TypyBench: Evaluating LLM Type Inference for Untyped Python Repositories

ICML 2025

Honghua Dong[1,2*], Jiacheng Yang[1,2*], Xun Deng[1*], **Yuhe Jiang**[1,2*], Gennady Pekhimenko[1,2,3], Fan Long[1], and Xujie Si[1,2,3]

University of Toronto[1], **Vector Institute**[2] and **CIFAR AI Chair**[3]

*Equal Contribution

Introduction

- **Advantages of Type Annotations**:
  - Enhance code clarity
  - Prevent type-related errors
  - Enable autocompletion
  - Facilitate maintenance & refactoring

```
class Greeter:
    def __init__(self, name: str) → None:
        self.name = name
    def greet(self) → str:
        return f"Hello, {self.name}"
    def get_name(self) → str:
        return self.name
```

Introduction

- **Advantages of Type Annotations**
- **Type Inference**
  - Given code, infer types

Untyped Code

```
class Greeter:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return f"Hello, {self.name}"
    def get_name(self):
        return self.name
```

Introduction

- **Advantages of Type Annotations**
- **Type Inference**
  - Given code, infer types

Type Annotated Code

```
class Greeter:
    def __init__(self, name: str) → None:
        self.name = name
    def greet(self) → str:
        return f"Hello, {self.name}"
    def get_name(self) → str:
        return self.name
```

Introduction

- **Advantages of Type Annotations**
- **Type Inference**
- **Benchmarks**

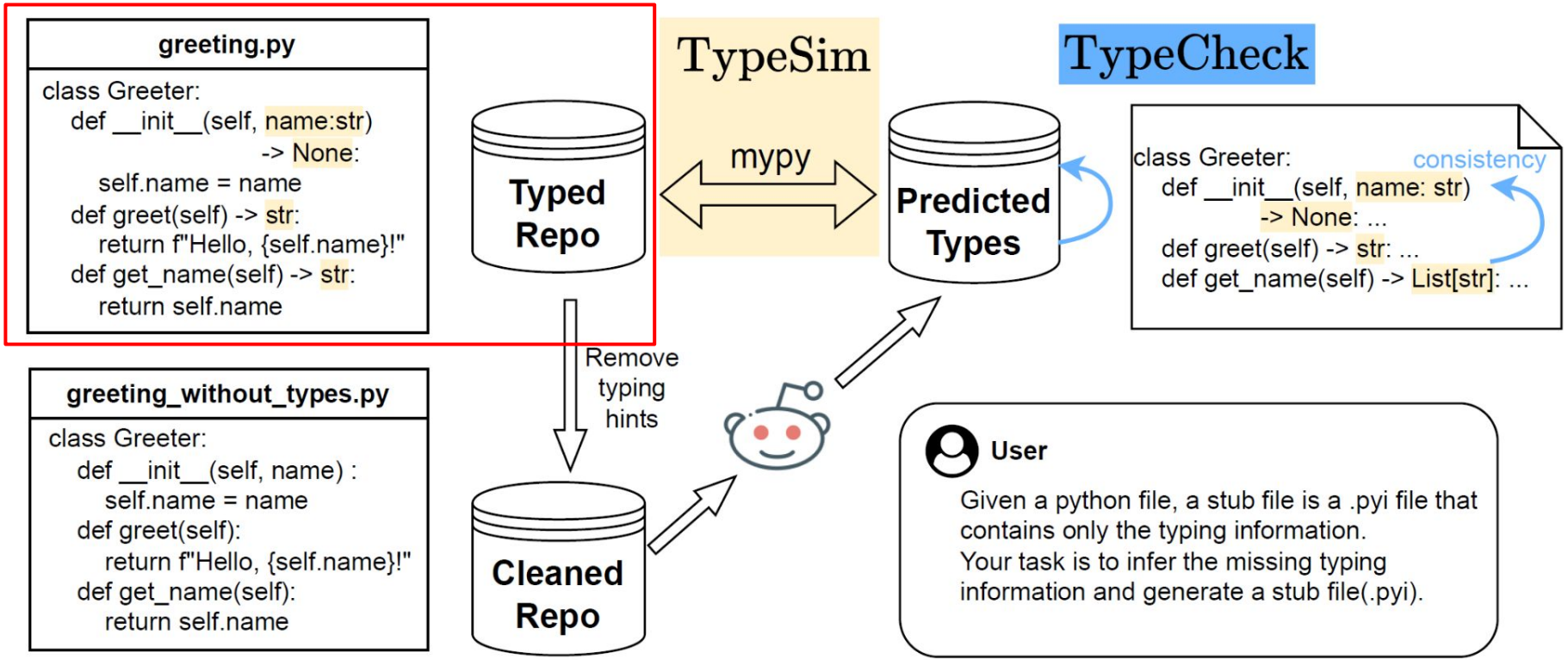
Existing Benchmarks
❌ Mostly rely on exact matching
❌ Less attention on type consistency
❌ No large-scale evaluation on LLMs

Introduction

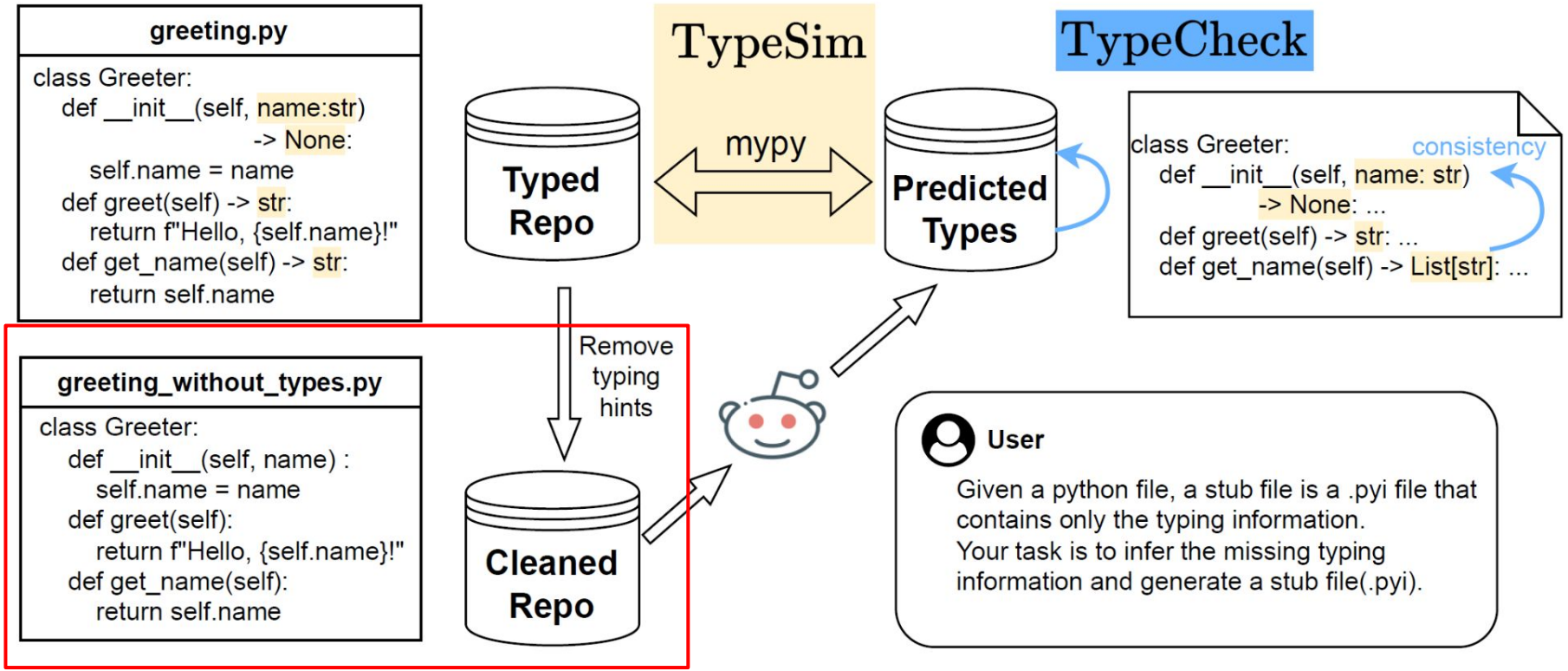
- **Advantages of Type Annotations**
- **Type Inference**
- **Benchmarks**

Existing Benchmarks	TypyBench
❌ Mostly rely on exact matching	✅ **TypeSim Metric**
❌ Less attention on type consistency	✅ **TypeCheck Metric**
❌ No large-scale evaluation on LLMs	✅ **Holistic Evaluation of SOTA LLMs**

TypyBench


greeting.py
class Greeter:
def __init__(self, name:str)
-> None:
self.name = name
def greet(self) -> str:
return f"Hello, {self.name}!"
def get_name(self) -> str:
return self.name
Typed Repo
TypeSim
mypy
Predicted Types
TypeCheck
class Greeter:
def __init__(self, name: str)
-> None: ...
def greet(self) -> str: ...
def get_name(self) -> List[str]: ...
consistency
Remove typing hints
greeting_without_types.py
class Greeter:
def __init__(self, name) :
self.name = name
def greet(self):
return f"Hello, {self.name}!"
def get_name(self):
return self.name
Cleaned Repo
User
Given a python file, a stub file is a .pyi file that contains only the typing information.
Your task is to infer the missing typing information and generate a stub file(.pyi).

TypyBench


greeting.py
class Greeter:
def __init__(self, name:str)
-> None:
self.name = name
def greet(self) -> str:
return f"Hello, {self.name}!"
def get_name(self) -> str:
return self.name
TypeSim
mypy
Typed Repo
Predicted Types
TypeCheck
class Greeter:
consistency
def __init__(self, name: str)
-> None: ...
def greet(self) -> str: ...
def get_name(self) -> List[str]: ...
Remove typing hints
greeting_without_types.py
class Greeter:
def __init__(self, name) :
self.name = name
def greet(self):
return f"Hello, {self.name}!"
def get_name(self):
return self.name
Cleaned Repo
User
Given a python file, a stub file is a .pyi file that contains only the typing information.
Your task is to infer the missing typing information and generate a stub file(.pyi).

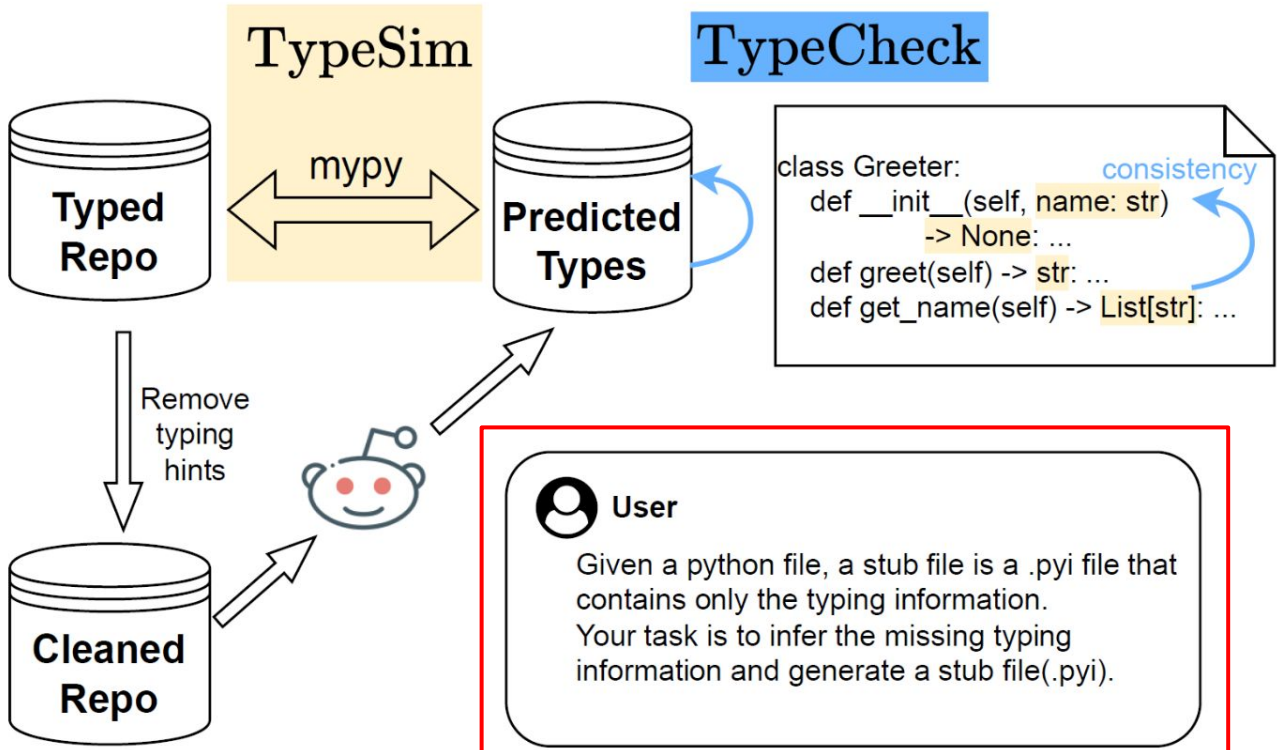
TypyBench

greeting.py

```
class Greeter:
    def __init__(self, name:str)
                        -> None:
        self.name = name
    def greet(self) -> str:
        return f"Hello, {self.name}!"
    def get_name(self) -> str:
        return self.name
```

greeting_without_types.py

```
class Greeter:
    def __init__(self, name) :
        self.name = name
    def greet(self):
        return f"Hello, {self.name}!"
    def get_name(self):
        return self.name
```



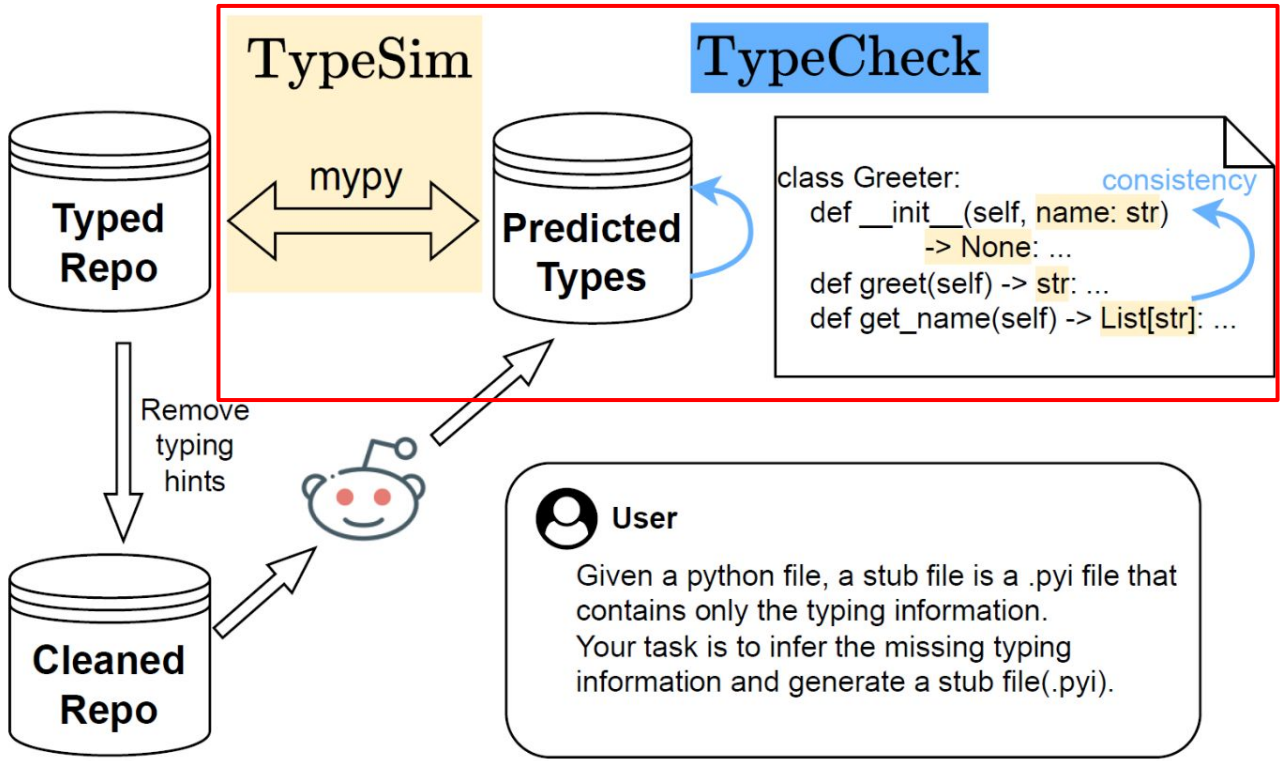
TypyBench

greeting.py

```
class Greeter:
    def __init__(self, name:str)
                        -> None:
        self.name = name
    def greet(self) -> str:
        return f"Hello, {self.name}!"
    def get_name(self) -> str:
        return self.name
```

greeting_without_types.py

```
class Greeter:
    def __init__(self, name) :
        self.name = name
    def greet(self):
        return f"Hello, {self.name}!"
    def get_name(self):
        return self.name
```



Two Novel Metrics – TypeSim

functional similarity + structural relationships

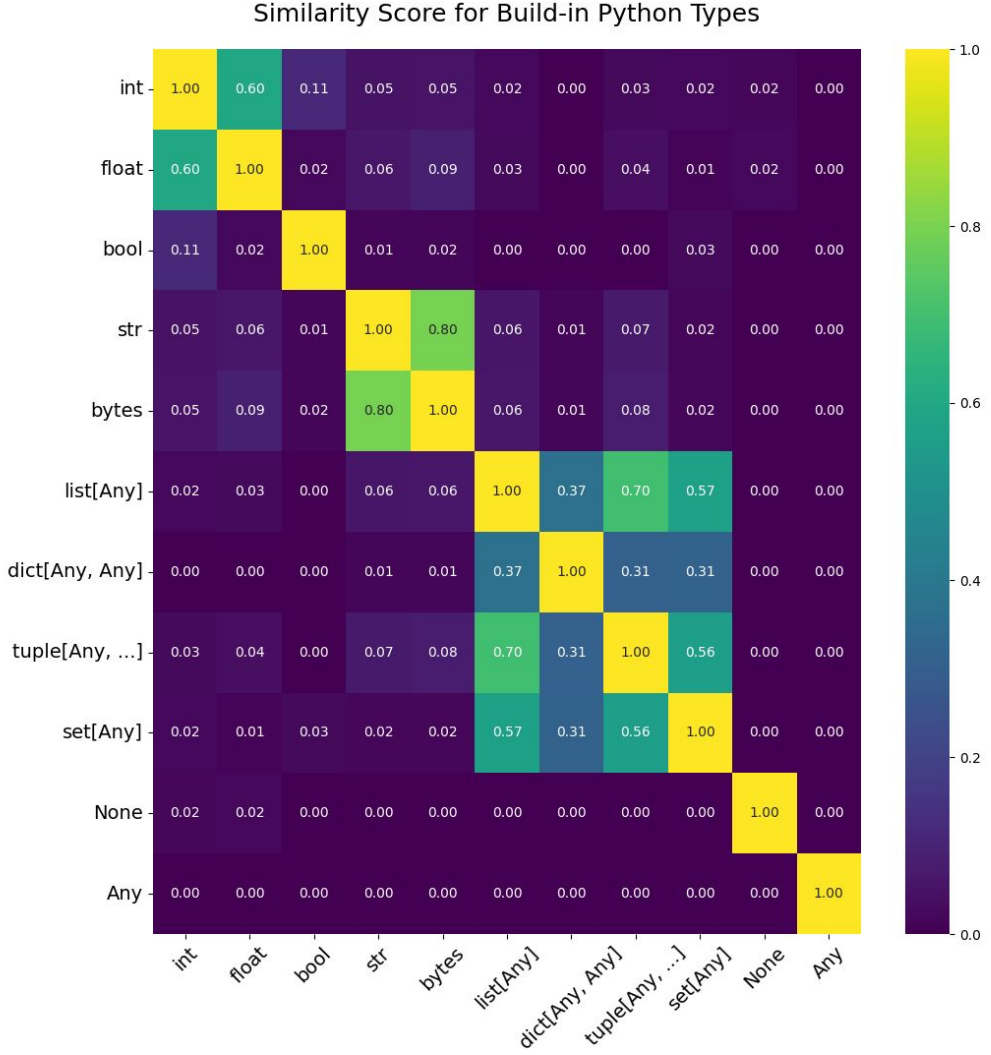
Two Novel Metrics – TypeSim

functional similarity + structural relationships

$$S(T1, T2) = \frac{|\ attrs(T1) \cap attrs(T2)\ |}{|\ attrs(T1) \cup attrs(T2)\ |}$$

Iterable vs Sequence: 0.92 > List vs Sequence: 0.7

int vs float : 0.6 > int vs str : 0.06

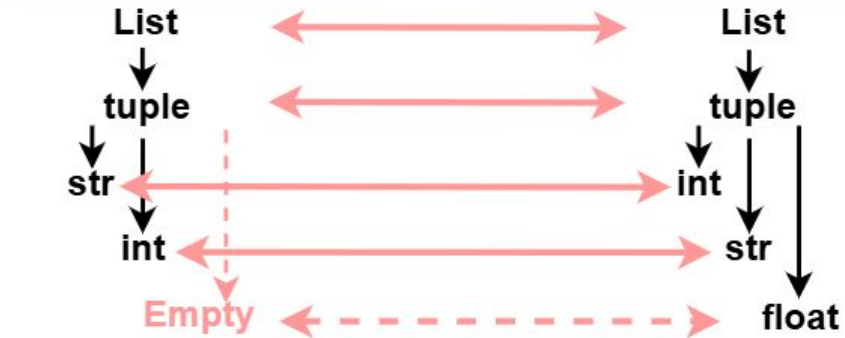

Similarity Score for Build-in Python Types
int 1.00 0.60 0.11 0.05 0.05 0.02 0.00 0.03 0.02 0.02 0.00
float 0.60 1.00 0.02 0.06 0.09 0.03 0.00 0.04 0.01 0.02 0.00
bool 0.11 0.02 1.00 0.01 0.02 0.00 0.00 0.00 0.03 0.00 0.00
str 0.05 0.06 0.01 1.00 0.80 0.06 0.01 0.07 0.02 0.00 0.00
bytes 0.05 0.09 0.02 0.80 1.00 0.06 0.01 0.08 0.02 0.00 0.00
list[Any] 0.02 0.03 0.00 0.06 0.06 1.00 0.37 0.70 0.57 0.00 0.00
dict[Any, Any] 0.00 0.00 0.00 0.01 0.01 0.37 1.00 0.31 0.31 0.00 0.00
tuple[Any, ...] 0.03 0.04 0.00 0.07 0.08 0.70 0.31 1.00 0.56 0.00 0.00
set[Any] 0.02 0.01 0.03 0.02 0.02 0.57 0.31 0.56 1.00 0.00 0.00
None 0.02 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00
Any 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
int float bool str bytes list[Any] dict[Any, Any] tuple[Any, ...] set[Any] None Any
1.0 0.8 0.6 0.4 0.2 0.0

Two Novel Metrics – TypeSim

functional similarity + structural relationships

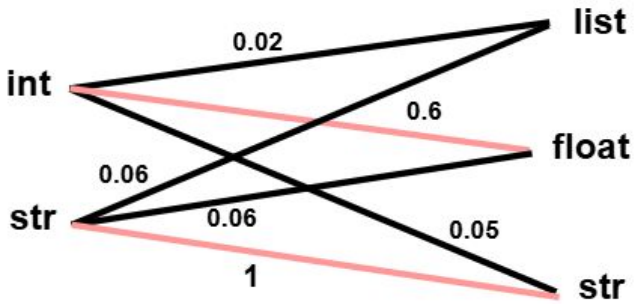
Ordered Arguments

List [tuple [str, int]] v.s. List [tuple [int, str, float]]



Permutable Arguments

Union (int, str) v.s. Union (list, float, str)



Two Novel Metrics – TypeCheck

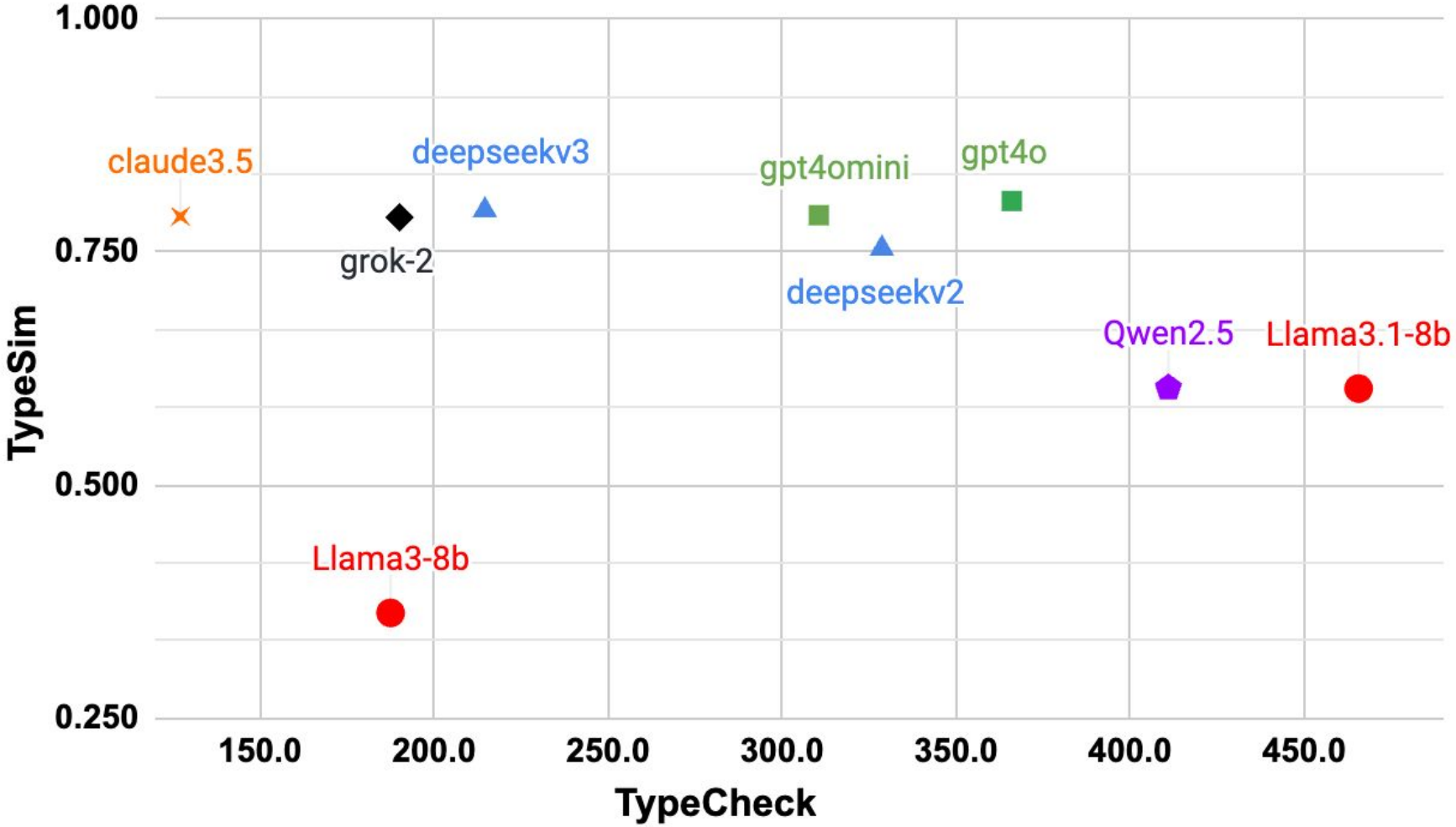
- **Predicted types** form a **coherent** system
- **Errors generated by static type checker:**
  - **Incompatible return types**
  - **Invalid argument types**

TypyBench Curation

We curate a benchmark dataset **TypyBench**, containing **50 well-typed** Python repositories from **GitHub** and **PyPI**.

Selection metrics: **Type Annotation Rate + Popularity + Complexity**

	# Repos	# Tokens	# Functions	# Cases
Train	20	8403760	31161	59966
Validation	10	4037666	13988	20177
Test	20	4983025	25101	46166


1.000
0.750
0.500
0.250
TypeSim
claude3.5
deepseekv3
grok-2
gpt4omini
gpt4o
deepseekv2
Qwen2.5
Llama3.1-8b
Llama3-8b
150.0
200.0
250.0
300.0
350.0
400.0
450.0
TypeCheck

Takeaways

- **Complementary metrics**
  - **TYPESIM** captures **functionality + structural** **similarity**
  - **TYPECHECK** highlights **consistency** issues.

Takeaways

- **Complementary metrics**
- **Future directions**:
  - Repo-level focus to improve **cross-file type consistency**
  - **Context length trade-offs**: challenges in handling large inputs/outputs