Neural Type Prediction with Search-based Validation

Michael Pradel\textsuperscript{1}, Georgios Gousios\textsuperscript{2}, Jason Liu\textsuperscript{3}, Satish Chandra\textsuperscript{3}

\textsuperscript{1} University of Stuttgart, \textsuperscript{2} TU Delft, \textsuperscript{3} Facebook

FSE 2020
Types in Dynamic PLs

- **Dynamically typed languages:** Extremely popular

- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support

- **Gradual types to the rescue**
Types in Dynamic PLs

- **Dynamically typed languages:** Extremely popular
- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support
- **Gradual types to the rescue**

But: Annotating types is painful
How to Add Type Annotations?

- Option 1: **Static type inference**
  - Guarantees type correctness, but very limited

- Option 2: **Dynamic type inference**
  - Depends on inputs and misses types

- Option 3: **Probabilistic type prediction**
  - Models learned from existing type annotations
Probabilistic Type Prediction

E.g., neural model to predict types

Identifiers
Comments
Code tokens

Neural model

Type annotations

Popular models:
- Deep Learning Type Inference, FSE’18
- NL2Type: Inferring JavaScript Function Types from Natural Language Information, ICSE’19
Challenges

- **Imprecision**
  - Some predictions are wrong
  - Developers must decide which suggestions to follow

- **Combinatorial explosion**
  - For each missing type: One or more suggestions
  - Exploring all combinations: Practically impossible
Example

def find_match(color):
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
Example

```python
def find_match(color):
    """
    Args:
        color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ['red', 'blue', 'green']
```

Predictions:
1) int
2) str
3) bool

1) str
2) Optional[str]
3) None

1) int
2) str
3) bool

1) List[str]
2) List[Optional[str]]
3) str
def find_match(color):
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    
    Args:
    color (str): color to match on and return
    
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ['red', 'blue', 'green']
Overview of TypeWriter

Lightweight static analysis

NL info

Neural type prediction

PL info

Probabilistic type prediction

Type predictions

Static type checker

Feedback-directed search

Search for consistent types

Program with type annotations

Program
Extracting NL and PL Info

- **NL information**
  - Names of functions and arguments
  - Function-level comments

- **PL information**
  - Occurrences of the to-be-typed code element
  - Types made available via imports
def find_match(color):
    ""
    Args:
      color (str): color to match on and return
    ""
    candidates = get_colors()
    for candidate in candidates:
      if color == candidate:
        return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    
    Args:
    color (str): color to match on and return
    
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    
    Args:
    color (str): color to match on and return

    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
def find_match(color):
    
    Args:
    color (str): color to match on and return
    
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
from ab import de
import x.y.z

def find_match(color):
    
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ['red', 'blue', 'green']
Neural Type Prediction Model

- Code tokens
  - Token embedding
  - RNN
  - Hidden layer + Softmax
  - Type vector

- Identifiers
  - Word embedding
  - RNN

- Comments
  - RNN

Available types
Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript

- **Combinatorial search problem**
  - For type slots $S$ and $k$ predictions per slot: 
    \((k + 1)^{|S|}\) possible type assignments
Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript

- **Combinatorial search problem**
  - For type slots $S$ and $k$ predictions per slot:
    \[(k + 1)^{|S|}\] possible type assignments

Too large to explore exhaustively!
Exploring the Search Space

Tree of variants of program $P$

- $P_{\text{origTypes}}$
- $P_{\text{addedTypes}_1}$
- $P_{\text{addedTypes}_2}$
- $P_{\text{addedTypes}_3}$
- $P_{\text{addedTypes}_4}$

... add, remove, or replace types
Exploring the Search Space

Tree of variants of program $P$

$P_{\text{origTypes}}$

$P_{\text{addedTypes}_1}$ $P_{\text{addedTypes}_2}$

$P_{\text{addedTypes}_3}$ $P_{\text{addedTypes}_4}$

Which variants to explore first?

... add, remove, or replace types
Feedback Function

- **Goal:** Minimize missing types without introducing type errors

- **Feedback score** (lower is better):

\[ v \cdot n_{\text{missing}} + w \cdot n_{\text{errors}} \]
Feedback Function

- **Goal:** Minimize missing types without introducing type errors

- **Feedback score (lower is better):**
  \[ v \cdot n_{\text{missing}} + w \cdot n_{\text{errors}} \]

- Default: \( v = 1, w = 2 \), i.e., higher weight for errors
Exploring the Search Space

- **Optimistic**: Add top-most predicted type everywhere and then remove types.
- Greedy or non-greedy
  - If score decreases, keep the type.
  - Backtrack to avoid local minima.
def find_match(color):
    
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
Example

```python
def find_match(color):
    
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def find_match(color):
    
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str
Example

def find_match(color):
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
Example

def find_match(color):
    
    """
    Args:
        color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ['red', 'blue', 'green']
Example

def find_match(color):
    """
    Args:
    color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]

1) List[str]
2) List[Optional[str]]
3) str

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Optional[str]]
3) str
Evaluation: Setup

- **Code corpora**
  - Facebook’s Python code
  - 5.8 millions lines of open-source code

- **Types**
  - Millions of argument and return types
  - 6-12% already annotated
  - Trivial types (e.g., type of self) ignored
## Effectiveness of Neural Model

<table>
<thead>
<tr>
<th>Approach</th>
<th>Top-1 Precision</th>
<th>Top-1 Recall</th>
<th>Top-1 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeWriter</td>
<td>65%</td>
<td>59%</td>
<td>62%</td>
</tr>
</tbody>
</table>

---

Open-source corpus; Combined prediction (arg. and return types)
# Effectiveness of Neural Model

<table>
<thead>
<tr>
<th>Approach</th>
<th>Top-1</th>
<th></th>
<th>Top-3</th>
<th></th>
<th>Top-5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec</td>
<td>Rec</td>
<td>F1</td>
<td>Prec</td>
<td>Rec</td>
<td>F1</td>
</tr>
<tr>
<td>TypeWriter</td>
<td>65%</td>
<td>59%</td>
<td>62%</td>
<td>80%</td>
<td>71%</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td>85%</td>
<td>75%</td>
<td>80%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequencies</td>
<td>12%</td>
<td>20%</td>
<td>15%</td>
<td>19%</td>
<td>35%</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td>22%</td>
<td>39%</td>
<td>28%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Open-source corpus; Combined prediction (arg. and return types)
# Effectiveness of Neural Model

<table>
<thead>
<tr>
<th>Approach</th>
<th>Top-1 Prec</th>
<th>Top-1 Rec</th>
<th>Top-1 F1</th>
<th>Top-3 Prec</th>
<th>Top-3 Rec</th>
<th>Top-3 F1</th>
<th>Top-5 Prec</th>
<th>Top-5 Rec</th>
<th>Top-5 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeWriter</td>
<td>65%</td>
<td>59%</td>
<td>62%</td>
<td>80%</td>
<td>71%</td>
<td>75%</td>
<td>85%</td>
<td>75%</td>
<td>80%</td>
</tr>
<tr>
<td>NL2Type</td>
<td>59%</td>
<td>55%</td>
<td>57%</td>
<td>73%</td>
<td>67%</td>
<td>70%</td>
<td>79%</td>
<td>71%</td>
<td>75%</td>
</tr>
<tr>
<td>Frequencies</td>
<td>12%</td>
<td>20%</td>
<td>15%</td>
<td>19%</td>
<td>35%</td>
<td>25%</td>
<td>22%</td>
<td>39%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Open-source corpus; Combined prediction (arg. and return types)
## Effectiveness of Search

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Top-$k$</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Type-correct</td>
</tr>
<tr>
<td>Greedy search</td>
<td>1</td>
<td>215 (70%)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>230 (75%)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>231 (75%)</td>
</tr>
<tr>
<td>Non-greedy search</td>
<td>1</td>
<td>216 (71%)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>178 (58%)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>164 (54%)</td>
</tr>
<tr>
<td>Pyre Infer</td>
<td>106 (35%)</td>
<td>78 (25%)</td>
</tr>
</tbody>
</table>

Ground truth: 306 annotations in 47 fully annotated files
Exploring up to $7 \cdot |S|$ states
## Effectiveness of Search

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Top-$k$</th>
<th>Annotations</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Type-correct</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>truth match</td>
<td></td>
</tr>
<tr>
<td>Greedy search</td>
<td>1</td>
<td>215 (70%)</td>
<td>194 (63%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>230 (75%)</td>
<td>196 (64%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>231 (75%)</td>
<td>198 (65%)</td>
<td></td>
</tr>
<tr>
<td>Non-greedy search</td>
<td>1</td>
<td>216 (71%)</td>
<td>195 (64%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>178 (58%)</td>
<td>148 (48%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>164 (54%)</td>
<td>141 (46%)</td>
<td></td>
</tr>
</tbody>
</table>

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states
## Effectiveness of Search

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Top-${k}$</th>
<th>Annotations</th>
<th>Type-correct</th>
<th>Ground truth match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy search</td>
<td>1</td>
<td>215 (70%)</td>
<td>194 (63%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>230 (75%)</td>
<td>196 (64%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>231 (75%)</td>
<td>198 (65%)</td>
<td></td>
</tr>
<tr>
<td>Non-greedy search</td>
<td>1</td>
<td>216 (71%)</td>
<td>195 (64%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>178 (58%)</td>
<td>148 (48%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>164 (54%)</td>
<td>141 (46%)</td>
<td></td>
</tr>
<tr>
<td>Pyre Infer</td>
<td>—</td>
<td>106 (35%)</td>
<td>78 (25%)</td>
<td></td>
</tr>
</tbody>
</table>

Ground truth: 306 annotations in 47 fully annotated files
Exploring up to $7 \cdot |S|$ states

Subsumes practically all types
Limitations

- Type-correctness vs. soundness
- Limited type vocabulary
- Gradual type checking (and hence TypeWriter) is relatively slow
Conclusion

Neural type prediction with search-based validation

- Probabilistic type prediction based on NL and PL information
- Ensure type correctness of added types via gradual type checker
- TypeWriter tool in use at Facebook