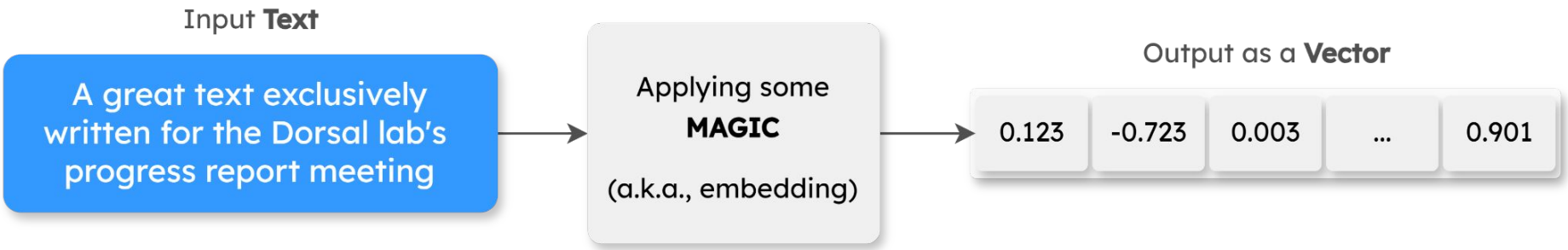
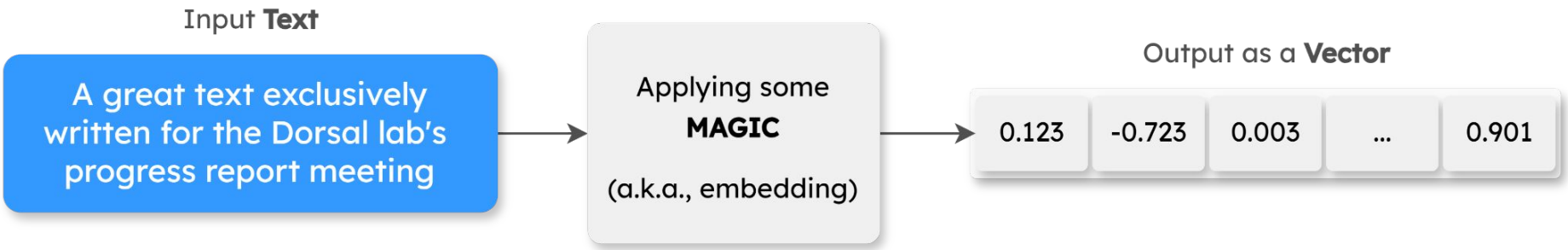
Performance-Oriented Code Representation

Kaveh Shahedi, Heng Li
Polytechnique Montréal
Summer 2024

What do we mean by “**Code Representation**”?

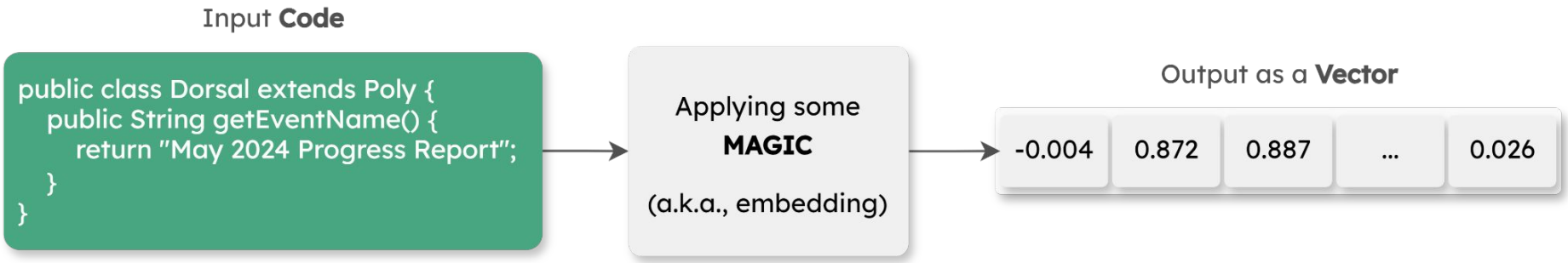


What do we mean by “**Code Representation**”?

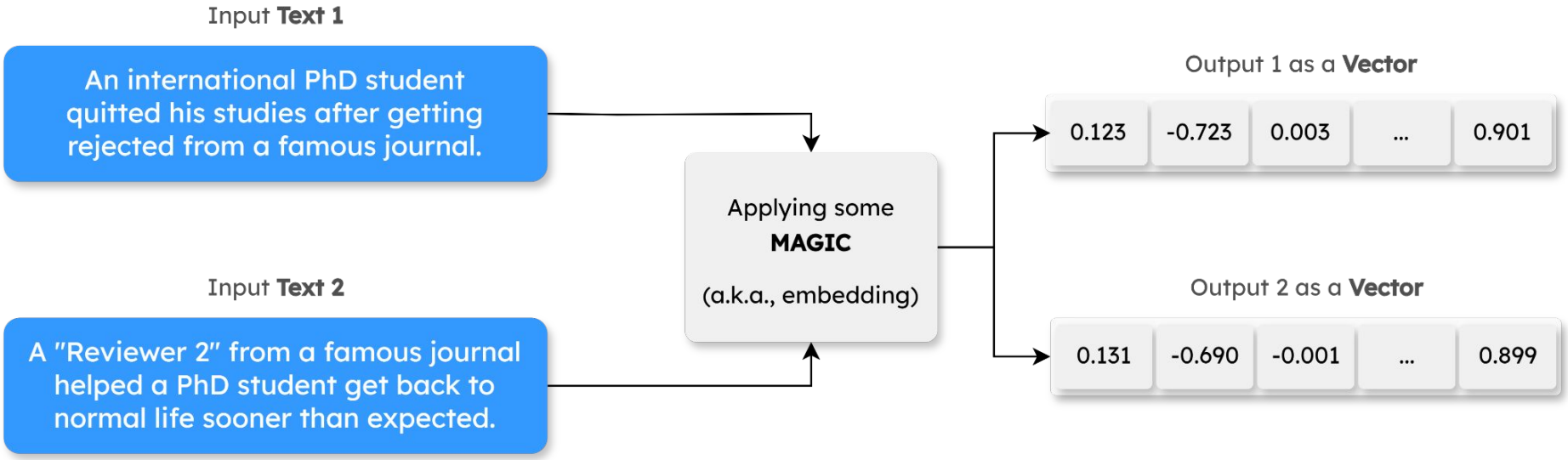


The **procedure** is the **same** for **code**!

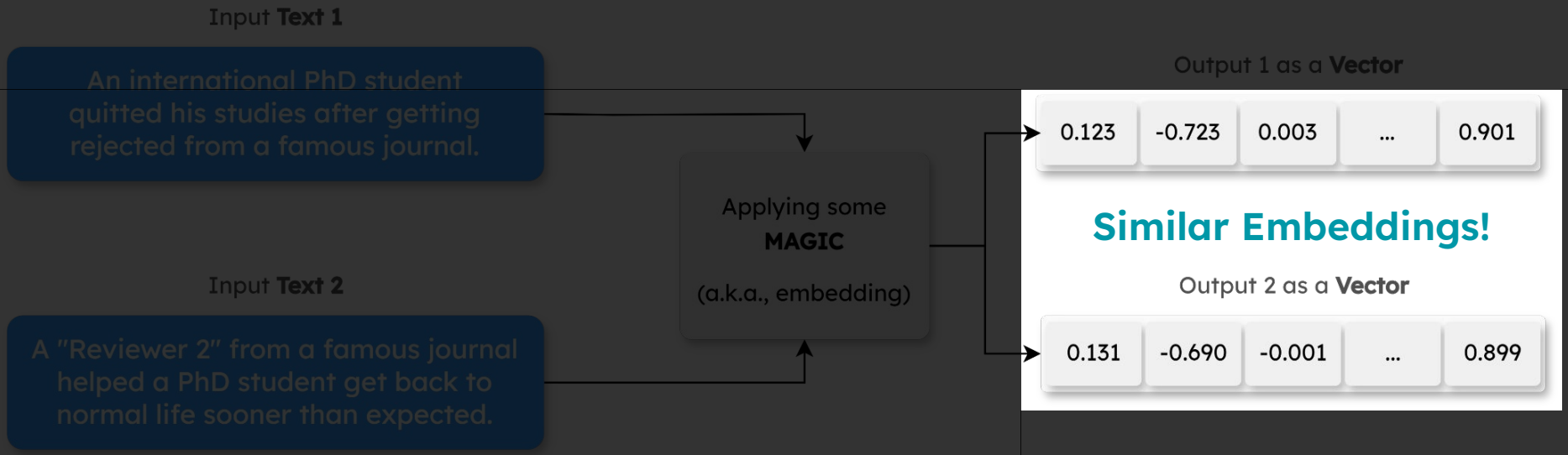
What do we mean by “**Code Representation**”?



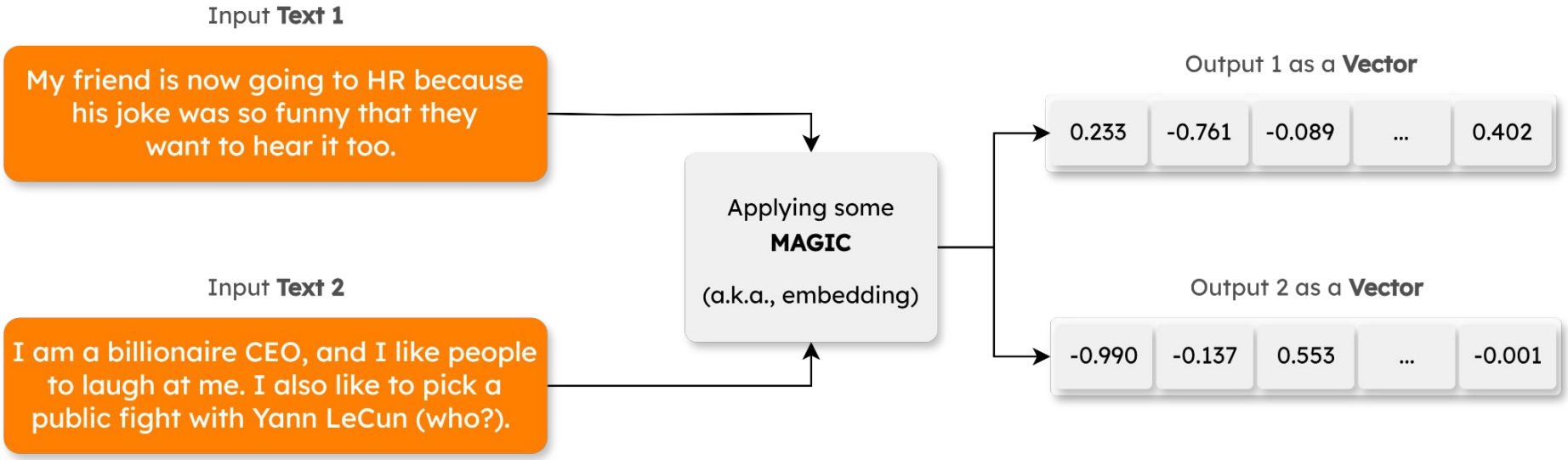
What's the usage of these **embeddings**?



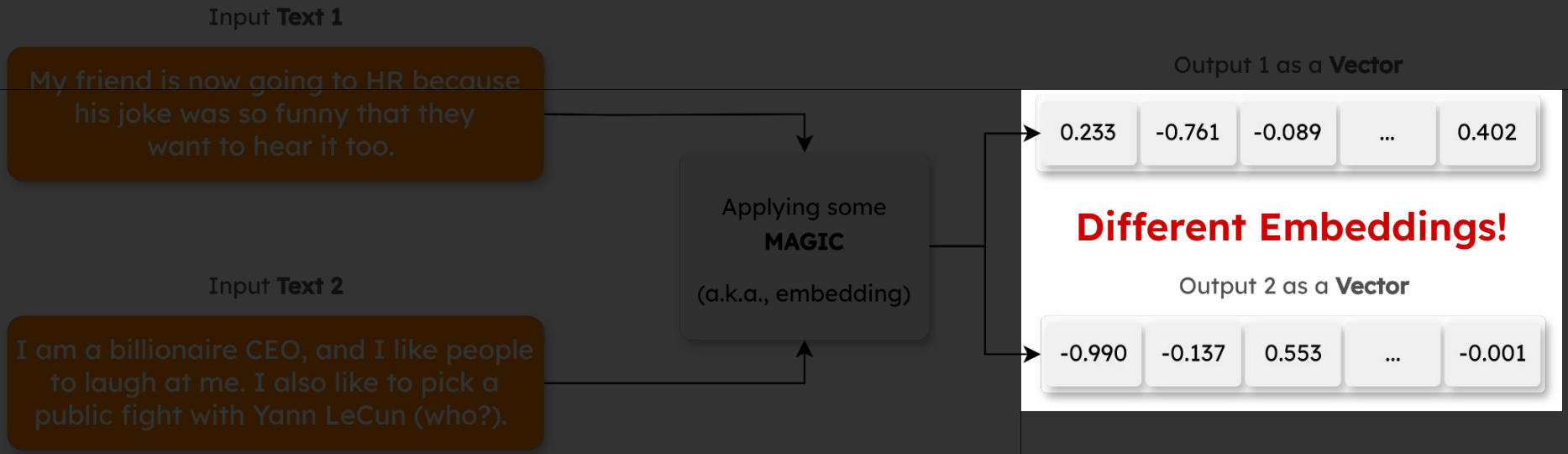
What's the usage of these **embeddings**?



What's the usage of these **embeddings**?



What's the usage of these **embeddings**?



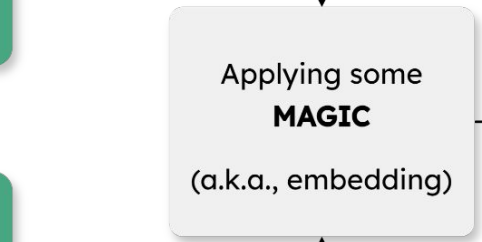
What's the usage of these **embeddings**?

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);
}
```



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**.

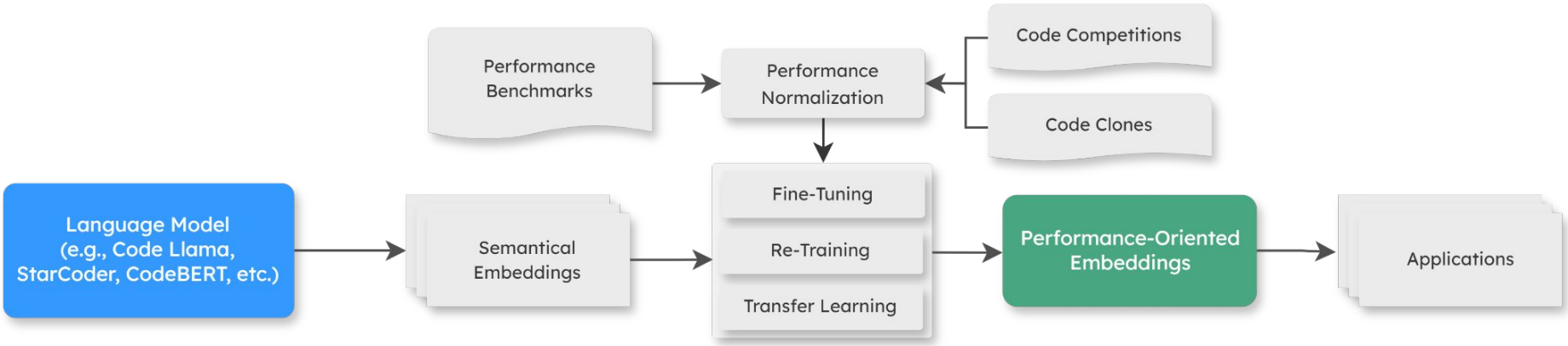
Why should we use such **code representation**?

→ for performance aspects

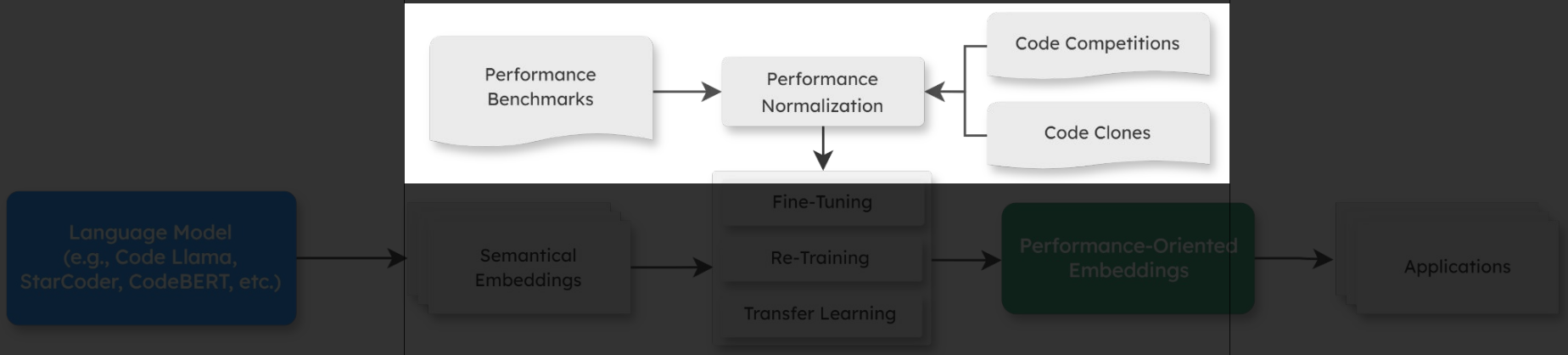
1. **Dynamic** approaches (e.g., instrumentation/tracing) are **expensive**, especially in resource-constrained systems
2. **Static** traditional approaches (e.g., static modeling) are **inaccurate**, since they cannot capture runtime information
3. **Code representation/embedding** is a **lightweight**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

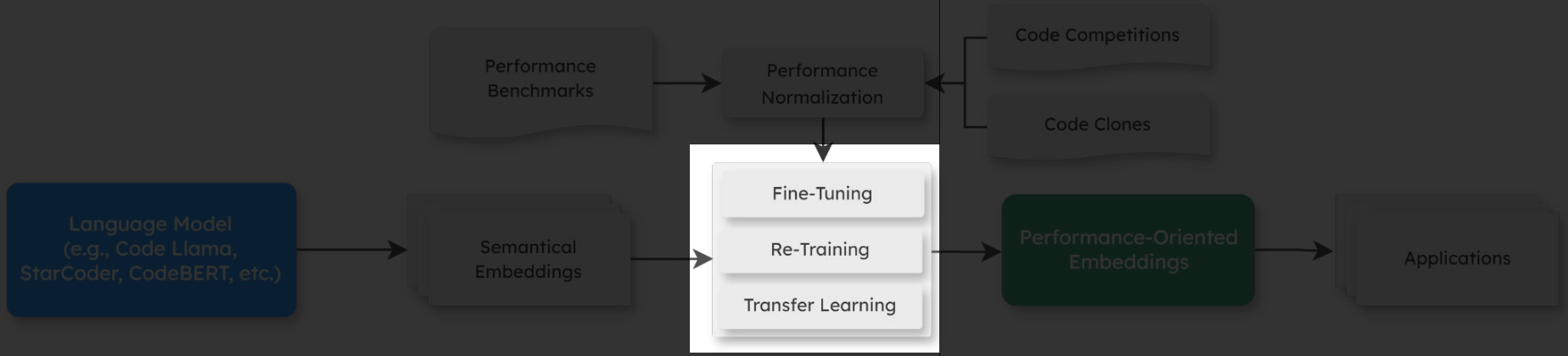
Code Competitions

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?

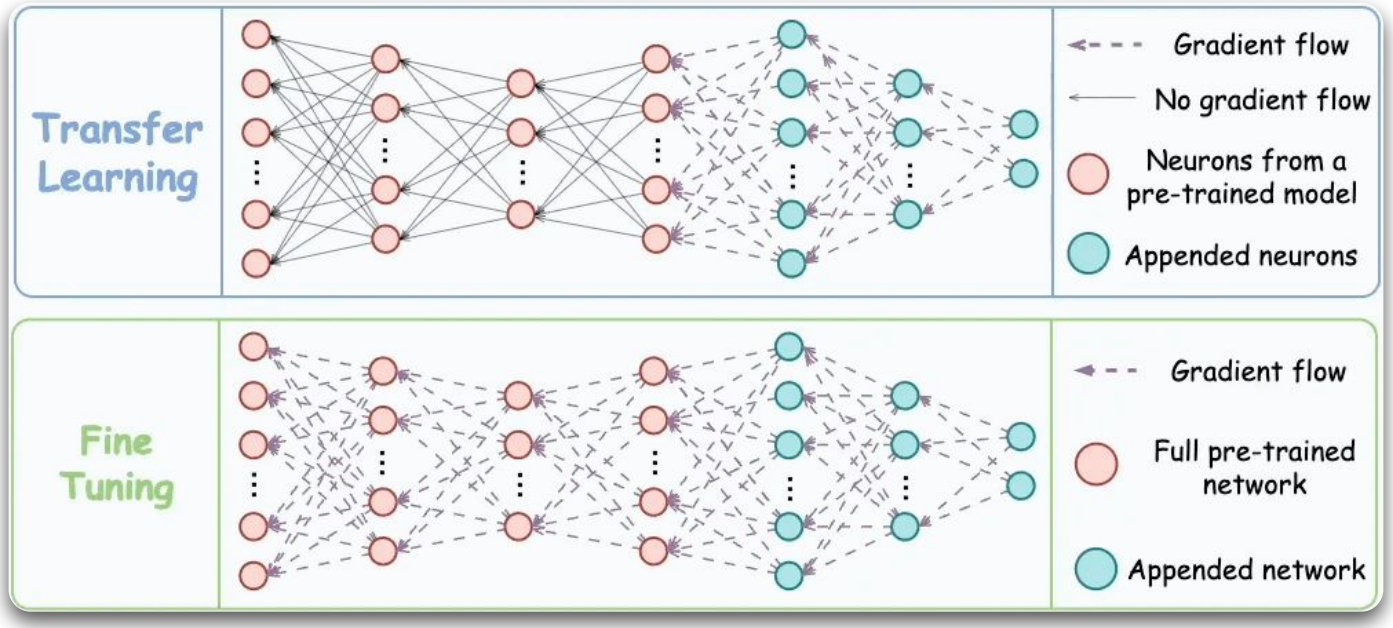


image source: https://www.blog.dailydoseofds.com/p/transfer-learning-vs-fine-tuning

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.

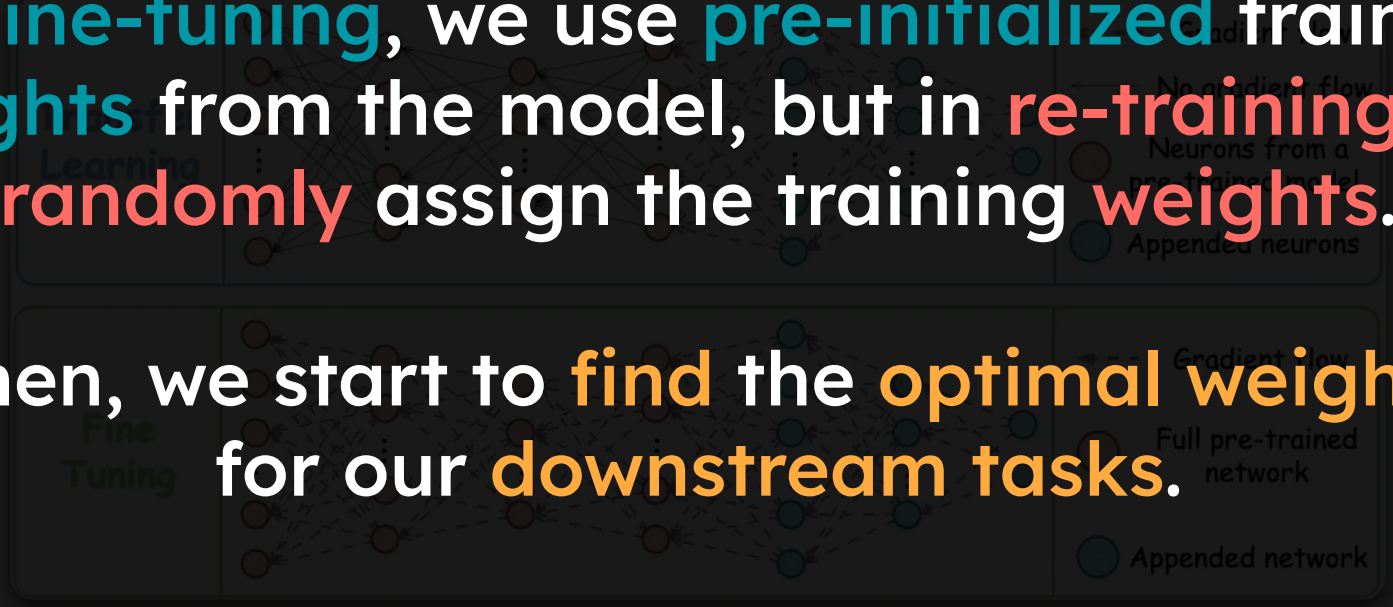


image source: https://www.blog.dailydoseofds.com/p/transfer-learning-vs-fine-tuning

Progress until now?

1. **Mined** Java **projects** from **GitHub** that matched our **selection criteria**
2. **Fetched** 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)

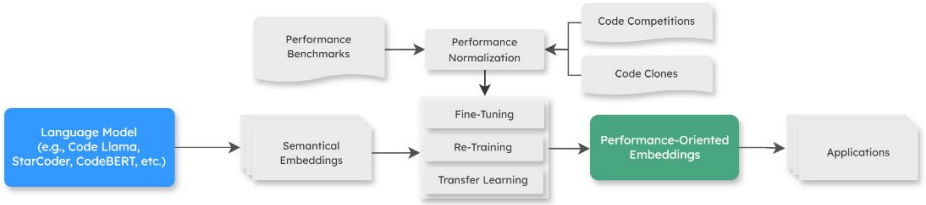
Output 1 as a **Vector**

-0.004	0.872	0.887	...	0.026

Output 2 as a **Vector**

0.001	0.800	0.802	...	0.051

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**!

So, what is the **difference** between them?

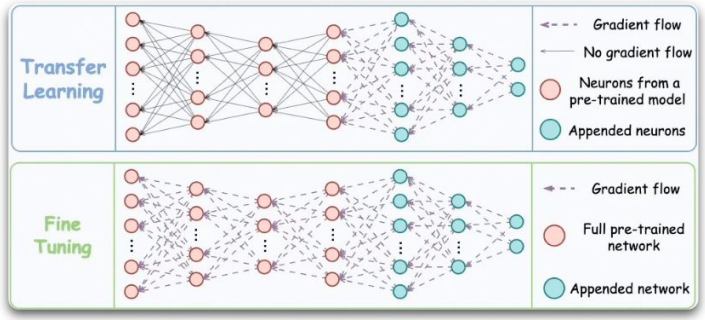


image source: https://www.blog.dailydoseofds.com/p/transfer-learning-vs-fine-tuning

Thanks!