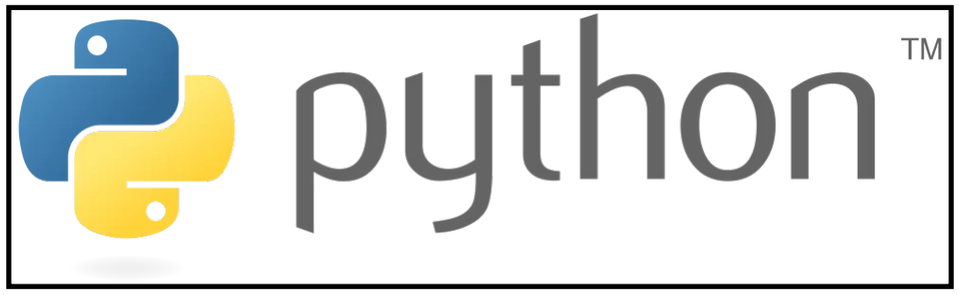
# TypeCare:
## Boosting Python Type Inference Models via Context-Aware Re-Ranking and Augmentation

Wonseok Oh, Hakjoo Oh
**Korea University**

ICSE'26 @ Rio De Janeiro, Brazil

# Type Annotation in Python




Dynamically Typed Language

# Type Annotation in Python


Dynamically Typed Language

Type Annotation

Code Readability

Bug Detection

...

# Type Annotation in Python


Dynamically Typed Language

Type Annotation

Code Readability

Bug Detection

...

**2024 Python Typing Survey Analysis**

Typing

91% of respondents saying they use types “Always” or “Often”

# Problem

## Type Annotation in Python

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11    def get(self, project_id: <FILL_IN>):
12        doc = self.doc.read(project_id)
13        return doc + 1
```

Goal:

Annotate Target Function

# Problem

## Type Annotation in Python

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11    def get(self, project_id: int):
12        doc = self.doc.read(project_id)
13        return doc + 1
```

Goal:

Annotate Target Function

# Problem

## Type Annotation in Python

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11     def get(self, project_id: int):
12         doc = self.doc.read(project_id)
13         return doc + 1
```

Goal:

Annotate Target Function

# SOTA Models

## Type Annotation in Python

- TypeGen: Generative Type Inference for Python (ICSE'23)

- TypeT5: Seq2seq Type Inference using Static Analysis (ICLR'23)

- TIGER: A Generating-Then-Ranking Framework for Practical Python Type Inference (ICSE'25)

**Learning-based Approaches**

# SOTA Models

## Type Annotation in Python

- TypeGen: Generative Type Inference for Python (ICSE'23)
- TypeT5: Seq2seq Type Inference using Static Analysis (ICLR'23)
- TIGER: A Generating-Then-Ranking Framework for Practical Python Type Inference (ICSE'25)

**Limitation:**

**They struggle to make accurate predictions for rare or unseen patterns (in the training data)**

# Observation

```python
class DocStore:
    def read(self, doc_id):
        if condition:
            return doc_id
        return -1

class Project:
    def __init__(self):
        self.doc = DocStore()

    def get(self, project_id: <FILL_IN>):
        doc = self.doc.read(project_id)
        return doc + 1
```

**TIGER (ICSE'25)**

**Model Output**

1. `str` ✗
2. `int` ✓
3. `Union[int, str]` ✗

# Observation

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11     def get(self, project_id: str):
12         doc = self.doc.read(project_id)
13         return doc + 1
```

TypeError: str + int

**TIGER (ICSE'25)**

**Model Output**

1. `str` ✗
2. `int` ✓
3. `Union[int, str]` ✗

# Observation

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11    def get(self, pr
12        doc = self.d
13        return doc +
```

**TIGER (ICSE'25)**

| Model Output | |
|---|---|
| 1. str | ✗ |
| 2. int | ✓ |
| 3. Union[int, str | ✗ |

Why does the model suddenly output "*str*" as the top-1?

# Observation

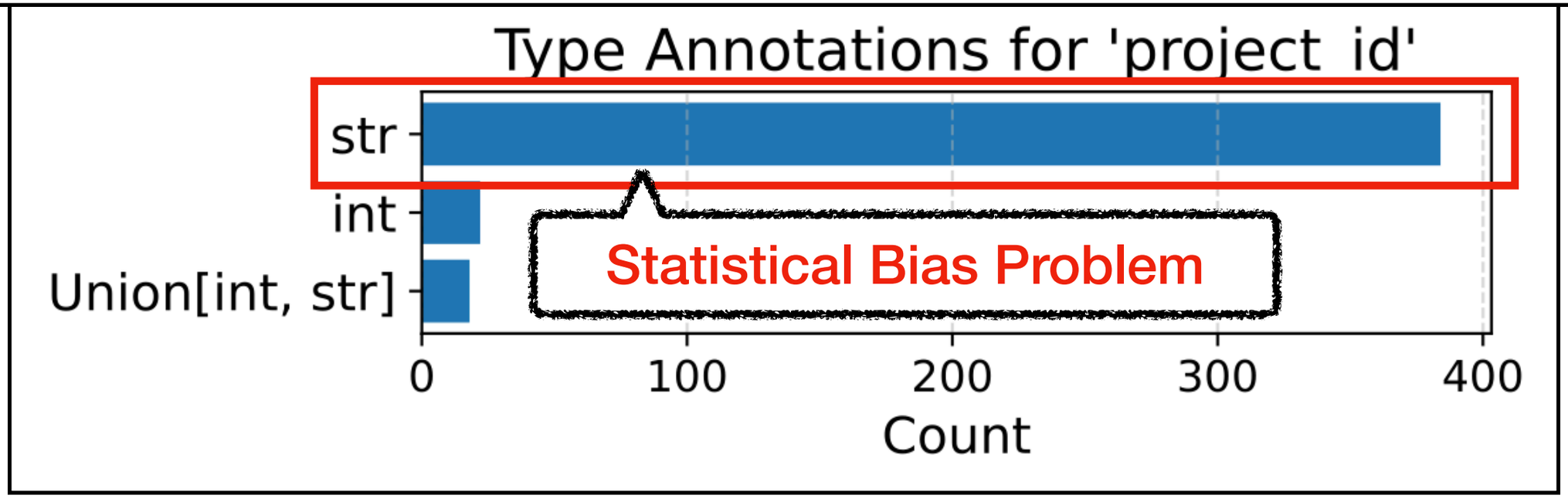
```python
class DocStore:
    def read(self, doc_id):
        if condition:
            return doc_id
        return -1

class Project:
    def __init__(self):
        self.doc = DocStore()
```

**TIGER (ICSE'25)**

**Model Output**

1. `str` ✗
2. `int` ✓
3. `Union[int, str]` ✗

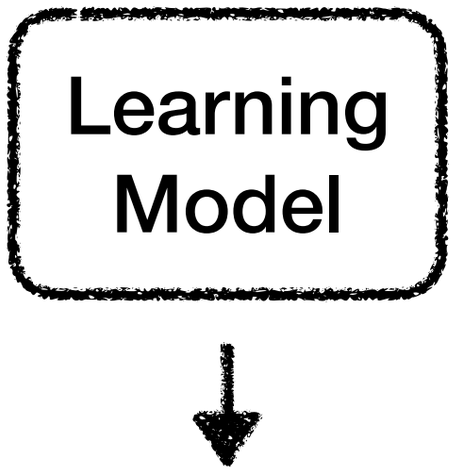
The number of annotations for *'project_id'* in the training dataset



# Our Work

## Type Annotation in Python

- To make the model more robust to rare or unseen data…



# Our Work

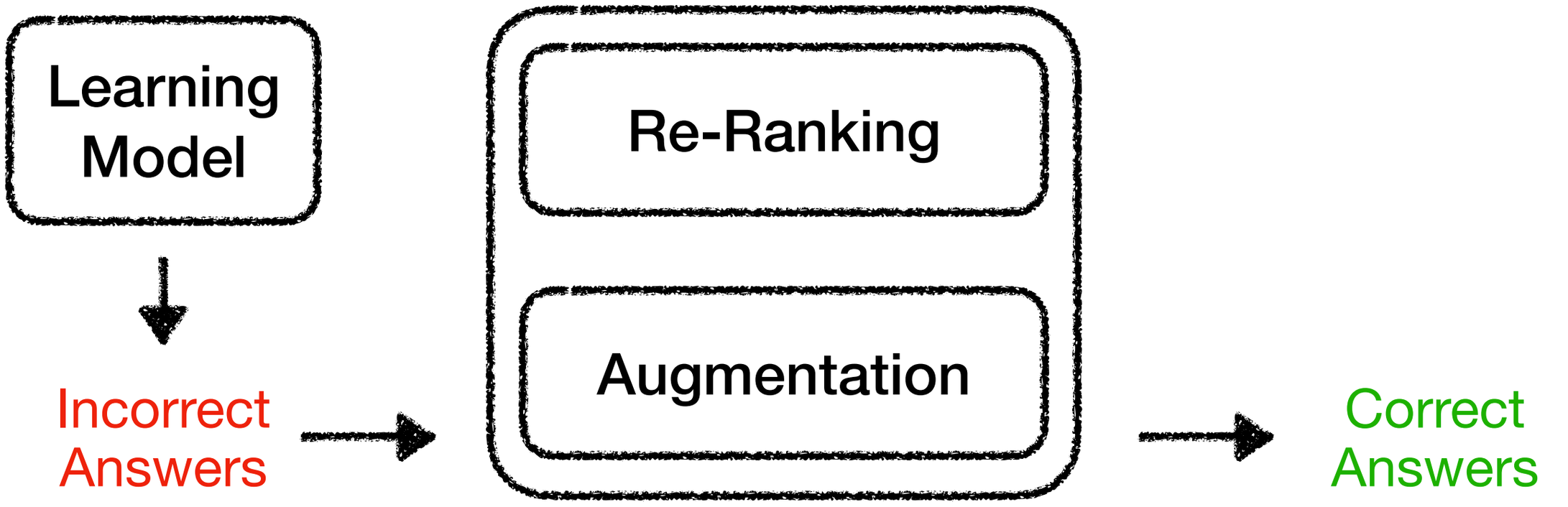
## Type Annotation in Python

- To make the model more robust to rare or unseen data…
- We made a post-processing method that refines the outputs of existing models

# Our Work

## Framework

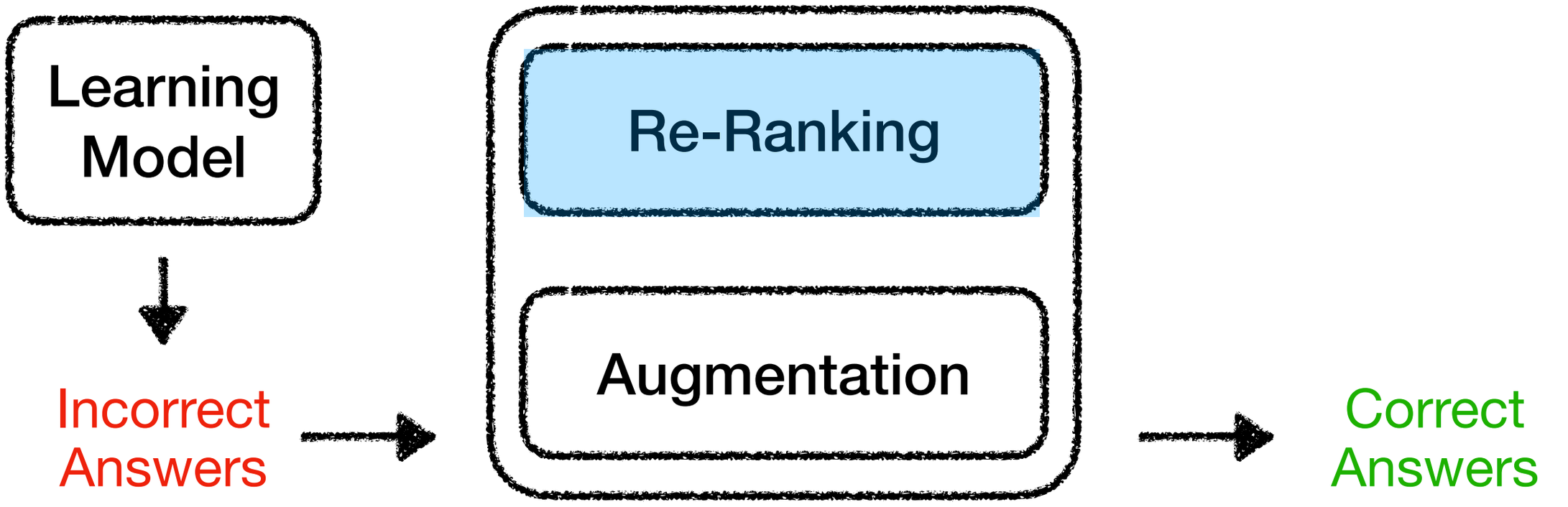
- Ours consists of two components:
  - Re-ranking Component
  - Augmentation Component



# Our Work

## Framework

- Ours consists of two components:
  - Re-ranking Component
  - Augmentation Component



# Re-ranking Component

## with Type Validity

```python
class DocStore:
    def read(self, doc_id):
        if condition:
            return doc_id
        return -1

class Project:
    def __init__(self):
        self.doc = DocStore()

    def get(self, project_id: <FILL_IN>):
        doc = self.doc.read(project_id)
        return doc + 1
```

### TIGER (ICSE'25)

**Model Output**

1. `str` ✗
2. `int` ✓
3. `Union[int, str]` ✗

# Re-ranking Component

## with Type Validity

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11    def get(self, project_id: <FILL_IN>):
12        doc = self.doc.read(project_id)
13        return doc + 1
```

TypeError: str + int

**TIGER (ICSE'25)**

| Model Output | |
|---|---|
| 1. str | ✗ |
| 2. int | ✓ |
| 3. Union[int, str] | ✗ |

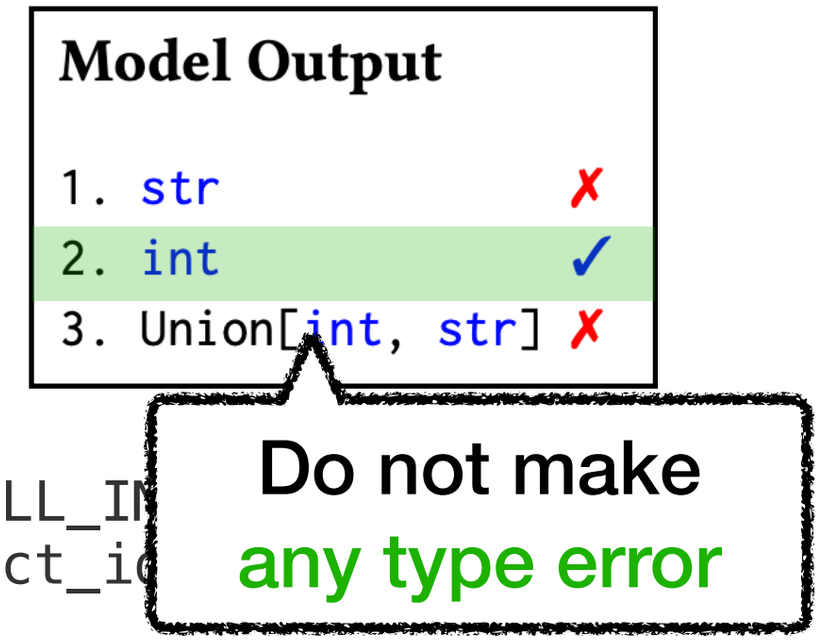
Make a potential type error

# Re-ranking Component

## with Type Validity

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11     def get(self, project_id: <FILL_IN
12         doc = self.doc.read(project_id
13         return doc + 1
```

**TIGER (ICSE'25)**

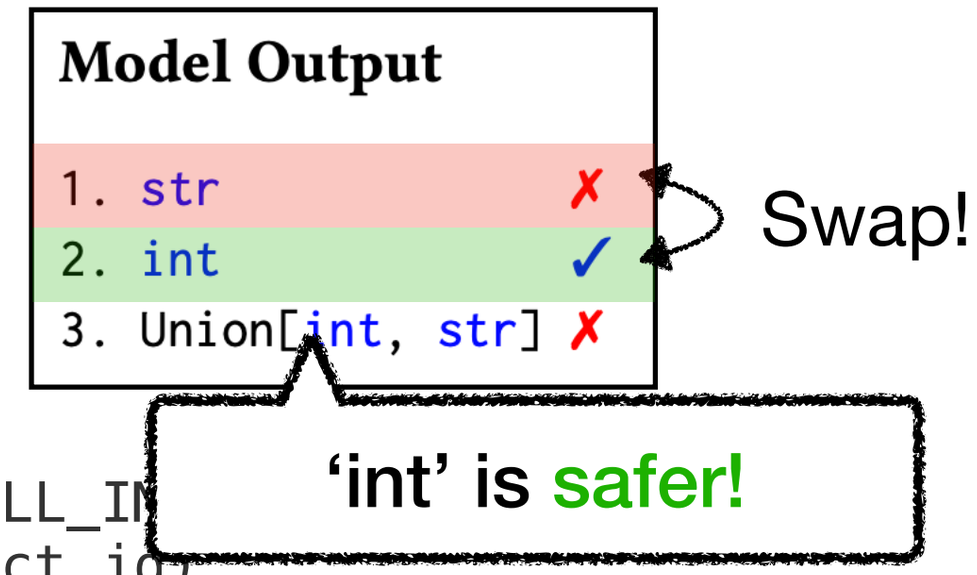


# Re-ranking Component

## with Type Validity

```
1 class DocStore:
2     def read(self, doc_id):
3         if condition:
4             return doc_id
5         return -1
6
7 class Project:
8     def __init__(self):
9         self.doc = DocStore()
10
11     def get(self, project_id: <FILL_IN
12         doc = self.doc.read(project_id)
13         return doc + 1
```

**TIGER (ICSE'25)**




Re-rank the outputs
with Type Validity (using Type Checker)

# Re-ranking Component

## with Code Usage Pattern

```
class A:
    text: Union[str, bytes]

    def ident(self, i_value: str):
        validate(i_value[1:])
        msg = self.text + i_value

    def data(self, d_value: bytes):
        d = d_value.decode('utf-8')
        return d

    def scope(self, s_value: <FILL_IN>):
        validate(s_value[1:])
        msg = self.text + s_value
```

### TIGER (ICSE'25)

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

# Re-ranking Component

## with Code Usage Pattern

```
1  class A:
2      text: Union[str, bytes]
3
4      def ident(self, i_value: str):
5          validate(i_value[1:])
6          msg = self.text + i_value
7
8      def data(self, d_value: bytes):
9          d = d_value.decode('utf-8')
10         return d
11
12     def scope(self, s_value: <FILL_IN>):
13         validate(s_value[1:])
14         msg = self.text + s_value
```

**TIGER (ICSE'25)**

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

Type checker flags
potential type errors
in all cases

# Re-ranking Component

## with Code Usage Pattern

```
1  class A:
2      text: Union[str, bytes]
3
4      def ident(self, i_value: str):
5          validate(i_value[1:])
6          msg = self.text + i_value
7
8      def data(self, d_value: bytes):
9          d = d_value.decode('utf-8')
10         return d
11
12     def scope(self, s_value: <FILL_IN>):
13         validate(s_value[1:])
14         msg = self.text + s_value
```

**TIGER (ICSE'25)**

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

Type checker flags
potential type errors
in all cases

"Type Correctness" alone is insufficient

# Re-ranking Component

## with Code Usage Pattern

```
class A:
    text: Union[str, bytes]

    def ident(self, i_value: str):
        validate(i_value[1:])
        msg = self.text + i_value

    def data(self, d_value: bytes):
        d = d_value.decode('utf-8')
        return d

    def scope(self, s_value: <FILL_IN>):
        validate(s_value[1:])
        msg = self.text + s_value
```

**TIGER (ICSE'25)**

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

# Re-ranking Component

## with Code Usage Pattern

```
1  class A:
2      text: Union[str, bytes]
3
4      def ident(self, i_value: str):
5          validate(i_value[1:])
6          msg = self.text + i_value
7
8      def data(self, d_value: bytes):
9          d = d_value.decode('utf-8')
10         return d
11
12     def scope(self, s_value: <FILL_IN>):
13         validate(s_value[1:])
14         msg = self.text + s_value
```

**TIGER (ICSE'25)**

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

Used in a **highly similar** **context**

# Re-ranking Component

## with Code Usage Pattern

```
1  class A:
2      text: Union[str, bytes]
3
4      def ident(self, i_value: str):
5          validate(i_value[1:])
6          msg = self.text + i_value
7
8      def data(self, d_value: bytes):
9          d = d_value.decode('utf-8')
10         return d
11
12     def scope(self, s_value: <FILL_IN>):
13         validate(s_value[1:])
14         msg = self.text + s_value
```

**TIGER (ICSE'25)**

**Model Output**

1. s_value: bytes ✗
2. s_value: int ✗
3. s_value: str ✓

Used in a **entirely distinct** **contexts**

# Re-ranking Component

## with Code Usage Pattern

```
1  class A:
2      text: Union[str, bytes]
3
4      def ident(self, i_value: str):
5          validate(i_value[1:])
6          msg = self.text + i_value
7
8                                  : bytes):
9                                  ('utf-8')
10         return m
11
12     def scope(self, s_value: <FILL_IN>):
13         validate(s_value[1:])
14         msg = self.text + s_value
```

Highly likely to be str

**TIGER (ICSE'25)**

| Model Output | |
|---|---|
| 1. s_value: bytes | ✗ |
| 2. s_value: int | ✗ |
| 3. s_value: str | ✓ |

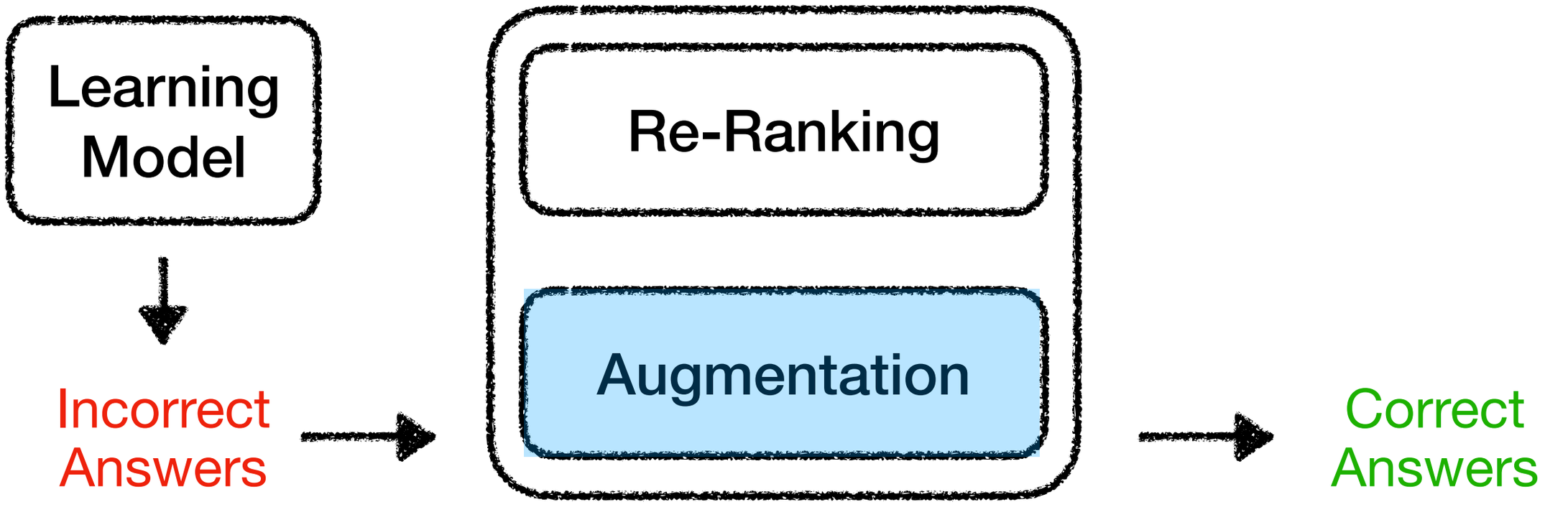
Swap!

Re-rank the outputs
with Code Usage Patterns (in the project)

# Our Work

## Framework

- Ours consists of two components:
  - Re-ranking Component
  - Augmentation Component



# Augmentation Component

## for uncertain variable

```
1 def make_identifier(
2     session: <FILL_IN>,
3     name: <FILL_IN>
4 ):
5     with session.cursor() as c:
6         c.execute(...)
7
8 def fetch_identifier(
9     d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

**TypeT5 (ICLR'23)**

**Model Output**

1. session: Session ✗
   name: str ✓

2. session: dict ✗
   name: str ✓

3. session: list ✗
   name: str ✓

**Answer**
session: DatabaseSession
name: str

# Augmentation Component

## for uncertain variable

```
1  def make_identifier(
2      session: <FILL_IN>,
3      name: <FILL_IN>
4  ):
5      with session.cursor() as c:
6          c.execute(...)
7
8  def fetch_identifier(
9      d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

**TypeT5 (ICLR'23)**

**Model Output**

1. session: Session ✗
   name: str ✓
2. session: dict ✗
   name: str ✓
3. session: list ✗
   name: str ✓

High confidence for *name*
Low confidence for *session*

# Augmentation Component

## for uncertain variable

```
1 def make_identifier(
2     session: <FILL_IN>,
3     name: <FILL_IN>
4 ):
5     with session.cursor() as c:
6         c.execute(...)
7
8 def fetch_identifier(
9     d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

**TypeT5 (ICLR'23)**

**Model Output**

1. session: Session ✗
   name: str ✓

2. session: dict ✗
   name: str ✓

3. session: list ✗
   name: str ✓

Augmenting predictions for low confidence

High confidence for *name*
Low confidence for *session*

# Augmentation Component

## for uncertain variable

```
1  def make_identifier(
2      session: <FILL_IN>,
3      name: <FILL_IN>
4  ):
5      with session.cursor() as c:
6          c.execute(...)
7
8  def fetch_identifier(
9      d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

### TypeT5 (ICLR'23)

**Model Output**

| | | |
|---|---|---|
| 1. | session: Session | ✗ |
| | name: str | ✓ |
| 2. | session: dict | ✗ |
| | name: str | ✓ |
| 3. | session: list | ✗ |
| | name: str | ✓ |

Used in a **highly similar** **context**

# Augmentation Component

## for uncertain variable

```
1  def make_identifier(
2      session: <FILL_IN>,
3      name: <FILL_IN>
4  ):
5      with session.cursor() as c:
6          c.execute(...)
7
8  def fetch_identifier(
9      d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

Used in a **highly similar** **context**

**TypeT5 (ICLR'23)**

**Model Output**

1. session: Session ✗
   name: str ✓

2. session: dict ✗
   name: str ✓

3. session: list ✗
   name: str ✓

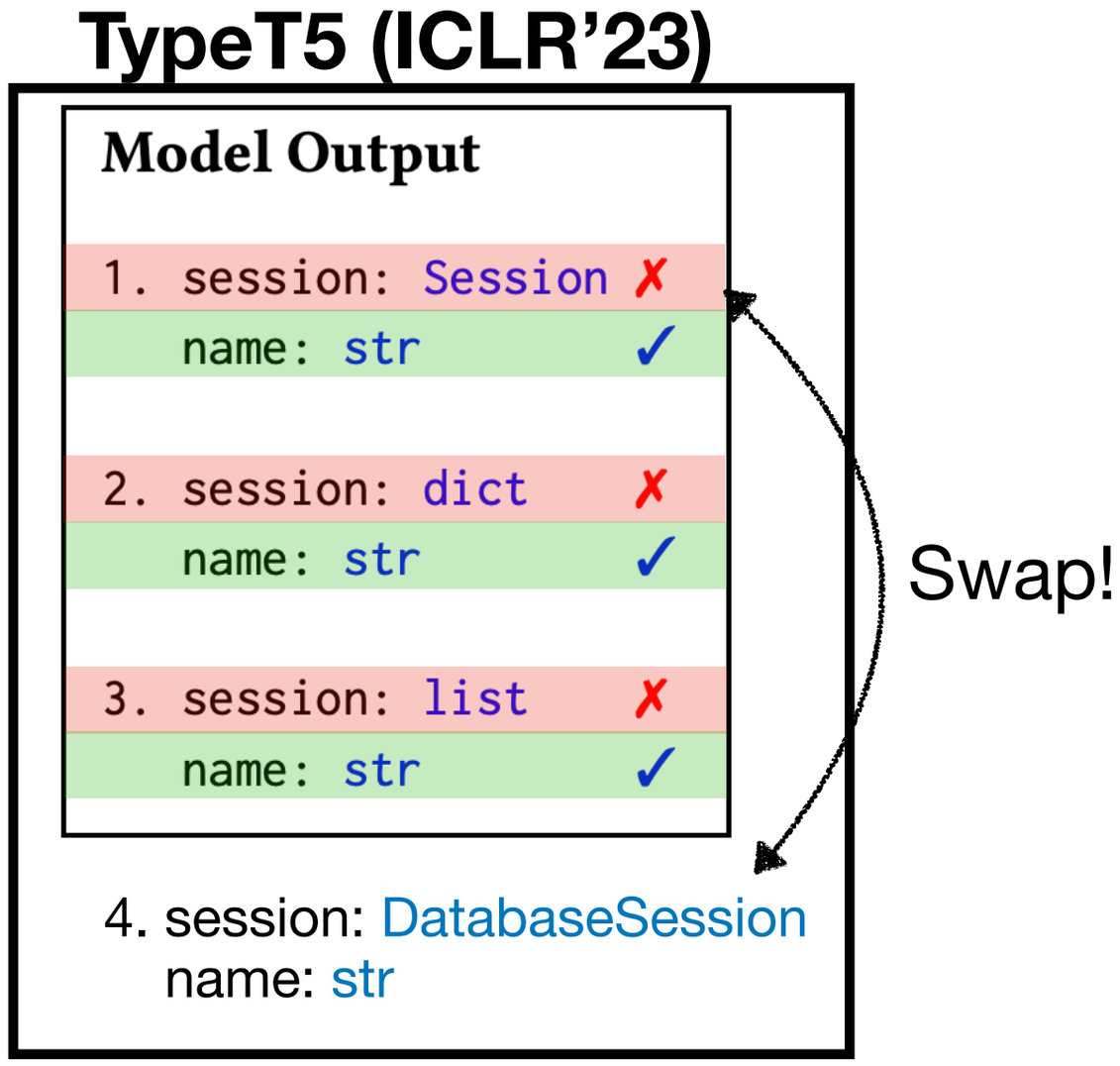
4. session: DatabaseSession
   name: str

**Augment** new answer

# Augmentation Component

## for uncertain variable

```
1 def make_identifier(
2     session: <FILL_IN>,
3     name: <FILL_IN>
4 ):
5     with session.cursor() as c:
6         c.execute(...)
7
8 def fetch_identifier(
9     d_session: DatabaseSession
10 ):
11     with d_session.cursor() as c:
12         c.execute(...)
```

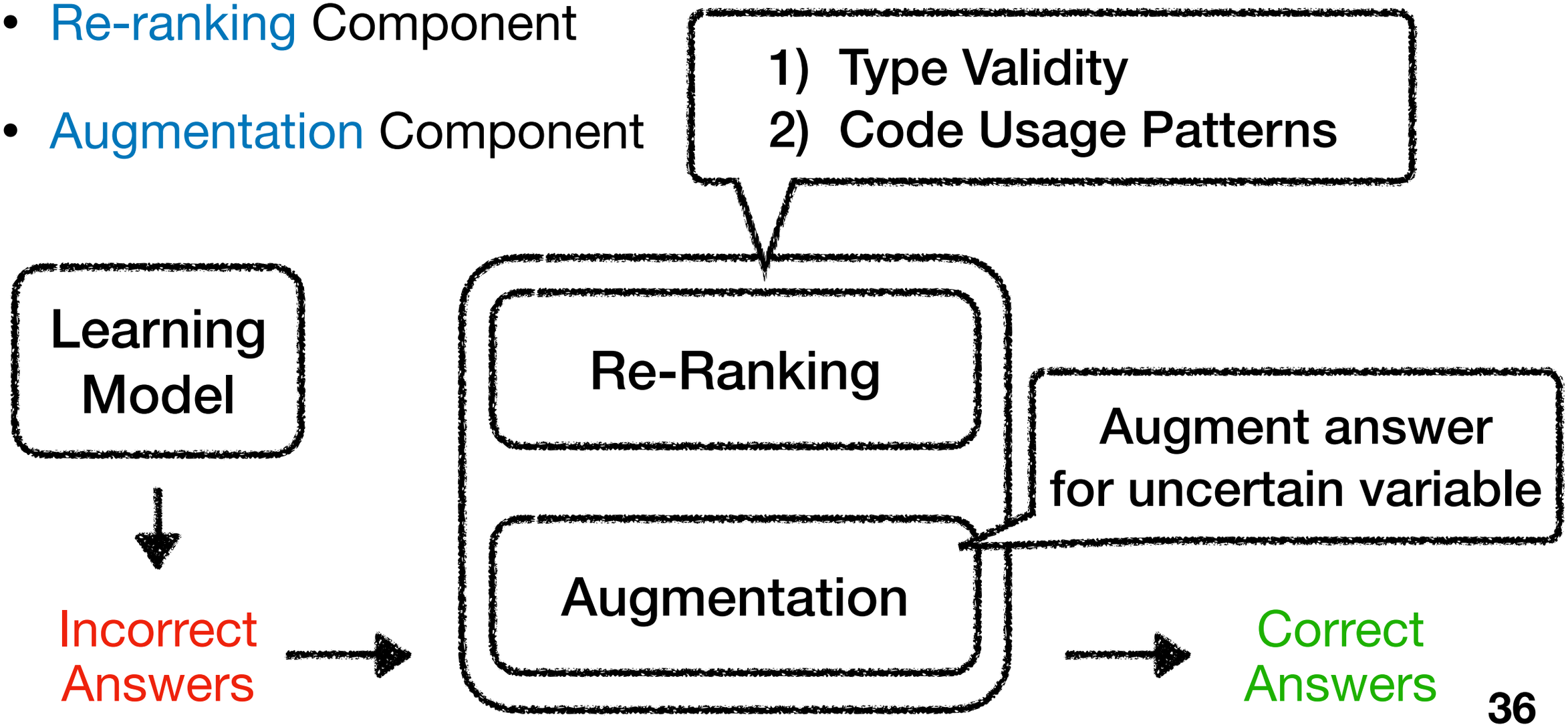



and then Re-rank them!

# Our Work

## Framework

- Ours consists of two components:
  - Re-ranking Component
  - Augmentation Component



# Evaluation

- **Setup:** Integrated our post-processing with SOTA models
  - Applied post-processing to the top-10 candidates

- **Dataset:** 10,928 parameters types and 6,983 return types

- **Metric:** String matching on normalized types

e.g., Optional[int] => Union[None, int]

# Effectiveness

- Type inference accuracy

| Dataset | Model | Exact Match | | | Base Match | | |
|---|---|---|---|---|---|---|---|
| | | Top 1 (%) | Top 3 (%) | Top 5 (%) | Top 1 (%) | Top 3 (%) | Top 5 (%) |
| Better Types 4Py | CodeT5$_B$ | 65.5% | 70.4% | 72.5% | 73.7% | 77.8% | 79.4% |
| | TypeT5 | 71.4% | 77.2% | 78.9% | 78.1% | 83.7% | 85.4% |
| | +TypeCare | (+13.4%) 81.0% | (+7.3%) 82.8% | (+5.8%) 83.5% | (+9.1%) 85.2% | (+4.2%) 87.2% | (+3.0%) 88.0% |
| Many Types 4Py | CodeT5$_M$ | 62.9% | 71.7% | 73.8% | 72.0% | 79.6% | 81.1% |
| | Tiger | 67.8% | 78.0% | 80.2% | 75.8% | 85.5% | 88.0% |
| | +TypeCare | (+15.2%) 78.1% | (+7.3%) 83.7% | (+6.0%) 85.0% | (+10.0%) 83.4% | (+4.9%) 89.7% | (+3.7%) 91.3% |
| | - | - | - | - | - | - | - |
| | TypeGen | 65.4% | 73.4% | 75.0% | 71.6% | 79.9% | 81.6% |
| | +TypeCare | (+12.5%) 73.6% | (+7.6%) 79.0% | (+6.8%) 80.1% | (+9.9%) 78.7% | (+5.4%) 84.2% | (+4.9%) 85.6% |

# Effectiveness

- Type infere

Performance Gains via SOTA methodologies

| Dataset | Model | Base Match | | | Base Match | | |
|---|---|---|---|---|---|---|---|
| | | Top 1 (%) | Top 3 (%) | Top 5 (%) | Top 1 (%) | Top 3 (%) | Top 5 (%) |
| Better Types 4Py | CodeT5$_B$ | 65.5% | 70.4% | 72.5% | 73.7% | 77.8% | 79.4% |
| | TypeT5 | 71.4% | 77.2% | 78.9% | 78.1% | 83.7% | 85.4% |
| | +TypeCare | (+13.4%) 81.0% | (+7.3%) 82.8% | (+5.8%) 83.5% | (+9.1%) 85.2% | (+4.2%) 87.2% | (+3.0%) 88.0% |
| Many Types 4Py | CodeT5$_M$ | 62.9% | 71.7% | 73.8% | 72.0% | 79.6% | 81.1% |
| | Tiger | 67.8% | 78.0% | 80.2% | 75.8% | 85.5% | 88.0% |
| | +TypeCare | (+15.2%) 78.1% | (+7.3%) 83.7% | (+6.0%) 85.0% | (+10.0%) 83.4% | (+4.9%) 89.7% | (+3.7%) 91.3% |
| | - | - | - | - | - | - | - |
| | TypeGen | 65.4% | 73.4% | 75.0% | 71.6% | 79.9% | 81.6% |
| | +TypeCare | (+12.5%) 73.6% | (+7.6%) 79.0% | (+6.8%) 80.1% | (+9.9%) 78.7% | (+5.4%) 84.2% | (+4.9%) 85.6% |

# Effectiveness

- Type inference accuracy

Performance Gains via our techniques

| Dataset | Model | | | | Base Match | | |
|---|---|---|---|---|---|---|---|
| | | | | | Top 1 (%) | Top 3 (%) | Top 5 (%) |
| Better Types 4Py | CodeT5$_B$ | 65.5% | 70.4% | 72.5% | 73.7% | 77.8% | 79.4% |
| | TypeT5 | 71.4% | 77.2% | 78.9% | 78.1% | 83.7% | 85.4% |
| | +TypeCare | (+13.4%) 81.0% | (+7.3%) 82.8% | (+5.8%) 83.5% | (+9.1%) 85.2% | (+4.2%) 87.2% | (+3.0%) 88.0% |
| Many Types 4Py | CodeT5$_M$ | 62.9% | 71.7% | 73.8% | 72.0% | 79.6% | 81.1% |
| | Tiger | 67.8% | 78.0% | 80.2% | 75.8% | 85.5% | 88.0% |
| | +TypeCare | (+15.2%) 78.1% | (+7.3%) 83.7% | (+6.0%) 85.0% | (+10.0%) 83.4% | (+4.9%) 89.7% | (+3.7%) 91.3% |
| | - | - | - | - | - | - | - |
| | TypeGen | 65.4% | 73.4% | 75.0% | 71.6% | 79.9% | 81.6% |
| | +TypeCare | (+12.5%) 73.6% | (+7.6%) 79.0% | (+6.8%) 80.1% | (+9.9%) 78.7% | (+5.4%) 84.2% | (+4.9%) 85.6% |

# Effectiveness

- Type inference accuracy

Accept *List* for *List[…]*

| Dataset | Model | Exact Match | | | Base Match | | |
|---|---|---|---|---|---|---|---|
| | | Top 1 (%) | Top 3 (%) | Top 5 (%) | Top 1 (%) | Top 3 (%) | Top 5 (%) |
| Better Types 4Py | CodeT5$_B$ | 65.5% | 70.4% | 72.5% | 73.7% | 77.8% | 79.4% |
| | TypeT5 | 71.4% | 77.2% | 78.9% | 78.1% | 83.7% | 85.4% |
| | +TypeCare | (+13.4%) 81.0% | (+7.3%) 82.8% | (+5.8%) 83.5% | (+9.1%) 85.2% | (+4.2%) 87.2% | (+3.0%) 88.0% |
| Many Types 4Py | CodeT5$_M$ | 62.9% | 71.7% | 73.8% | 72.0% | 79.6% | 81.1% |
| | Tiger | 67.8% | 78.0% | 80.2% | 75.8% | 85.5% | 88.0% |
| | +TypeCare | (+15.2%) 78.1% | (+7.3%) 83.7% | (+6.0%) 85.0% | (+10.0%) 83.4% | (+4.9%) 89.7% | (+3.7%) 91.3% |
| | - | - | - | - | - | - | - |
| | TypeGen | 65.4% | 73.4% | 75.0% | 71.6% | 79.9% | 81.6% |
| | +TypeCare | (+12.5%) 73.6% | (+7.6%) 79.0% | (+6.8%) 80.1% | (+9.9%) 78.7% | (+5.4%) 84.2% | (+4.9%) 85.6% |

Achieved significant performance gains, particularly in Top-1 accuracy

# Effectiveness

- Type inference accuracy across different type categories

int, str

List[Dict[int, str]

MyClass

| Model | Element (#: 1405) | | | Parameterized (#: 483) | | | User-defined (#: 799) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Top-1 | Top-3 | Top-5 | Top-1 | Top-3 | Top-5 | Top-1 | Top-3 | Top-5 |
| TIGER | 88.2% | 94.2% | 94.5% | 29.2% | 41.0% | 41.8% | 55.3% | 71.8% | 78.1% |
| +TypeCare | 92.0% | 95.2% | 95.9% | 37.7% | 46.4% | 48.9% | 78.1% | 86.1% | 87.6% |
| **Improve** | **+4.3%** | **+1.1%** | **+1.5%** | **+29.1%** | **+13.2%** | **+17.0%** | **+41.2%** | **+19.9%** | **+12.2%** |
| TYPEGEN | 84.3% | 90.0% | 90.7% | 30.2% | 42.9% | 45.8% | 53.4% | 62.8% | 65.2% |
| +TypeCare | 89.0% | 92.4% | 93.0% | 37.5% | 50.9% | 54.0% | 68.2% | 72.3% | 73.2% |
| **Improve** | **+5.6%** | **+2.7%** | **+2.5%** | **+24.2%** | **+18.6%** | **+17.9%** | **+27.7%** | **+15.1%** | **+12.3%** |

# Effectiveness

- Type inference accuracy across different type categories

int, str

List[Dict[int, str]

MyClass

| Model | Element (#: 1405) | | | Parameterized (#: 483) | | | User-defined (#: 799) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Top-1 | Top-3 | Top-5 | Top-1 | Top-3 | Top-5 | Top-1 | Top-3 | Top-5 |
| TIGER | 88.2% | 94.2% | 94.5% | 29.2% | 41.0% | 41.8% | 55.3% | 71.8% | 78.1% |
| +TypeCare | 92.0% | 95.2% | 95.9% | 37.7% | 46.4% | 48.9% | 78.1% | 86.1% | 87.6% |
| **Improve** | **+4.3%** | **+1.1%** | **+1.5%** | **+29.1%** | **+13.2%** | **+17.0%** | **+41.2%** | **+19.9%** | **+12.2%** |
| TYPEGEN | 84.3% | 90.0% | 90.7% | 30.2% | 42.9% | 45.8% | 53.4% | 62.8% | 65.2% |
| +TypeCare | 89.0% | 92.4% | 93.0% | 37.5% | 50.9% | 54.0% | 68.2% | 72.3% | 73.2% |
| **Improve** | **+5.6%** | **+2.7%** | **+2.5%** | **+24.2%** | **+18.6%** | **+17.9%** | **+27.7%** | **+15.1%** | **+12.3%** |

Substantial performance gains on rare and unseen types

# Summary

- We proposed **a model-agnostic post-processing technique** to enhance model robustness
  - Re-ranking with Type Correctness & Code Usage Pattern
  - Augmentation for uncertain variable
- Achieved **significant performance gains** across diverse models

Thank you!

Paper