Predicting TypeScript Type Annotations and Definitions With Machine Learning

Ming-Ho Yee

Northeastern University

September 13, 2023

Thesis Proposal

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
other	

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
other	

Large language models for code

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

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

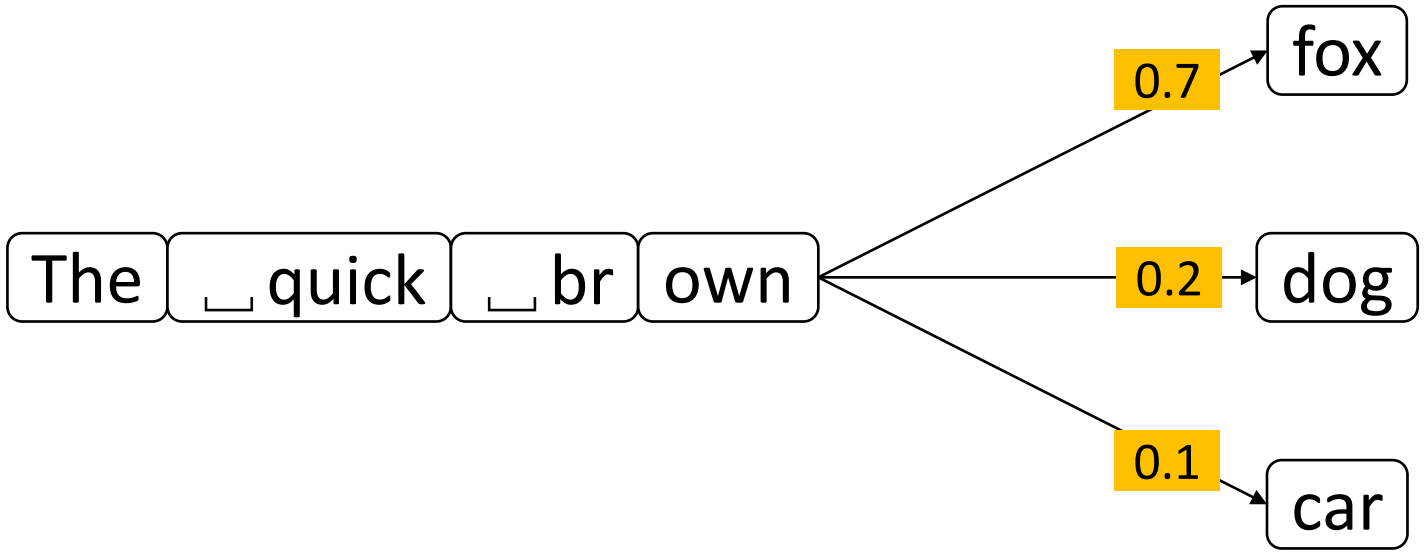
Large language models (LLMs)

Large language models (LLMs)

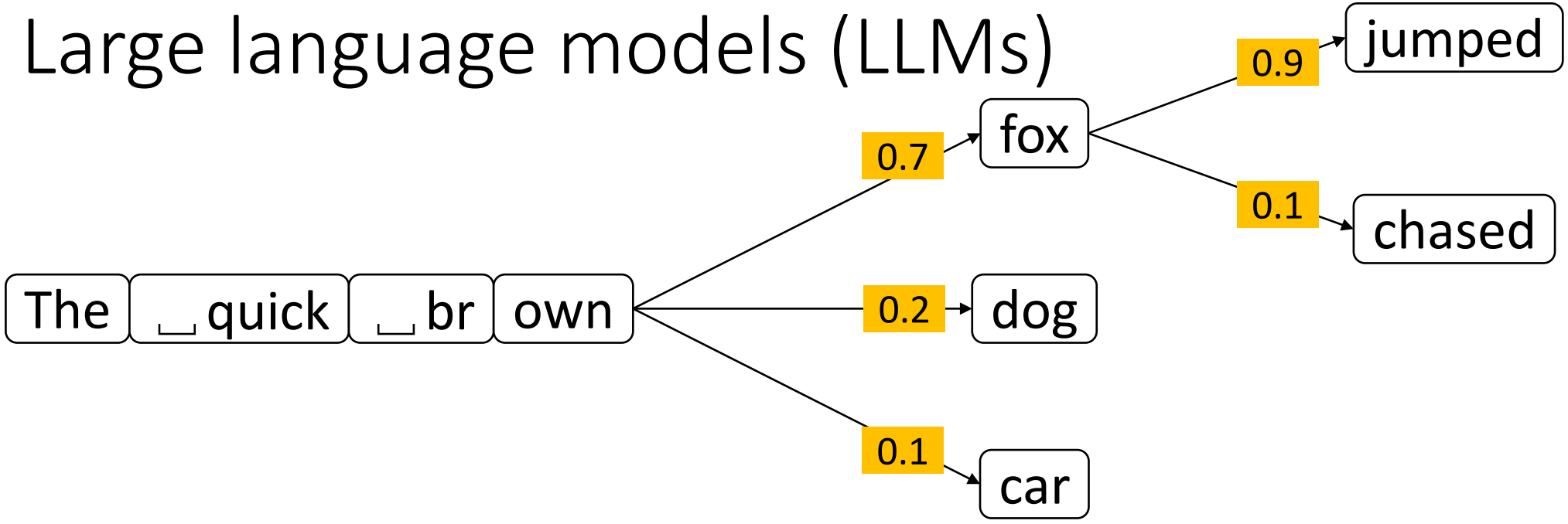
The quick brown

Large language models (LLMs)

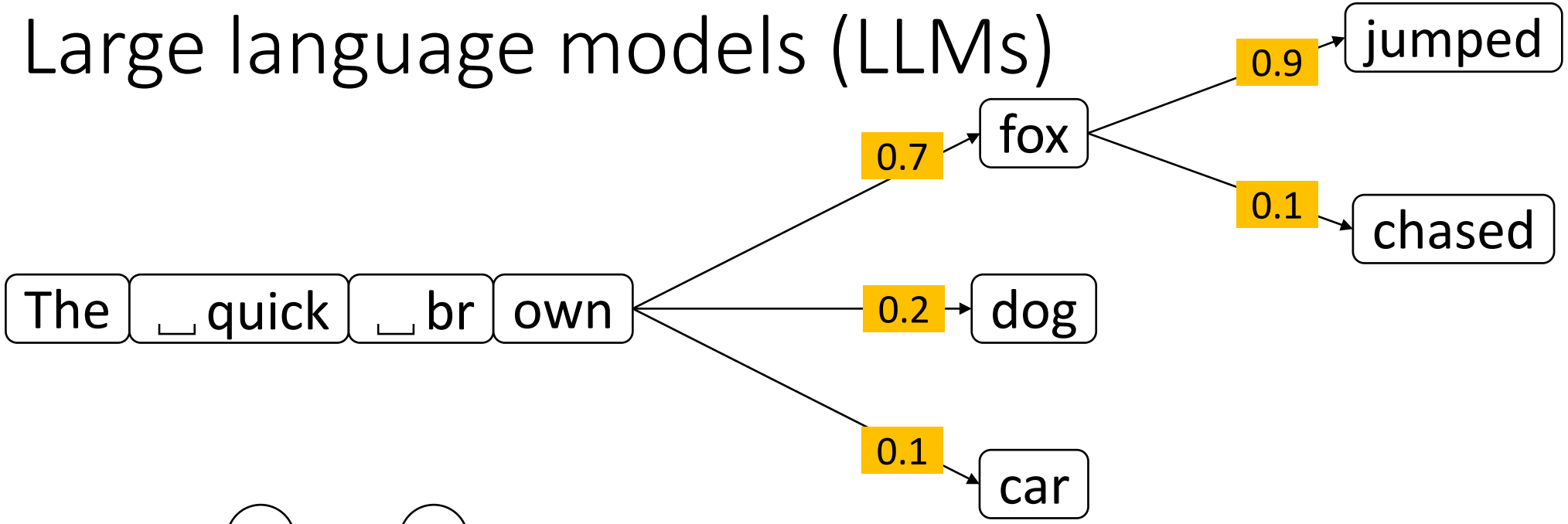
Large language models (LLMs)

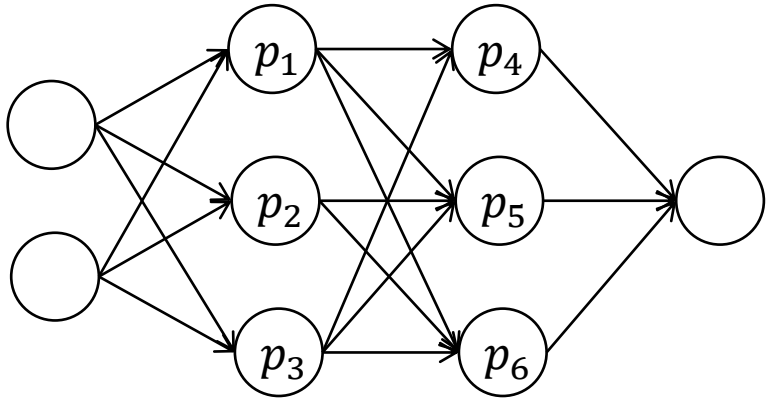


Large language models (LLMs)

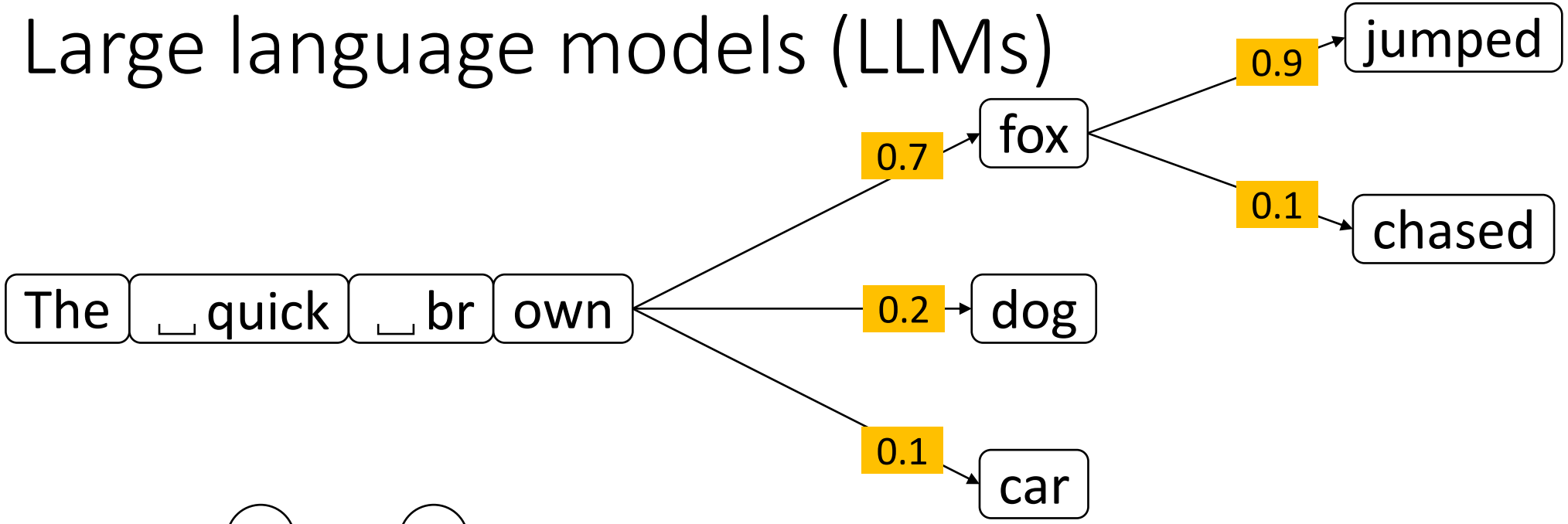


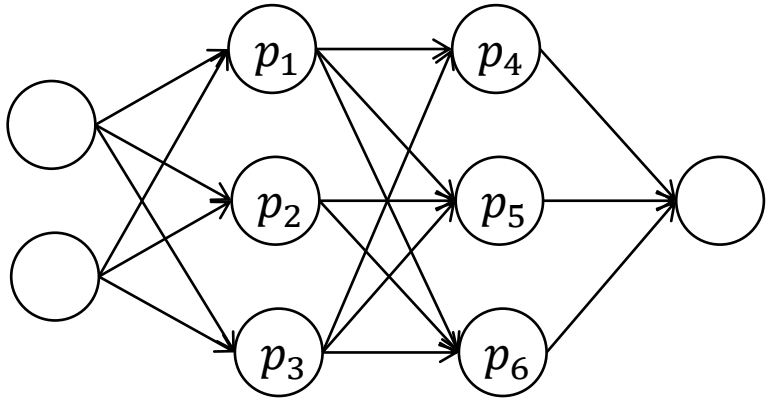
Large language models (LLMs)





Large language models (LLMs)

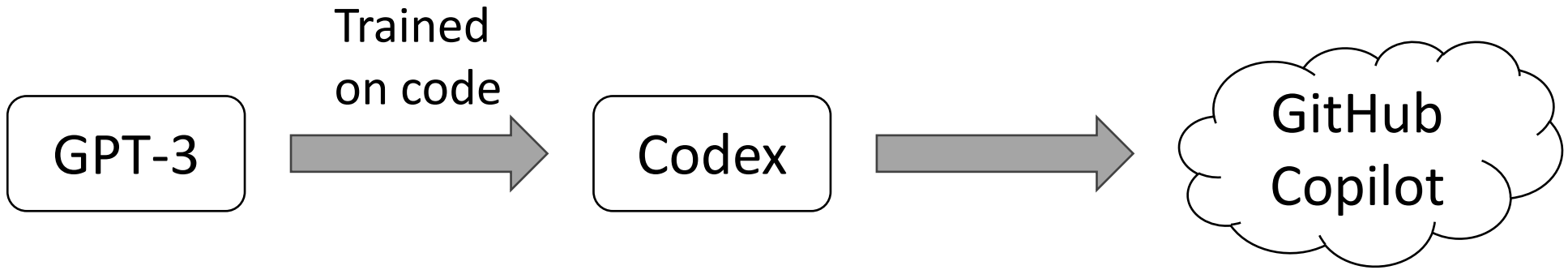




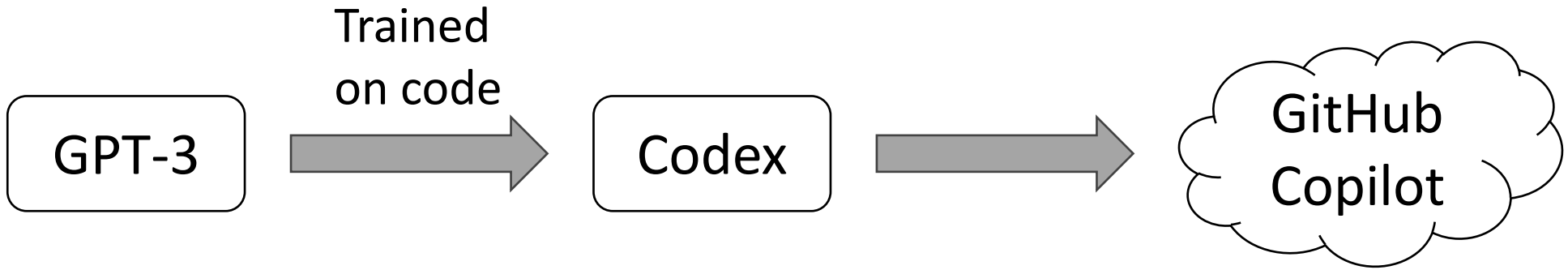
Model	Parameters	Training data
GPT-1 [2018]	117 million	4.5 GB
GPT-2 [2019]	1.5 billion	40 GB
GPT-3 [2020]	175 billion	570 GB
GPT-3.5 [2022]	175 billion	Undisclosed
GPT-4 [2023]	Undisclosed	Undisclosed

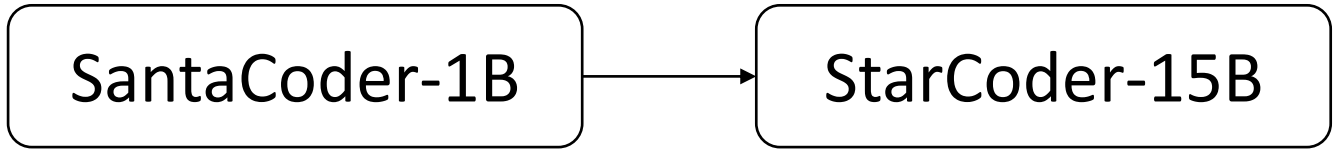
ChatGPT

Large language models for code (code LLMs)

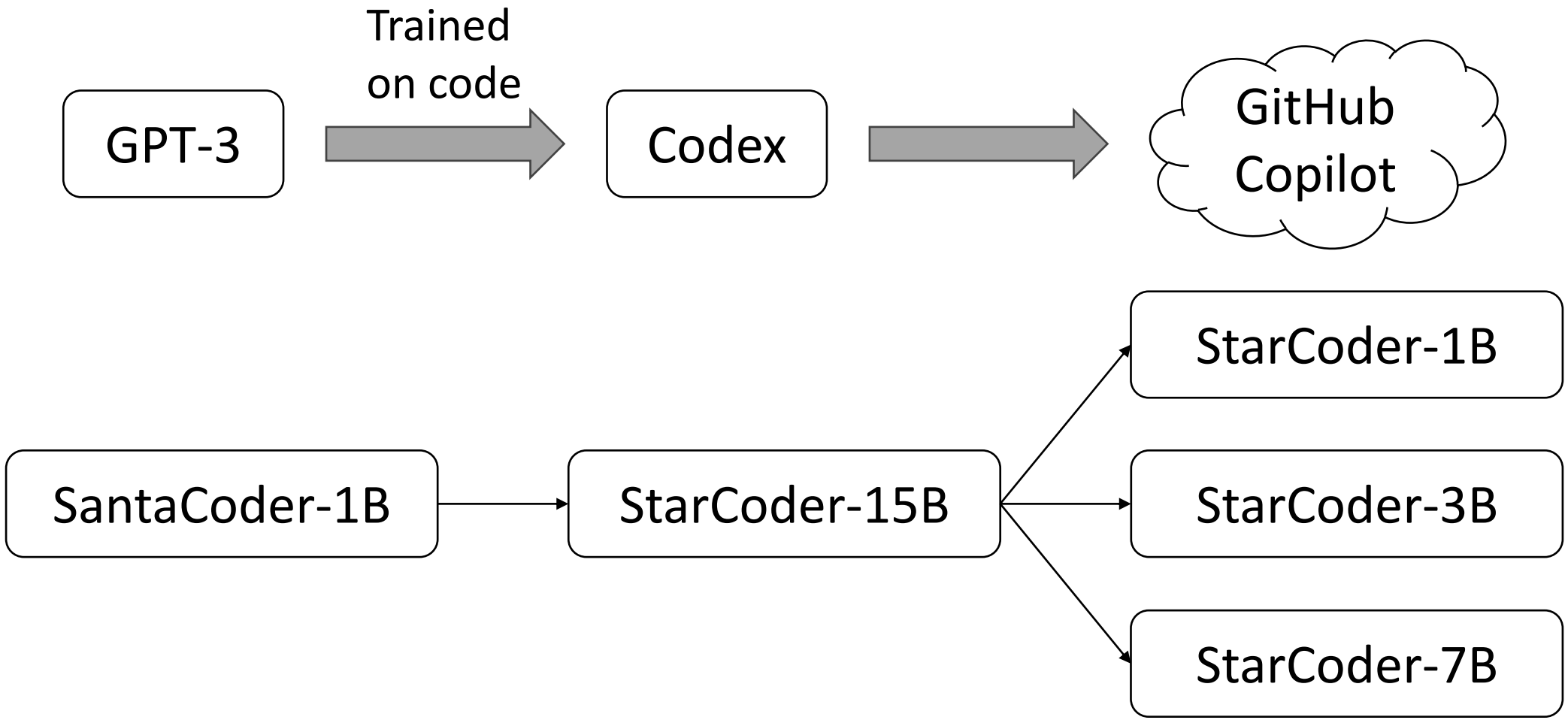


Large language models for code (code LLMs)





Large language models for code (code 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

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 NeurIPS 2023]

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 NeurIPS 2023]

Cassano, Yee, Shinn, Guha, and Holtzen

Type Definitions

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

Generating TypeScript Type Definitions With Machine Learning [proposed work]

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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

Evaluating type prediction models

What is the likelihood that a predicted type annotation is correct?

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

Evaluating type prediction models

What is the likelihood that a predicted type annotation is correct?

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

```
type S = number;
function f(w, x, y, z) { ... }
```

Identifier	Ground truth	Prediction	
w	number	number	✅
x	A | B	B | A	❌
y	S	number	❌
z	number	any	❌

Accuracy: 0.25

Evaluating type prediction models

What is the likelihood that a predicted type annotation is correct?

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

```
type S = number;
function f(w, x, y, z) { ... }
```

Identifier	Ground truth	Prediction	
w	number	number	✅
x	A | B	B | A	❌
y	S	number	❌
z	number	any	❌

Accuracy: 0.25

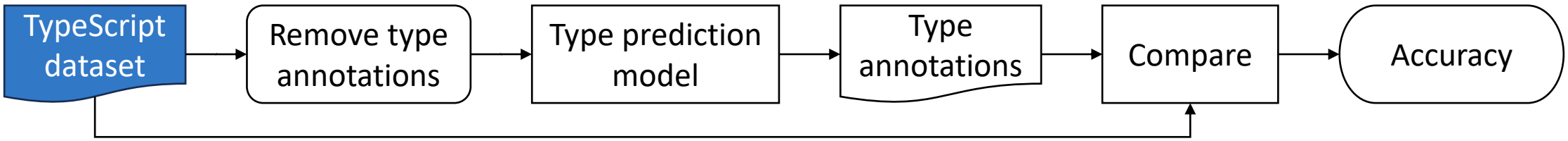
Limitations of accuracy:

- Requires exact match
- Requires ground truth
- Predictions may not type check

TypeWeaver: type check the type annotations

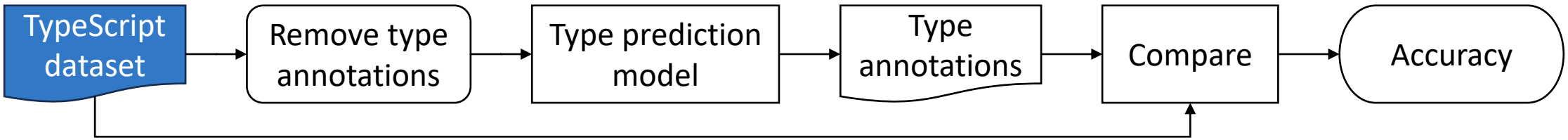
TypeWeaver: type check the type annotations

Prior work:

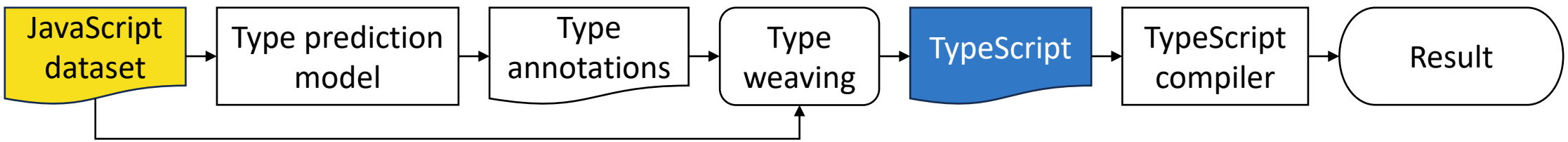


TypeWeaver: type check the type annotations

Prior work:



TypeWeaver:



Constructing the JavaScript dataset

1. Top 1,000 most downloaded packages

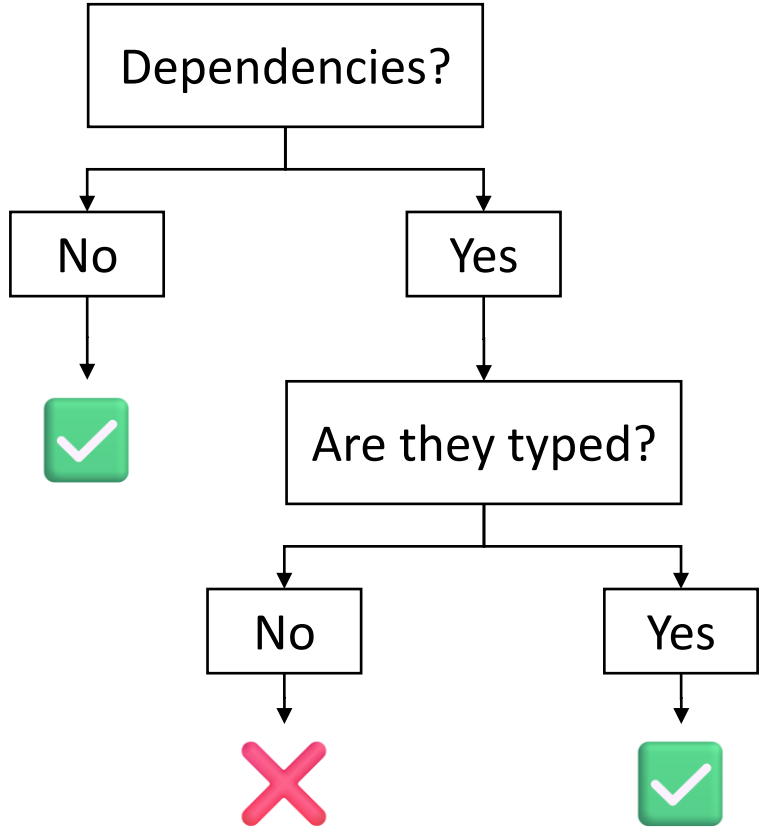
2. Download source code

3. Filter and clean

4. Check dependencies

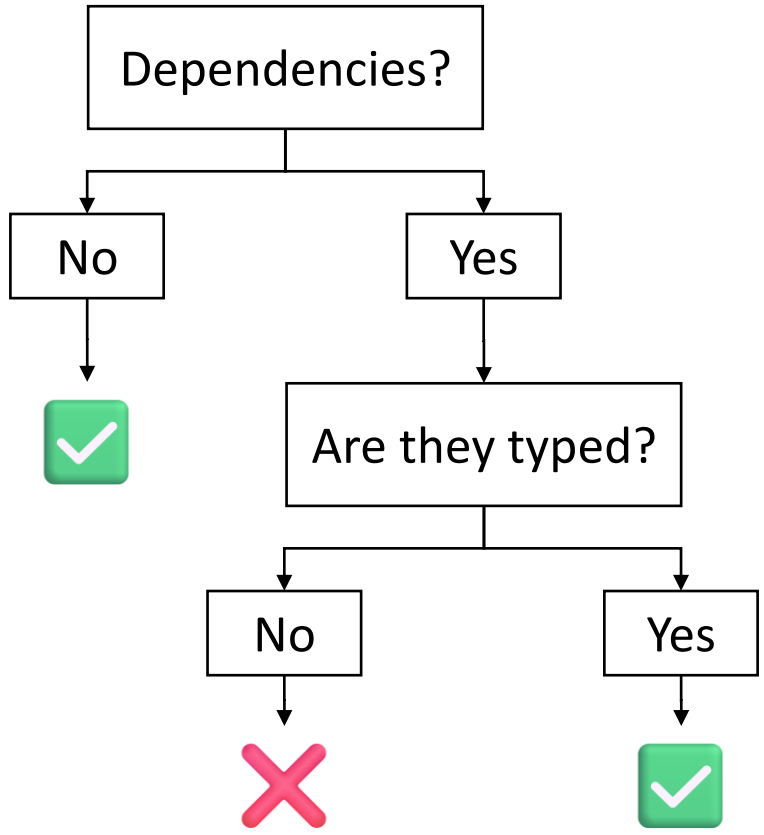
Constructing the JavaScript dataset

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: 513 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

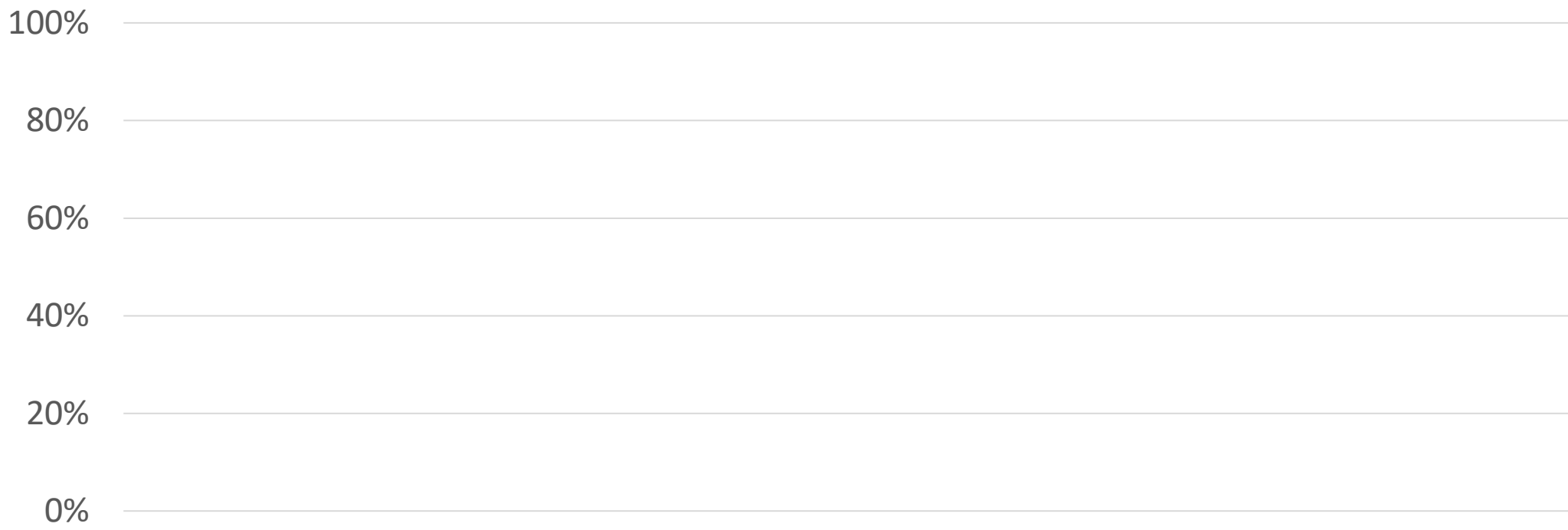
```
<fim_prefix>function sum_list(l:
<fim_suffix>) {
    let sum = 0;
    for (let i = 0; i < l.length; i++) {
        sum += l[i];
    }
    return sum;
}<fim_middle>any[]): number {
    if (l.length === 0) {
        throw 'Empty list!';
    }
    if (l.length === 1) {
        return l[0];
    }
  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

Type prediction front end

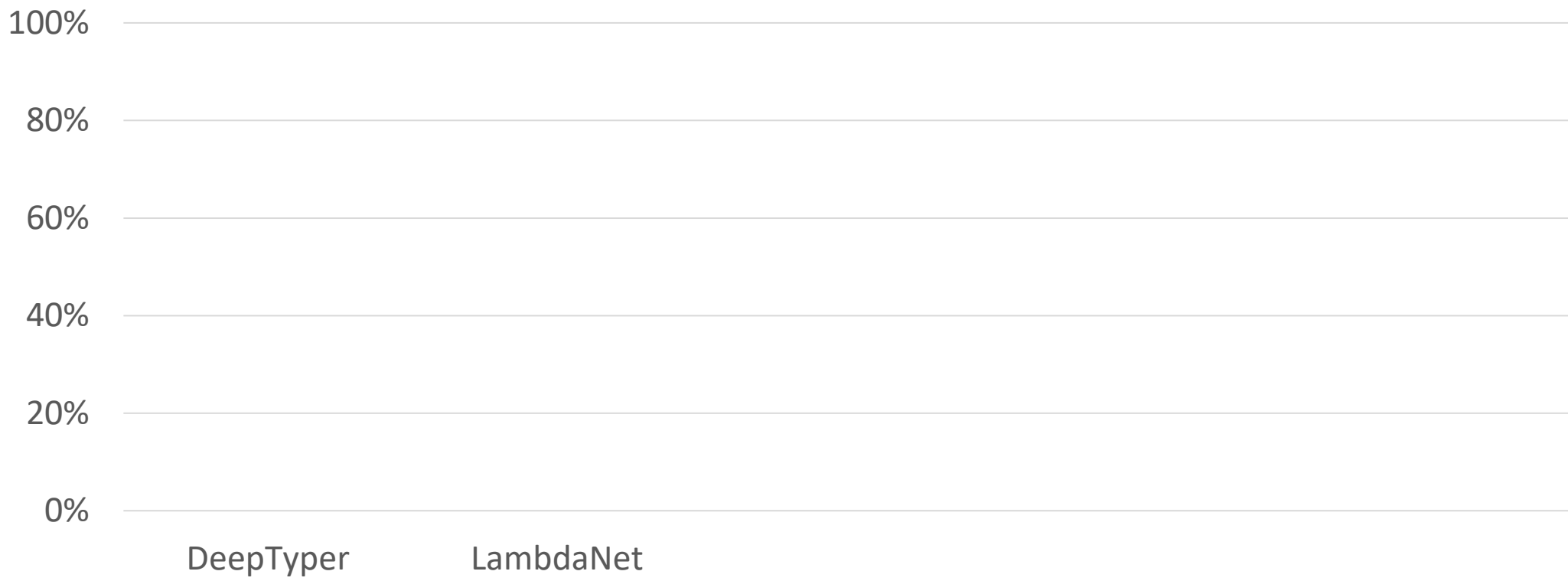
Result

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

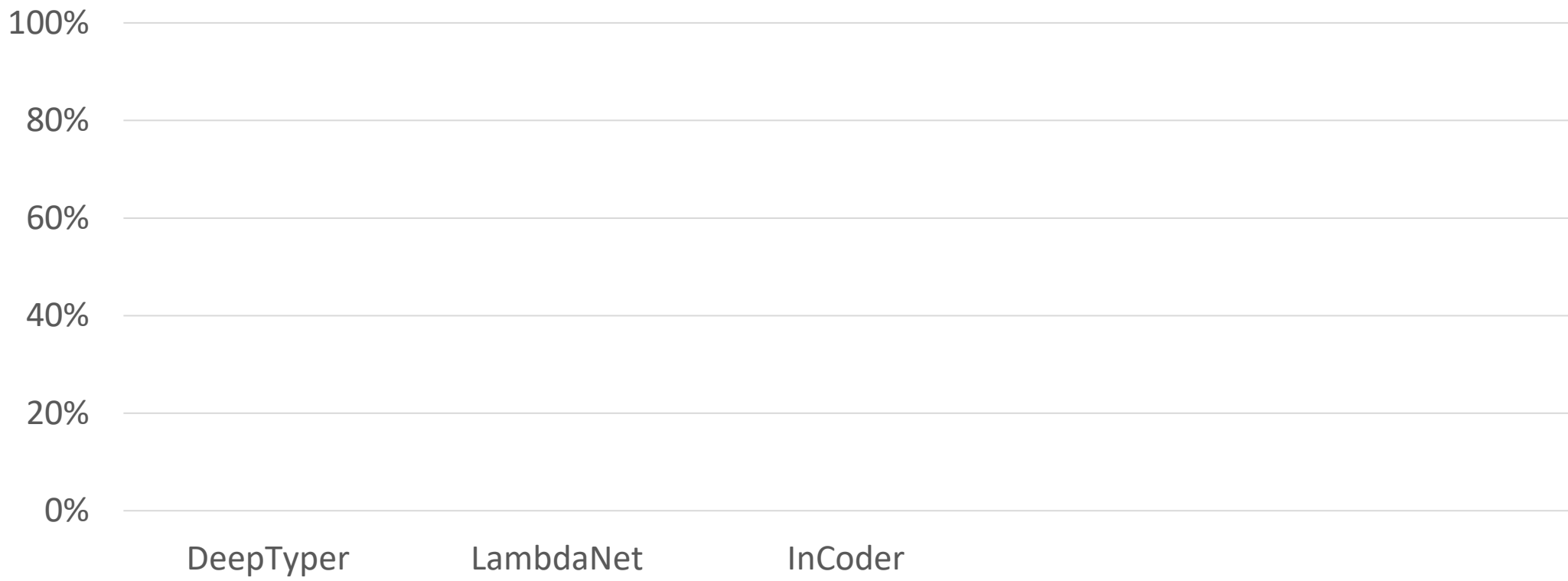
Percentage of packages that type check



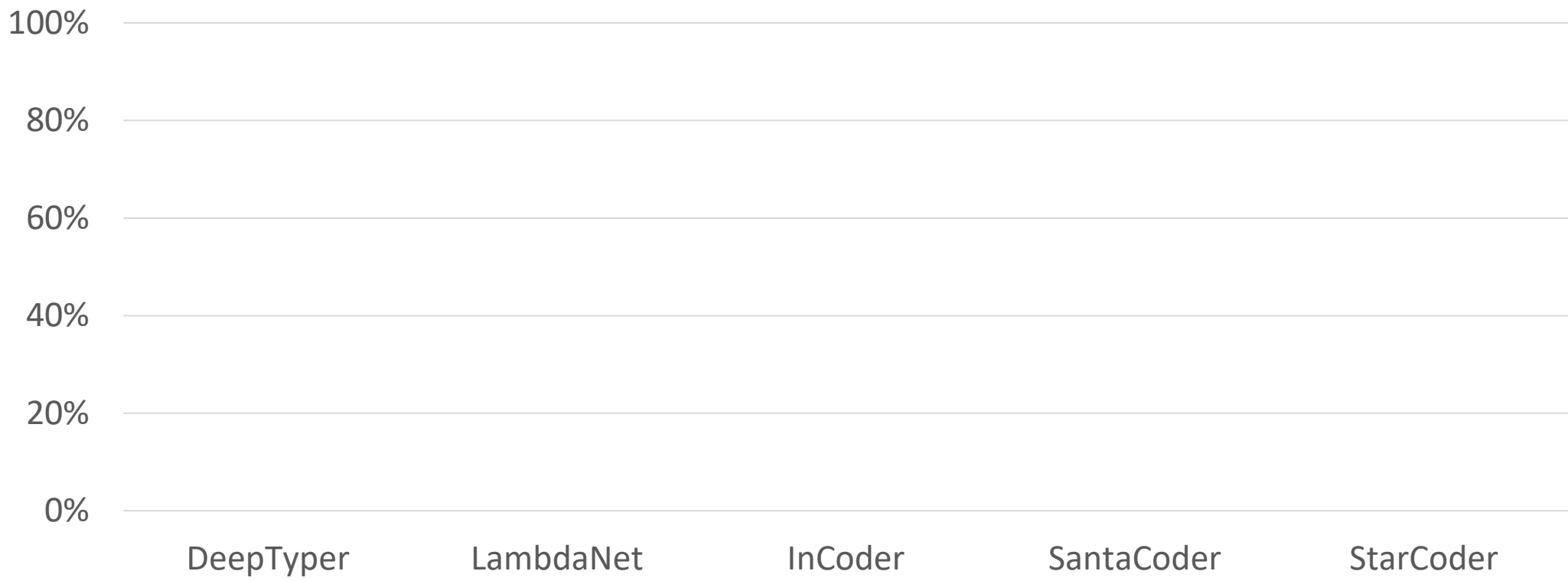
Percentage of packages that type check



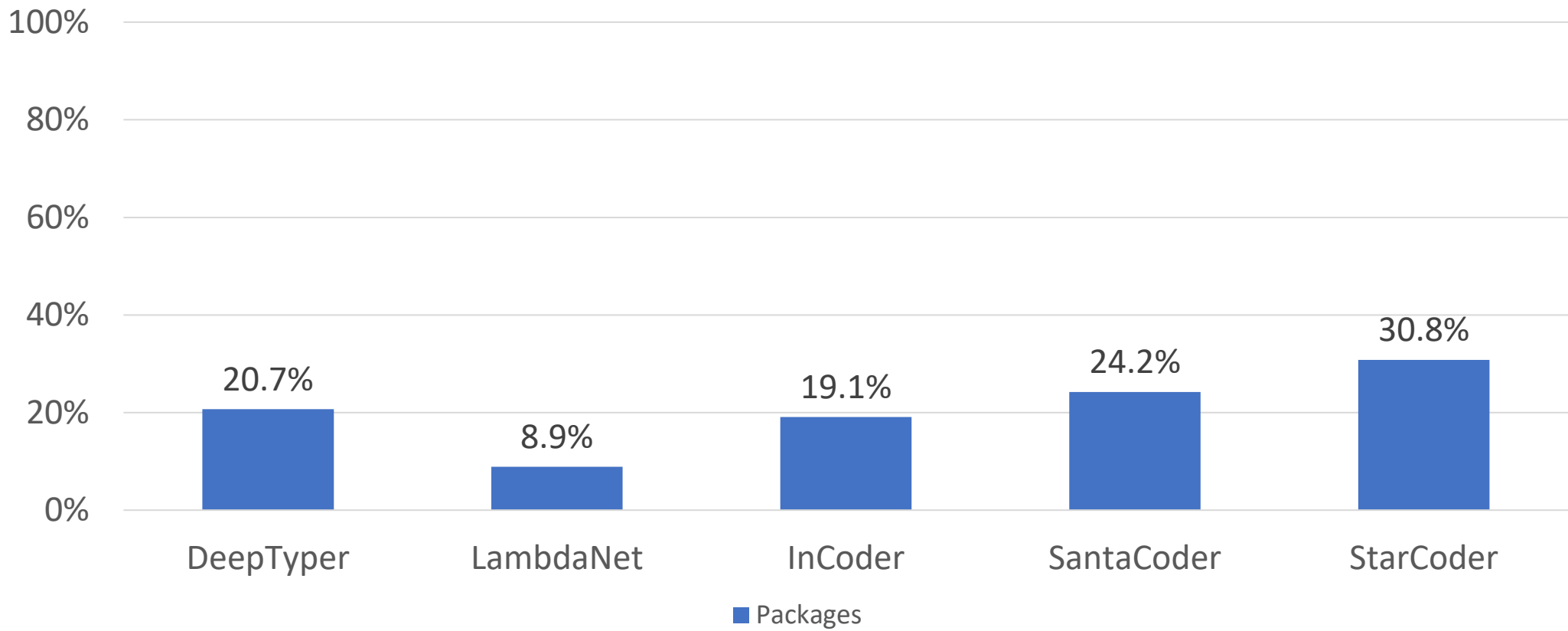
Percentage of packages that type check



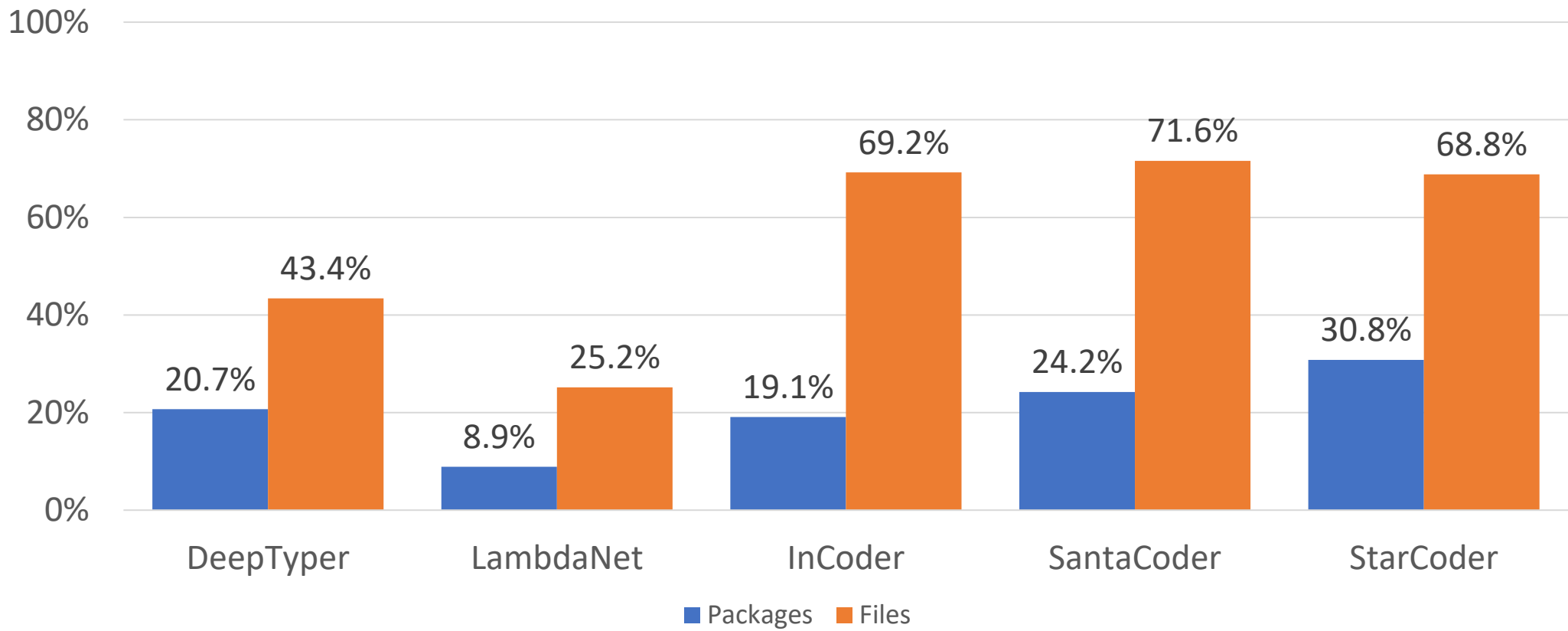
Percentage of packages that type check



Percentage of packages that type check

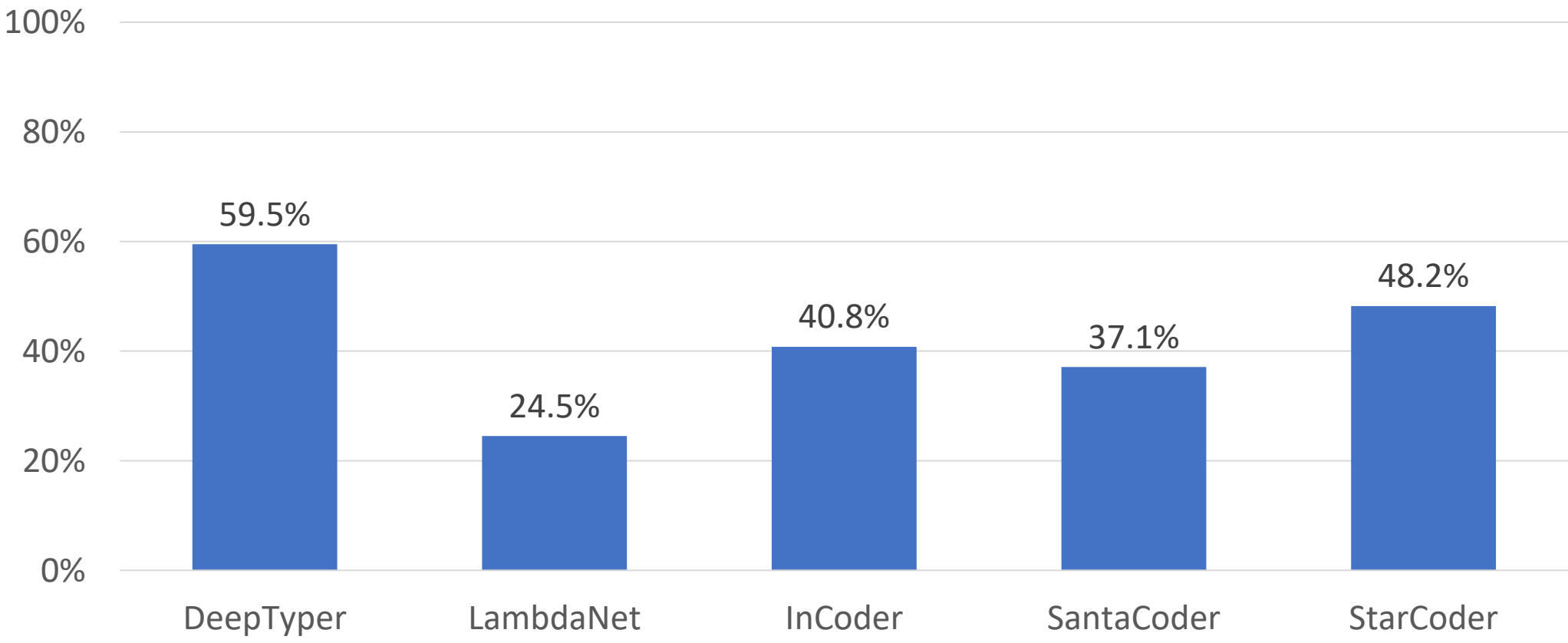


Percentage of packages/files that type check



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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

Improving type prediction

Improving type prediction

Dataset quality

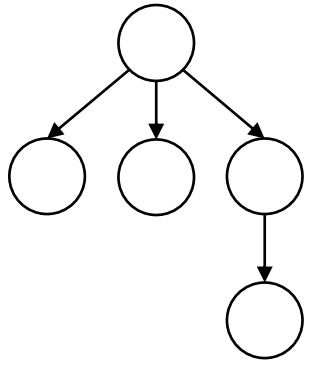
TypeScript dataset

Improving type prediction

Dataset quality

TypeScript dataset

Program decomposition

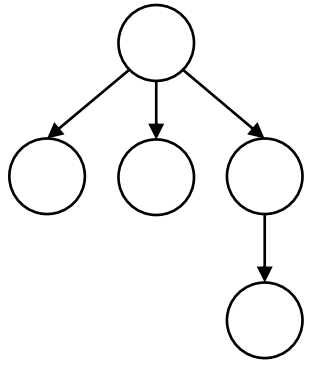


Improving type prediction

Dataset quality

TypeScript dataset

Program decomposition



Fill-in-the-type training

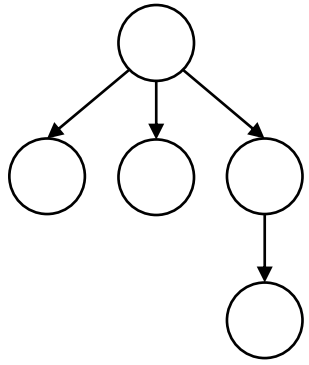
```
function f(x: _hole_) {
    return x + 1;
}
```

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

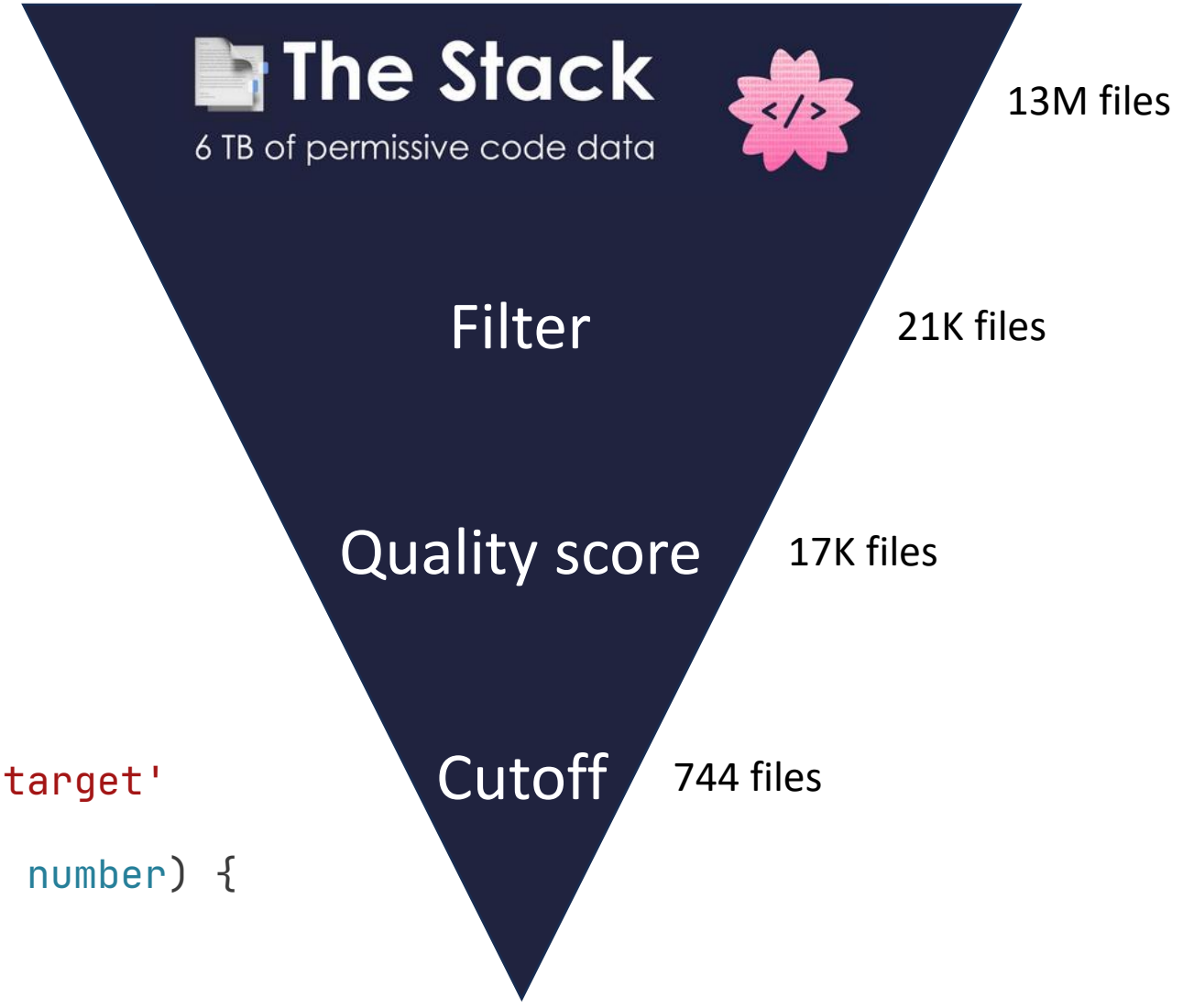
```
function f(x) {
    return x + 1;
}


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];
}
```

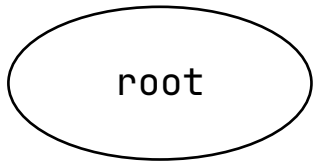


Program decomposition

```
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;
}
```

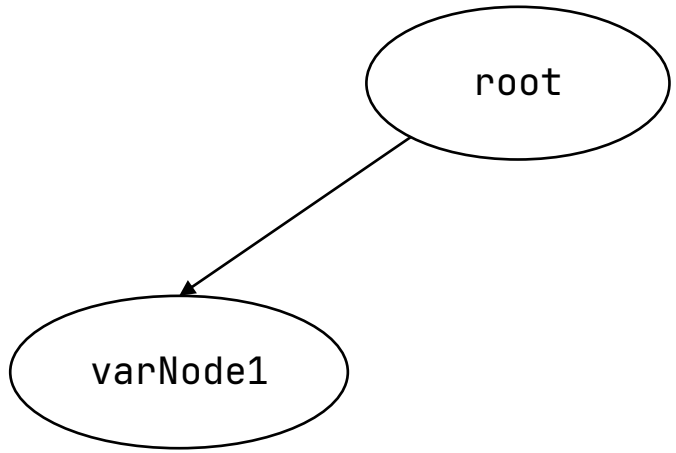
Program decomposition

```
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;
}
```



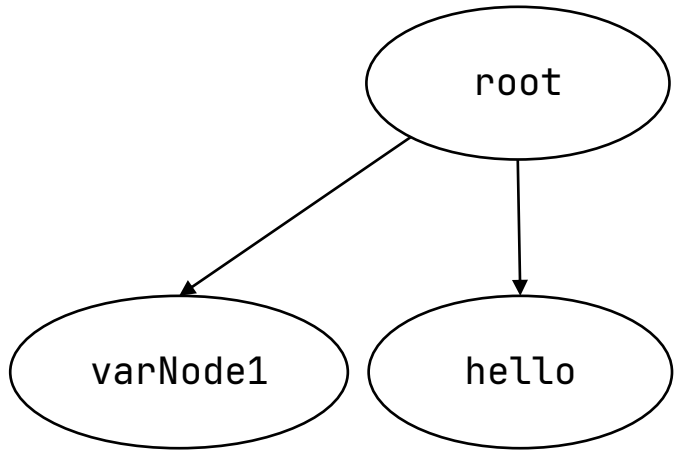
Program decomposition

```
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;
}
```



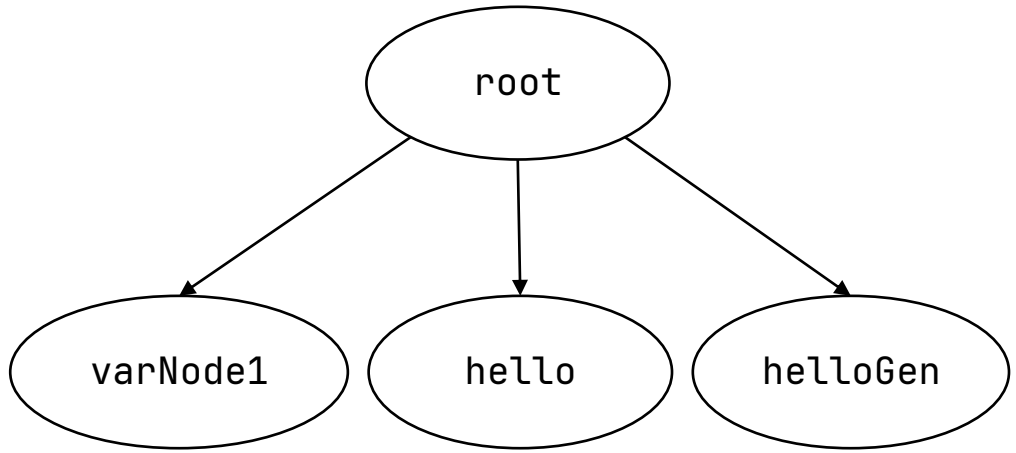
Program decomposition

```
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;
}
```



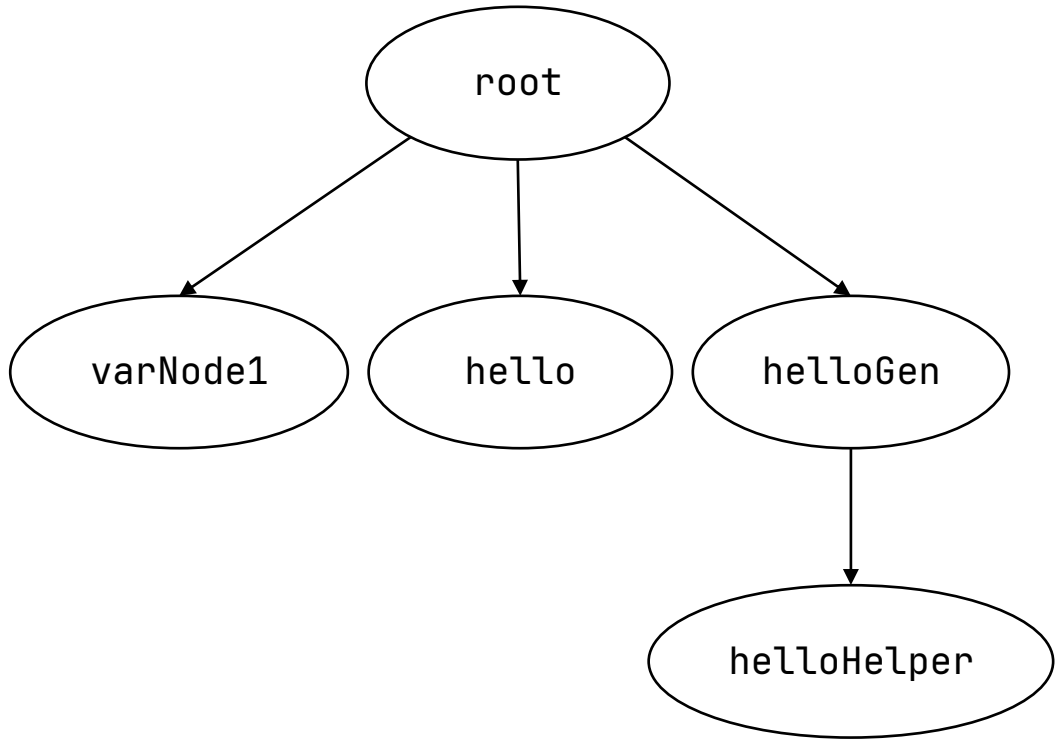
Program decomposition

```
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;
}
```



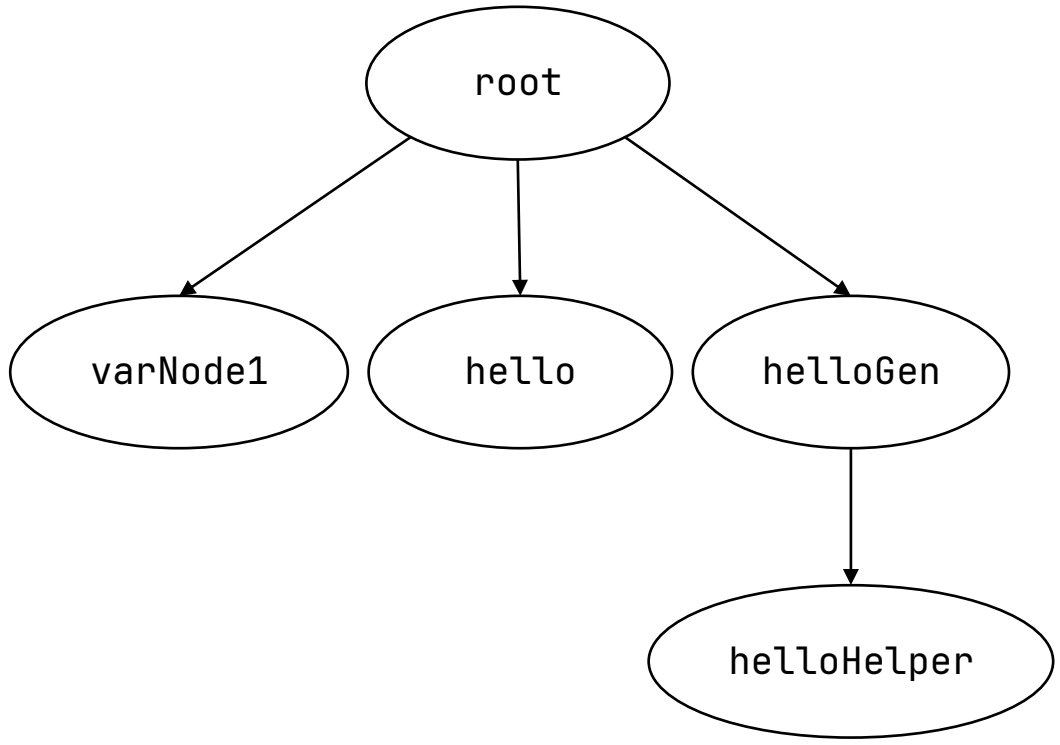
Program decomposition

```
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;
}
```



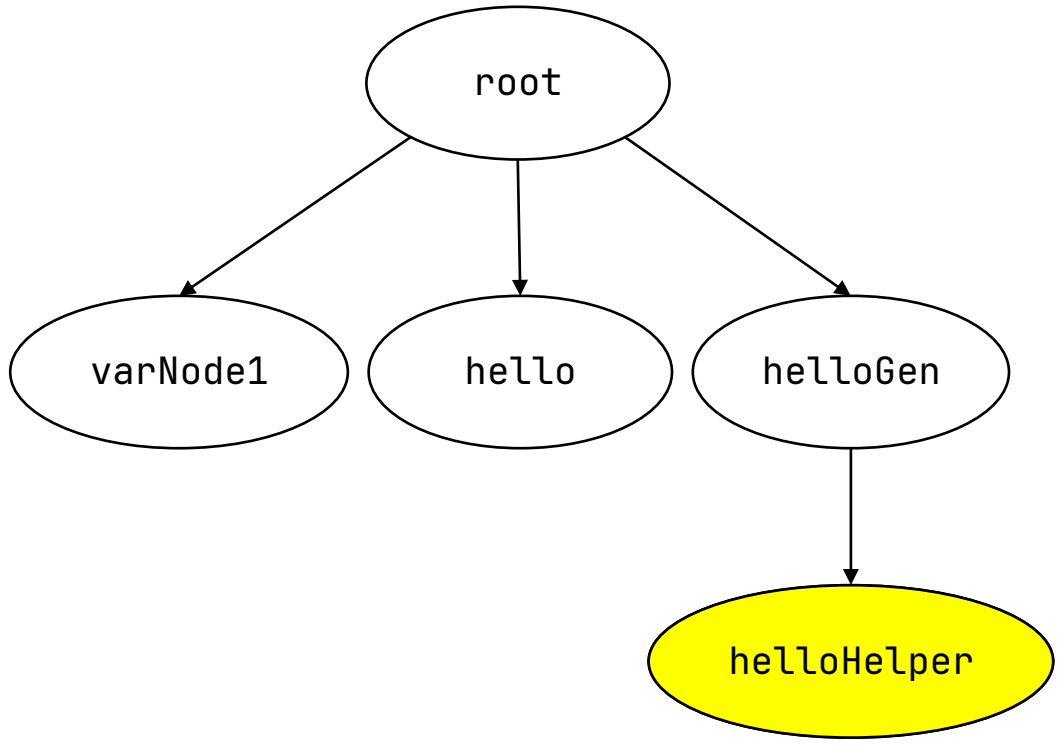
Program decomposition

```
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;
}
```



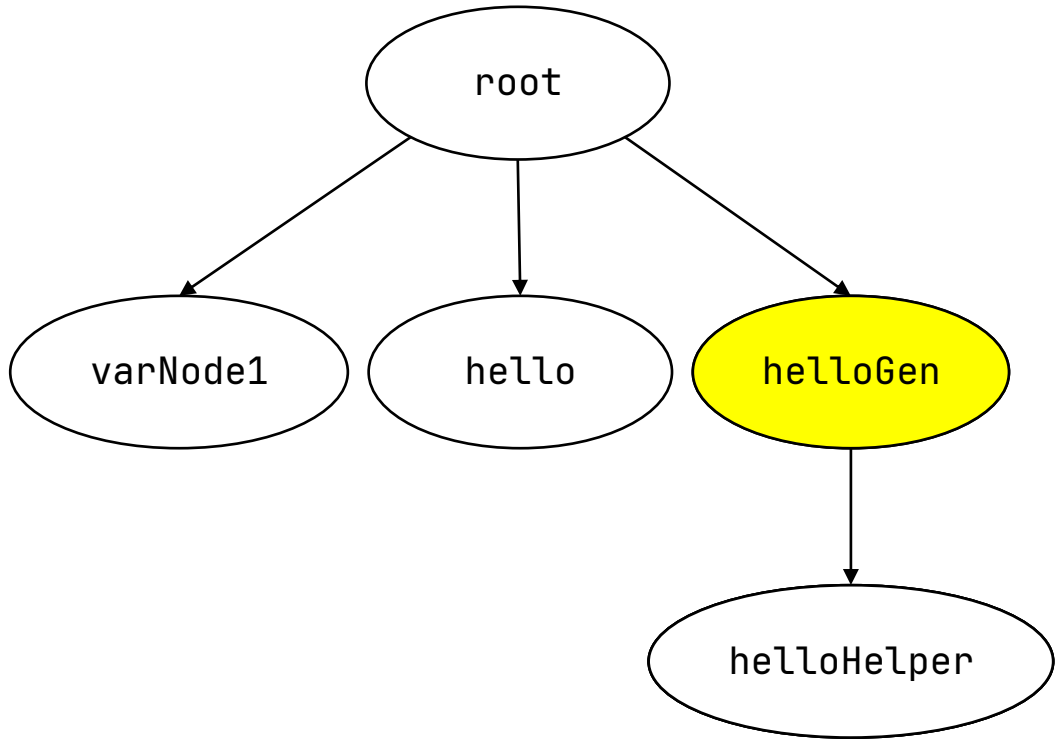
Program decomposition

```
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;
}
```

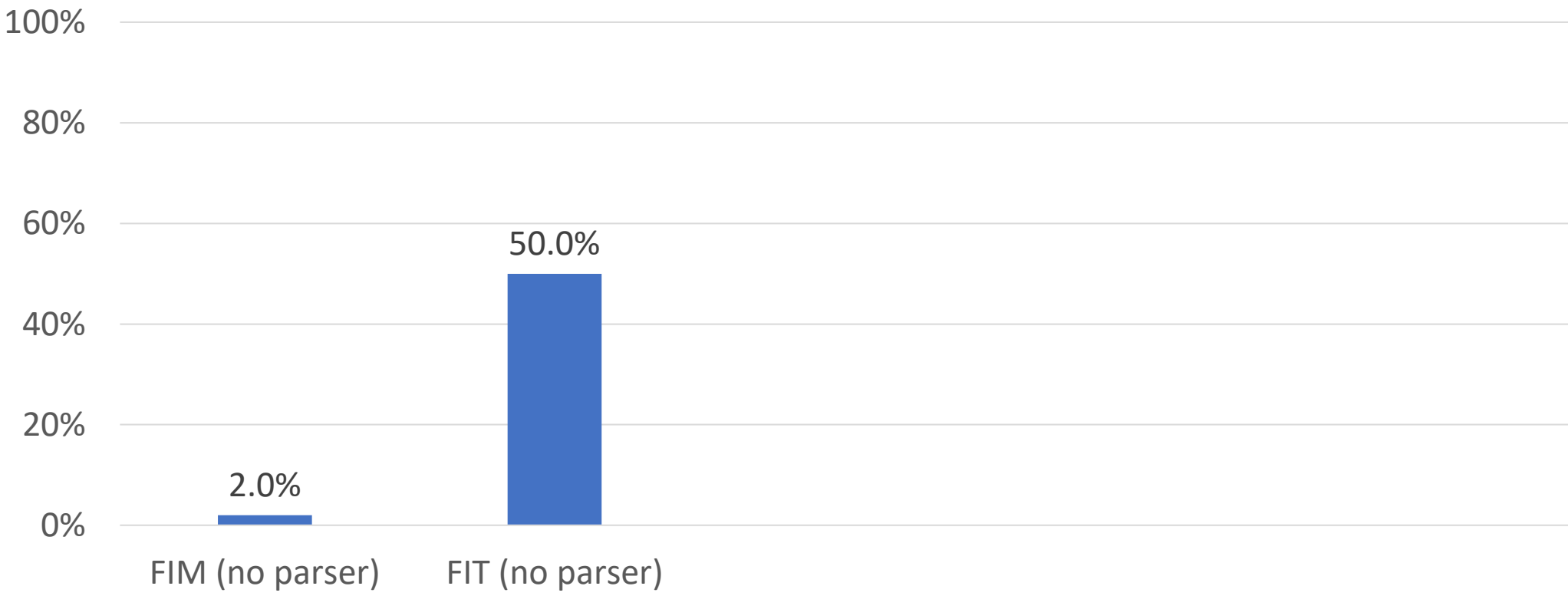
Score: 0

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

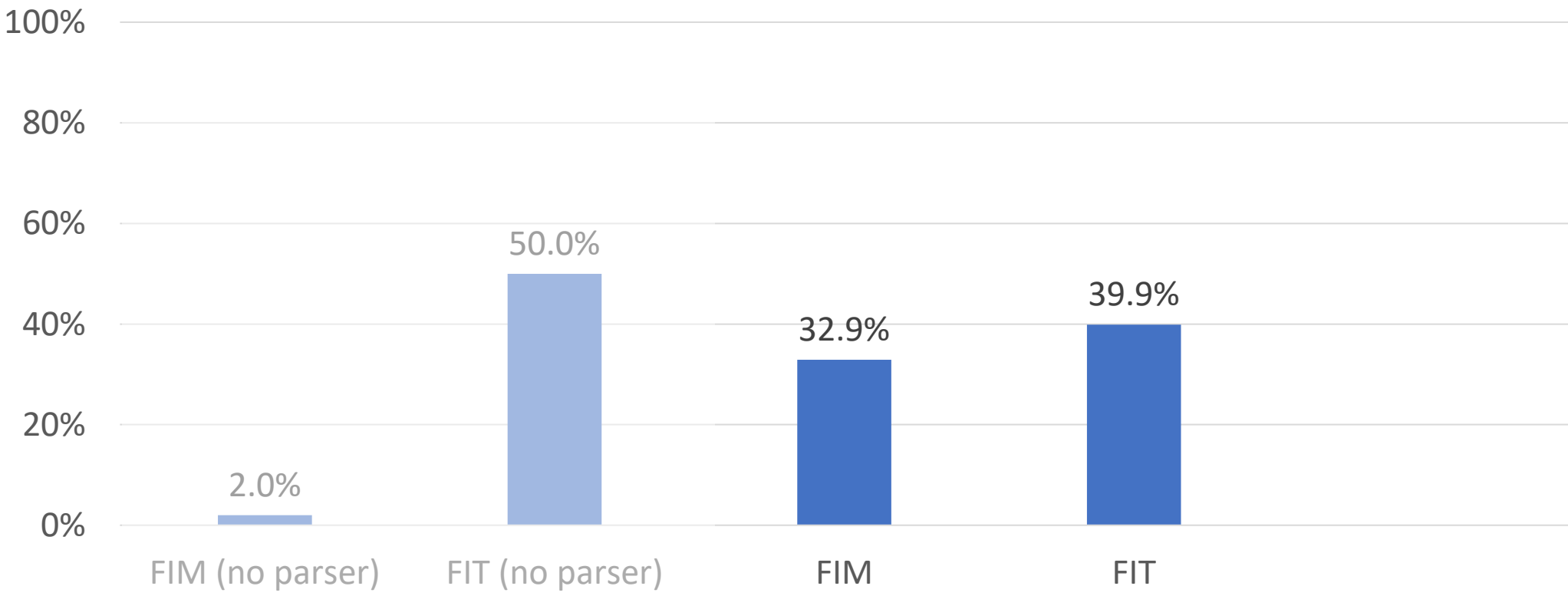
We also use this metric during type prediction

Percentage of files that type check

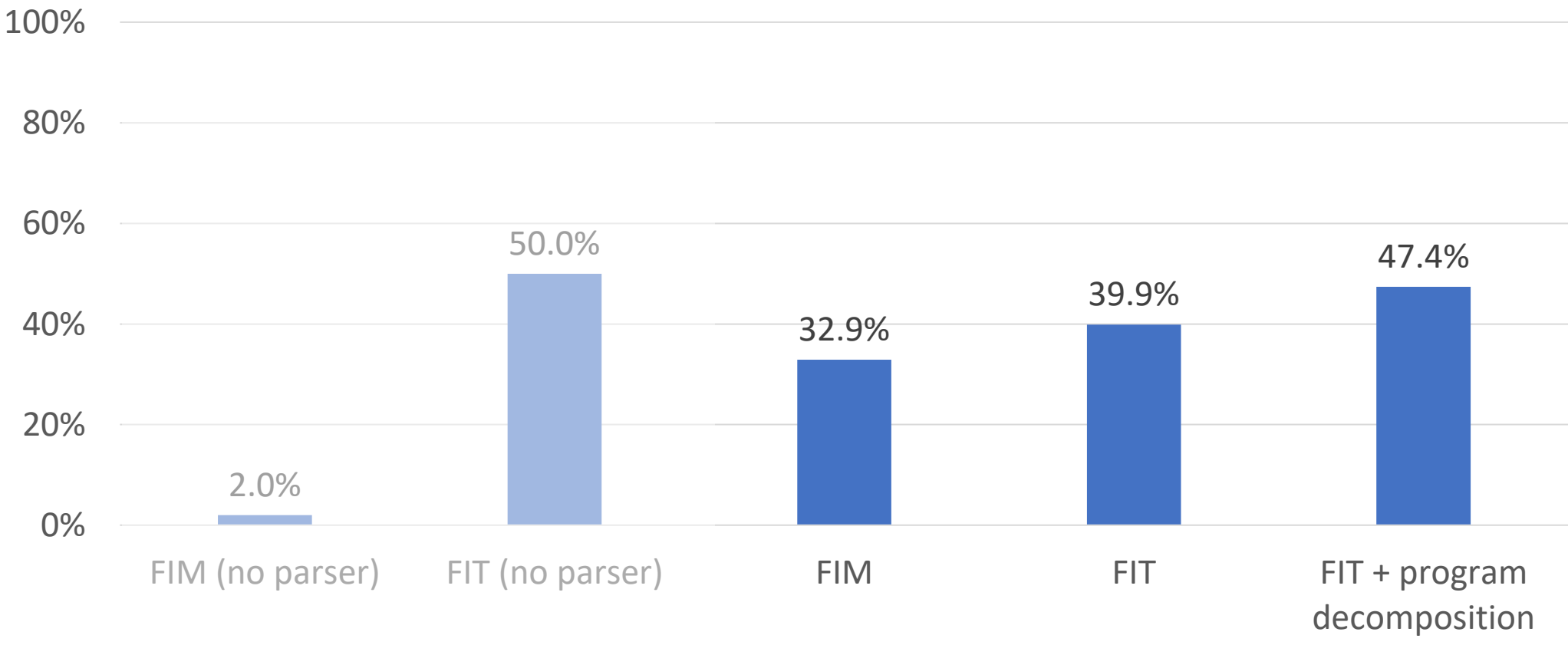
Percentage of files that type check



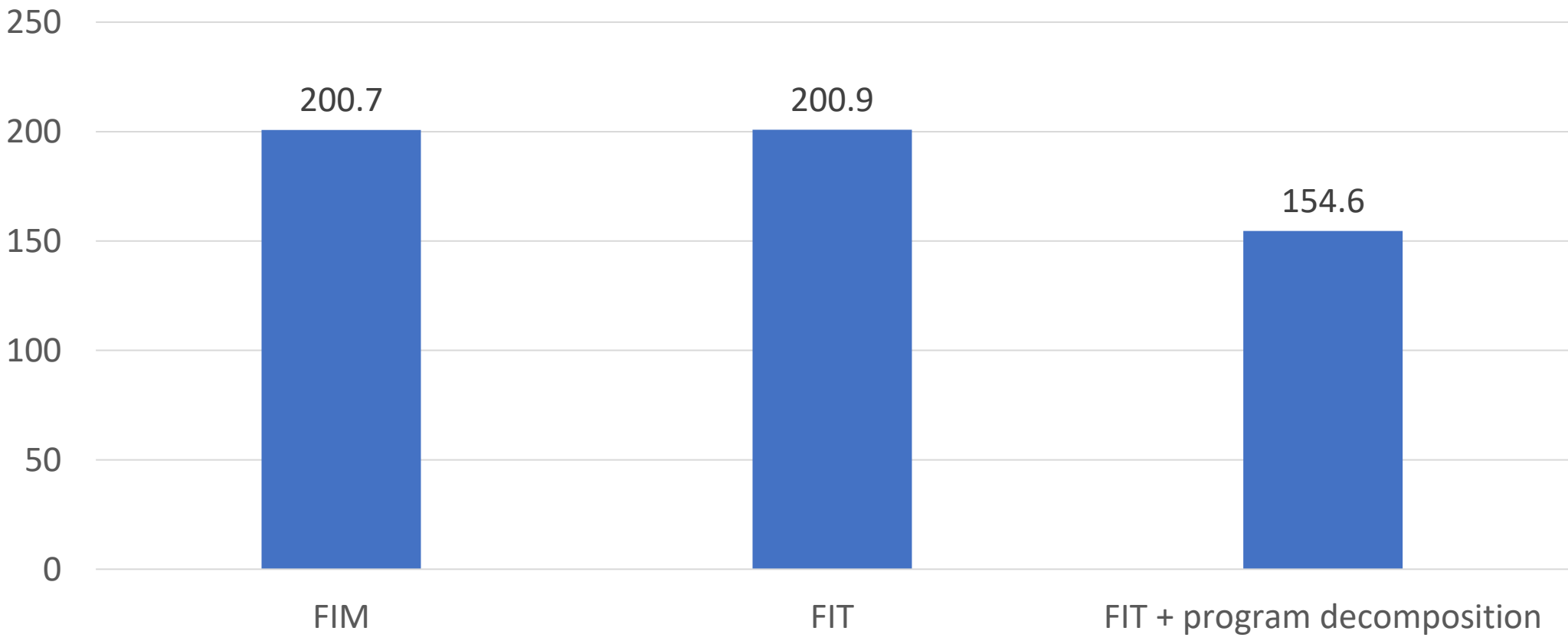
Percentage of files that type check



Percentage of files that type check



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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

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

Approach

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

Approach

```
<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

```
<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

```
<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

```
<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>Migrate to TypeScript
<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

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

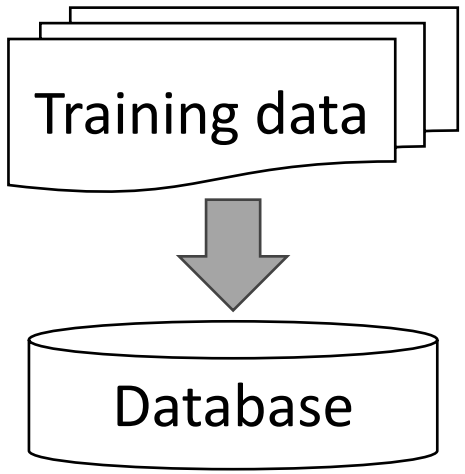
Approach

```
<commit_before>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);
}
<commit_msg>Add the type definition for Point
<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);
}
```

Alternative approaches

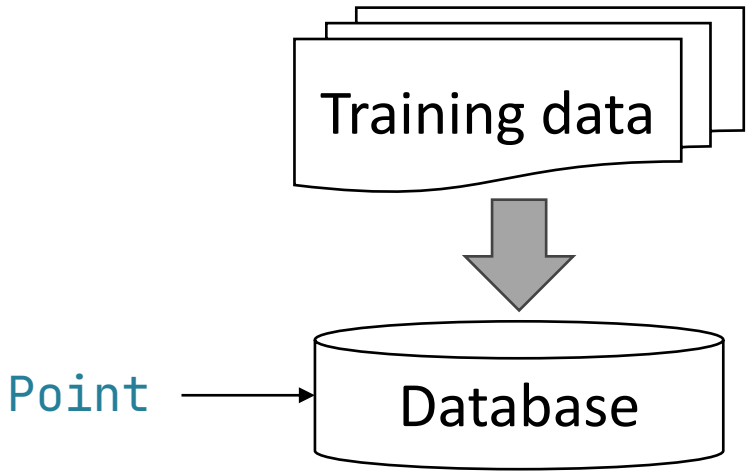
Alternative approaches

Database of types



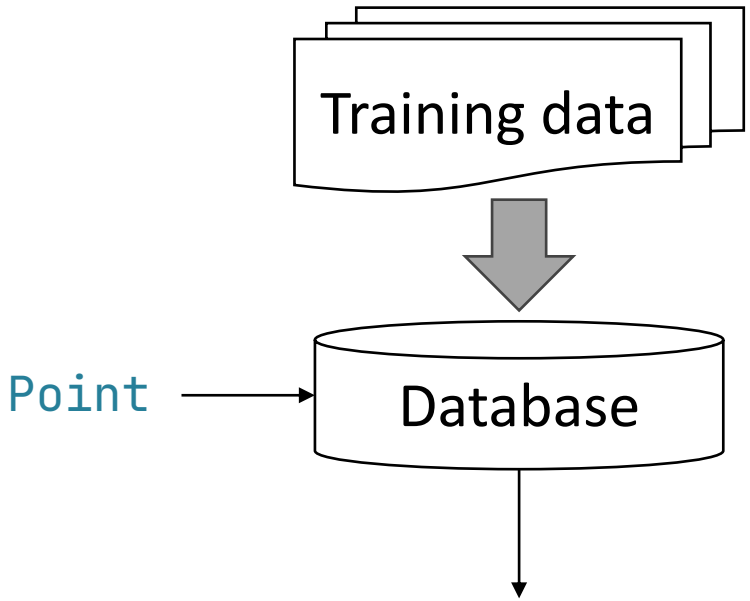
Alternative approaches

Database of types



Alternative approaches

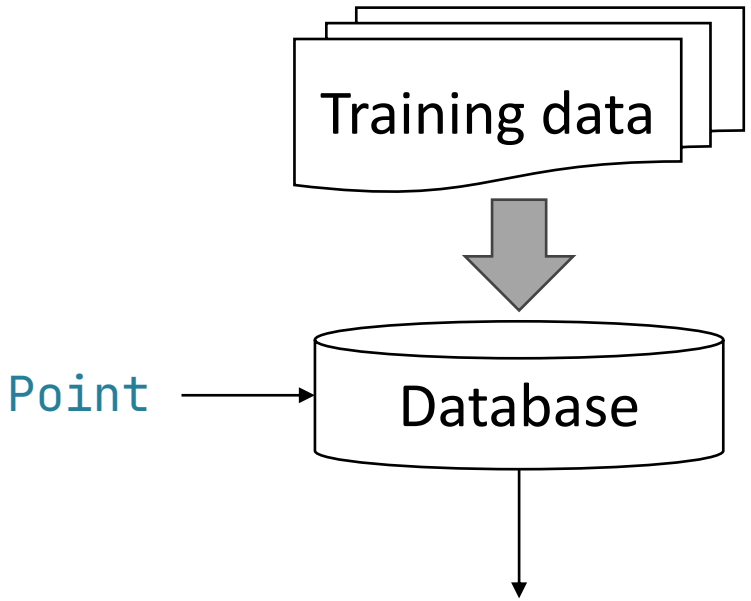
Database of types




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

Alternative approaches

Database of types



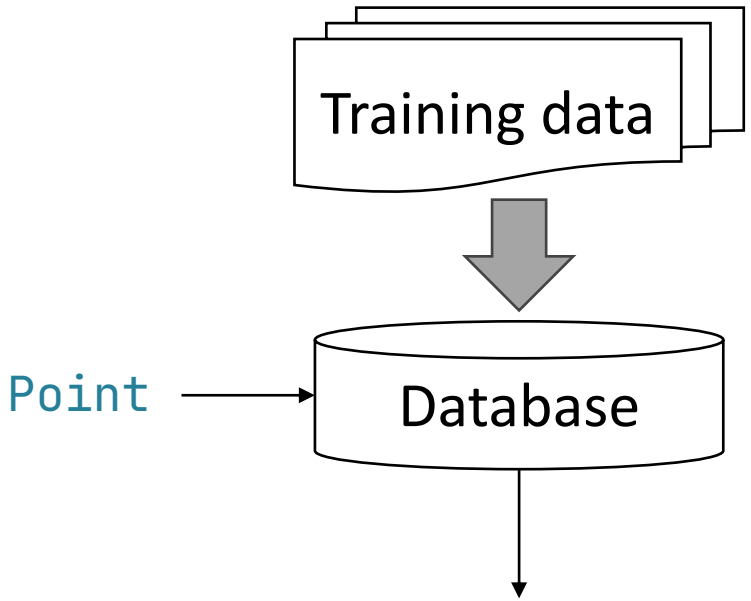

```
interface Point {
    x: number,
    y: number
}
```

Type definitions first

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

Alternative approaches

Database of types




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

Type definitions first

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

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

Status report

Completed

- Test harness
- Baseline experiments
- Initial fine-tuning
- Initial evaluation

Status report

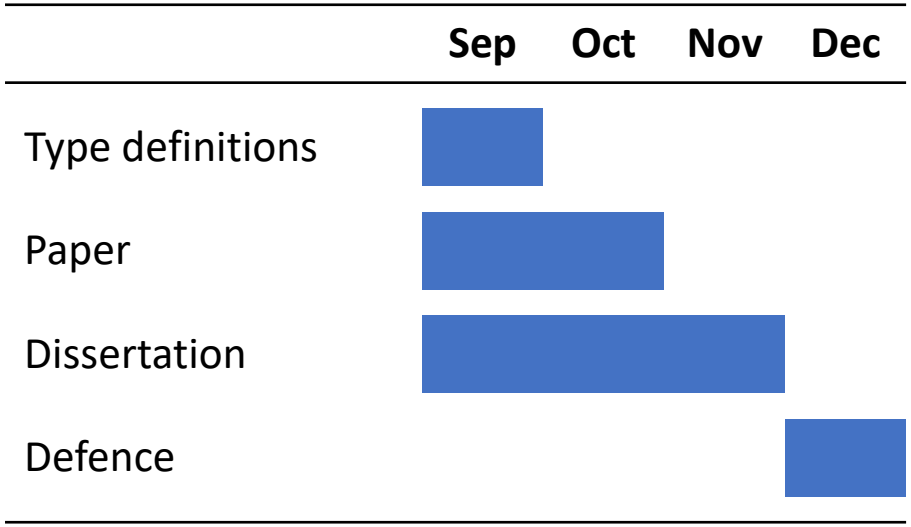
Completed

- Test harness
- Baseline experiments
- Initial fine-tuning
- Initial evaluation

Next steps

- Analyze results
- Different training formats
- More rigorous evaluation
- Ablation studies

Schedule


Sep
Oct
Nov
Dec
Type definitions
Paper
Dissertation
Defence

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

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

[submitted to NeurIPS 2023]

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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

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

Conclusion

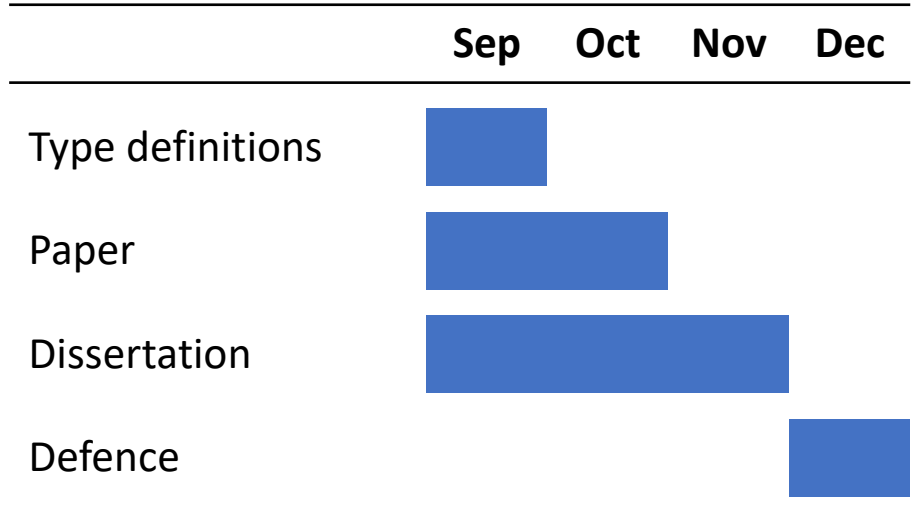
Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

Generating TypeScript Type Definitions With Machine Learning [proposed work]

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



Conclusion

Thank you!

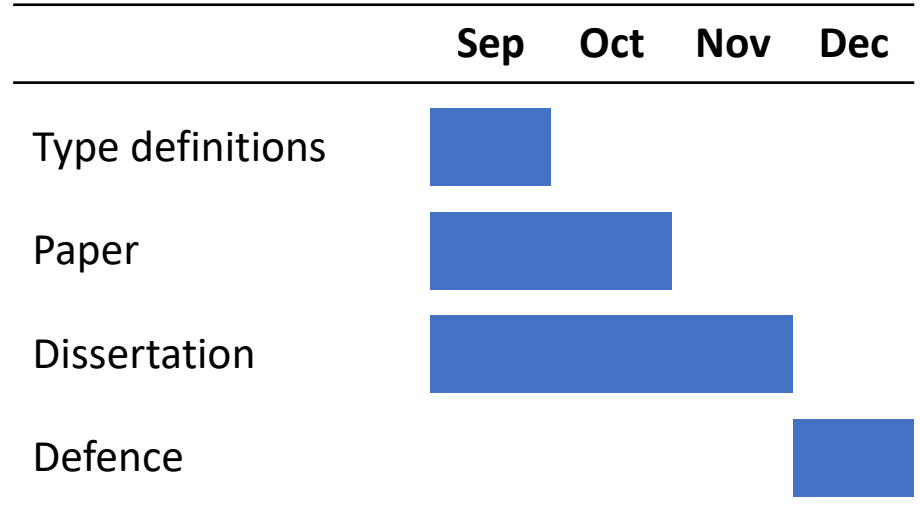
Machine learning can be used to partially migrate JavaScript programs to TypeScript, by predicting type annotations and generating type definitions.

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 NeurIPS 2023]
Cassano, Yee, Shinn, Guha, and Holtzen

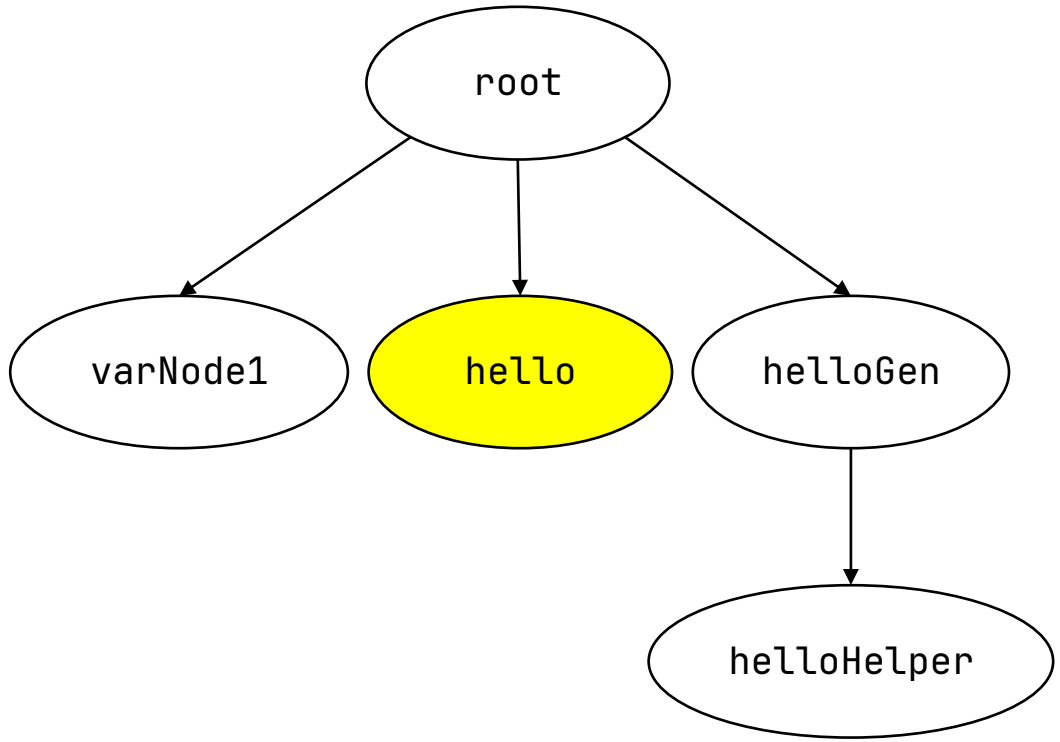
Generating TypeScript Type Definitions With Machine Learning [proposed work]

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



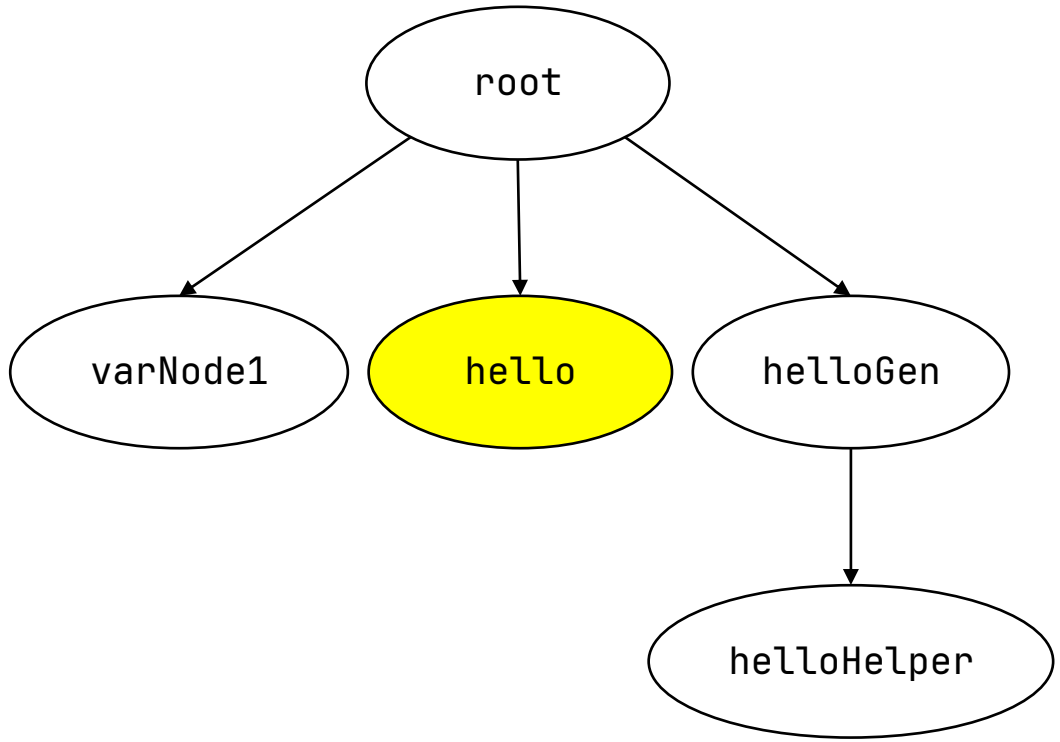
Program decomposition – with usages

```
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 – with usages

```
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 – with usages

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

/* Example usages of 'hello' are shown below:
    hello(name) + 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;
}
```