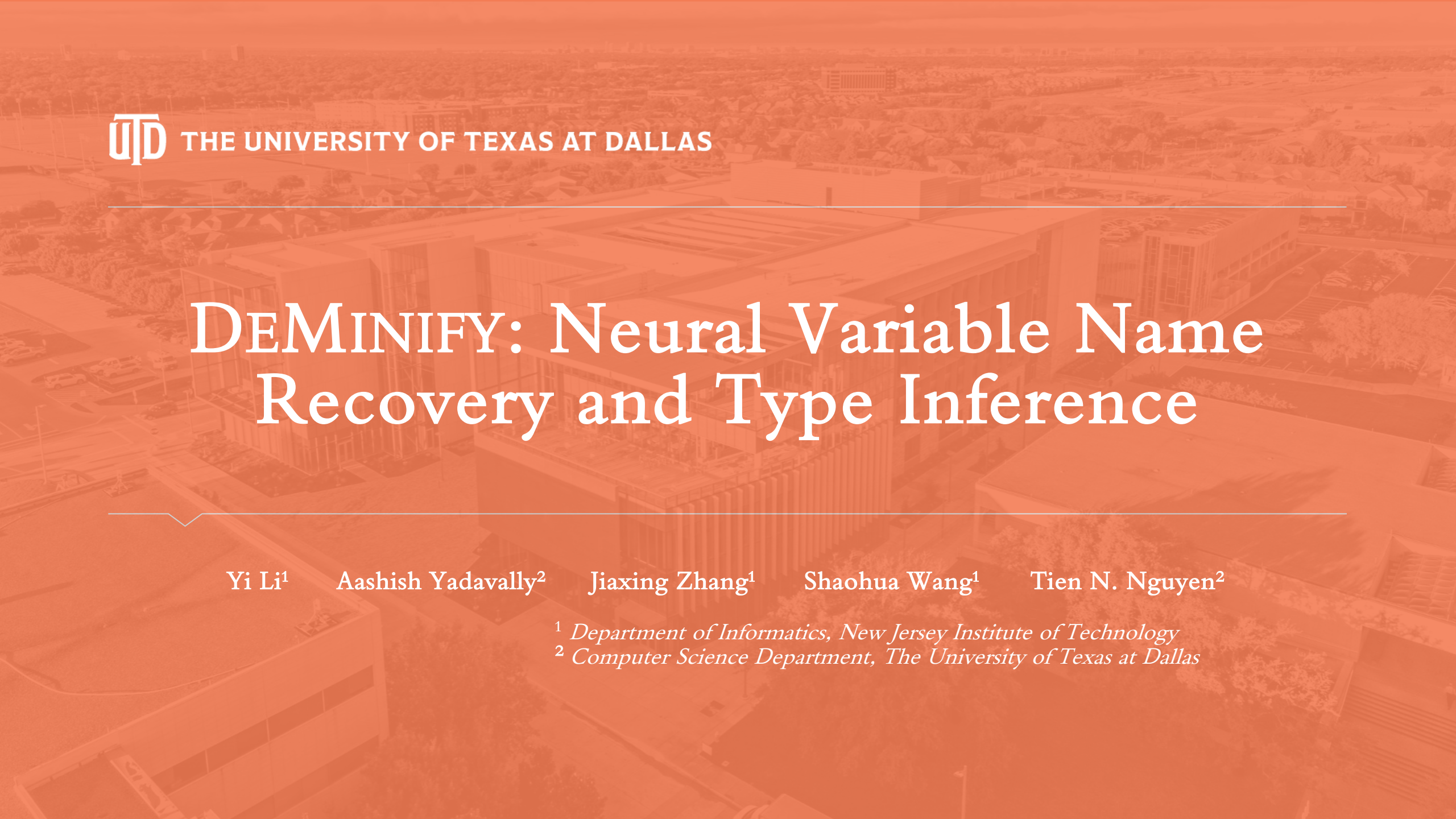


THE UNIVERSITY OF TEXAS AT DALLAS

DeMinify: Neural Variable Name Recovery and Type Inference

Yi Li[1] Aashish Yadavally[2] Jiaxing Zhang[1] Shaohua Wang[1] Tien N. Nguyen[2]

[1] *Department of Informatics, New Jersey Institute of Technology*
[2] *Computer Science Department, The University of Texas at Dallas*

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                        ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                        ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                        ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                        ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          …, range_.startOffset, range_.endOffset)

                              …
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                              …
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                              …
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)

                              …
    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                              …
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:

                              …
```

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          …, range_.startOffset, range_.endOffset)

                              …
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                              …
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                              …
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)

                              …
    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                              …
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                                b, w, r.startContainer)
        if y != 1:

                              …
```

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        p
        p
          range_.startCon
                                ...
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)
                                ...
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
                                ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
                                ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                                ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                                b, w, r.startContainer)
        if y != 1:
                                ...
```

Observation

Dual-Task Learning between Variable Name and Type Prediction.

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        p
        p
          range_.startCon

    trail

    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()

    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                                b, w, r.startContainer)
        if y != 1:
```

Observation — *Dual-Task Learning between Variable Name and Type Prediction.*

Key Idea - I — *Variable Name Learning and Type Learning mutually benefit each other.*

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          …, range_.startOffset, range_.endOffset)

                              …
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                              …
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                              …
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)

                              …
    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                              …
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:

                              …
```

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        r
        p
        preSelectionRange
        preSelectionRange.setEnd(
          range_.startContainer, range_.startOffset)

                          ...

    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                          ...

    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                          ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        d.setEnd(
          r.startContainer, r.startOffset)

                          ...

    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                          ...

    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:

                          ...
```

Observation

The name and type of a variable are affected by the names and types of the surrounding variables in the code.

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        r
        p
        preSelectionRange
        preSelectionRange.setEnd(
          range_.startContainer, range_.startOffset)

    trailingImageCount =
          root, selection


    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                      doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
                              ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
                                  d.setEnd(
                                    r.startContainer, r.startOffset)
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                                  b, w, r.startContainer)
        if y != 1:
                              ...
```

Observation

The name and type of a variable are affected by the names and types of the surrounding variables in the code.

Key Idea - II

"Tell Me Your Friends, I'll Tell You Who You Are"

We treat the problem of variable name and type generation as predicting the missing features in a graph neural network – leveraging Missing Feature Graph Convolutional Network (GCN_{MF}) for this purpose.

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          …, range_.startOffset, range_.endOffset)
                              ...
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)
                              ...
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
                              ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                              ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                              ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                              ...
```

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
        preSelectionRange = range_.cloneRange()
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          …, range_.startOffset, range_.endOffset)

                              …
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                              …
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                              …
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)

                              …
    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                              …
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:

                              …
```

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if se
        range_ = selectio
        preSelectionRange
        preSelectionRange.selectNodeContents(root)
        preSelectionRange.setEnd(
          range_.startContainer, range_.startOffset)

                          ...

    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)

                          ...

    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:

                          ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    a = b.getSelection()
        d.selectNodeContents(w)
        d.setEnd(
          r.startContainer, r.startOffset)

                          ...

    a = self.getTrailingImageCount(
          w, p, r.endContainer)

                          ...

    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:

                          ...
```

Observation

The actual variable name must be in accordance with the names of the accessed fields and called methods.

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if se
        range_ = selectio
        preSelectionRange
        preSelectionRange.selectNodeContents(root)
        p
    trailingImageCount = self.getTrailingImageCount(
          root, selectionState, range_.endContainer)
                          ...
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
                          ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    s = b.getSelection()
        d.selectNodeContents(w)
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                          ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                          ...
```

Observation *The actual variable name must be in accordance with the names of the accessed fields and called methods.*

Observation *The fields and methods of a variable are kept intact after minification.*

Motivating Example

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if se
        range_ = selectio
        preSelectionRange
        preSelectionRange.selectNodeContents(root)
        p
    trailingImageCount = self.getTrailingImageCount(
    if start != 0:
        emptyBlocksIndex =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                    doc, root, range_.startContainer)
        if emptyBlocksIndex != 1:
                        ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    g = b.getSelection()
        d.selectNodeContents(w)
    a = self.getTrailingImageCount(
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                          b, w, r.startContainer)
        if y != 1:
                        ...
```

Observation

The actual variable name must be in accordance with the names of the accessed fields and called methods.

Observation

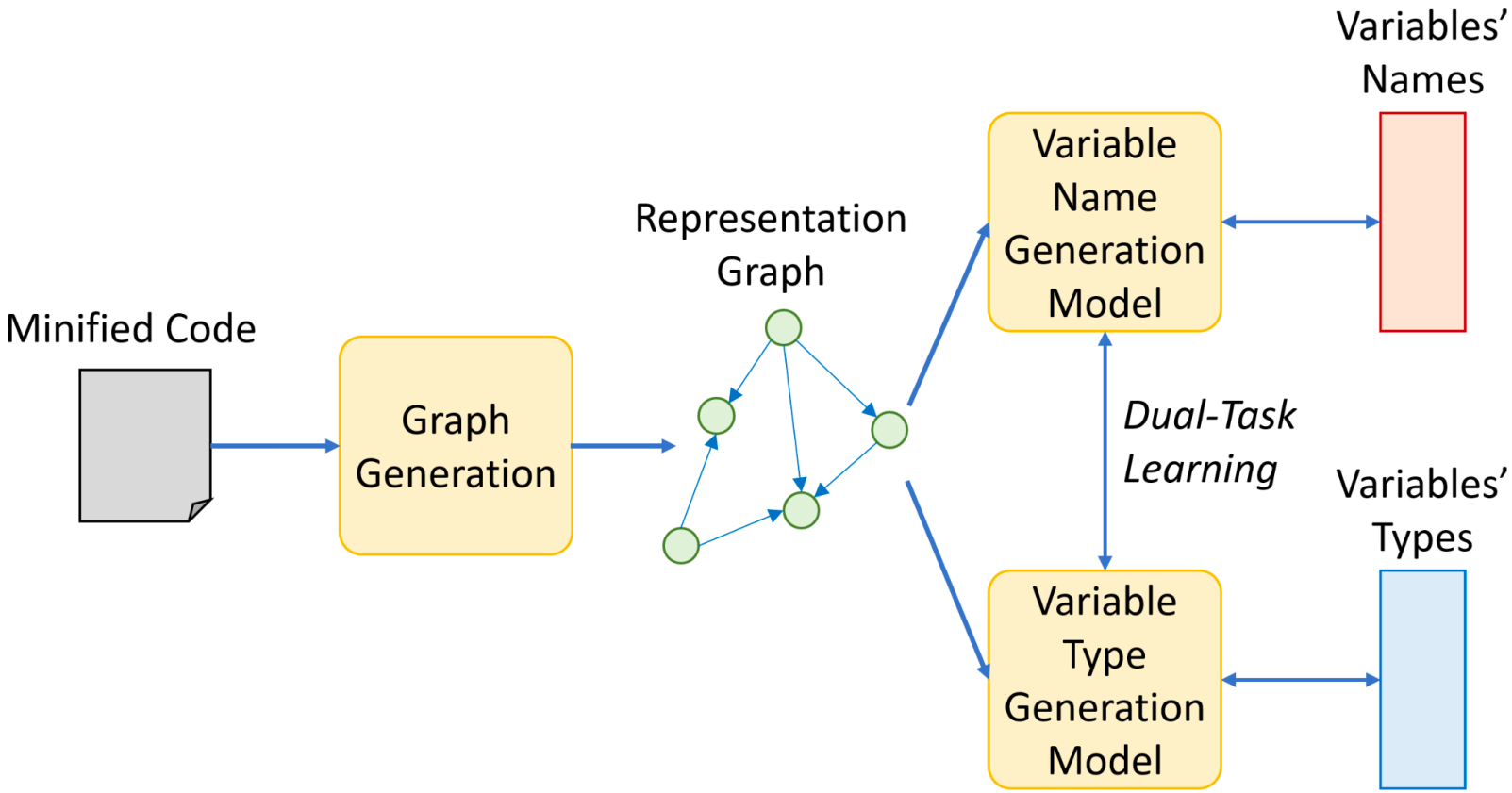
The fields and methods of a variable are kept intact after minification.

Key Idea - III

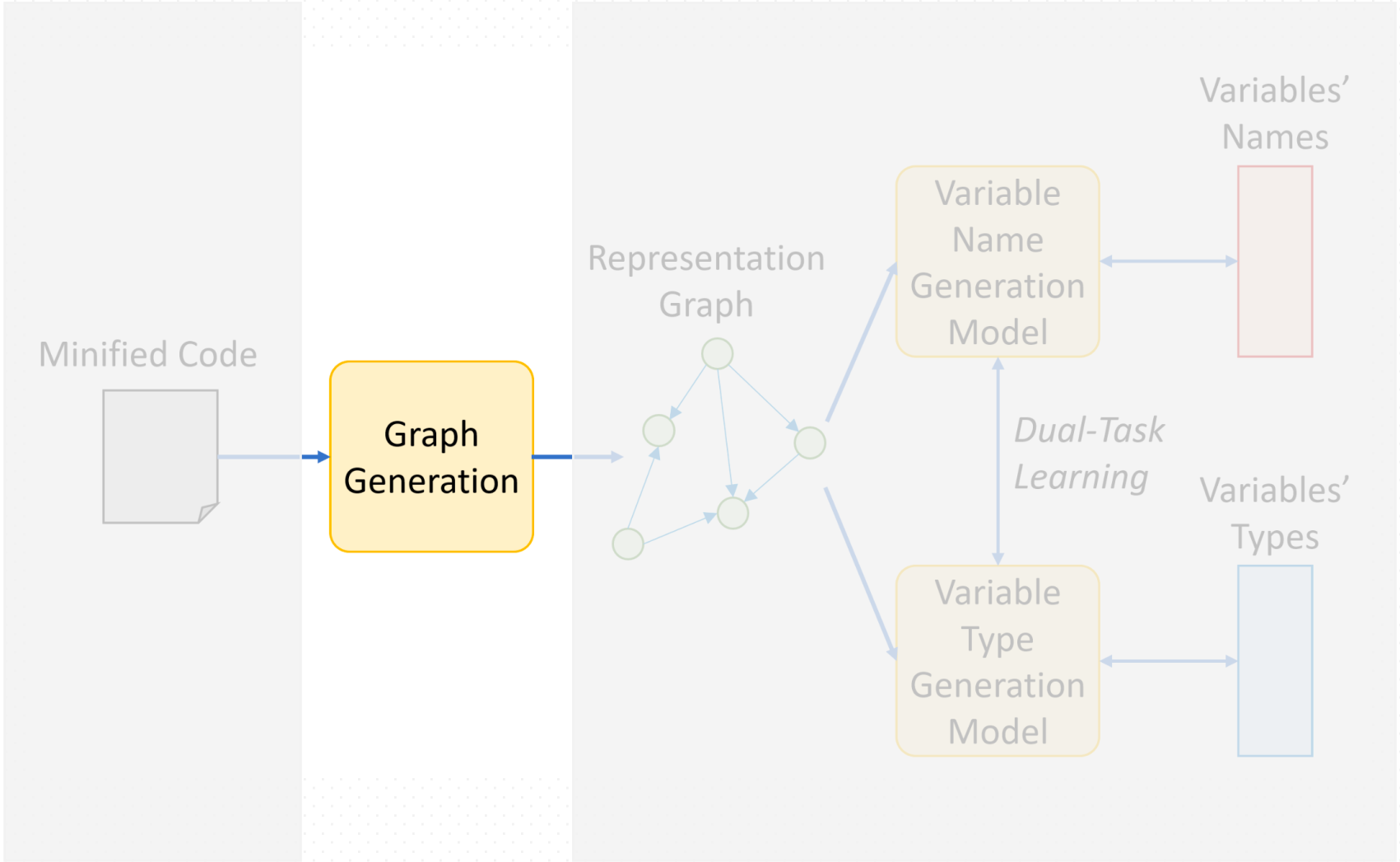
"Properties of a Variable"

The name of a variable is in accordance with its own properties.

DeMinify: Architecture Overview



DeMinify: Architecture Overview



Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

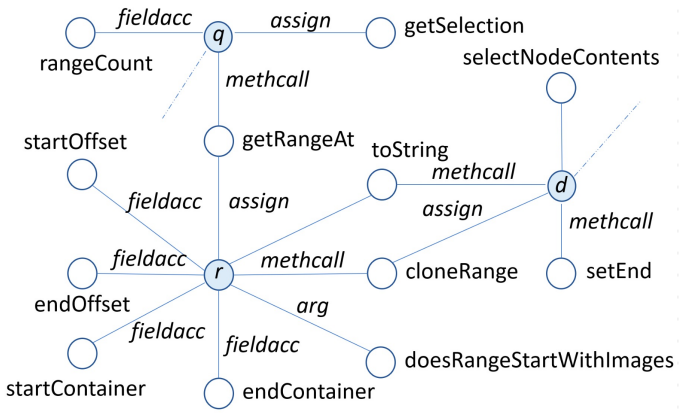
[1] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Recovering Variable Names for Minified Code with Usage Contexts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, 1165–1175.

Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                          ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                          ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                          ...
```



[1] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Recovering Variable Names for Minified Code with Usage Contexts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, 1165–1175.

Part I: Graph Generation

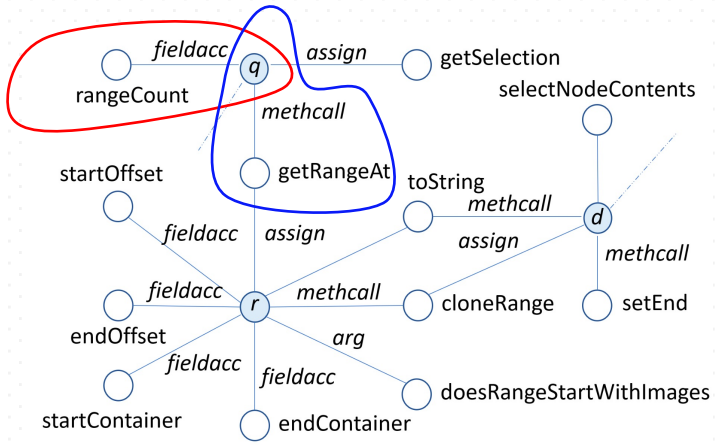
Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                          ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                          ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                          ...
```

Attribute and Behavior Triplets

< variable, field/method, fieldAccess/methodCall >



[1] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Recovering Variable Names for Minified Code with Usage Contexts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, 1165–1175.

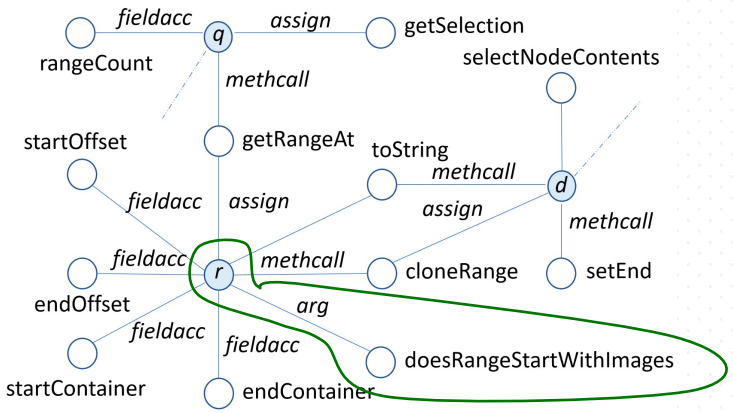
Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                          ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                          ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                          ...
```

Argument Relation



[1] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Recovering Variable Names for Minified Code with Usage Contexts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, 1165–1175.

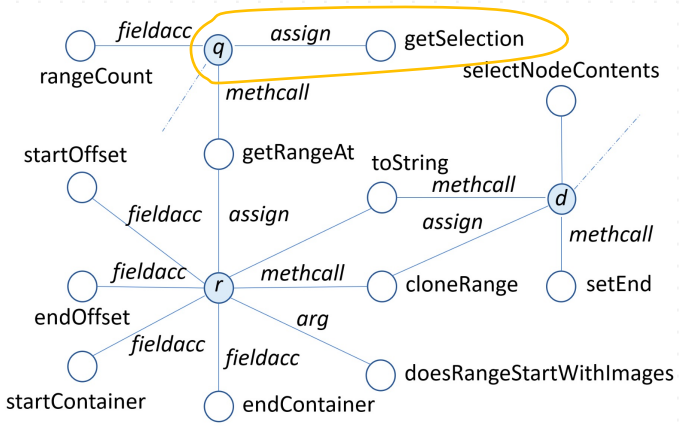
Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                          ...
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                          ...
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                              b, w, r.startContainer)
        if y != 1:
                          ...
```

Assignment Relation



[1] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Recovering Variable Names for Minified Code with Usage Contexts. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE '19). IEEE Press, 1165–1175.

Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

- **Type Dependency Graph** [2]
  - represents the types of the variables in a function/method according to the type inference rules.

[2] Yun Peng, Cuiyun Gao, Zongjie Li, Bowei Gao, David Lo, Qirun Zhang, and Michael Lyu. 2022. Static Inference Meets Deep Learning: A Hybrid Type 1229 Inference Approach for Python. In Proceedings of the 44th International Conference on Software Engineering (Pittsburgh, Pennsylvania) (ICSE '22). Association for Computing Machinery, New York, NY, USA, 2019–2030.

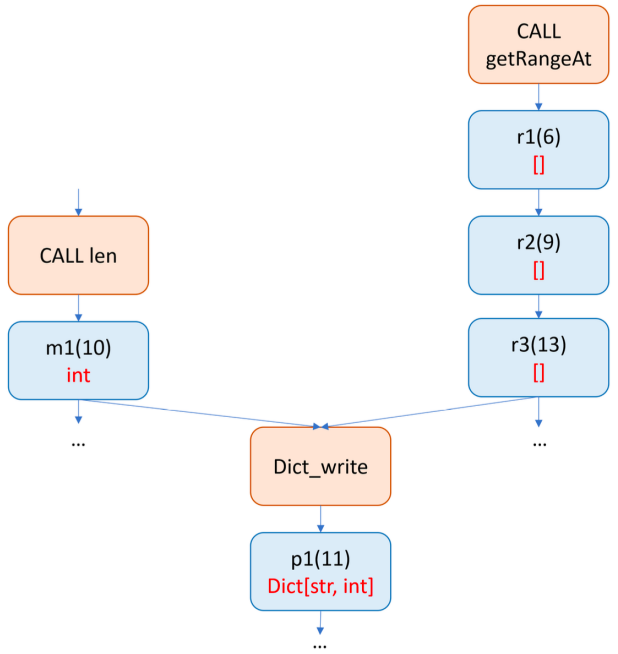
Part I: Graph Generation

Given minified code, it is first parsed and two feature graphs are extracted:

- **Relation Graph** [1]
  - represents the relations among the variables including the ones via field accesses and method calls.

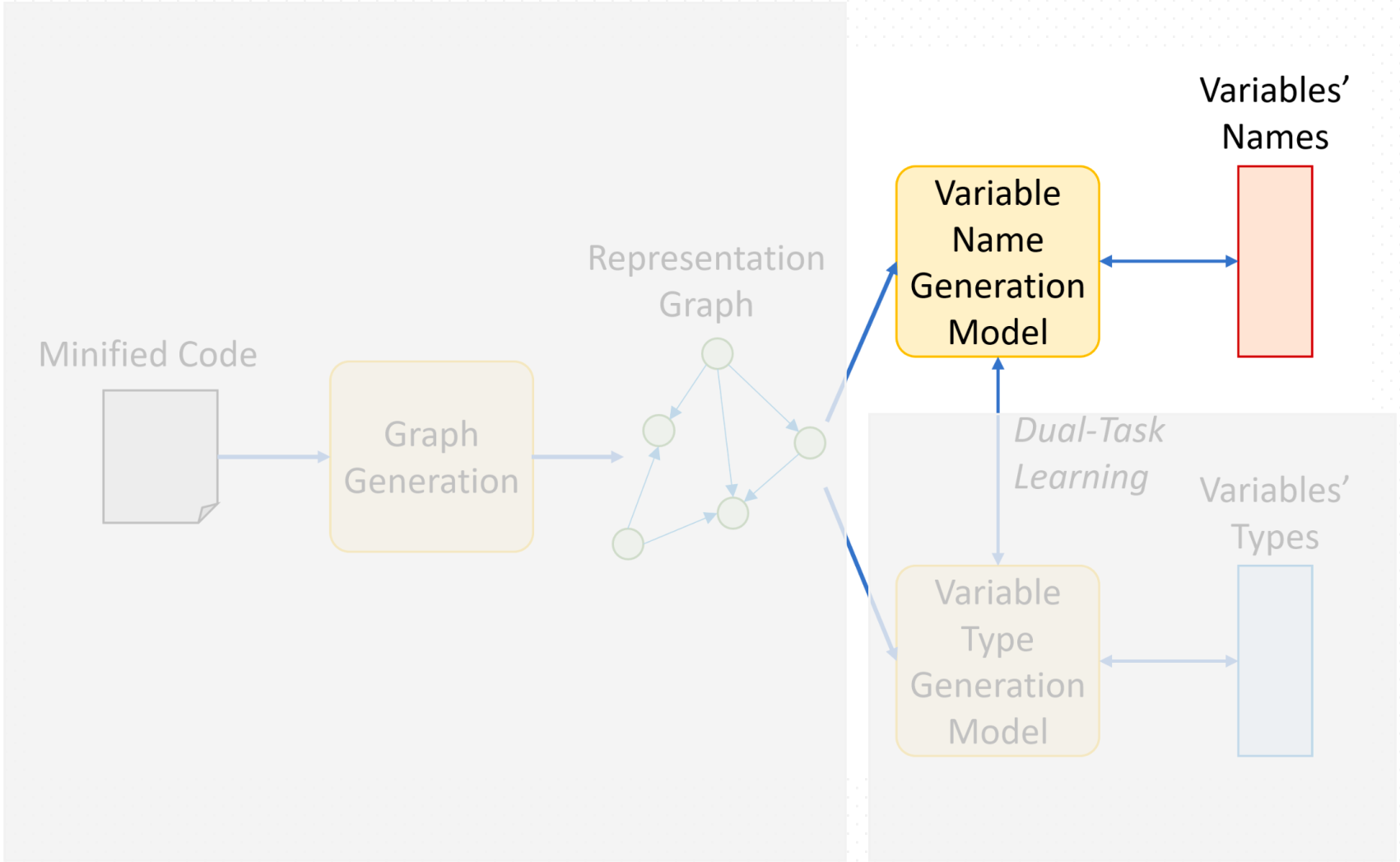
- **Type Dependency Graph** [2]
  - represents the types of the variables in a function/method according to the type inference rules.

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
        d = r.cloneRange()
        d.selectNodeContents(w)
        d.setEnd(
          …, r.startOffset, r.endOffset)
                              …
    a = self.getTrailingImageCount(
          w, p, r.endContainer)
                              …
    if m != 0:
        y =
          self.getIndexRelativeToAdjacentEmptyBlocks(
                                b, w, r.startContainer)
        if y != 1:
                              …
```

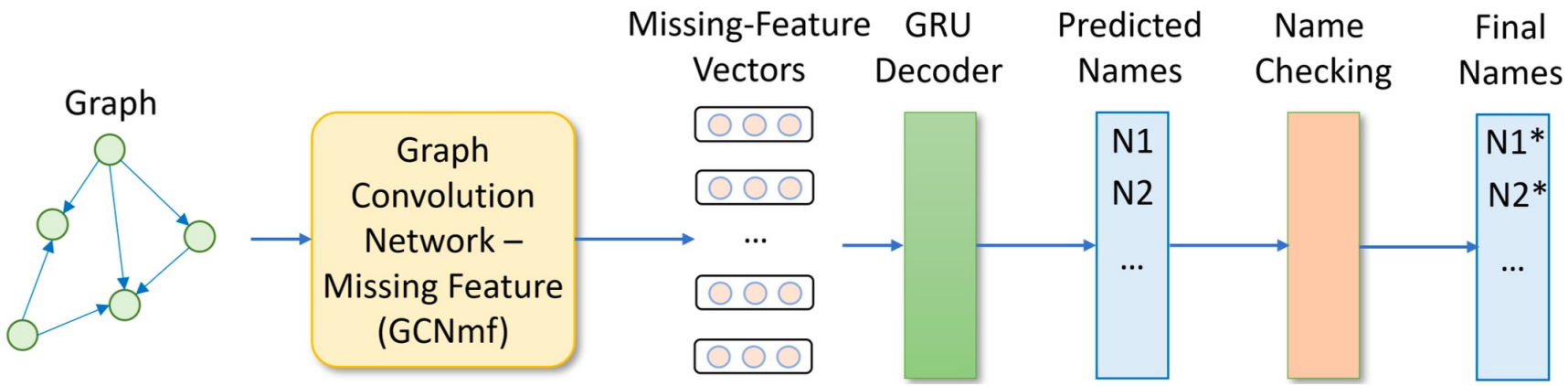


[2] Yun Peng, Cuiyun Gao, Zongjie Li, Bowei Gao, David Lo, Qirun Zhang, and Michael Lyu. 2022. Static Inference Meets Deep Learning: A Hybrid Type 1229 Inference Approach for Python. In Proceedings of the 44th International Conference on Software Engineering (Pittsburgh, Pennsylvania) (ICSE '22). Association for Computing Machinery, New York, NY, USA, 2019–2030.

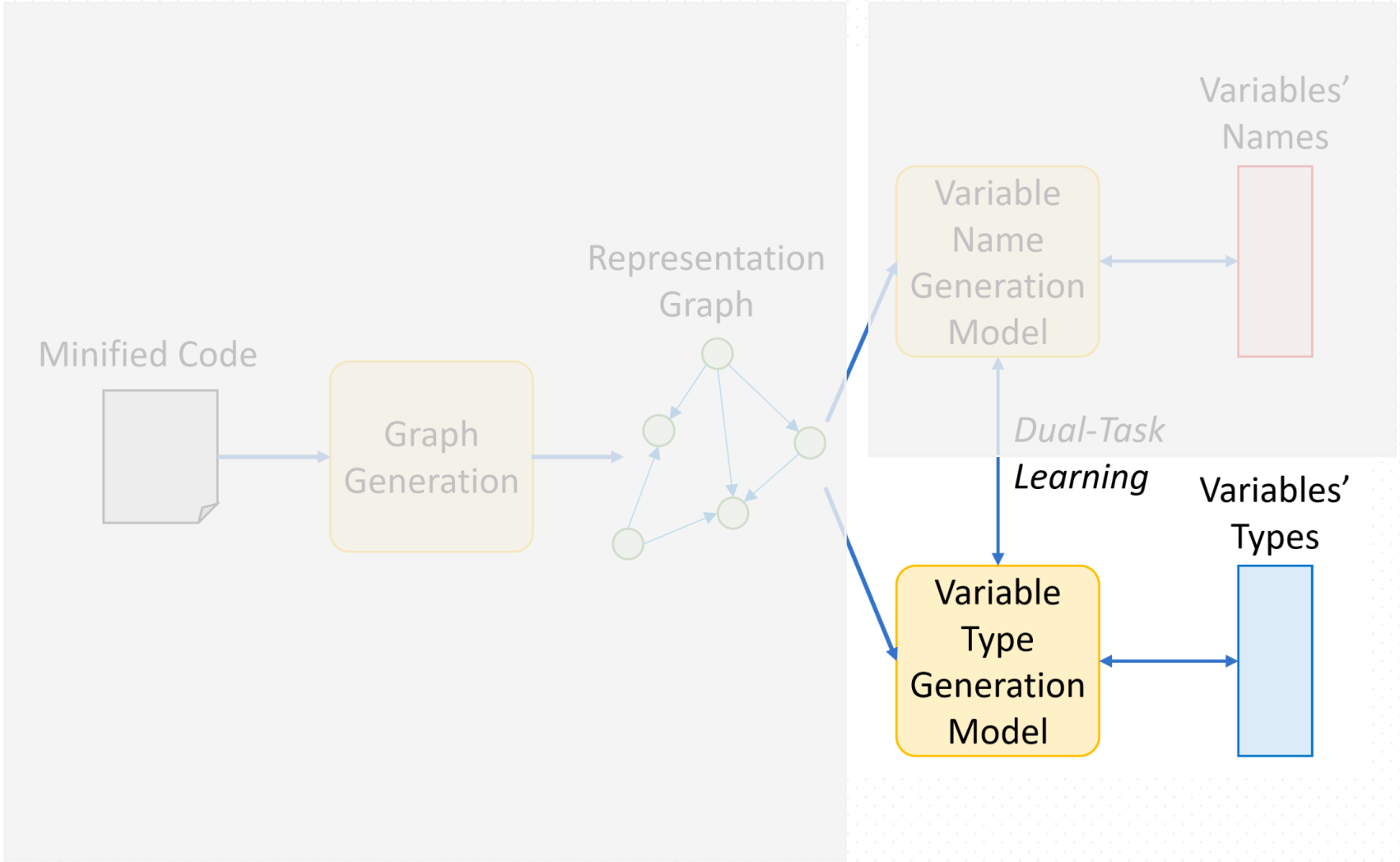
DeMinify: Architecture Overview



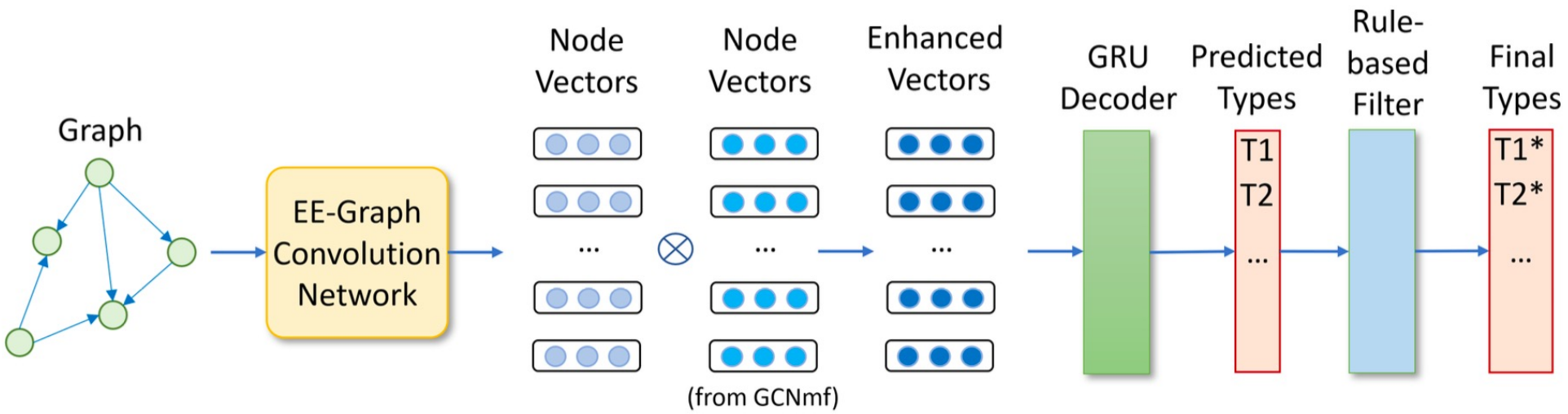
Part II: Variable Name Generation



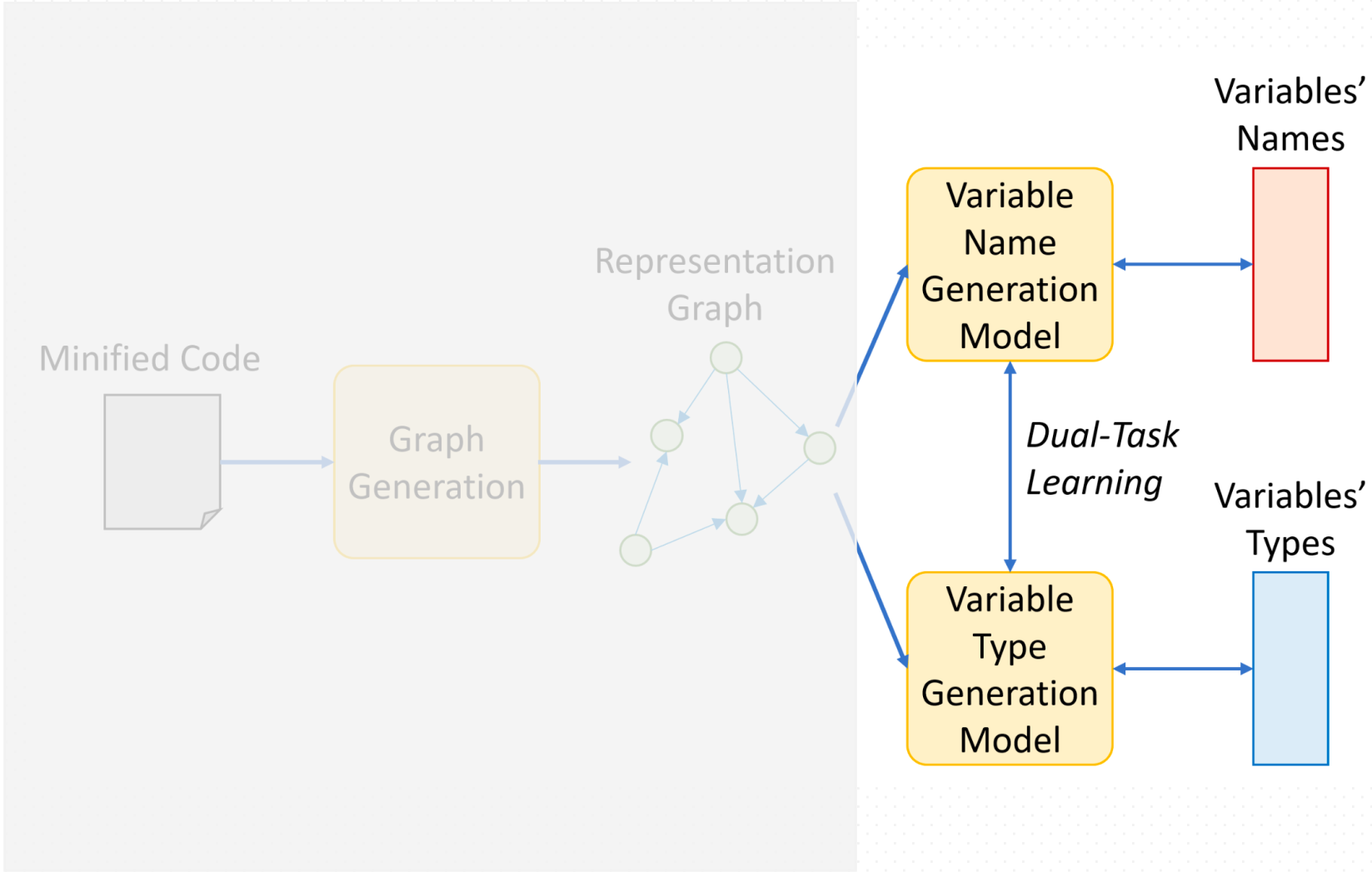
DeMinify: Architecture Overview



Part III: Variable Type Generation



DeMinify: Architecture Overview



Empirical Evaluation (RQ1)

	Top-1		Top-3		Top-5	
	Local	All	Local	All	Local	All
JSNice [31]	41.6	54.5	52.2	63.0	59.5	67.8
JSNaughty [36]	48.3	59.6	59.8	69.7	66.3	75.0
JSNeat [34]	58.2	66.5	65.3	75.4	71.6	80.1
DeMinify	67.5	76.7	75.4	84.3	82.1	90.2

Table 1. Comparative Study on **Variable Name Prediction**

Empirical Evaluation (RQ1)

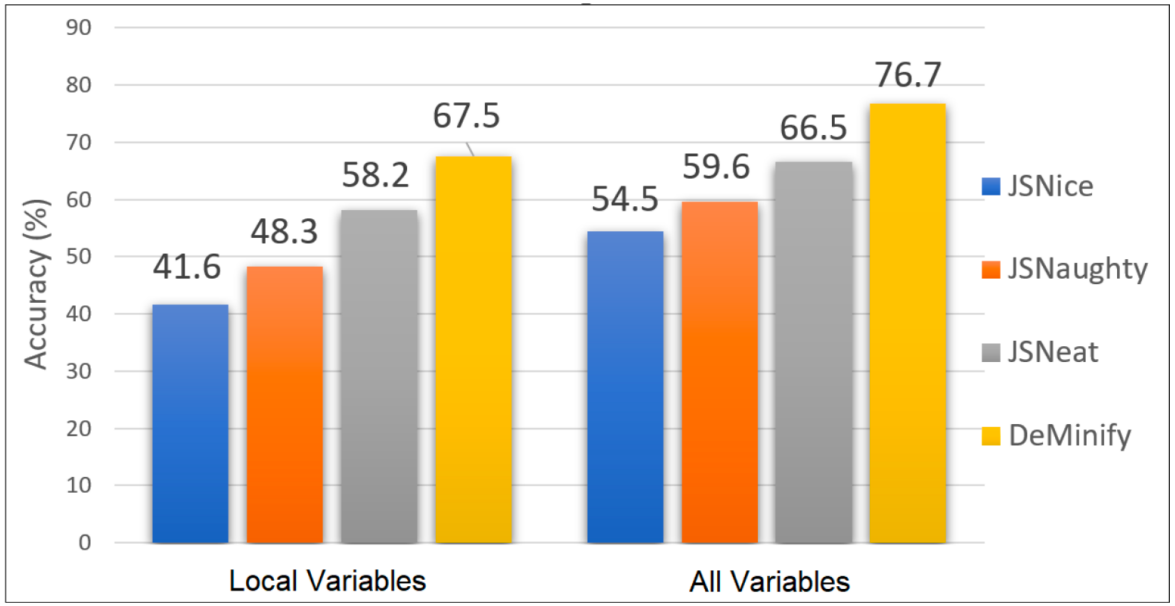


Figure. Top-1 Accuracy on **Variable Name Prediction**

Empirical Evaluation (RQ1)

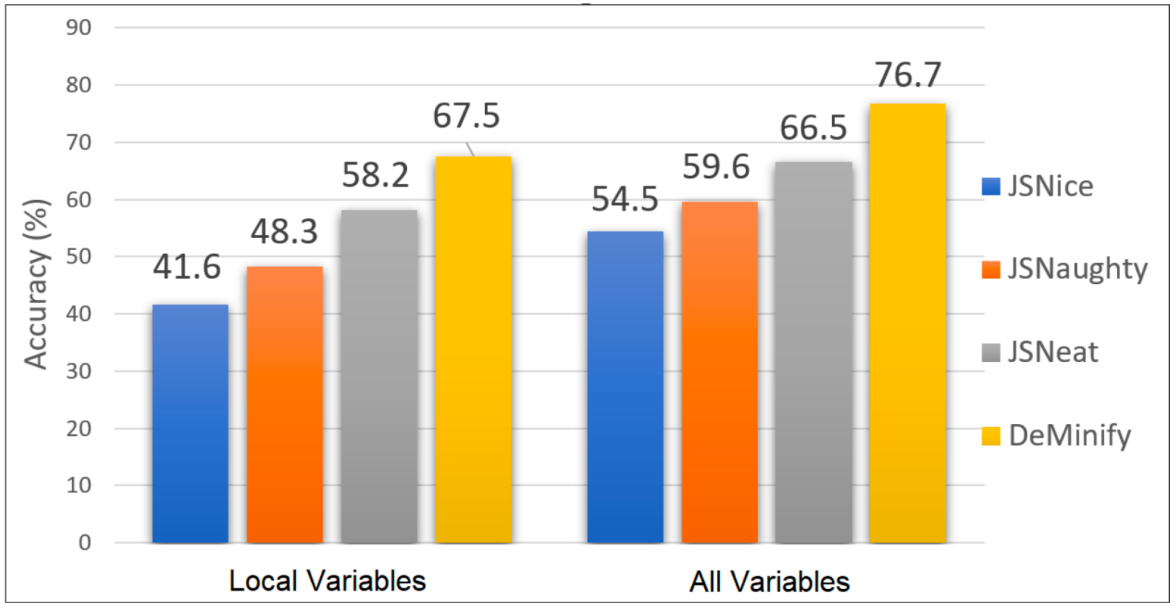


Figure. Top-1 Accuracy on **Variable Name Prediction**

DeMinify *improves over the state-of-the-art approaches for* ***predicting variable names*** *by* ***10.2% - 22.2%****.*

Empirical Evaluation (RQ2)

	Top-1		Top-3		Top-5	
	EM	PM	EM	PM	EM	PM
Ivanov *et al.* [15]	52	58	55	63	60	67
TypeWriter [27]	55	61	59	66	62	70
Typilus [4]	59	66	63	71	64	73
Type4Py [20]	62	66	66	72	67	73
HiTyper [24]	69	77	72	81	72	82
DeMinify	79	89	87	90	88	94

Table 2. Comparative Study on **Variable Type Prediction**

Empirical Evaluation (RQ2)

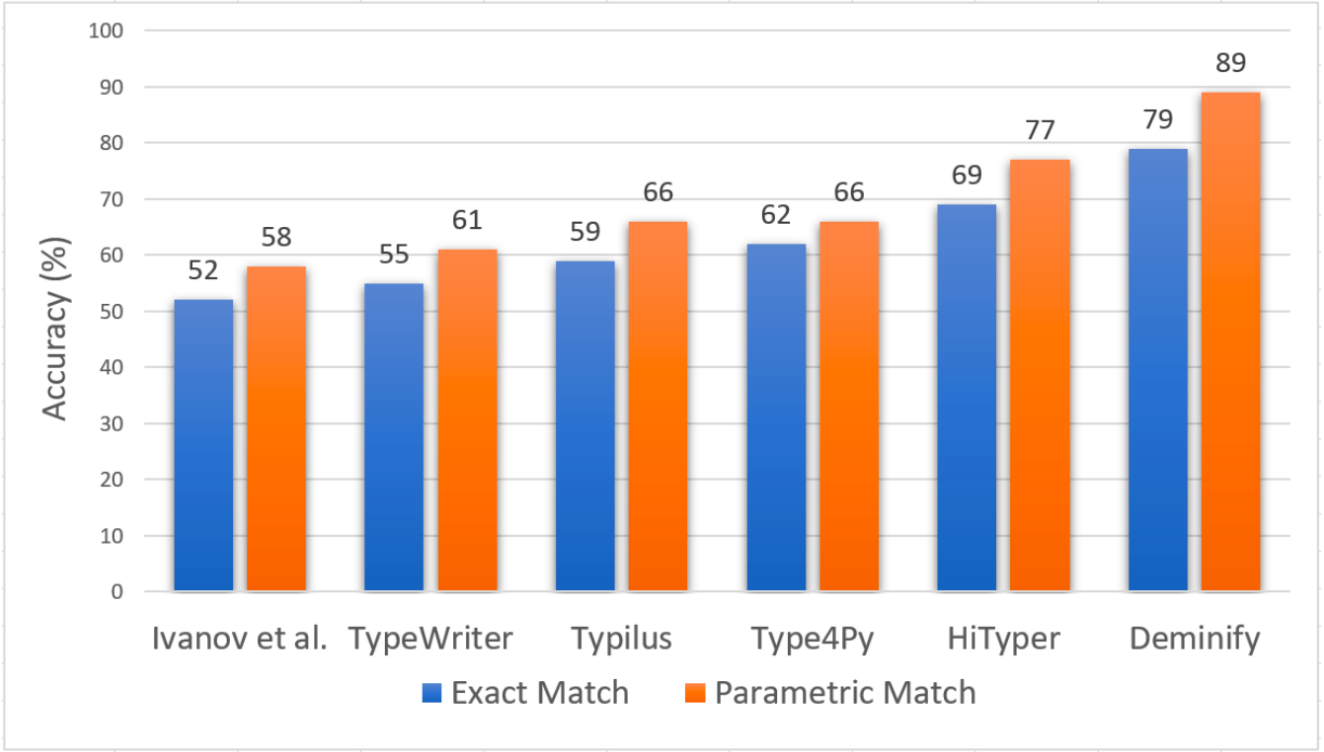


Figure. Top-1 Accuracy on **Variable Type Prediction**

Empirical Evaluation (RQ2)

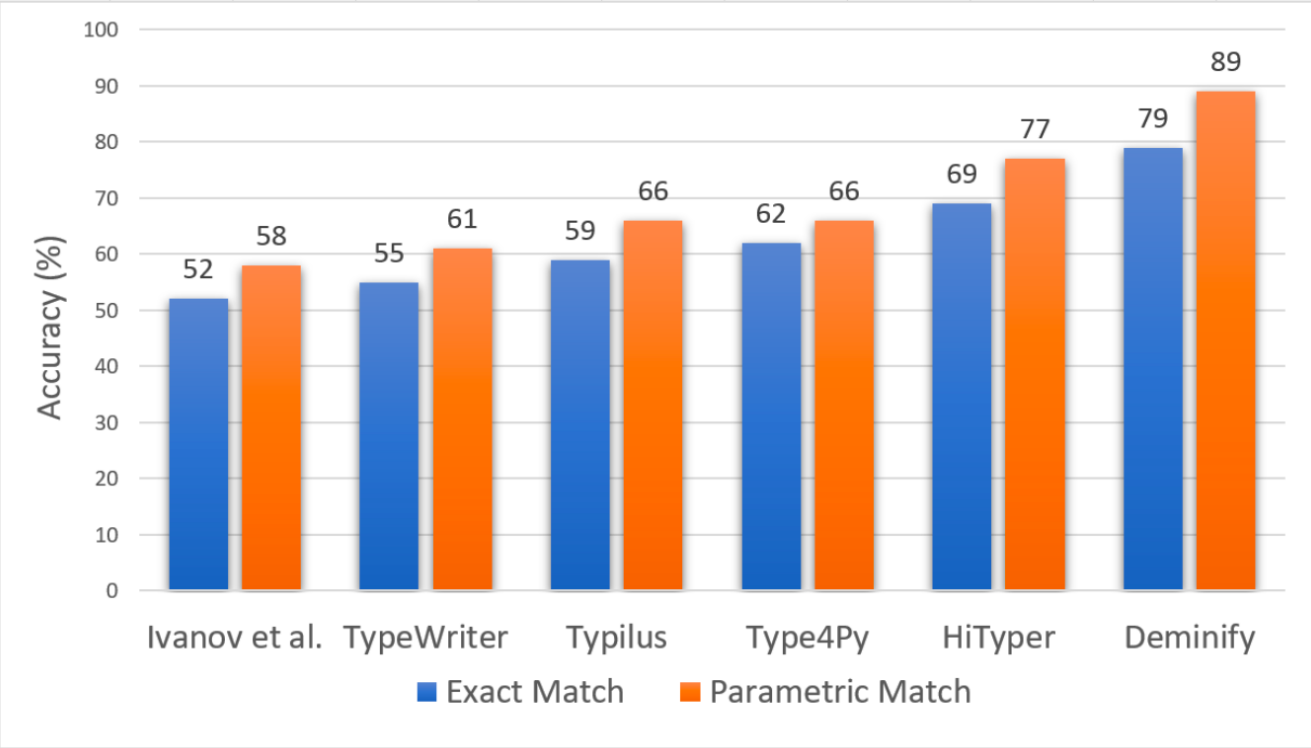


Figure. Top-1 Accuracy on **Variable Type Prediction**

DeMinify *improves over the state-of-the-art approaches for* ***predicting variable types*** *by* ***10% - 27%****.*

Empirical Evaluation (RQ3)

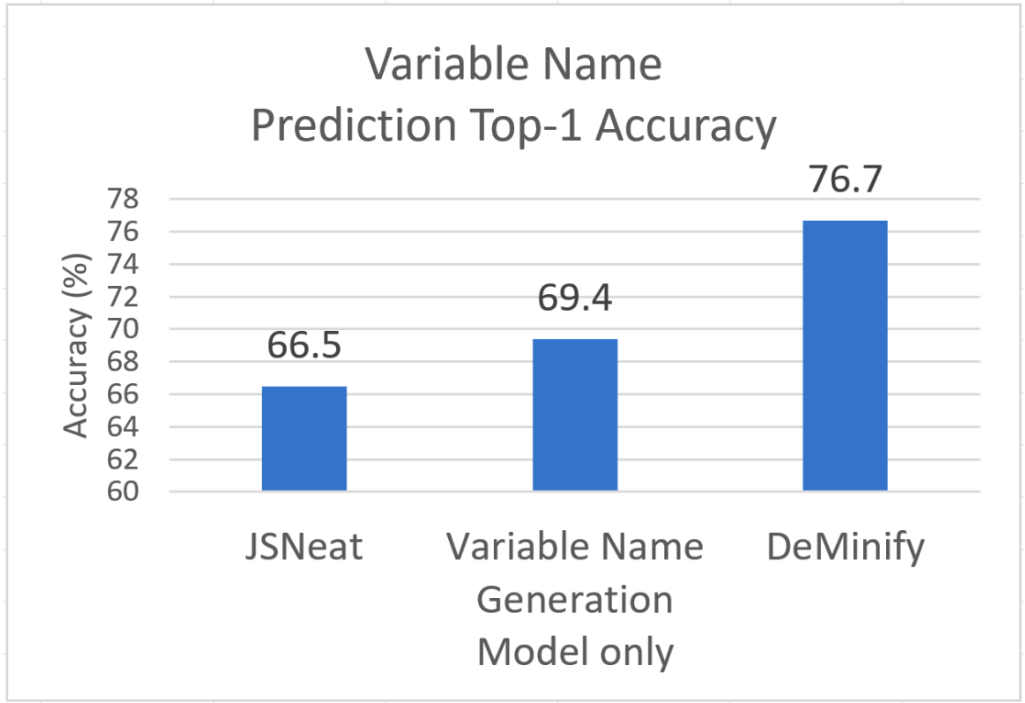


Figure. Impact of Dual-Task Learning on **Variable Name Prediction** (*left*)

Empirical Evaluation (RQ3)

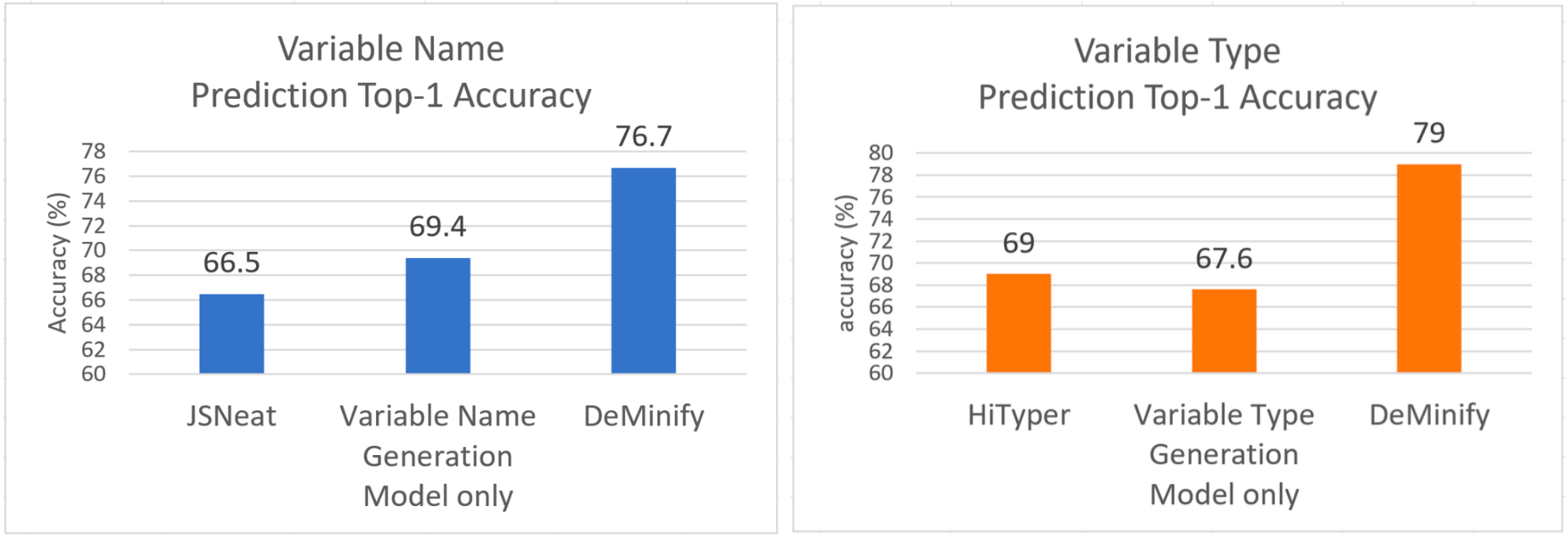


Figure. Impact of Dual-Task Learning on **Variable Name Prediction** (*left*), and **Variable Type Prediction** (*right*).

Empirical Evaluation (RQ3)

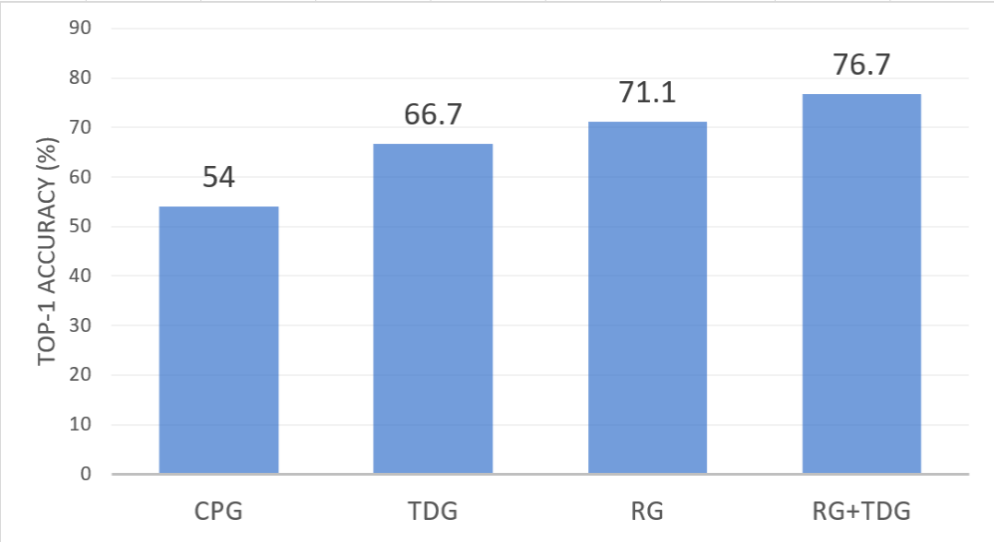


Figure. Impact of input graphs on **Variable Name Prediction** (*left*)
Here, **CPG**: Code Property Graph, **TDG**: Type Dependency Graph, **RG**: Relation Graph

Empirical Evaluation (RQ3)

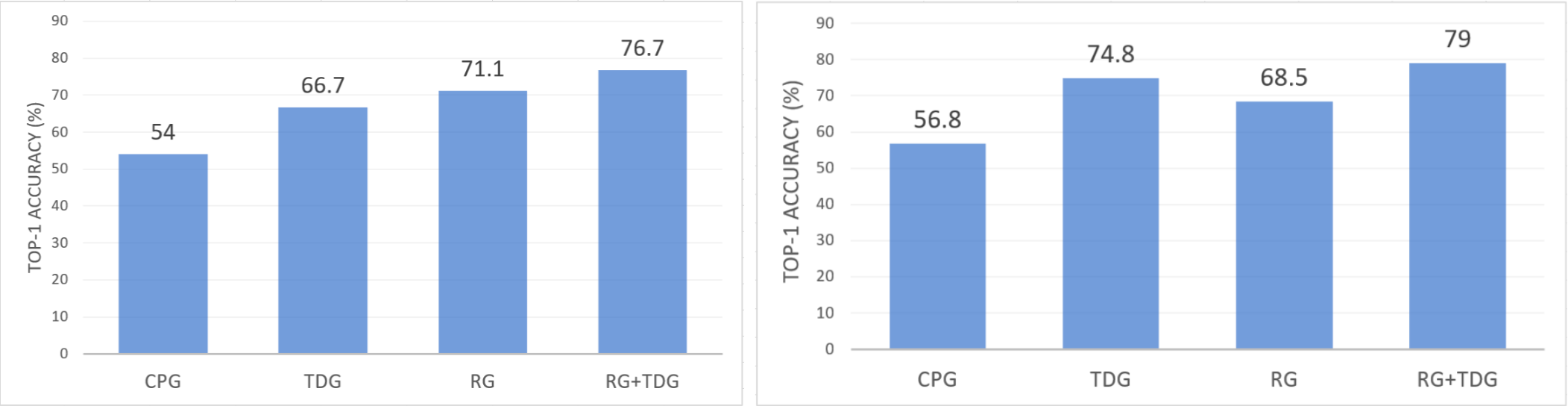


Figure. Impact of input graphs on **Variable Name Prediction** (*left*), and **Variable Type Prediction** (*right*).
Here, **CPG**: Code Property Graph, **TDG**: Type Dependency Graph, **RG**: Relation Graph

DeMinify in Action

```
1  def get(self, request):
2    client_ip = self.extract_ip_from(request)
3    is_limited = self.check_request_limit(client_ip)
4    if is_limited:
5      return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
6    count = BirdNameDatabase.objects.count() - 1
7    index = randint(0, count)
8    bn = BirdNameDatabase.objects.all()[index]
9    serialized = BirdNameSerializer(bn, many=False)
10   self.save_general_statistics(client_ip, bn)
11   return Response(serialized.data)
12
13 def get(self, r):
14   q = self.extract_ip_from(r)
15   p = self.check_request_limit(q)
16   if p:
17     return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
18   m = BirdNameDatabase.objects.count() - 1
19   n = randint(0, m)
20   t = BirdNameDatabase.objects.all()[n]
21   s = BirdNameSerializer(t, many=False)
22   self.save_general_statistics(q, t)
23   return Response(s.data)
```

Figure. Python code listing **correctly** deobfuscated by DeMinify

Conclusion

Scan QR code to access the replication package, data, and supplementary material.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                    ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                    ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

usion

Scan QR code to access the replication package, data, and supplementary material.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

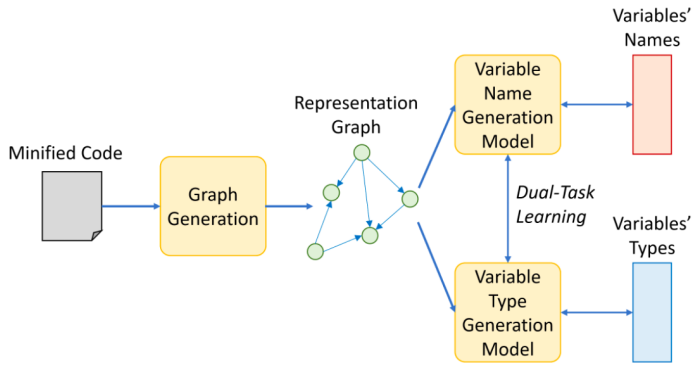
```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                    ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                    ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

DeMinify: Architecture Overview



Scan QR code to access the replication package, data, and supplementary material.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

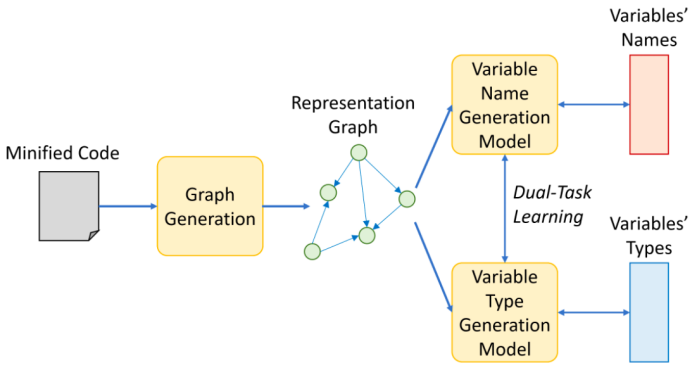
```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                    ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                    ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

DeMinify: Architecture Overview



Empirical Evaluation

- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable names*** *by* ***10.2% - 22.2%***.
- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable types*** *by* ***10% - 27%***.
- *All model components and design choices in* DeMinify *contribute to an improved performance in both variable name and type prediction.*

Scan QR code to access the replication package, data, and supplementary material.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

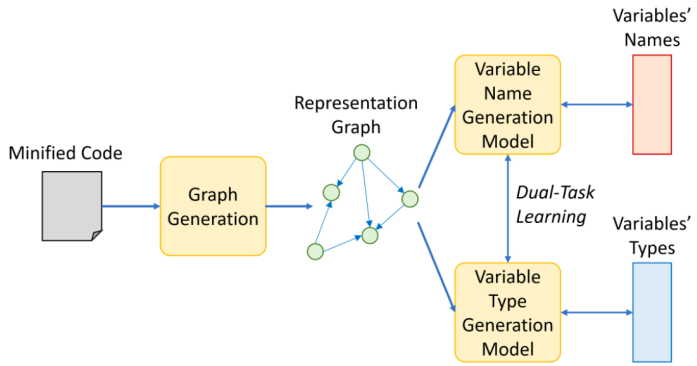
```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                    ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                    ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

DeMinify: Architecture Overview



Empirical Evaluation

- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable names*** *by* ***10.2% - 22.2%****.*
- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable types*** *by* ***10% - 27%****.*
- *All model components and design choices in* DeMinify *contribute to an improved performance in both variable name and type prediction.*

DeMinify in Action

```
def get(self, request):
  client_ip = self.extract_ip_from(request)
  is_limited = self.check_request_limit(client_ip)
  if is_limited:
    return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
  count = BirdNameDatabase.objects.count() - 1
  index = randint(0, count)
  bn = BirdNameDatabase.objects.all()[index]
  serialized = BirdNameSerializer(bn, many=False)
  self.save_general_statistics(client_ip, bn)
  return Response(serialized.data)

def get(self, r):
  q = self.extract_ip_from(r)
  p = self.check_request_limit(q)
  if p:
    return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
  m = BirdNameDatabase.objects.count() - 1
  n = randint(0, m)
  t = BirdNameDatabase.objects.all()[n]
  s = BirdNameSerializer(t, many=False)
  self.save_general_statistics(q, t)
  return Response(s.data)
```

Figure: Python code listing **correctly** deobfuscated by DeMinify

THE UNIVERSITY OF TEXAS AT DALLAS

Scan QR code to access the replication package, data, and supplementary material.

Background

- When the source code needs to be executed on the client side, it is often obfuscated, replacing the variable names with short, opaque, and meaningless names.
- The minification of variable names in this manner helps hide the business logic from the readers.

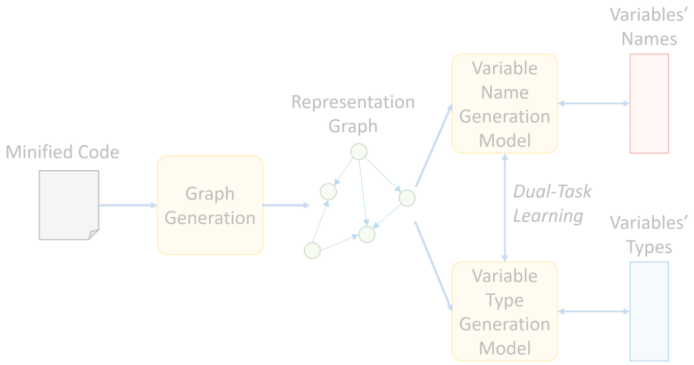
```
def exportSelection(self, root, doc):
    if not root:
        return null
    selection = doc.getSelection()
    if selection.rangeCount > 0:
        range_ = selection.getRangeAt(0)
                    ...
```

```
def exportSelection(self, w, b):
    if not w:
        return null
    q = b.getSelection()
    if q.rangeCount > 0:
        r = q.getRangeAt(0)
                    ...
```

Figure. An original code snippet from a project in GitHub (*left*), and its minified version (*right*).

- The deminification of the same too, is highly relevant in reverse engineering, helping cybersecurity and software analysts identify potentially malicious code.

DeMinify: Architecture Overview



Thank you!

Empirical Evaluation

- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable names*** *by* ***10.2% - 22.2%****.*
- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable types*** *by* ***10% - 27%****.*
- *All model components and design choices in* DeMinify *contribute to an improved performance in both variable name and type prediction.*

DeMinify in Action

```
1  def get(self, request):
2   client_ip = self.extract_ip_from(request)
3   is_limited = self.check_request_limit(client_ip)
4   if is_limited:
5    return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
6   count = BirdNameDatabase.objects.count() - 1
7   index = randint(0, count)
8   bn = BirdNameDatabase.objects.all()[index]
9   serialized = BirdNameSerializer(bn, many=False)
10  self.save_general_statistics(client_ip, bn)
11  return Response(serialized.data)
12
13 def get(self, r):
14  q = self.extract_ip_from(r)
15  p = self.check_request_limit(q)
16  if p:
17   return Response({}, status=rest_framework.status.HTTP_403_FORBIDDEN)
18  m = BirdNameDatabase.objects.count() - 1
19  n = randint(0, m)
20  t = BirdNameDatabase.objects.all()[n]
21  s = BirdNameSerializer(t, many=False)
22  self.save_general_statistics(q, t)
23  return Response(s.data)
```

Figure: Python code listing **correctly** deobfuscated by DeMinify

Scan QR code to access the replication package, data, and supplementary material.

EXTRA SLIDES

Empirical Evaluation

- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable names*** *by* ***10.2% - 22.2%****.*
- DeMinify *improves over the state-of-the-art approaches for* ***predicting variable types*** *by* ***10% - 27%****.*
- *All model components and design choices in* DeMinify *contribute to an improved performance in both variable name and type prediction.*