

Predicting TypeScript Type Annotations and Definitions with Machine Learning

Ming-Ho Yee

Northeastern University

March 29, 2024

Ph.D. Dissertation Defense

Type migration: JavaScript to TypeScript



- Incremental migration
- Static type checking
- Better documentation
- Editor integration

Type migration: JavaScript to TypeScript



- Incremental migration
- Static type checking
- Better documentation
- Editor integration

```
function f(s) {  
  return s;  
}
```

abc f
abc S

```
function f(s: string) {  
  return s;  
}
```

Symbol interface Symbolvar
charAt
charCodeAt
codePointAt
concat

Machine learning for type prediction

Predict the most likely type annotation for the given code fragment

Machine learning for type prediction

Predict the most likely type annotation for the given code fragment

Classification

```
function f(x) {  
    return x + 1;  
}
```

Type of x	Probability
number	0.4221
any	0.2611
string	0.2558
<i>other</i>	

Machine learning for type prediction

Predict the most likely type annotation for the given code fragment

Classification

```
function f(x) {  
    return x + 1;  
}
```

Type of x	Probability
number	0.4221
any	0.2611
string	0.2558
<i>other</i>	

Large language models for code

```
function f(x: _hole_) {  
    return x + 1;  
}
```

```
function f(x: number) {  
    return x + 1;  
}
```

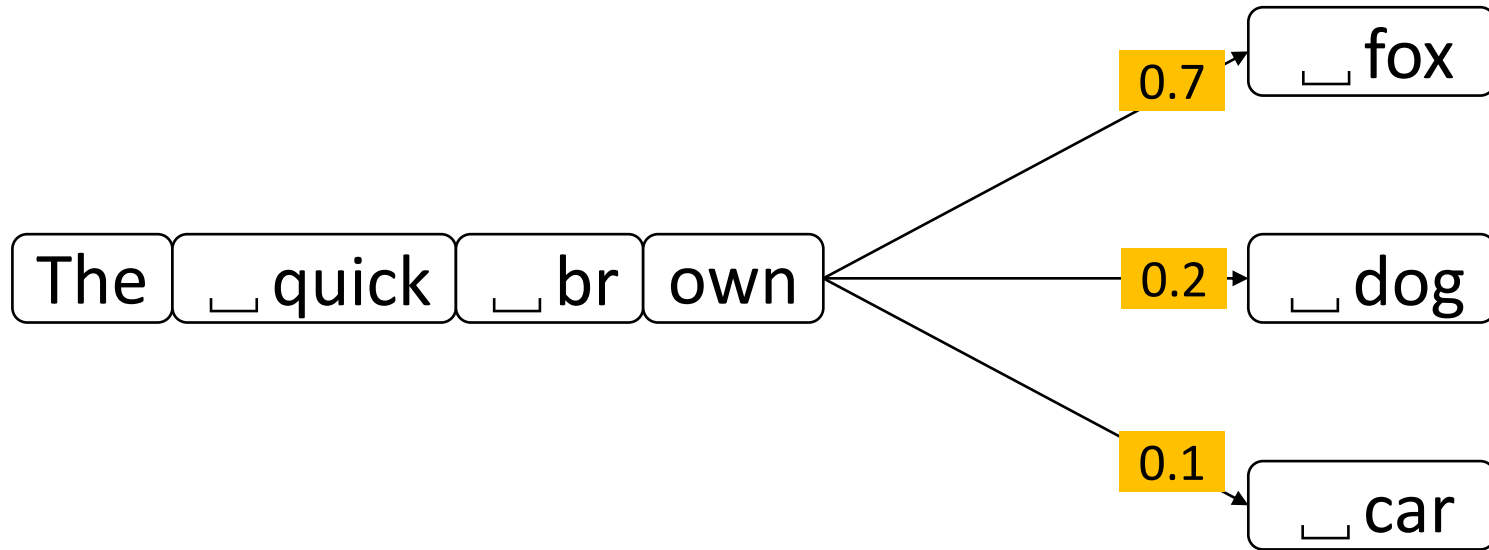
Large language models (LLMs)

The quick brown

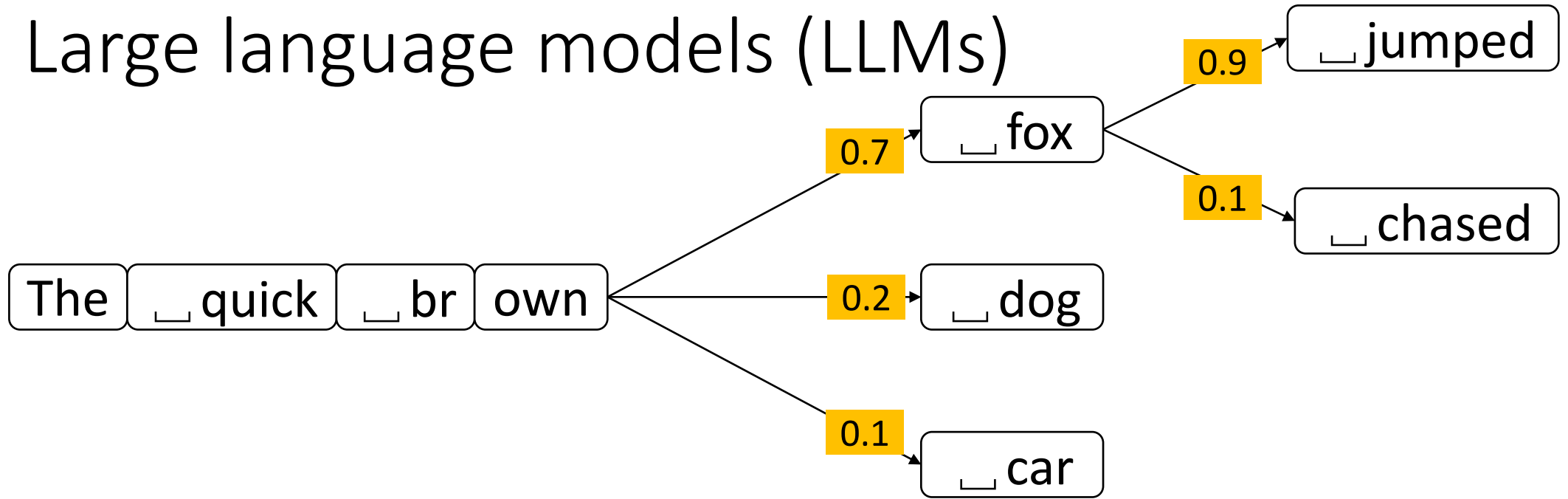
Large language models (LLMs)

The _ quick _ br own

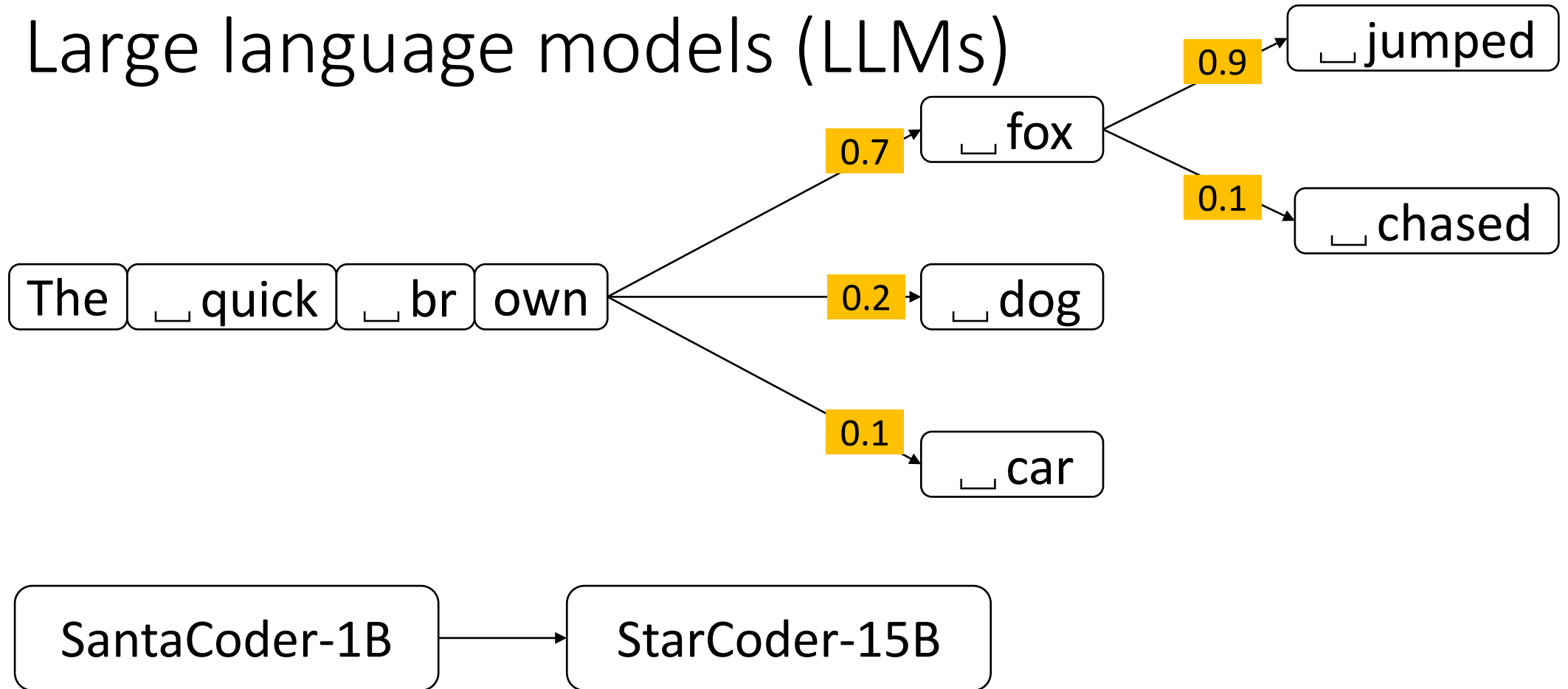
Large language models (LLMs)



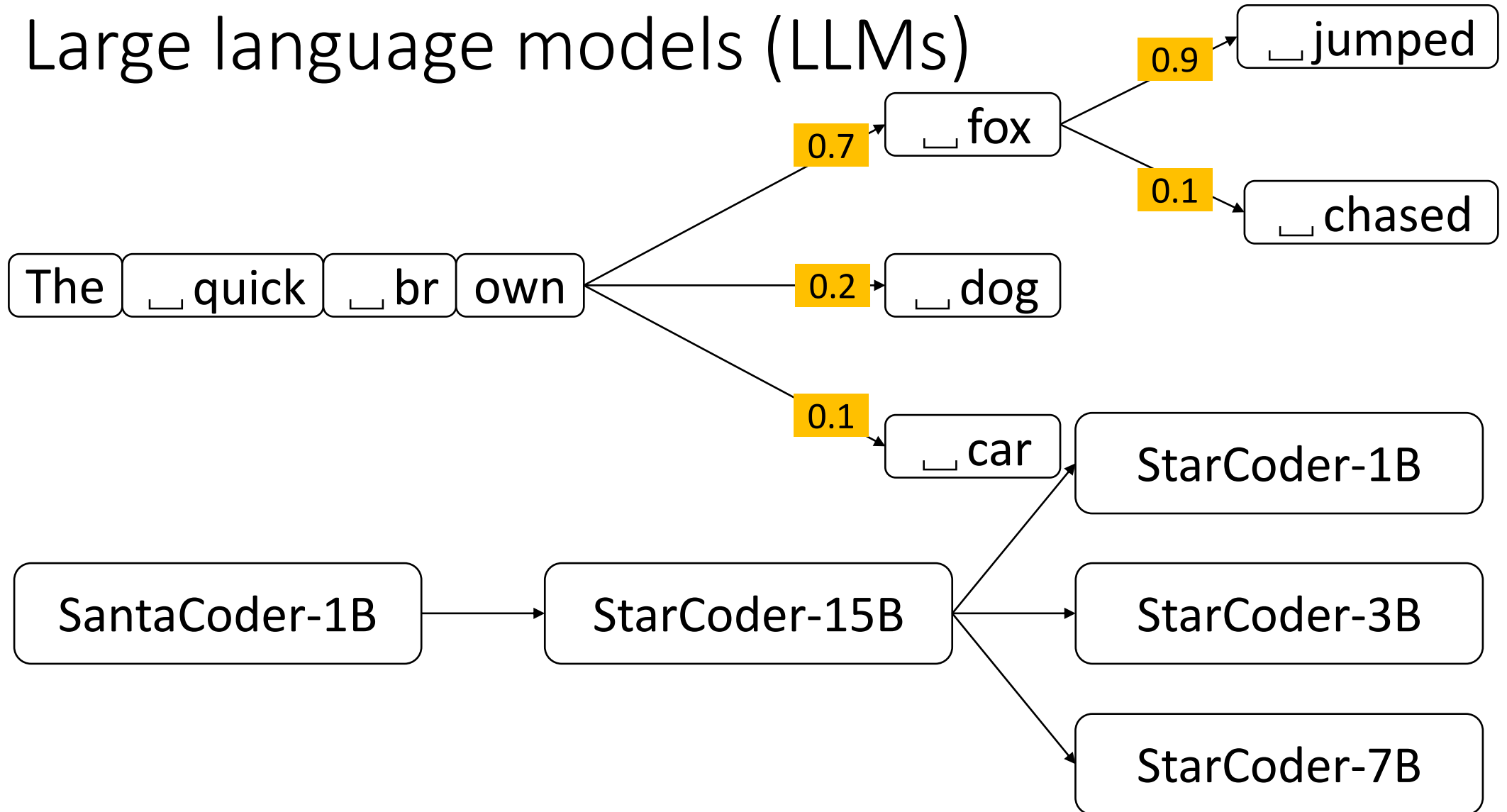
Large language models (LLMs)



Large language models (LLMs)



Large language models (LLMs)



Fill in the middle (FIM)

Fill in the middle (FIM)

Training

```
function fact(n) {  
    if (n == 0) return 1;  
    return n * fact(n-1);  
}
```

Fill in the middle (FIM)

Training

```
function fact(n) {  
    if (n == 0) return 1;  
    return n * fact(n-1);  
}
```

Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_middle>if (n == 0) return 1;  
<fim_suffix>return n * fact(n-1);  
}
```


Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_suffix>return n * fact(n-1);  
}<fim_middle>if (n == 0) return 1;
```

Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_suffix>return n * fact(n-1);  
><fim_middle>if (n == 0) return 1;
```

Inference

```
function f(x: _hole_) {  
    return x + 1;  
}
```

Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_suffix>return n * fact(n-1);  
<fim_middle>if (n == 0) return 1;
```

Inference

```
<fim_prefix>function f(x: <fim_suffix>) {  
    return x + 1;  
<fim_middle>
```

Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_suffix>return n * fact(n-1);  
><fim_middle>if (n == 0) return 1;
```

Inference

```
<fim_prefix>function f(x: <fim_suffix>) {  
    return x + 1;  
><fim_middle>number
```

Fill in the middle (FIM)

Training

```
<fim_prefix>function fact(n) {  
<fim_suffix>return n * fact(n-1);  
}<fim_middle>if (n == 0) return 1;
```

Inference

```
function f(x: number) {  
    return x + 1;  
}
```

Limitations of existing approaches

Limitations of existing approaches

Evaluation

```
function f(x: string) {  
    return x * 1;  
}
```

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Limitations of existing approaches

Evaluation

```
function f(x: string) {  
    return x * 1;  
}
```

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Fill in the Middle

```
function f(x: hole) {  
    return x + 1;  
}
```

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training
[submitted to [COLM 2024](#)]
Cassano, Yee, Shinn, Guha, and Holtzen

Limitations of existing approaches

Evaluation

```
function f(x: string) {  
    return x * 1;  
}
```

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Fill in the Middle

```
function f(x: hole) {  
    return x + 1;  
}
```

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training
[submitted to [COLM 2024](#)]
Cassano, Yee, Shinn, Guha, and Holtzen

Type Definitions

```
interface Point {  
    x: number,  
    y: number  
}
```

Generating TypeScript Type
Definitions with Machine
Learning

Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Thesis

Machine learning can be used to partially migrate **JavaScript programs to TypeScript**, by predicting type annotations and generating type definitions.

Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]

Yee and Guha

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training

[submitted to [COLM 2024](#)]

Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type
Definitions with Machine
Learning

Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

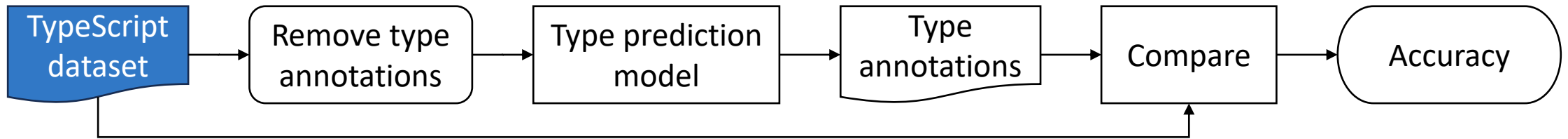
Type Prediction With
Program Decomposition and
Fill-in-the-Type Training
[submitted to [COLM 2024](#)]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type
Definitions with Machine
Learning

TypeWeaver: type check the type annotations

TypeWeaver: type check the type annotations

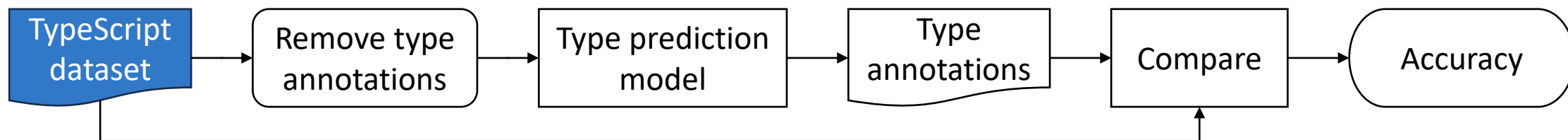
Prior work:



TypeWeaver: type check the type annotations

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{total predictions}}$$

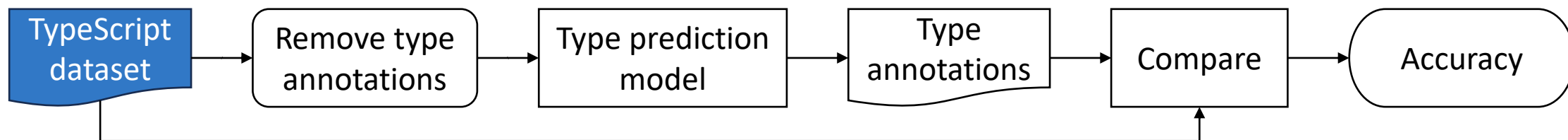
Prior work:



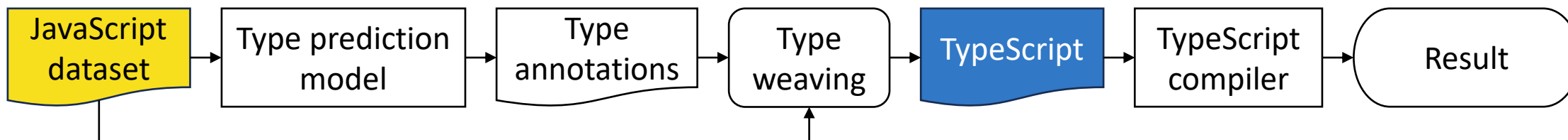
TypeWeaver: type check the type annotations

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{total predictions}}$$

Prior work:



TypeWeaver:



Constructing the JavaScript dataset

Constructing the JavaScript dataset

1. Top 1,000 most downloaded packages



2. Download source code

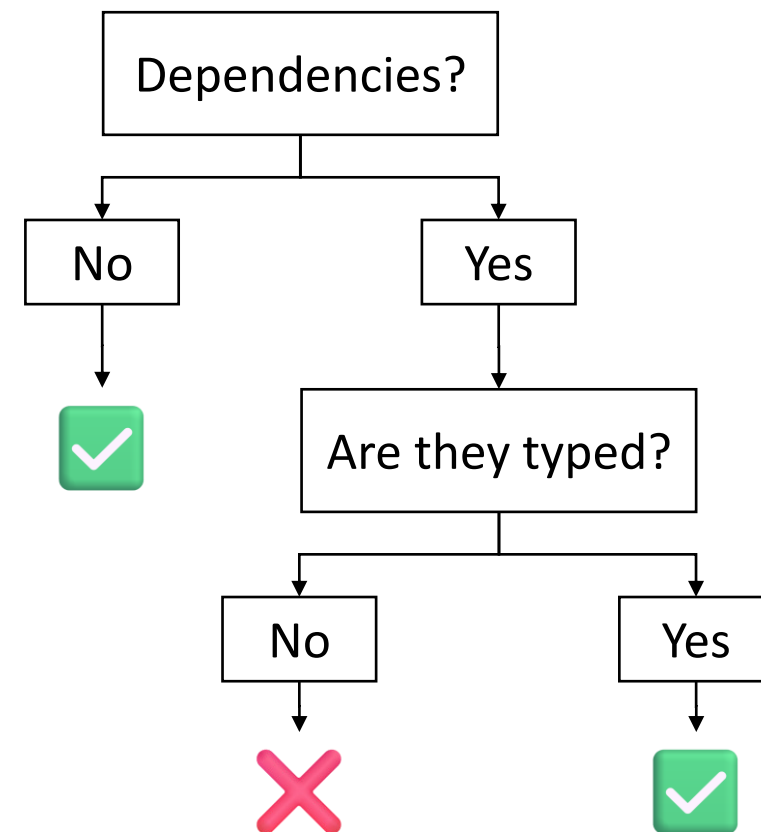


3. Filter and clean

4. Check dependencies

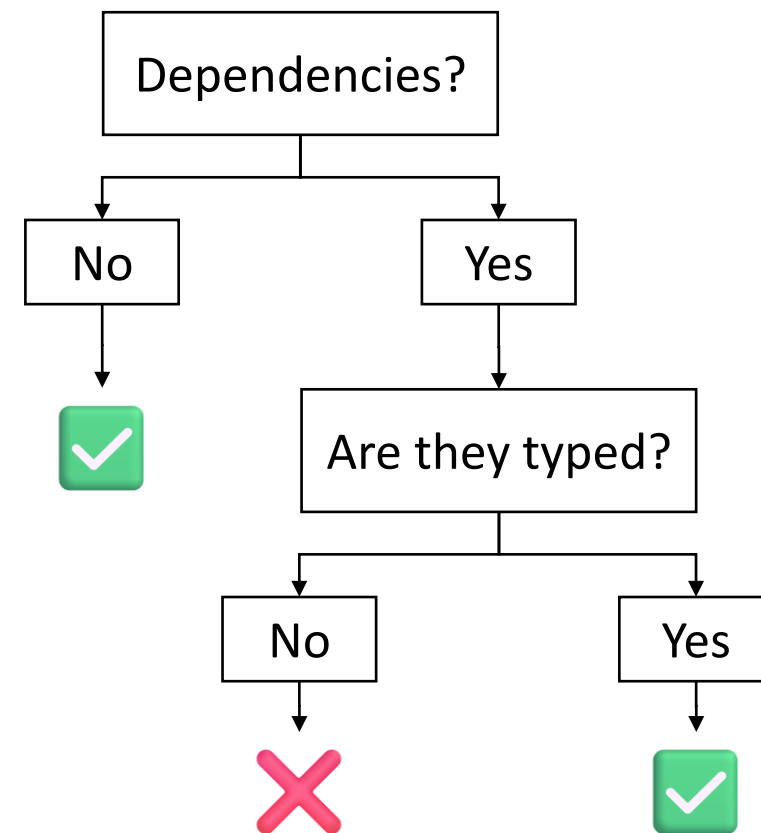
Constructing the JavaScript dataset

1. Top 1,000 most downloaded packages
2. Download source code
3. Filter and clean
4. Check dependencies



Constructing the JavaScript dataset

1. Top 1,000 most downloaded packages
2. Download source code
3. Filter and clean
4. Check dependencies



Result: 506 packages

Type weaving: JS + type annotations = TS

```
function f(x, y) {  
    return x + y;  
}
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x, y) {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x, y) {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
    Parameter
      Identifier
    Parameter
      Identifier
  Block
    ReturnStatement
      ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x, y): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
    Parameter
      Identifier
    Parameter
      Identifier
  Block
    ReturnStatement
      ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x, y): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x: string, y): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x: string, y): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x: string, y: number): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x: string, y: number): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type weaving: JS + type annotations = TS

```
function f(x: string, y: number): string {
  return x + y;
}
```

```
FunctionDeclaration
  Identifier
  Parameter
    Identifier
  Parameter
    Identifier
  Block
    ReturnStatement
    ...
```

Token	Type	Probability
function		
f	string	0.6381
(
x	string	0.4543
,		
y	number	0.4706
)		
{		
return		
x	number	0.3861
+		
y	number	0.5039
;		
}		

Type prediction front end

Original code

```
function sum_list(l) {  
  let sum = 0;  
  for (let i = 0; i < l.length; i++) {  
    sum += l[i];  
  }  
  return sum;  
}
```

Type prediction front end

Insert hole

```
function sum_list(l: _hole_) {  
  let sum = 0;  
  for (let i = 0; i < l.length; i++) {  
    sum += l[i];  
  }  
  return sum;  
}
```

Type prediction front end

Reformat input

```
<fim_prefix>function sum_list(l:  
<fim_suffix>) {  
    let sum = 0;  
    for (let i = 0; i < l.length; i++) {  
        sum += l[i];  
    }  
    return sum;  
}<fim_middle>
```

Type prediction front end

Fill in the middle

```
<fim_prefix>function sum_list(l:  
<fim_suffix>) {  
    let sum = 0;  
    for (let i = 0; i < l.length; i++) {  
        sum += l[i];  
    }  
    return sum;  
}<fim_middle>any[]): number {  
    if (l.length === 0) {  
        throw 'Empty list!';  
    }  
    if (l.length === 1) {  
        return l[0];  
    }  
    return sum
```

Type prediction front end

Extract type

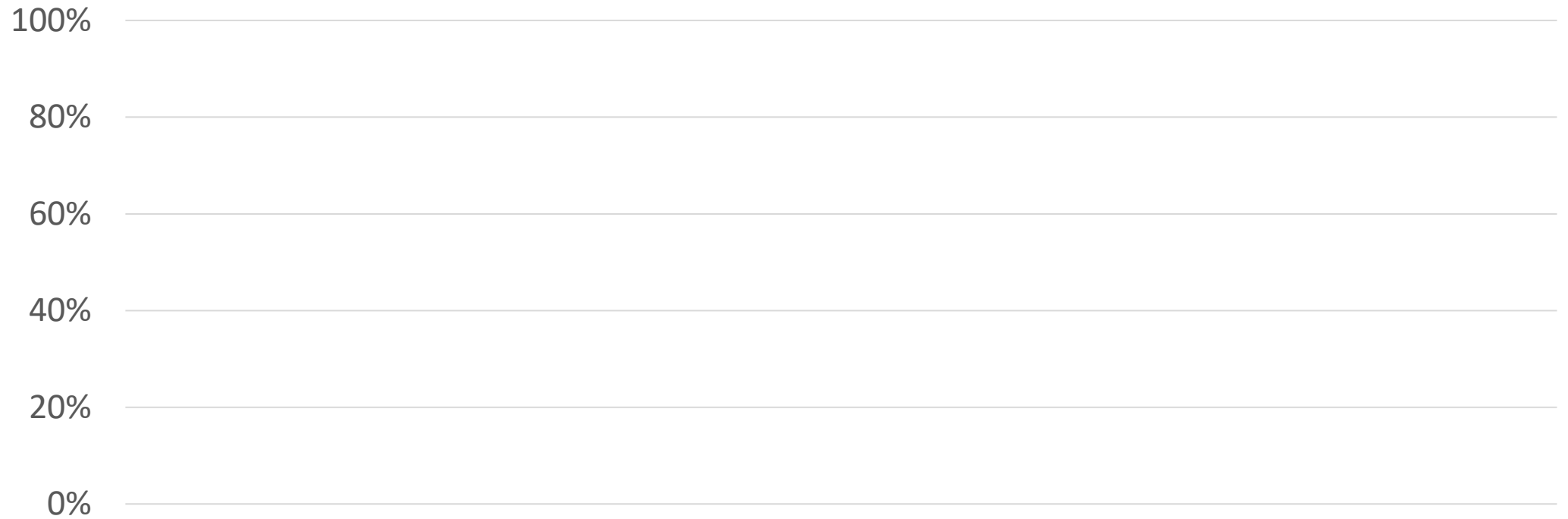
```
<fim_prefix>function sum_list(l:  
<fim_suffix>) {  
  let sum = 0;  
  for (let i = 0; i < l.length; i++) {  
    sum += l[i];  
  }  
  return sum;  
}<fim_middle>any[]): number {  
  if (l.length === 0) {  
    throw 'Empty list!';  
  }  
  if (l.length === 1) {  
    return l[0];  
  }  
  return sum
```

Type prediction front end

Result

```
function sum_list(l: any[]) {  
  let sum = 0;  
  for (let i = 0; i < l.length; i++) {  
    sum += l[i];  
  }  
  return sum;  
}
```

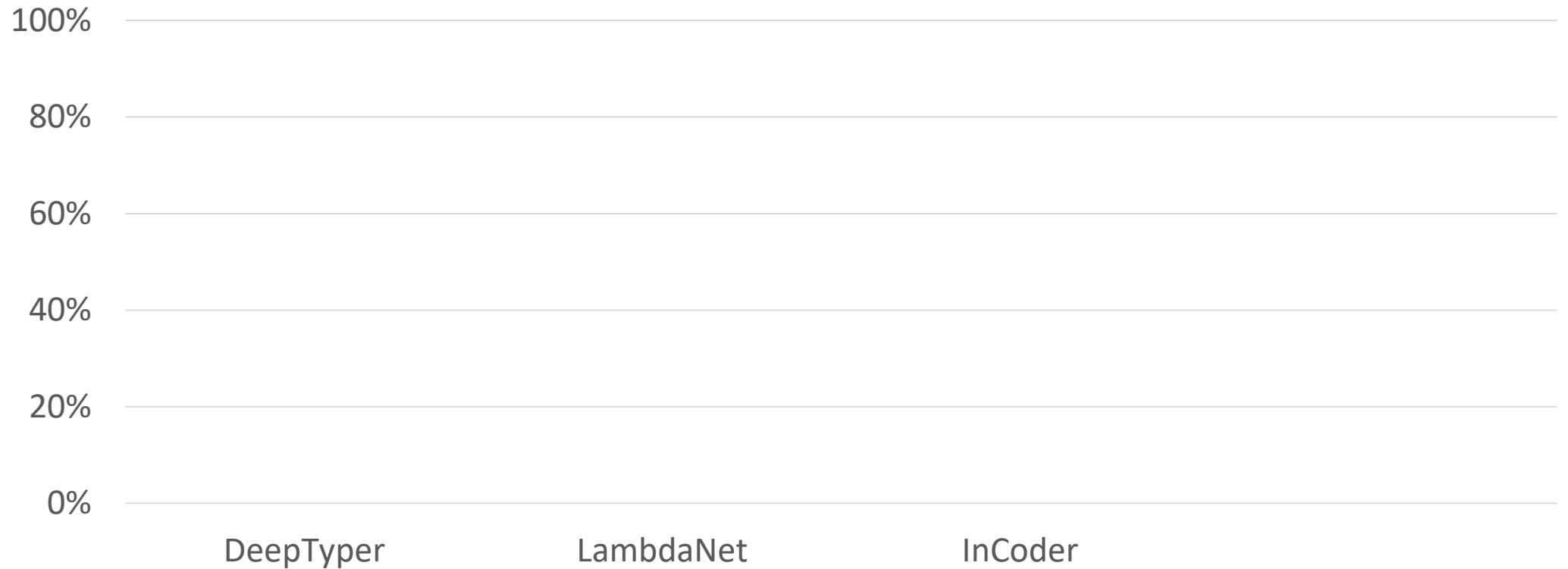
Percentage of packages that type check



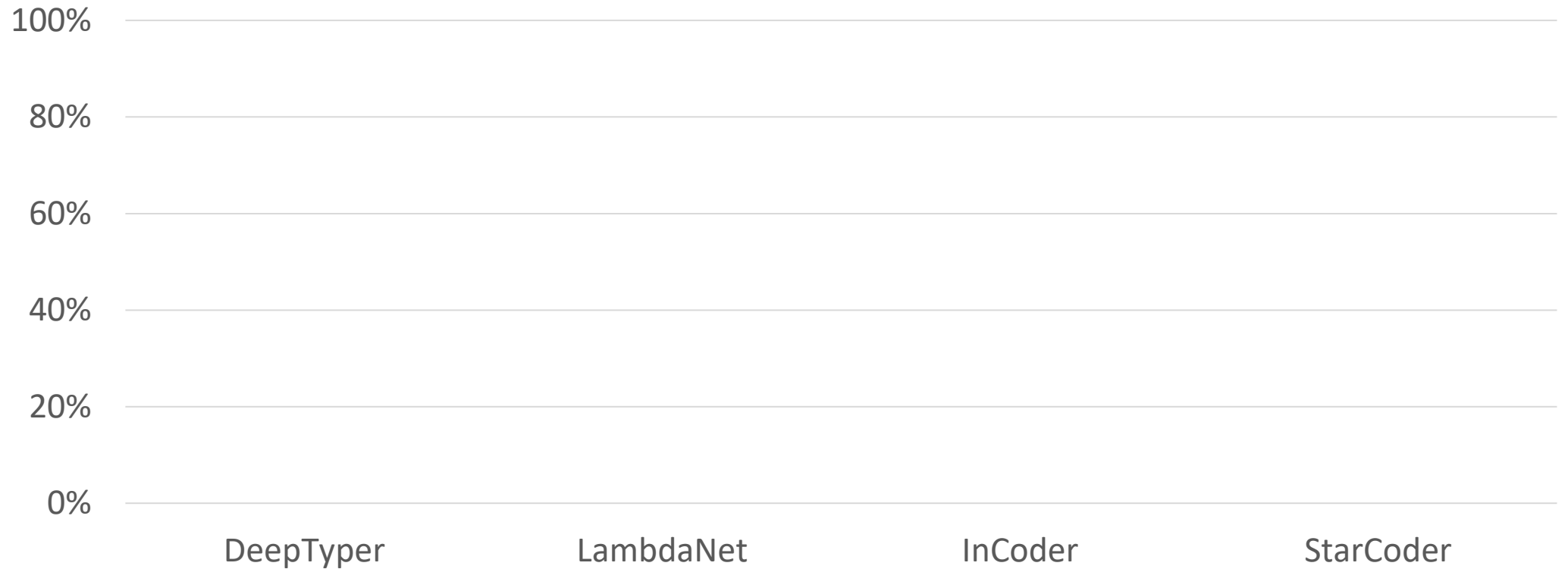
Percentage of packages that type check



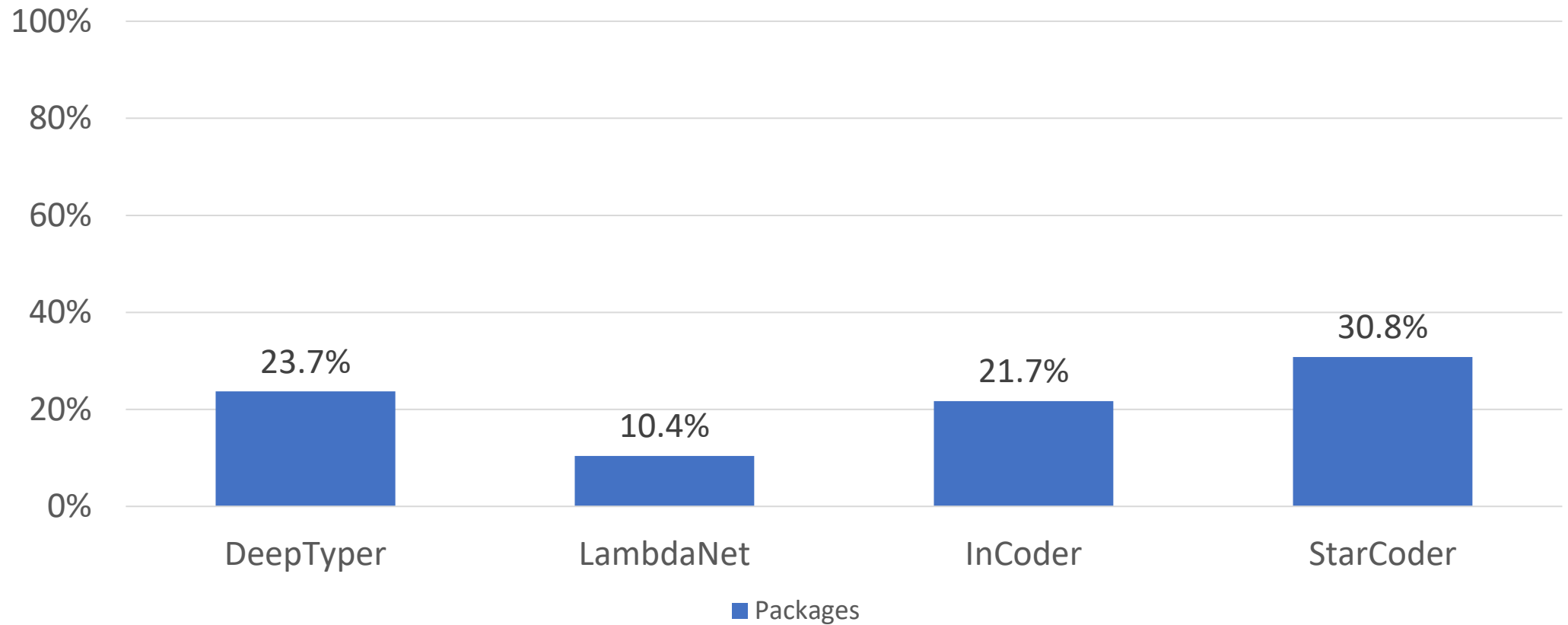
Percentage of packages that type check



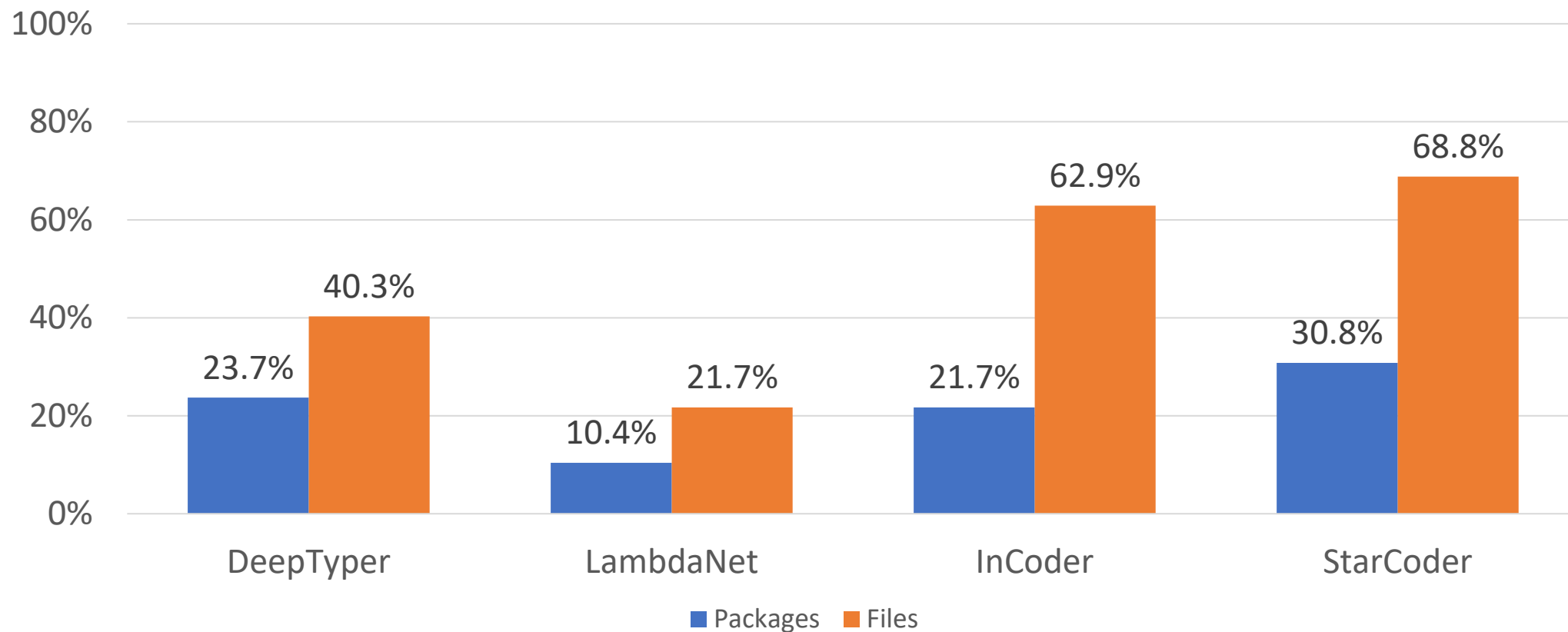
Percentage of packages that type check



Percentage of packages that type check

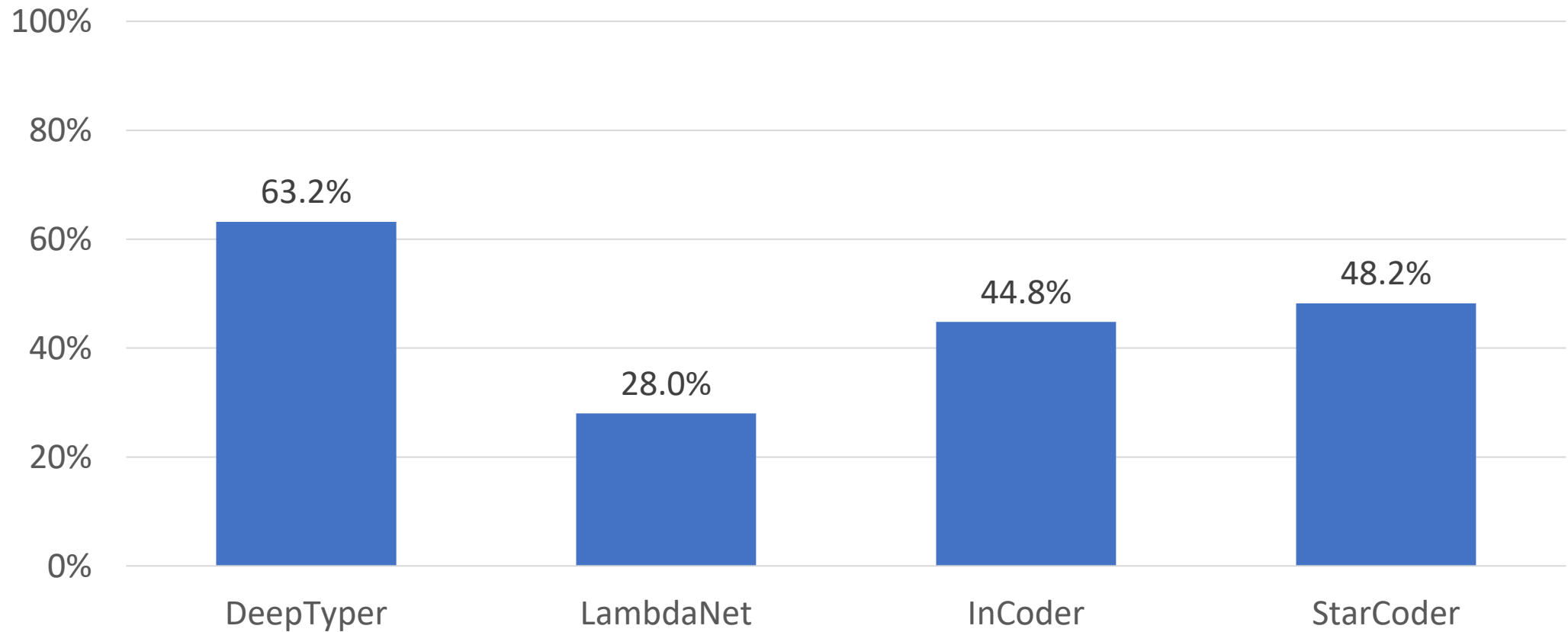


Percentage of packages/files that type check



Percentage of trivial annotations (in files that type check)

Percentage of trivial annotations (in files that type check)



Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by **predicting type annotations** and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training
[submitted to [COLM 2024](#)]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type
Definitions with Machine
Learning

Improving type prediction

Improving type prediction

Dataset
quality

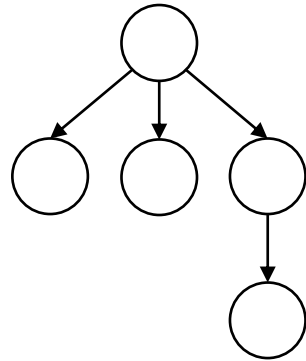
TypeScript
dataset

Improving type prediction

Dataset
quality

Program
decomposition

TypeScript
dataset

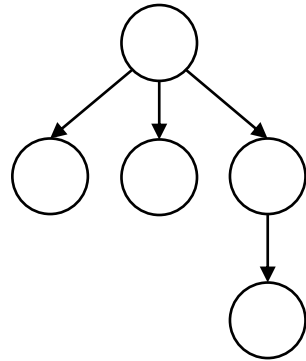


Improving type prediction

Dataset
quality

TypeScript
dataset

Program
decomposition



Fill-in-the-type
training

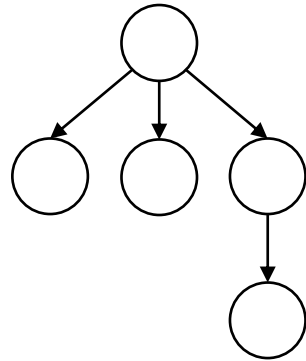
```
function f(x: _hole_) {  
  return x + 1;  
}
```

Improving type prediction

Dataset
quality

TypeScript
dataset

Program
decomposition



Fill-in-the-type
training

```
function f(x: _hole_) {  
  return x + 1;  
}
```

Program
typedness

```
function f(x: any) {  
  return x + 1;  
}
```

Dataset quality

Dataset quality

```
function f(x) {  
    return x + 1;  
}
```

~~~~~

# Dataset quality

```
function f(x) {  
    return x + 1;  
}
```

~~~~~

```
export default {  
    group: "typography",  
    currentPage: 2  
}
```


Dataset quality

```
function f(x) {  
    return x + 1;  
}
```

~~~~~

```
export default {  
    group: "typography",  
    currentPage: 2  
}
```

```
export const TabIcons = [  
    'tab', 'code-braces', 'tags', 'target'  
]  
export function getTabIcon(tabType: number) {  
    return TabIcons[tabType];  
}
```

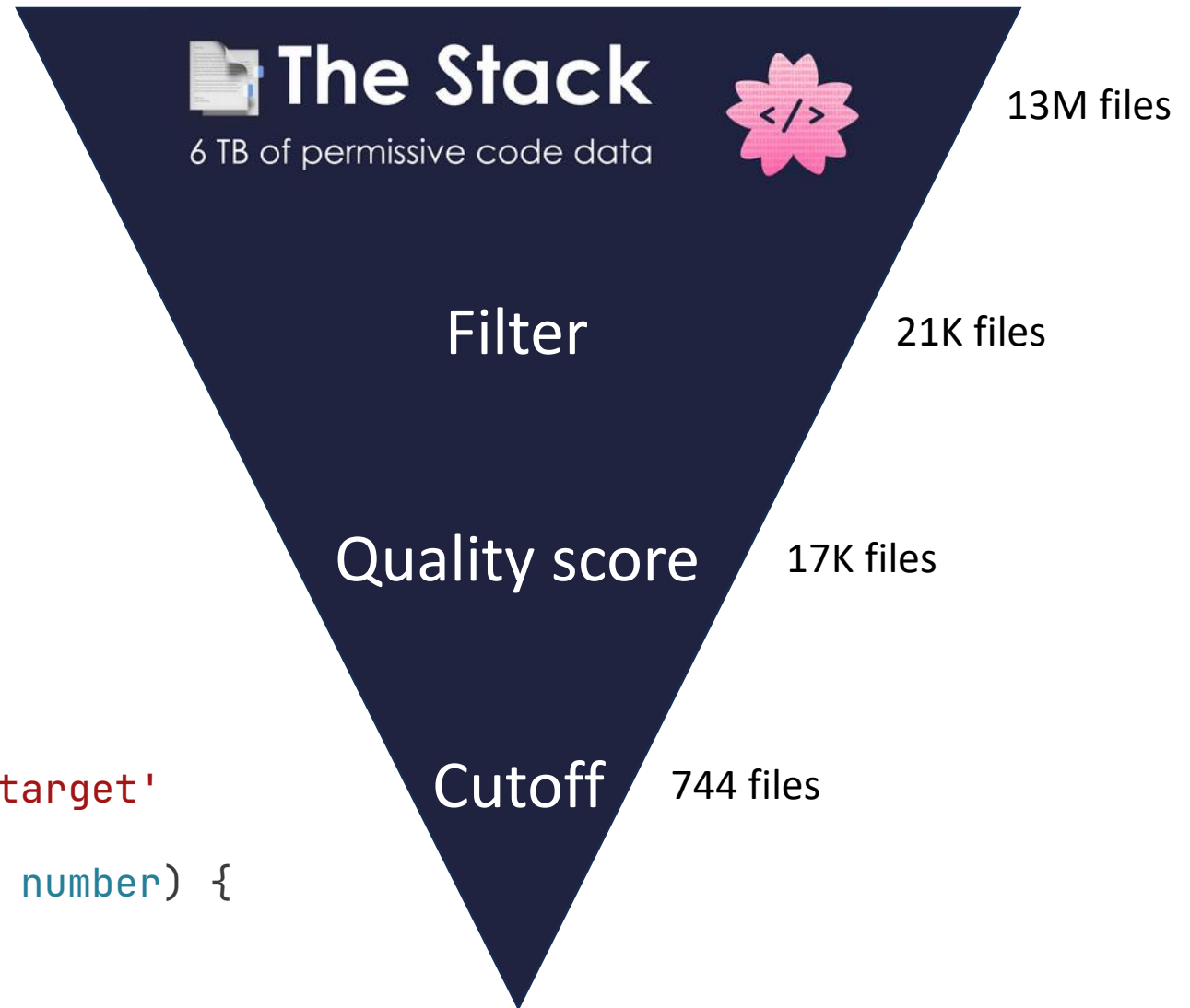
# Dataset quality

```
function f(x) {  
  return x + 1;  
}
```

~~~~~

```
export default {  
  group: "typography",  
  currentPage: 2  
}
```

```
export const TabIcons = [  
  'tab', 'code-braces', 'tags', 'target'  
]  
export function getTabIcon(tabType: number) {  
  return TabIcons[tabType];  
}
```



Program decomposition

```
let greeting = "Hello";
let suffix = "!";

// Produces a greeting for the given name
const hello = (name) => {
  return greeting + " " + name;
};

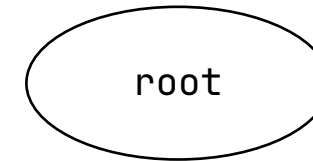
function helloGen(name) {
  const helloHelper = () => {
    return hello(name) + suffix;
  };
  return helloHelper;
}
```

Program decomposition

```
let greeting = "Hello";
let suffix = "!";

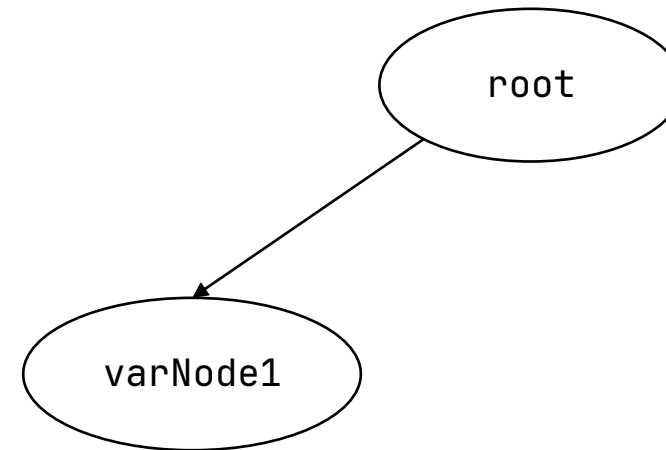
// Produces a greeting for the given name
const hello = (name) => {
  return greeting + " " + name;
};

function helloGen(name) {
  const helloHelper = () => {
    return hello(name) + suffix;
  };
  return helloHelper;
}
```



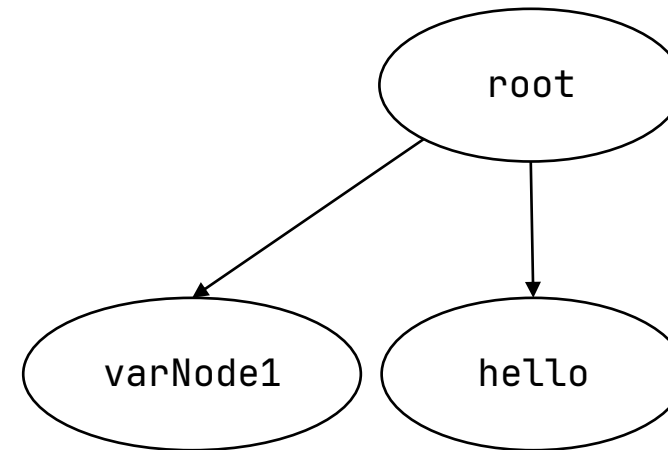
Program decomposition

```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
  return greeting + " " + name;  
};  
  
function helloGen(name) {  
  const helloHelper = () => {  
    return hello(name) + suffix;  
  };  
  return helloHelper;  
}
```



Program decomposition

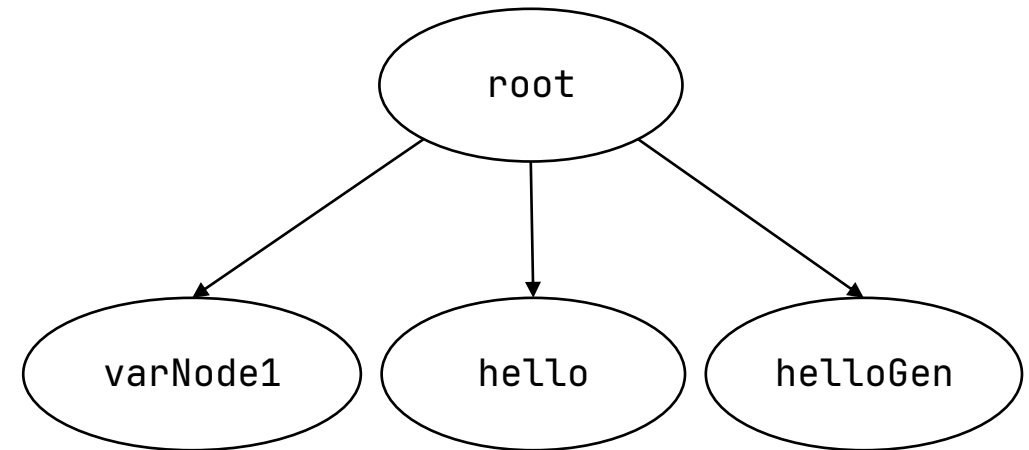
```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
    return greeting + " " + name;  
};  
  
function helloGen(name) {  
    const helloHelper = () => {  
        return hello(name) + suffix;  
    };  
    return helloHelper;  
}
```



Program decomposition

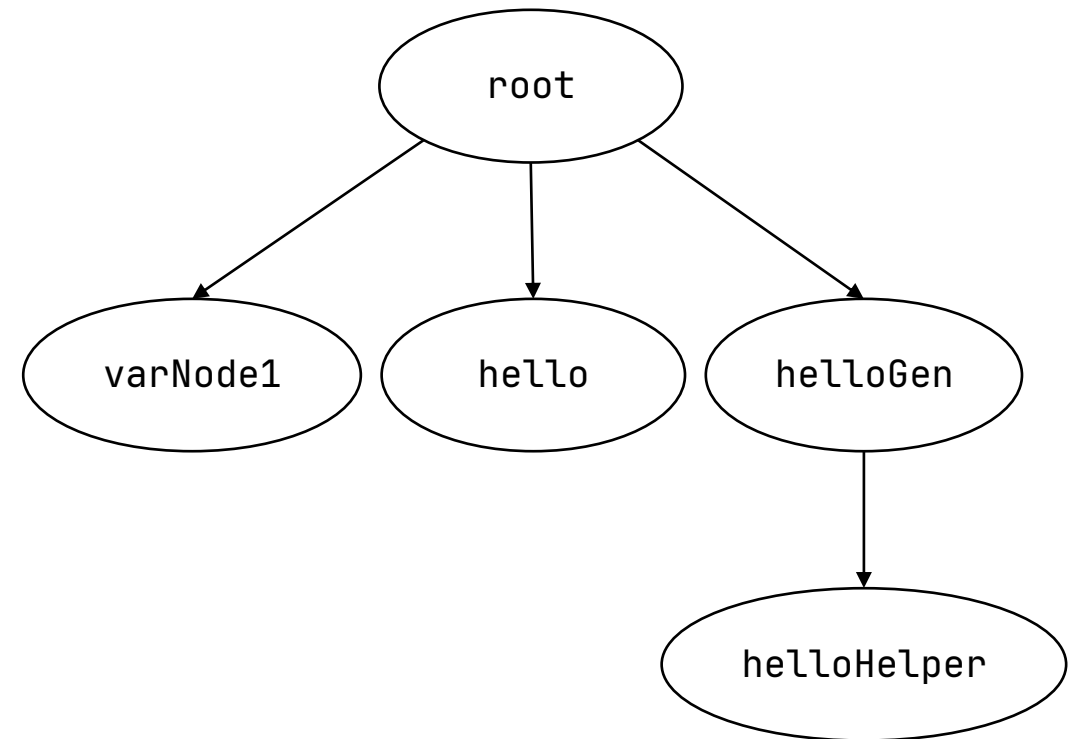
```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
  return greeting + " " + name;  
};
```

```
function helloGen(name) {  
  const helloHelper = () => {  
    return hello(name) + suffix;  
  };  
  return helloHelper;  
}
```



Program decomposition

```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
  return greeting + " " + name;  
};  
  
function helloGen(name) {  
  const helloHelper = () => {  
    return hello(name) + suffix;  
  };  
  return helloHelper;  
}
```

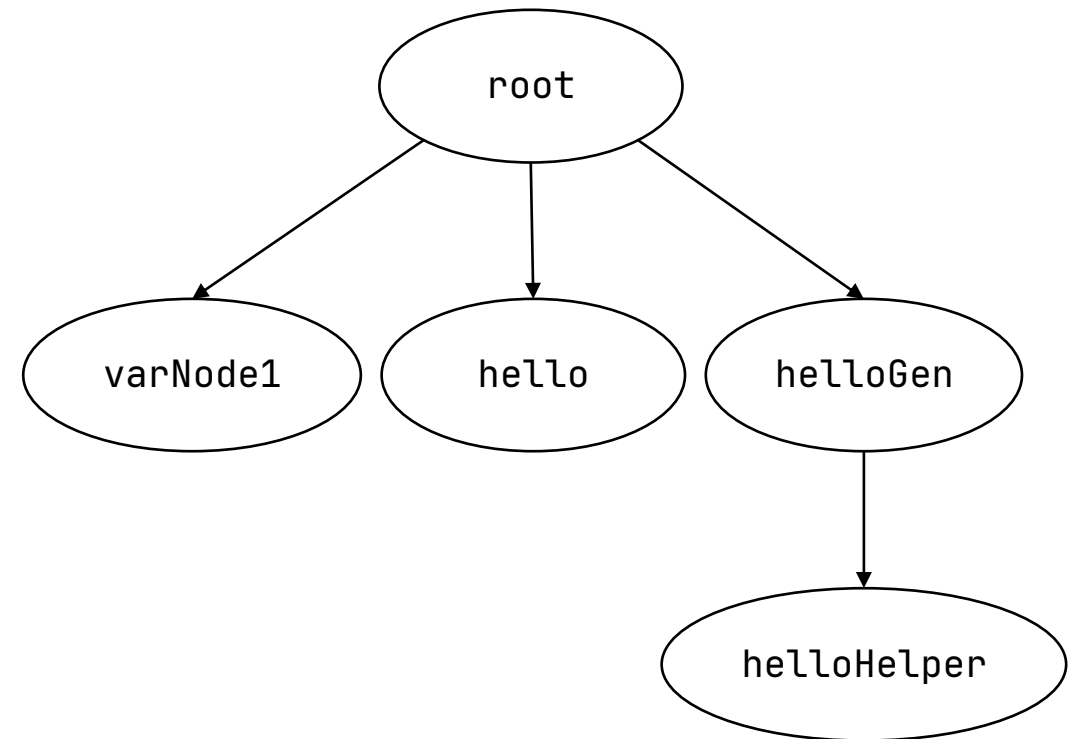


Program decomposition

```
let greeting = "Hello";
let suffix = "!";

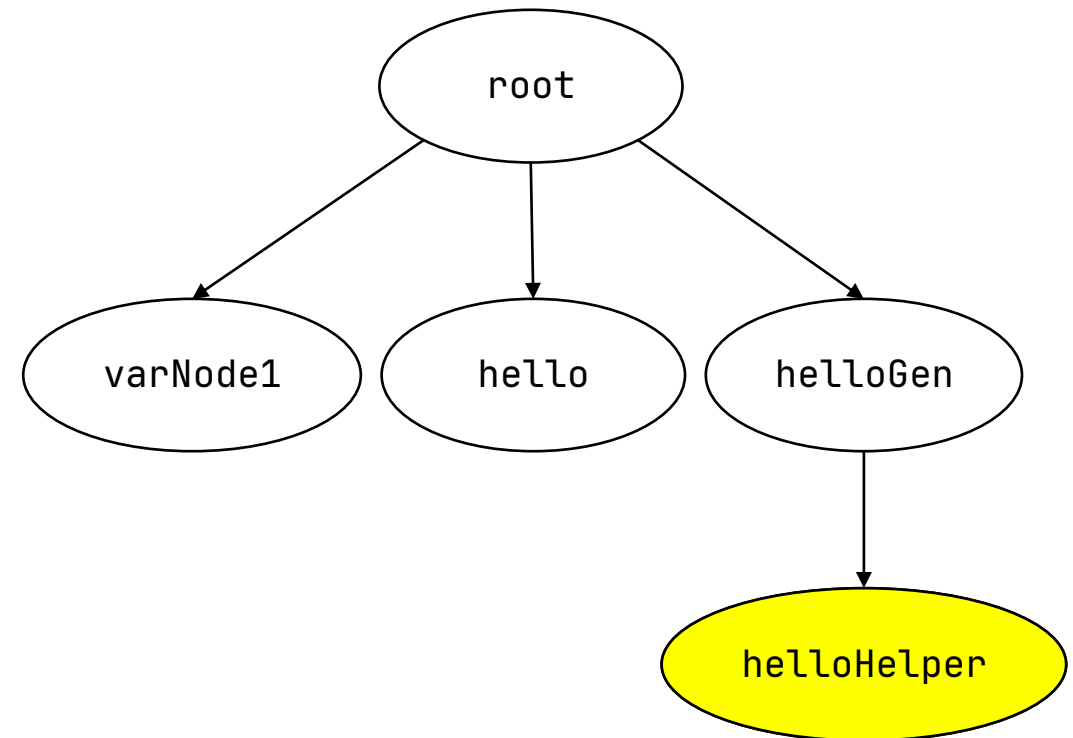
// Produces a greeting for the given name
const hello = (name) => {
  return greeting + " " + name;
};

function helloGen(name) {
  const helloHelper = () => {
    return hello(name) + suffix;
  };
  return helloHelper;
}
```



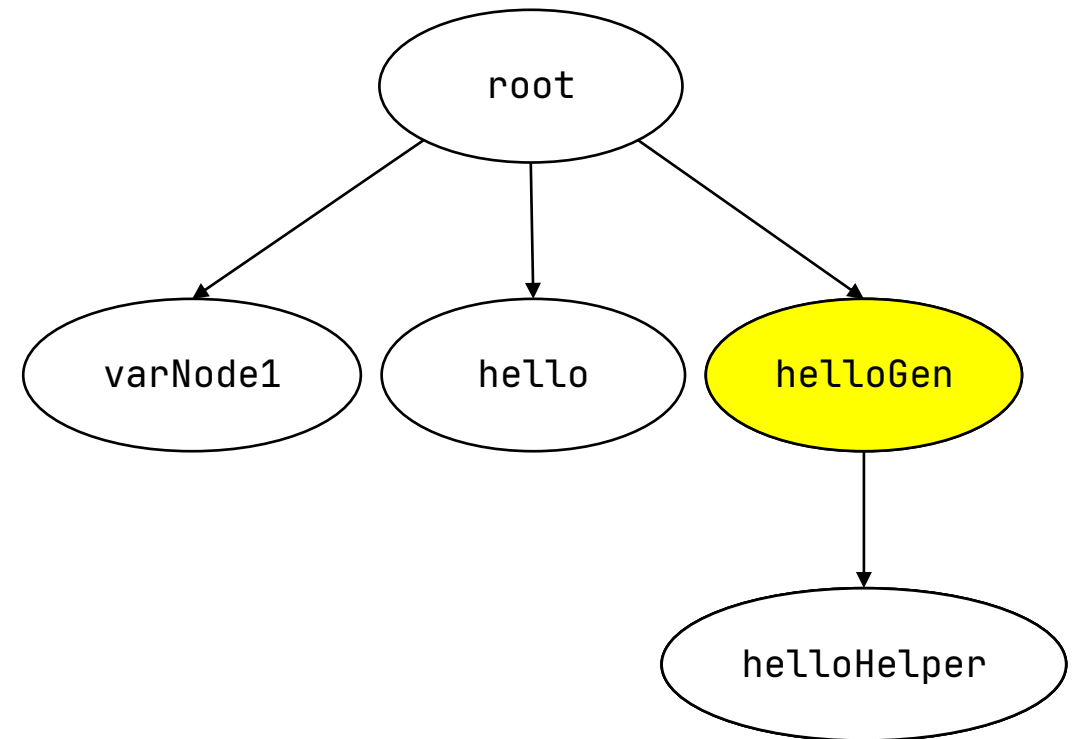
Program decomposition

```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
  return greeting + " " + name;  
};  
  
function helloGen(name) {  
  const helloHelper = (): string => {  
    return hello(name) + suffix;  
  };  
  return helloHelper;  
}
```



Program decomposition

```
let greeting = "Hello";  
let suffix = "!";  
  
// Produces a greeting for the given name  
const hello = (name) => {  
  return greeting + " " + name;  
};  
  
function helloGen(name): () => string {  
  const helloHelper = (): string => {  
    return hello(name) + suffix;  
  };  
  return helloHelper;  
}
```



Fill-in-the-type training

```
function sum_list(l: _hole_) {  
  let sum = 0;  
  for (let i = 0; i < l.length; i++) {  
    sum += l[i];  
  }  
  return sum;  
}
```

Fill-in-the-type training

```
function sum_list(l: any[]): number {  
  if (l.length === 0) {  
    throw 'Empty list!';  
  }  
  if (l.length === 1) {  
    return l[0];  
  }  
  return sum) {  
    let sum = 0;  
    for (let i = 0; i < l.length; i++) {  
      sum += l[i];  
    }  
    return sum;  
  }  
}
```

Fill-in-the-type training

Fill in the middle

```
function sumThree(a: number, b: number, c: number): number {  
    return a + b + c;  
}
```

Fill-in-the-type training

Fill in the middle

```
function sumThree(a: number, b: number, c: number): number {  
    return a + b + c;  
}
```

Fill-in-the-type training

Fill in the middle

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>}  
<fim_middle>number, c: number): number {  
    return a + b + c;
```


Fill-in-the-type training

Fill in the middle

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>}  
<fim_middle>number, c: number): number {  
    return a + b + c;
```

Fill in the type

```
function sumThree(a: number, b: number, c: number): number {  
    return a + b + c;  
}
```

Fill-in-the-type training

Fill in the middle

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>}  
<fim_middle>number, c: number): number {  
    return a + b + c;  
}
```

Fill in the type

```
function sumThree(a: number, b: number, c: number): number {  
    return a + b + c;  
}
```

Fill-in-the-type training

Fill in the middle

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>}  
<fim_middle>number, c: number): number {  
    return a + b + c;
```

Fill in the type

```
function sumThree(a: number, b: number, c) {  
    return a + b + c;  
}
```

Fill-in-the-type training

Fill in the middle

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>}  
<fim_middle>number, c: number): number {  
    return a + b + c;
```

Fill in the type

```
<fim_prefix>function sumThree(a: number, b:  
<fim_suffix>, c) {  
    return a + b + c;  
}<fim_middle>number
```

Program typedness

Both programs type check

```
function f(x: any) {  
  return x + 1;  
}
```

```
function f(x: number) {  
  return x + 1;  
}
```

Program typedness

Both programs type check

```
function f(x: any) {  
  return x + 1;  
}
```

```
function f(x: number) {  
  return x + 1;  
}
```

Type annotation	Score
unknown	1.0
any	0.5
Function	0.5
undefined	0.2
null	0.2

Program typedness

Both programs type check

```
function f(x: any) {  
  return x + 1;  
}
```

Score: 500

```
function f(x: number) {  
  return x + 1;  
}
```

Score: 0

Type annotation	Score
unknown	1.0
any	0.5
Function	0.5
undefined	0.2
null	0.2

Program typedness

Both programs type check

```
function f(x: any) {  
  return x + 1;  
}
```

Score: 500

```
function f(x: number) {  
  return x + 1;  
}
```

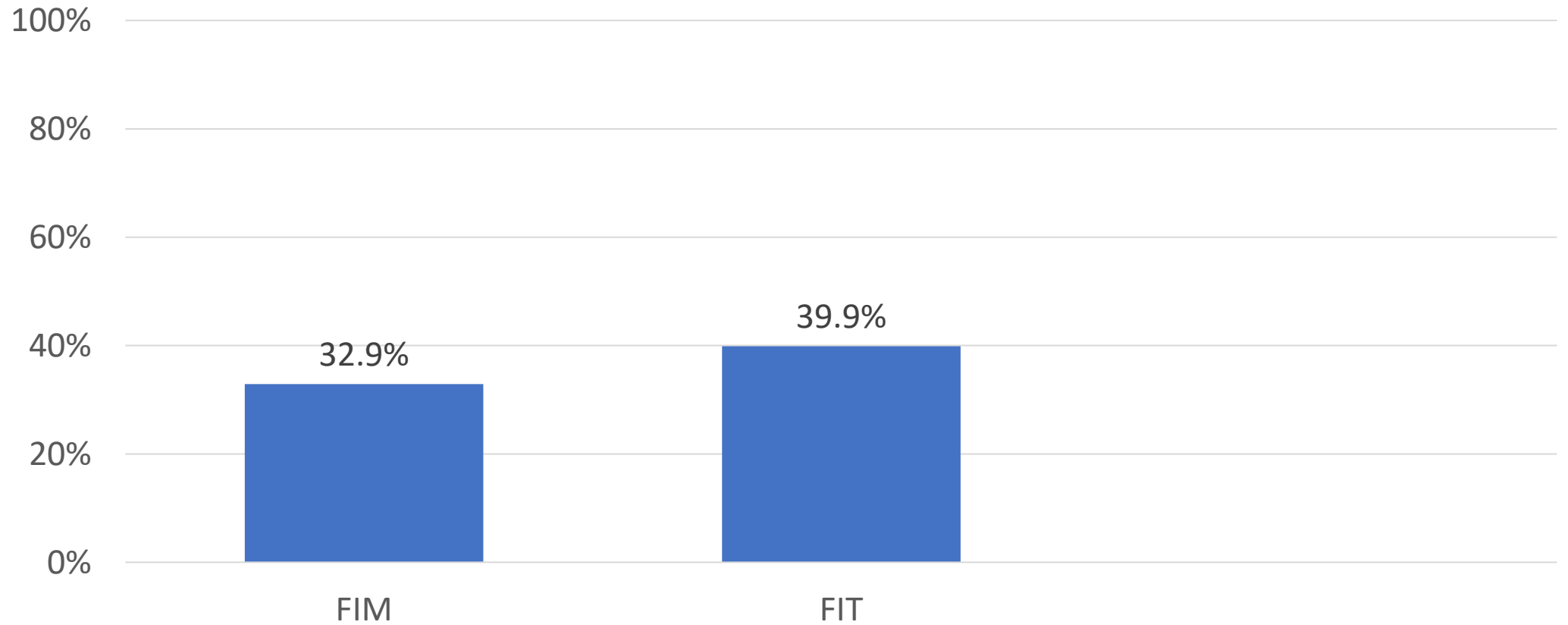
Score: 0

Type annotation	Score
unknown	1.0
any	0.5
Function	0.5
undefined	0.2
null	0.2

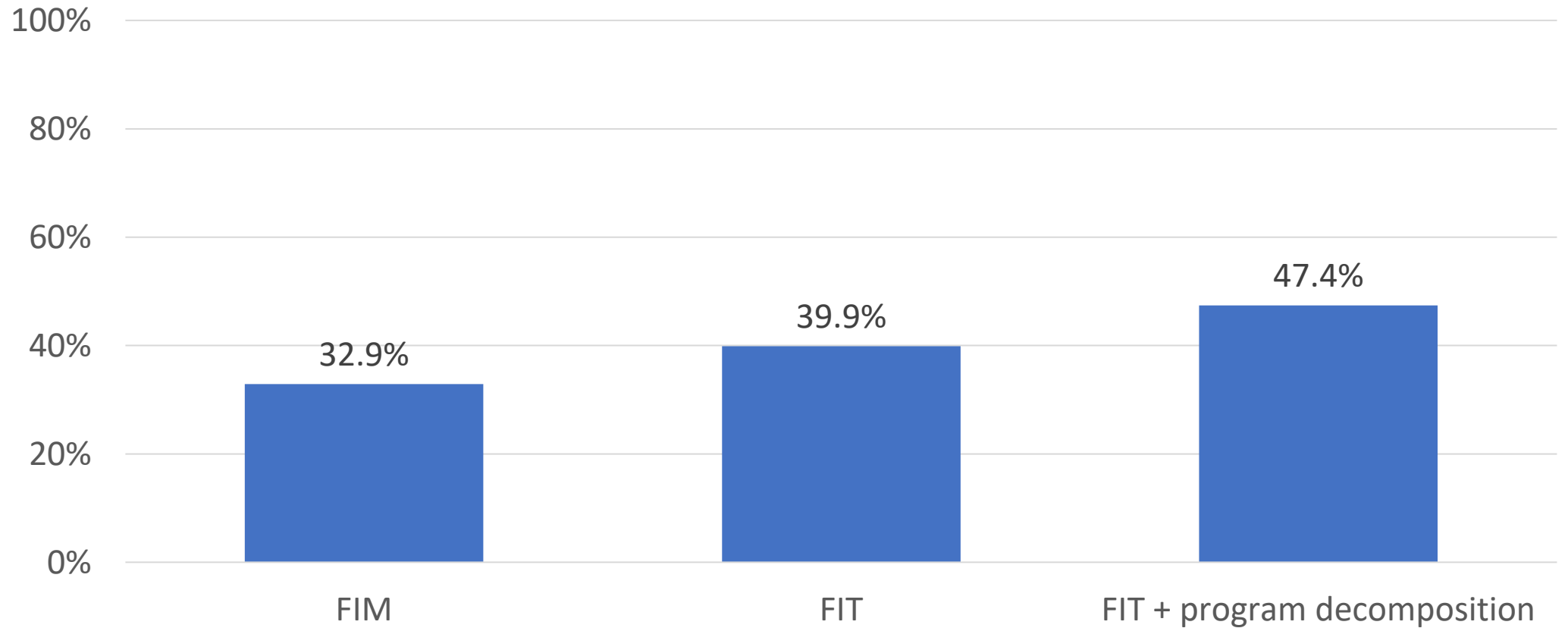
We also use this metric during type prediction

Percentage of files that type check

Percentage of files that type check

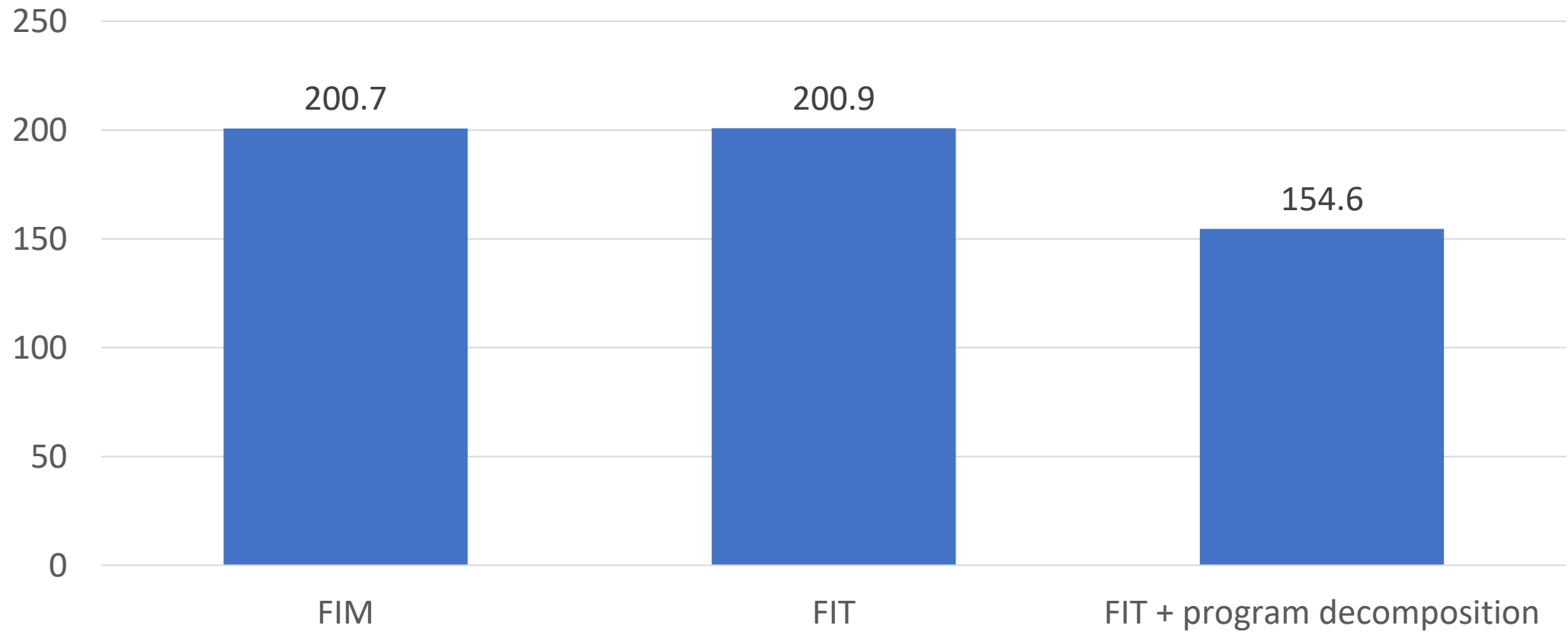


Percentage of files that type check



Typedness scores

Typedness scores



Thesis

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and **generating type definitions**.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training
[submitted to [COLM 2024](#)]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type
Definitions with Machine
Learning

Problem definition

```
function dist(p1, p2) {  
  const dx = p2.x - p1.x;  
  const dy = p2.y - p1.y;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```

Problem definition

```
function dist(p1: Point, p2: Point) {  
  const dx = p2.x - p1.x;  
  const dy = p2.y - p1.y;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```


Problem definition

```
function dist(p1: Point, p2: Point) {  
  const dx = p2.x - p1.x;  
  const dy = p2.y - p1.y;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```

```
interface Point {  
  x: number,  
  y: number  
}
```

Approach: single-step migration

Approach: single-step migration

```
<commit_before>...  
<commit_msg>...  
<commit_after>...
```

Approach: single-step migration

```
<commit_before>...  
<commit_msg>...  
<commit_after>interface Point {  
    x: number,  
    y: number  
}  
  
function dist(p1: Point, p2: Point) {  
    const dx = p2.x - p1.x;  
    const dy = p2.y - p1.y;  
    return Math.sqrt(dx*dx + dy*dy);  
}
```

Approach: single-step migration

```
<commit_before>function dist(p1, p2) {  
  const dx = p2.x - p1.x;  
  const dy = p2.y - p1.y;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```

```
<commit_msg>...  
<commit_after>interface Point {  
  x: number,  
  y: number  
}
```

```
function dist(p1: Point, p2: Point) {  
  const dx = p2.x - p1.x;  
  const dy = p2.y - p1.y;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```

Approach: single-step migration

```
<commit_before>function dist(p1, p2) {  
    const dx = p2.x - p1.x;  
    const dy = p2.y - p1.y;  
    return Math.sqrt(dx*dx + dy*dy);  
}
```

```
<commit_msg>Add type annotations and interfaces
```

```
<commit_after>interface Point {  
    x: number,  
    y: number  
}
```

```
function dist(p1: Point, p2: Point) {  
    const dx = p2.x - p1.x;  
    const dy = p2.y - p1.y;  
    return Math.sqrt(dx*dx + dy*dy);  
}
```

Approach: single-step migration

```
<commit_before>function dist(p1, p2) {  
    const dx = p2.x - p1.x;  
    const dy = p2.y - p1.y;  
    return Math.sqrt(dx*dx + dy*dy);  
}
```

```
<commit_msg>Add type annotations and interfaces  
<commit_after>
```

Approach: multi-step migration, annotations

Approach: multi-step migration, annotations

```
<commit_before>...  
<commit_msg>...  
<commit_after>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

Approach: multi-step migration, annotations

```
<commit_before>function circleArea(c) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r) {  
    return r.width * r.height;  
}  
<commit_msg>...  
<commit_after>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

Approach: multi-step migration, annotations

```
<commit_before>function circleArea(c) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r) {  
    return r.width * r.height;  
}  
<commit_msg>Add type annotations  
<commit_after>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

Approach: multi-step migration, annotations

```
<commit_before>function circleArea(c) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r) {  
    return r.width * r.height;  
}  
<commit_msg>Add type annotations  
<commit_after>
```

Approach: multi-step migration, definitions

Approach: multi-step migration, definitions

```
<commit_before>...  
<commit_msg>...  
<commit_after>interface Circle {  
    position: Point;  
    radius: number;  
}  
function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

Approach: multi-step migration, definitions

```
<commit_before>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}  
<commit_msg>...  
<commit_after>interface Circle {  
    position: Point;  
    radius: number;  
}  
function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

Approach: multi-step migration, definitions

```
<commit_before>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}  
<commit_msg>Add a type alias or interface for Circle  
<commit_after>interface Circle {  
    position: Point;  
    radius: number;  
}  
function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}
```

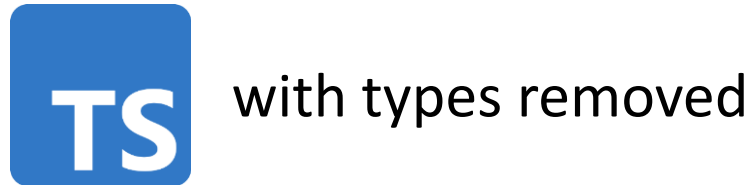

Approach: multi-step migration, definitions

```
<commit_before>function circleArea(c: Circle) {  
    return Math.PI * c.radius * c.radius;  
}  
function rectangleArea(r: Rectangle) {  
    return r.width * r.height;  
}  
<commit_msg>Add a type alias or interface for Circle  
<commit_after>
```

Evaluation datasets

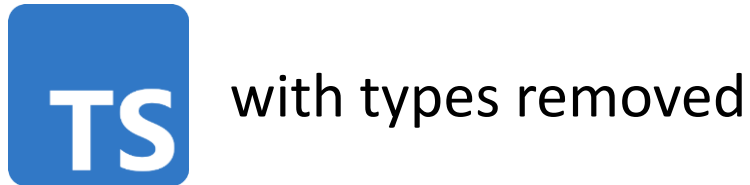
Evaluation datasets

UNTYPED

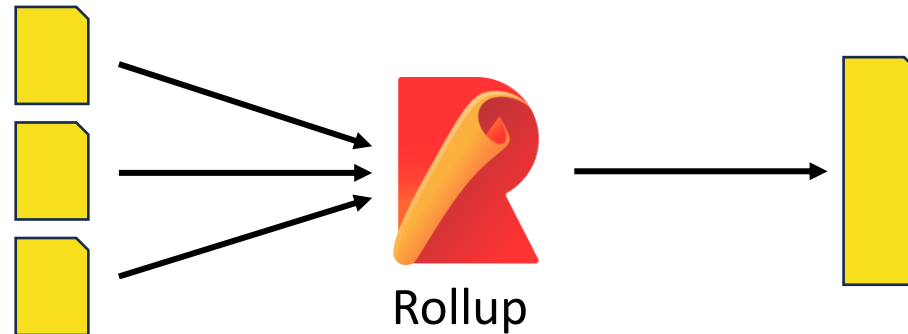


Evaluation datasets

UNTYPED

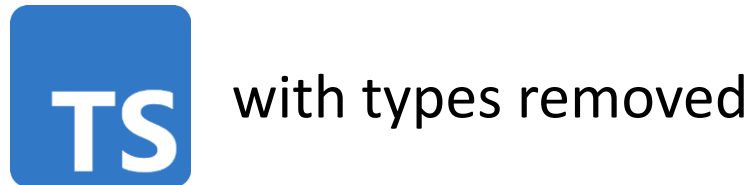


UNTYPED-HARD

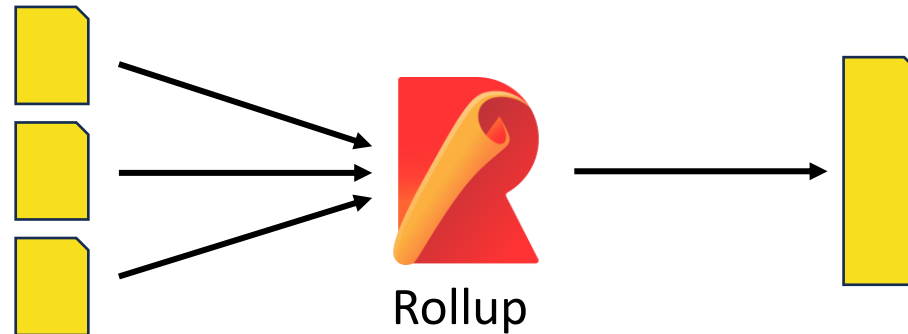


Evaluation datasets

UNTYPED

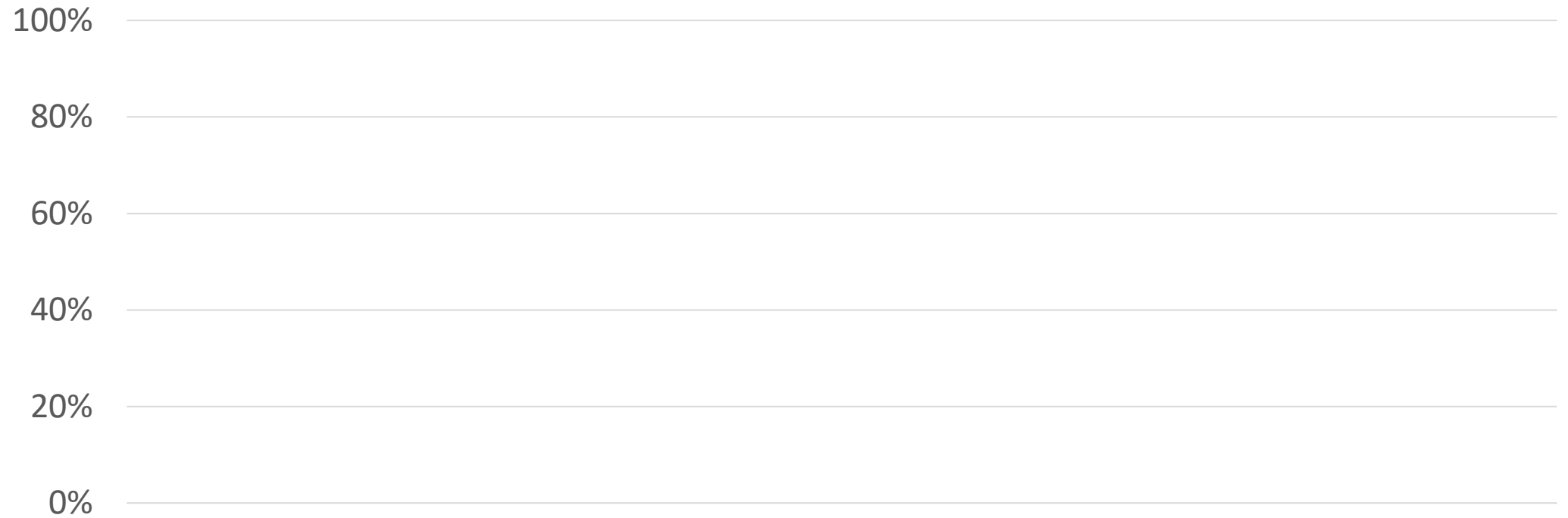


UNTYPED-HARD

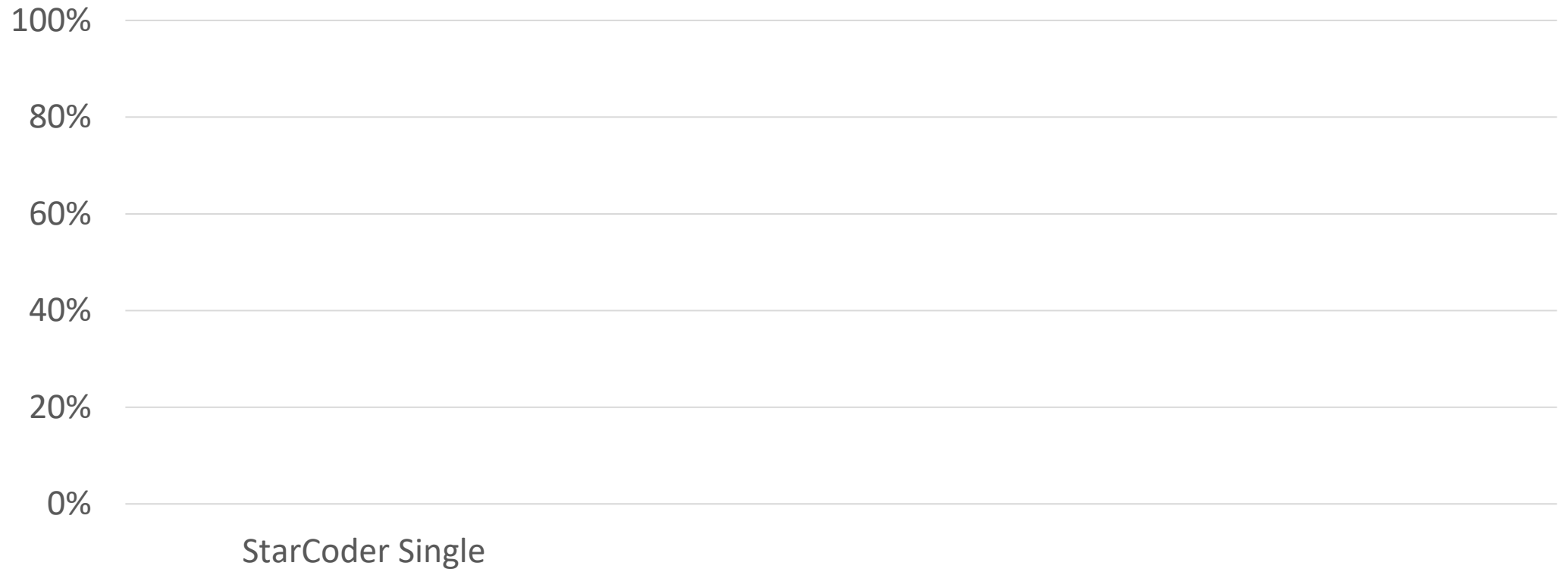


Dataset	Packages	Files	LOC	Functions
UNTYPED	50	50	6,339	455
UNTYPED-HARD	50	91	7,645	723

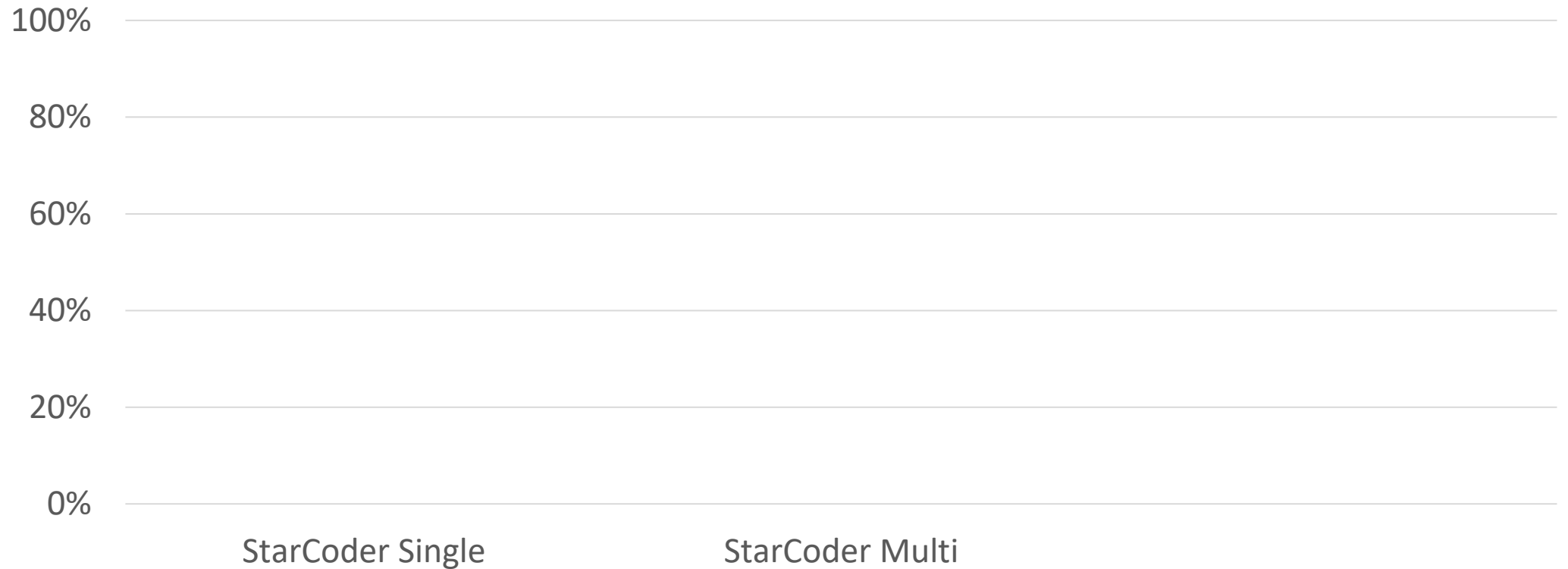
Percentage of files that type check



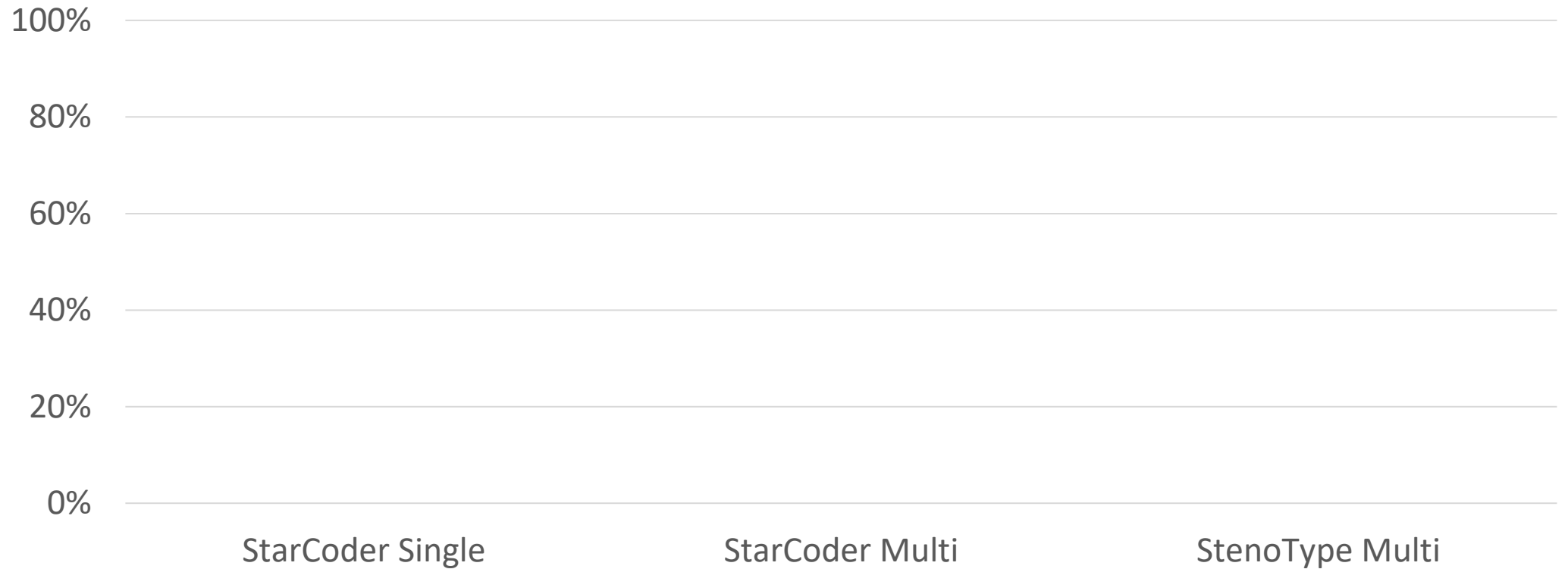
Percentage of files that type check



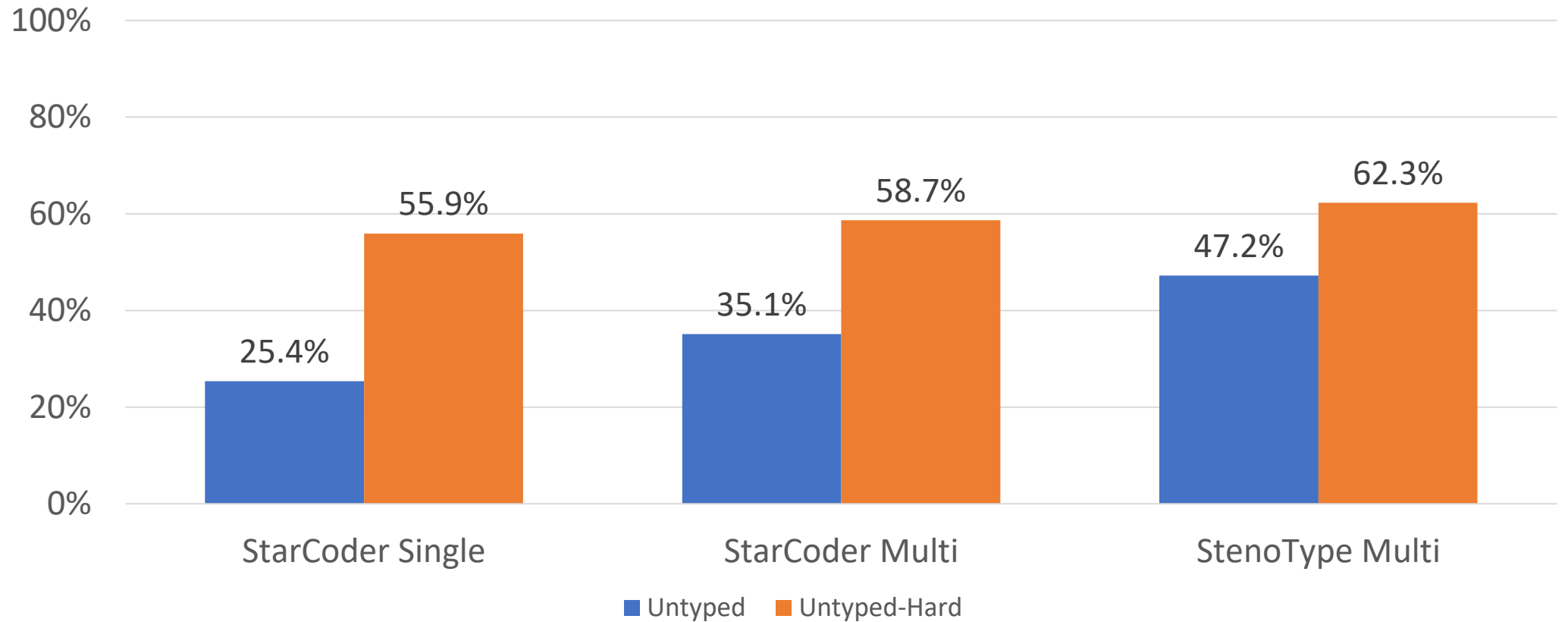
Percentage of files that type check



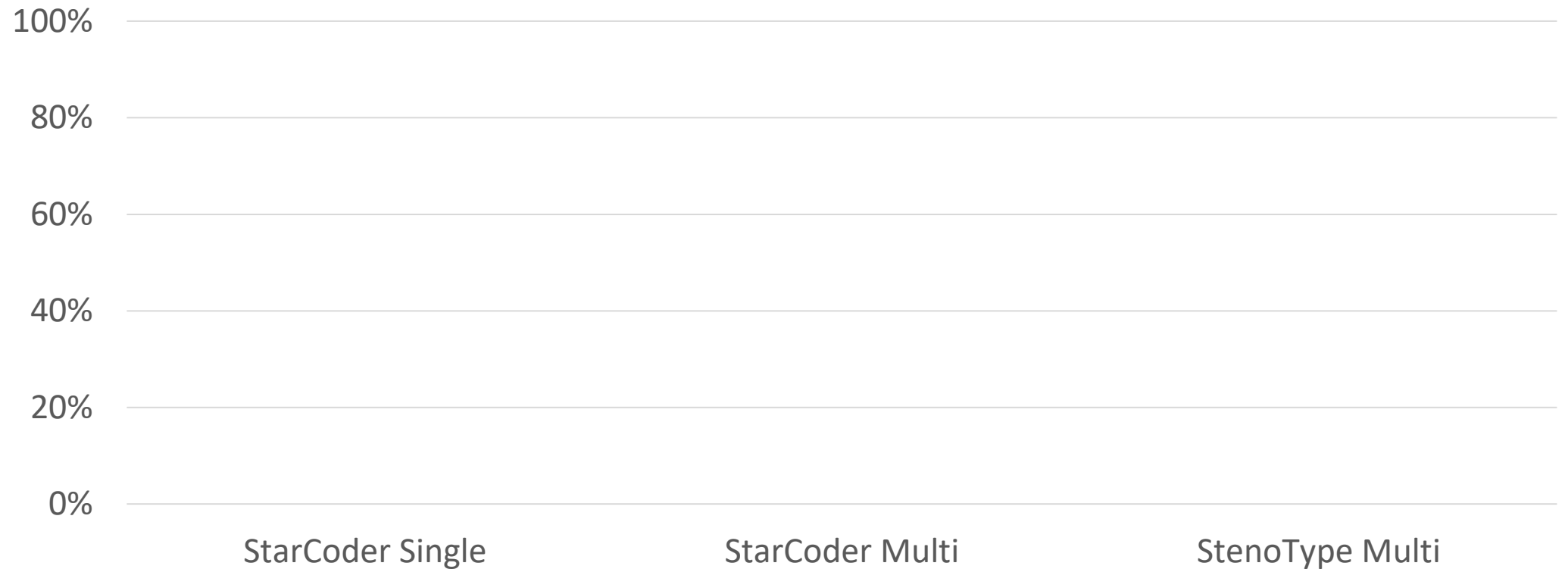
Percentage of files that type check



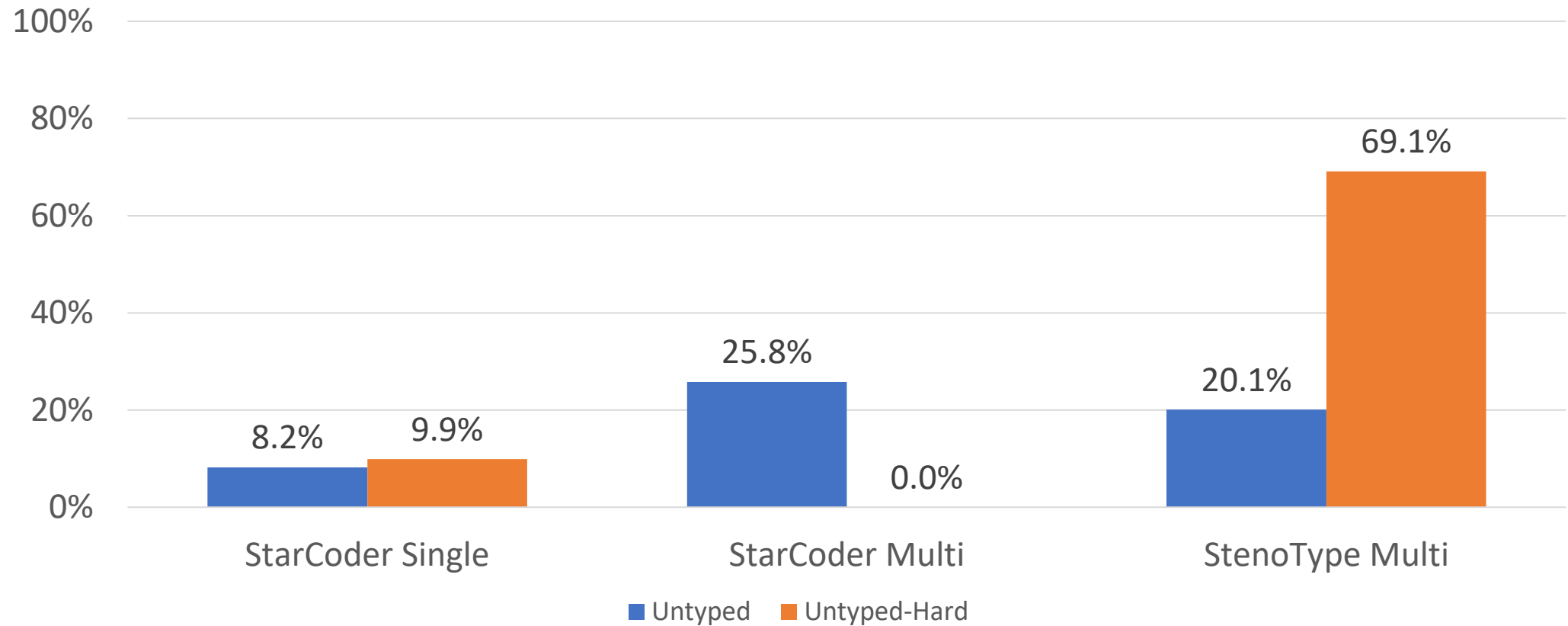
Percentage of files that type check



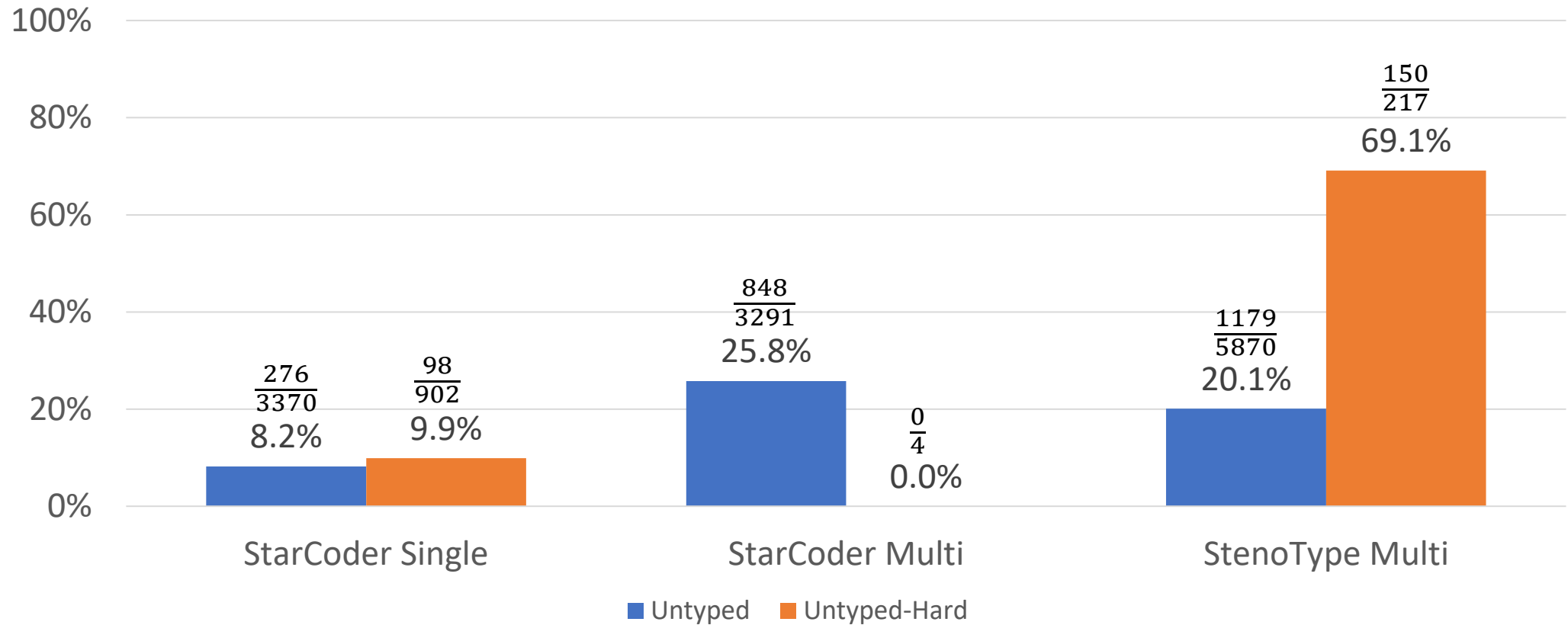
Percentage of trivial annotations (in files that type check)



Percentage of trivial annotations (in files that type check)

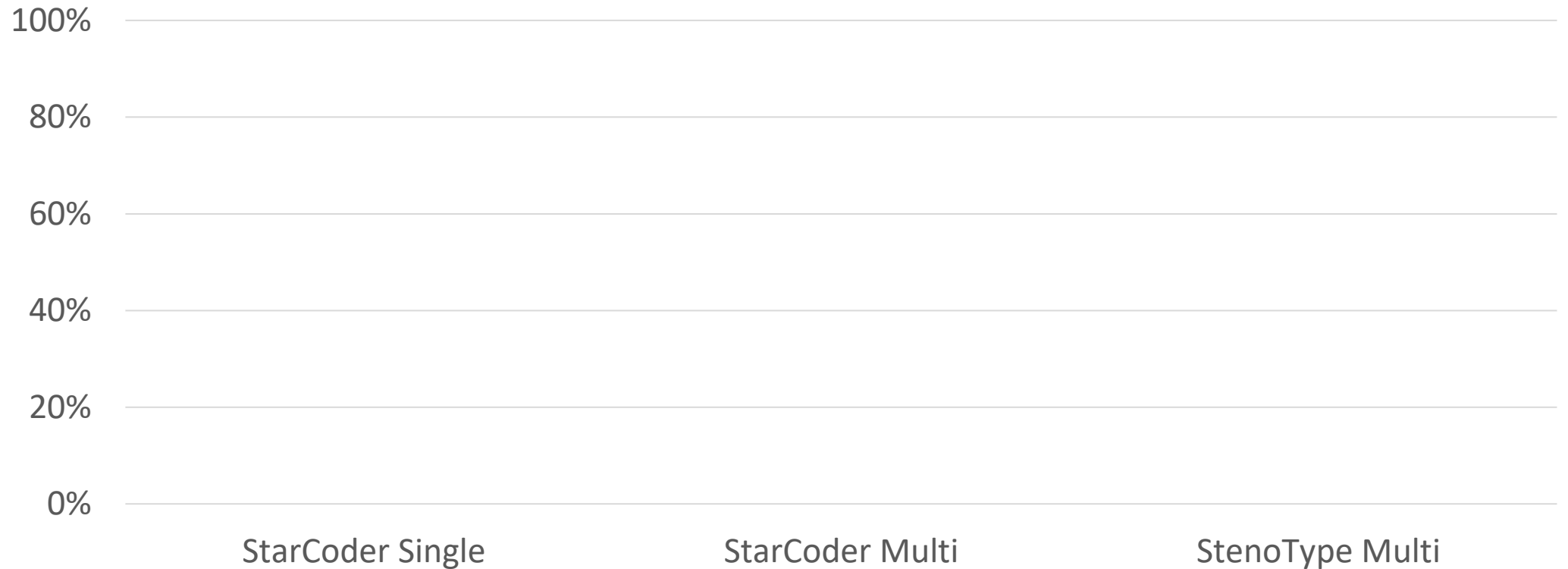


Percentage of trivial annotations (in files that type check)



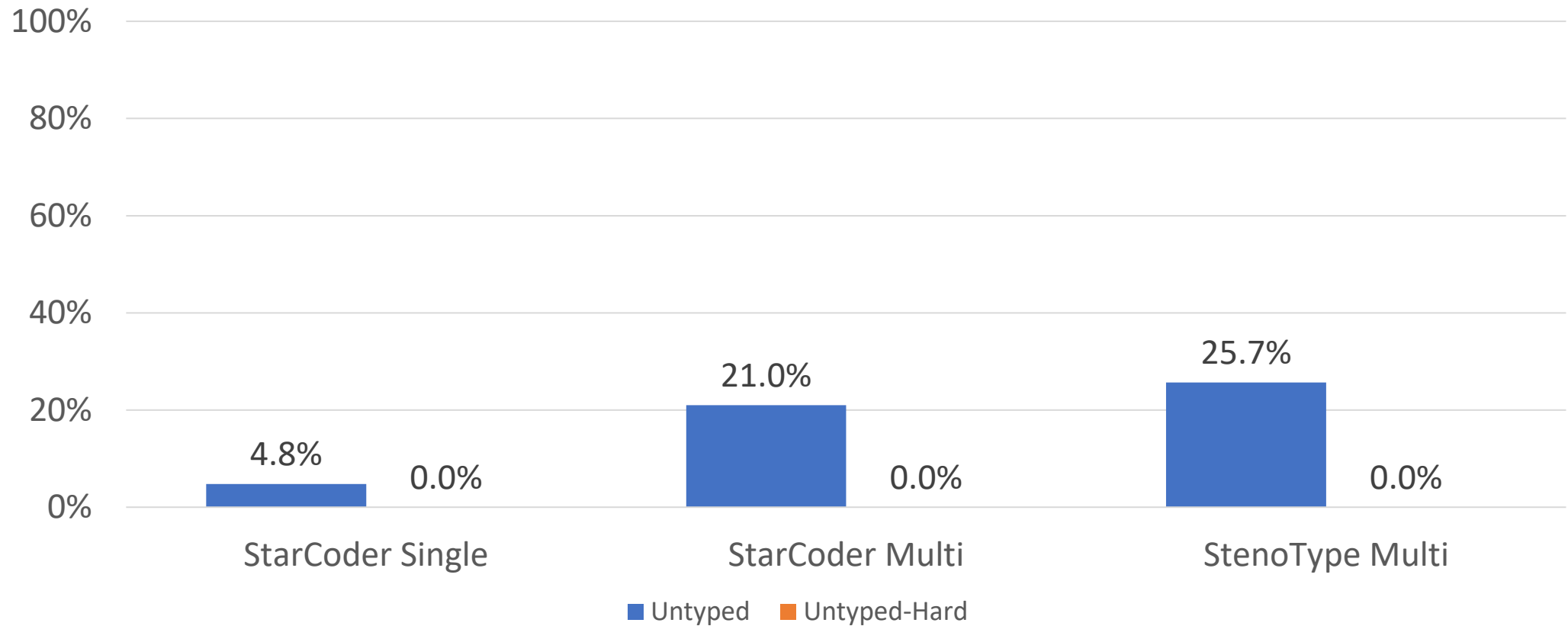
Percentage of files correctly migrated

Correct = type checks + no mutations + some types added



Percentage of files correctly migrated

Correct = type checks + no mutations + some types added



Future work

Future work

- Dataset quality

Future work

- Dataset quality
- Type prediction, revisited

Future work

- Dataset quality
- Type prediction, revisited
- Generating type definitions, revisited

Future work

- Dataset quality
- Type prediction, revisited
- Generating type definitions, revisited
- Fully automated type migration

Conclusion

Conclusion

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Conclusion

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]
Yee and Guha

Conclusion

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]

Yee and Guha

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training

[submitted to [COLM 2024](#)]

Cassano, Yee, Shinn, Guha, and Holtzen

Conclusion

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models
Produce TypeScript Types
That Type Check? [[ECOOP 2023](#)]

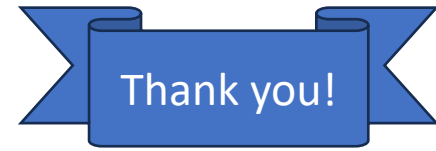
Yee and Guha

Type Prediction With
Program Decomposition and
Fill-in-the-Type Training

[submitted to [COLM 2024](#)]

Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type
Definitions with Machine
Learning



Conclusion

Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

Do Machine Learning Models Produce TypeScript Types That Type Check? [[ECOOP 2023](#)]

Yee and Guha

Type Prediction With Program Decomposition and Fill-in-the-Type Training

[submitted to [COLM 2024](#)]

Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions with Machine Learning