

# Performance-Oriented Code Representation

**Kaveh Shahedi**, Heng Li  
Polytechnique Montréal  
Summer 2024



# What do we mean by “Code Representation”?

Input **Text**

A great text exclusively  
written for the Dorsal lab's  
progress report meeting



Applying some  
**MAGIC**  
(a.k.a., embedding)



Output as a **Vector**

0.123

-0.723

0.003

...

0.901

# What do we mean by “Code Representation”?



The **procedure** is the **same** for **code**!

# What do we mean by “Code Representation”?

Input **Code**

```
public class Dorsal extends Poly {  
    public String getEventName() {  
        return "May 2024 Progress Report";  
    }  
}
```

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output as a **Vector**

|        |       |       |     |       |
|--------|-------|-------|-----|-------|
| -0.004 | 0.872 | 0.887 | ... | 0.026 |
|--------|-------|-------|-----|-------|

# What's the usage of these embeddings?

Input **Text 1**

An international PhD student  
quitted his studies after getting  
rejected from a famous journal.

Input **Text 2**

A "Reviewer 2" from a famous journal  
helped a PhD student get back to  
normal life sooner than expected.

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

|       |        |       |     |       |
|-------|--------|-------|-----|-------|
| 0.123 | -0.723 | 0.003 | ... | 0.901 |
|-------|--------|-------|-----|-------|

Output 2 as a **Vector**

|       |        |        |     |       |
|-------|--------|--------|-----|-------|
| 0.131 | -0.690 | -0.001 | ... | 0.899 |
|-------|--------|--------|-----|-------|

# What's the usage of these embeddings?

Input **Text 1**

An international PhD student  
quitted his studies after getting  
rejected from a famous journal.

Input **Text 2**

A "Reviewer 2" from a famous journal  
helped a PhD student get back to  
normal life sooner than expected.

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

|       |        |       |     |       |
|-------|--------|-------|-----|-------|
| 0.123 | -0.723 | 0.003 | ... | 0.901 |
|-------|--------|-------|-----|-------|

**Similar Embeddings!**

Output 2 as a **Vector**

|       |        |        |     |       |
|-------|--------|--------|-----|-------|
| 0.131 | -0.690 | -0.001 | ... | 0.899 |
|-------|--------|--------|-----|-------|

# What's the usage of these embeddings?

Input **Text 1**

My friend is now going to HR because his joke was so funny that they want to hear it too.

Input **Text 2**

I am a billionaire CEO, and I like people to laugh at me. I also like to pick a public fight with Yann LeCun (who?).

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

0.233   -0.761   -0.089   ...   0.402

Output 2 as a **Vector**

-0.990   -0.137   0.553   ...   -0.001

# What's the usage of these embeddings?

Input **Text 1**

My friend is now going to HR because his joke was so funny that they want to hear it too.

Input **Text 2**

I am a billionaire CEO, and I like people to laugh at me. I also like to pick a public fight with Yann LeCun (who?).

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

|       |        |        |     |       |
|-------|--------|--------|-----|-------|
| 0.233 | -0.761 | -0.089 | ... | 0.402 |
|-------|--------|--------|-----|-------|

**Different Embeddings!**

Output 2 as a **Vector**

|        |        |       |     |        |
|--------|--------|-------|-----|--------|
| -0.990 | -0.137 | 0.553 | ... | -0.001 |
|--------|--------|-------|-----|--------|



# What's the usage of these embeddings?

Input **Code 1**

```
public int factorialForLoop(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++)  
        result *= i;  
    return result;  
}
```

Input **Code 2**

```
public int factorialRecursive(int n) {  
    if (n == 0 || n == 1)  
        return 1;  
    else  
        return n * factorialRecursive(n - 1);  
}
```

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

|        |       |       |     |       |
|--------|-------|-------|-----|-------|
| -0.004 | 0.872 | 0.887 | ... | 0.026 |
|--------|-------|-------|-----|-------|

Output 2 as a **Vector**

|       |       |       |     |       |
|-------|-------|-------|-----|-------|
| 0.001 | 0.800 | 0.802 | ... | 0.051 |
|-------|-------|-------|-----|-------|

# What's the usage of these embeddings?

Input **Code 1**

```
public int factorialForLoop(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++)  
        result *= i;  
    return result;  
}
```

Input **Code 2**

```
public int factorialRecursive(int n) {  
    if (n == 0 || n == 1)  
        return 1;  
    else  
        return n * factorialRecursive(n - 1);  
}
```

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a **Vector**

|        |       |       |     |       |
|--------|-------|-------|-----|-------|
| -0.004 | 0.872 | 0.887 | ... | 0.026 |
|--------|-------|-------|-----|-------|

**Similar Embeddings!**

Output 2 as a **Vector**

|       |       |       |     |       |
|-------|-------|-------|-----|-------|
| 0.001 | 0.800 | 0.802 | ... | 0.051 |
|-------|-------|-------|-----|-------|

# Emmm, how does this **MAGIC** part works?

**NLP**

AI + Computers → **Understand human language**

**LMs**

Statistical models + Input text → **Predict and generate new texts**

**CE**

Input text + LM → **Represent as a vector**

# Emmm, how does this **MAGIC** part works?

Word2Vec

Doc2Vec

AST-Based

LLMs

---

Code Generation and Completion

Code Clone Detection

Bug Detection and Code Smells

Code Document Generation

Okay cool, but what do  
“**Code Representation**” and “**Code Performance**”  
have to do with each other?

# Why should we use such code representation?

└─→ for performance aspects

1. **Dynamic** approaches (e.g., instrumentation/tracing) are **expensive**, especially in resource-constrained systems
2. **Static** traditional approaches (e.g., static modeling) are **inaccurate**, since they cannot capture runtime information
3. **Code representation/embedding** is a **lightweight** approach that can capture **semantic** and **contextual** information of **code**.

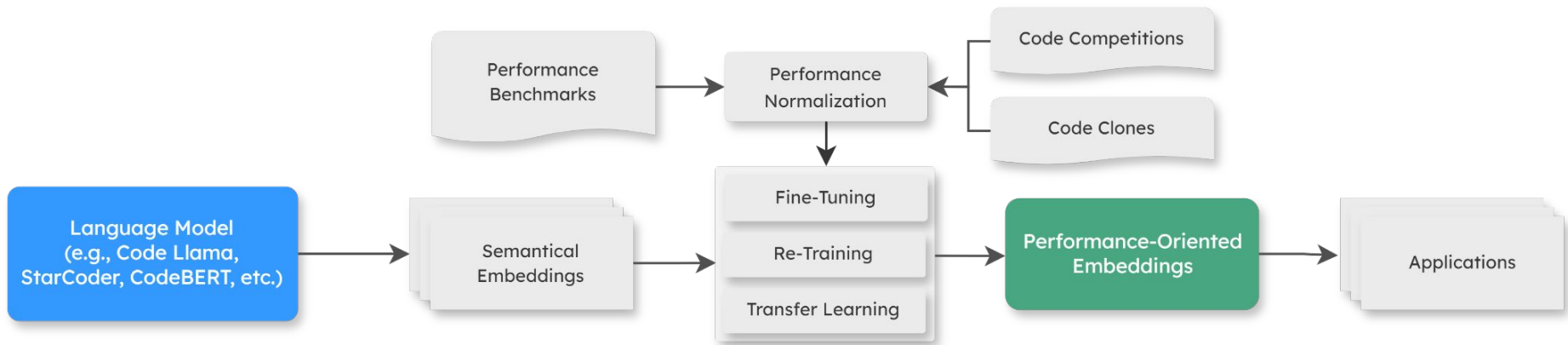
# Why should we use such code representation?

└─→ for performance aspects

1. **Dynamic** approaches (e.g., instrumentation/tracing) are **expensive**, especially in resource-constrained systems
2. **Static** traditional approaches (e.g., static modeling) are **inaccurate**, since they cannot capture runtime information
3. **Code representation/embedding** is a **lightweight** approach that can capture **semantic** and **contextual** information of **code**.

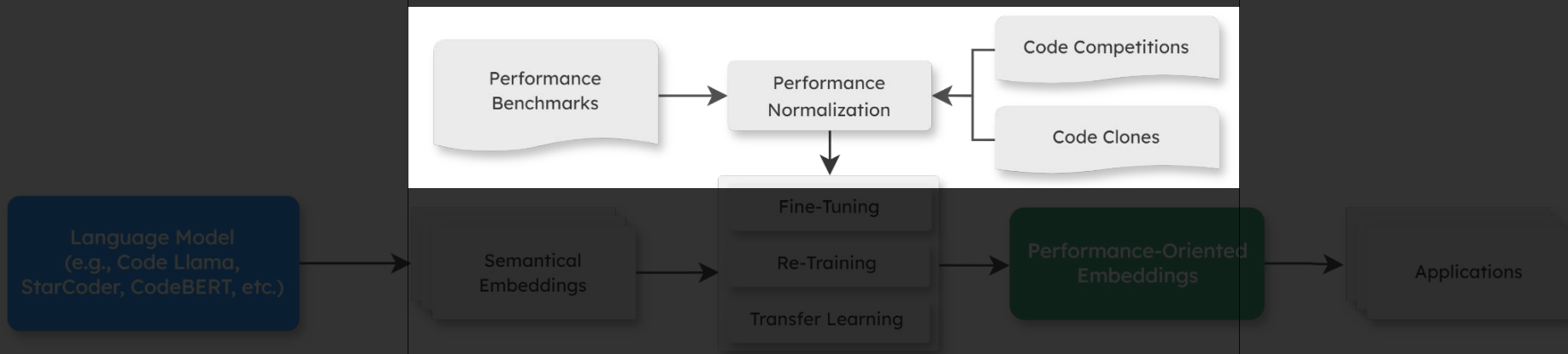
It is **NOT AWARE** of Performance Aspects of Code!

# How do we want to generate this awareness?





# How do we want to generate this awareness?



# Model training sources

## Performance Benchmarks

1. **Project repositories** (i.e., Java) that have **JMH benchmarks**
2. **Iterate** through **commit history**, and fetch **method changes**
3. **Run benchmarks** to assess **performance** in each commit
4. **List of function versions** and their **performance**

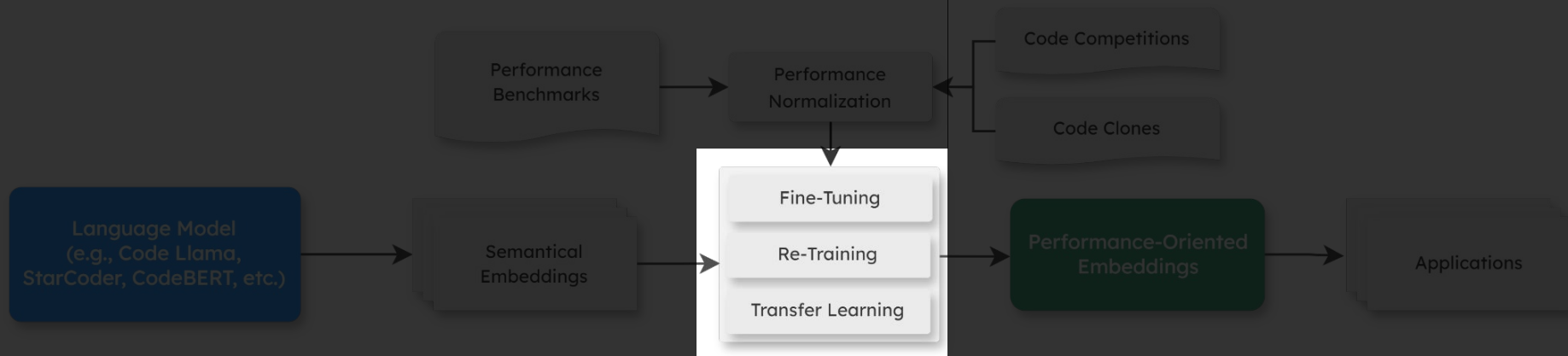
## Code Competitions

1. **Same** question, **different** codes, **same** functionality, **different** performance!
  - a. From **different users**
  - b. From **same users**
2. Evaluate each **pair of code's performance**
3. A **list of code pairs** that do the **same thing**, but with **different performance**!

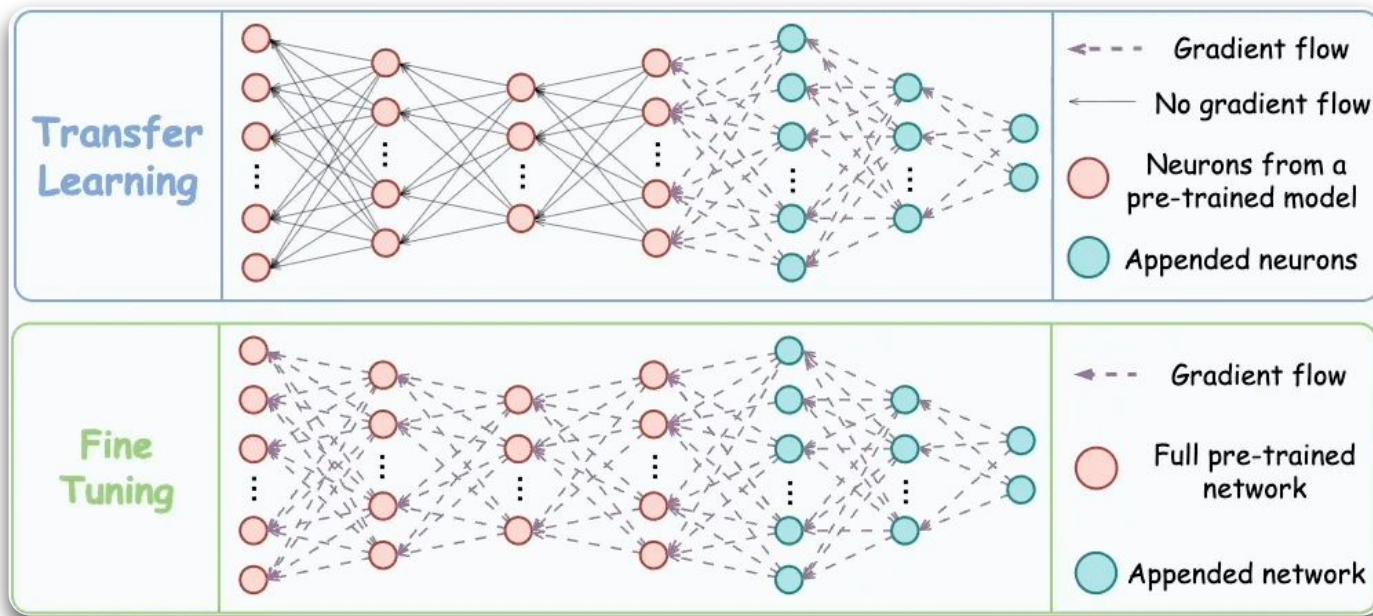
## Code Clones

1. **Datasets** of **codes** that do the **same thing**, but with **different implementation**
2. Hence, **same functionality**, **different performance**
3. Again, a **list of code pairs** that do the **same thing**, but with **different performance**!

# How do we want to generate this awareness?



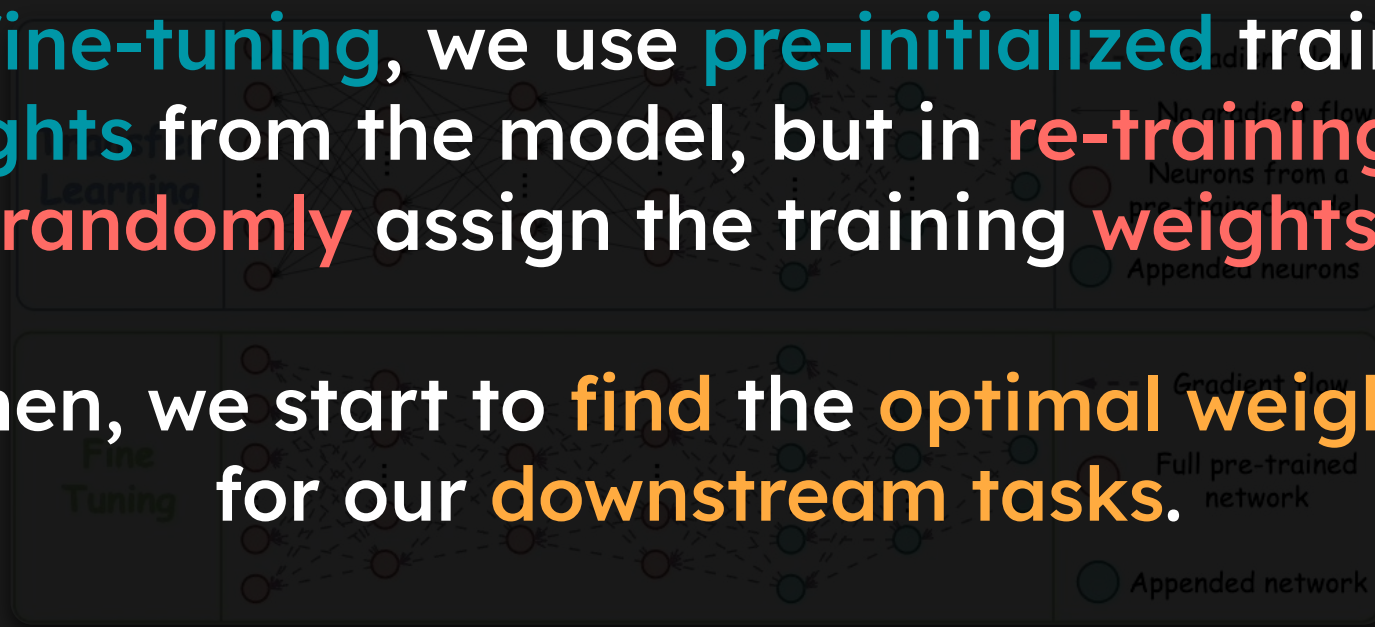
# So, what is the difference between them?



So, what is the difference between them?

In **fine-tuning**, we use **pre-initialized** training **weights** from the model, but in **re-training**, we **randomly** assign the training **weights**.

Then, we start to **find** the **optimal weights** for our **downstream tasks**.



# Progress until now?

1. **Mined** Java **projects** from **GitHub** that matched our **selection criteria**
2. **Fetches** their **functions' history** through the **commits**
3. **Implemented** a **lightweight Java instrumentation** agent
  - a. **Associate benchmarks** and their **target functions**
  - b. **Assess** the **performance** (i.e., execution time) of functions
4. **Run** the **benchmarks** for a sample of commits in each project (W.I.P)

## What's the usage of these embeddings?

Input Code 1

```
public int factorialForLoop(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
        result *= i;
    return result;
}
```

Input Code 2

```
public int factorialRecursive(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorialRecursive(n - 1);
}
```

Applying some  
**MAGIC**  
(a.k.a., embedding)

Output 1 as a Vector

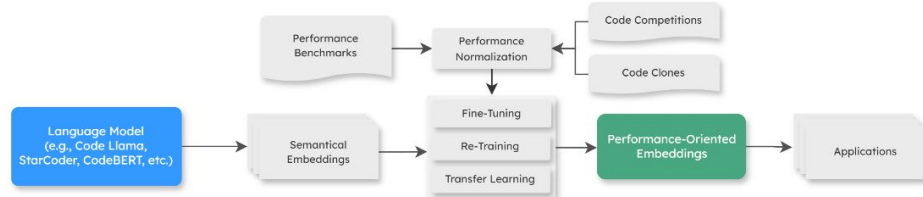
-0.004 0.872 0.887 ... 0.026

Output 2 as a Vector

0.001 0.800 0.802 ... 0.051

9

## How do we want to generate this awareness?



16

## Model training sources

### Performance Benchmarks

1. Project repositories (i.e., Java) that have JMH benchmarks
2. Iterate through commit history, and fetch method changes
3. Run benchmarks to assess performance in each commit
4. List of function versions and their performance

### Code Competitions

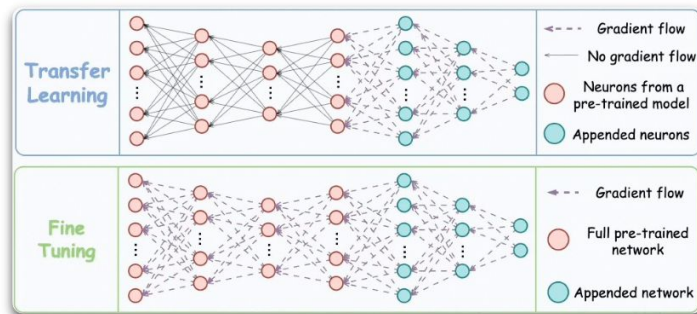
1. Same question, **different** codes, **same** functionality, **different** performance!
  - a. From **different users**
  - b. From **same users**
2. Evaluate each pair of code's performance
3. A list of code pairs that do the same thing, but with different performance!

### Code Clones

1. Datasets of codes that do the same thing, but with **different implementation**
2. Hence, **same functionality**, **different performance**
3. Again, a list of code pairs that do the same thing, but with **different performance**!

18

## So, what is the difference between them?



20

**Thanks!**