contributors and co-authors

Interested? Let’s talk!

(discourse @svkeerthy)