TYPYBENCH: EVALUATING LLM TYPE INFERENCE FOR UNTYPED PYTHON REPOSITORIES

Yuhe Jiang[*1,2], **Xun Deng**[*1], **Jiacheng Yang**[*1,2], **Honghua Dong**[*1,2]
Gennady Pekhimenko[1,2], **Fan Long**[1], **Xujie Si**[1,2]
[1]University of Toronto, [2]Vector Institute, [*]Equal contribution, alphabetically ordered

ABSTRACT

Type inference for dynamic languages like Python is a persistent challenge in software engineering. While large language models (LLMs) have shown promise in code understanding, their type inference capabilities remain underexplored. We introduce TYPYBENCH, a benchmark designed to evaluate LLMs' type inference across entire Python repositories. TYPYBENCH features two novel metrics: TYPESIM, which captures nuanced semantic relationships between predicted and ground truth types, and TYPECHECK, which assesses type consistency across codebases. Our evaluation of various LLMs on a curated dataset of 50 high-quality Python repositories reveals that, although LLMs achieve decent TYPESIM scores, they struggle with complex nested types and exhibit significant type consistency errors. These findings suggest that future research should shift focus from improving type similarity to addressing repository-level consistency. TYPYBENCH provides a foundation for this new direction, offering insights into model performance across different type complexities and usage contexts.

1 INTRODUCTION

Type inference, the ability to automatically deduce the types of variables and expressions in a program, has been a long-standing challenge in programming language research (Raychev et al., 2015; Hellendoorn et al., 2018). In dynamically-typed languages like Python, where explicit type annotations are optional, type inference involves analyzing code to determine appropriate type annotations that could have been written by developers. This capability has become increasingly important as codebases grow in size and complexity.

The significance of type information in modern software development cannot be overstated. Type annotations serve multiple crucial purposes: (1) they enhance code clarity by making developers' intentions explicit, (2) prevent type-related errors through early detection, (3) enable rich IDE features like autocompletion, and (4) facilitate maintenance and refactoring operations. The introduction of type hints through PEP 484[1] marked a pivotal moment for Python, acknowledging the growing importance of static typing in large-scale software development.

While the benefits of type annotations are clear, manually adding them to existing codebases is time-consuming and error-prone. This challenge has sparked interest in developing automatic type inference tools like Mypy (Lehtosalo et al.), Pyright (Microsoft) and MonkeyType (Instagram), as well as learning-based algorithms (Wei et al., 2020; Allamanis et al., 2020). Moreover, recent advances in large language models (LLMs) have shown promising results in code understanding tasks (Brown et al., 2020; Achiam et al., 2023; Chen et al., 2021), and in type inference tasks (Wei et al., 2023; Peng et al., 2023) with better performance than previous methods (Shivarpatna Venkatesh et al., 2024). Such tools can significantly reduce developer effort while improving code quality and maintainability.

Despite these advances, current evaluation benchmarks and approaches (Mir et al., 2021; Allamanis et al., 2020; Shivarpatna Venkatesh et al., 2024) for type inference methods face significant limitations. Traditional evaluation metrics rely heavily on exact matching (or up to parametric type (Allamanis et al., 2020)), which fails to capture important semantic relationships between types – for instance,

[1]https://peps.python.org/pep-0484/

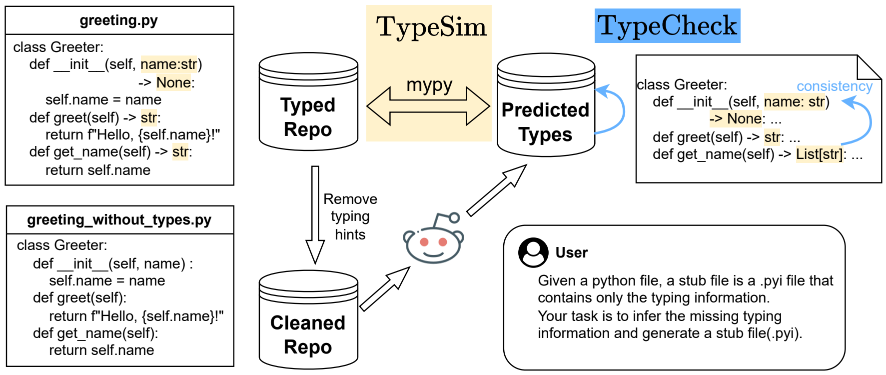


Figure 1: Overview of TYPYBENCH. We collect well-typed Python repositories and remove their typing information as the inputs. The outputs predicted by type inference methods are then evaluated using TYPESIM and TYPECHECK, where TYPESIM measures the functionality similarity between predicted and human-annotated types, and TYPECHECK evaluates type consistency across entire codebases through static type checking.

the functional similarity between `List` and `Sequence`, where developers may use them interchangeably. Furthermore, existing benchmarks often evaluate type inference in isolation, focusing on individual functions or files rather than considering type consistency across entire codebases. This disconnect between local correctness and global coherence makes it challenging to reliably assess the real-world effectiveness of type inference methods.

To address these challenges, we introduce two novel evaluation metrics, TYPESIM and TYPECHECK, as illustrated in Figure 1. TYPESIM measures the functional similarity between predicted and human-annotated types by incorporating both structural relationships and type hierarchies, providing a more nuanced evaluation than exact matching. Complementarily, TYPECHECK measures type consistency across entire codebases via static type checking[2], verifying that inferred types integrate coherently.

We then present TYPYBENCH, a benchmark of 50 high-quality Python repositories selected based on their type coverage, complexity, and domain diversity. Unlike existing benchmarks (Pradel et al., 2020; Mir et al., 2021), TYPYBENCH emphasizes repository-level evaluation, allowing the assessment of both local type accuracy and global type consistency. Our extensive evaluation of state-of-the-art LLMs on TYPYBENCH reveals key insights. While modern LLMs achieve respectable TYPESIM scores (up to 0.80), they struggle with TYPECHECK, highlighting a gap between local type accuracy and global consistency compared to human annotations. The main contributions of this paper are:

- Two novel metrics for evaluating type inference: TYPESIM for measuring semantic similarity between types, and TYPECHECK for assessing repository-wide type consistency.
- TYPYBENCH: A comprehensive benchmark of 50 high-quality Python repositories, designed to evaluate both local and global aspects of type inference.
- An extensive empirical study of state-of-the-art LLMs on type inference, revealing critical gaps between type prediction accuracy and consistency.

These contributions establish a foundation for future research in automated type inference, providing both the tools and insights needed to develop more effective type inference systems.

2 RELATED WORK

2.1 TYPE INFERENCE METHODS

Conventional Methods. Traditional type inference relies on static analysis and runtime tracing, as implemented in tools like Mypy (Lehtosalo et al.), Pyright (Microsoft), and MonkeyType (Instagram). These approaches offer high precision but are limited in coverage and require explicit type annotations or runtime information.

[2]We use the number of Mypy check errors to estimate the consistency of types.

Learning-based Methods. Early learning approaches like JSNice (Raychev et al., 2015) pioneered using probabilistic models to learn from existing codebases. This direction evolved to leverage various program representations, from natural language information (Malik et al., 2019) to graph structures (Hellendoorn et al., 2018; Wei et al., 2020; Allamanis et al., 2020; Cassano et al., 2023), though struggling to handle complex type structures and rare types.

LLM-based Methods. Recent work has shown that large language models can match or exceed traditional approaches in type inference tasks (Jesse et al., 2021; Wei et al., 2023; Peng et al., 2023). These methods benefit from pre-training on large code corpora and can leverage natural language understanding for improved type prediction.

2.2 Type Inference Benchmarks

Previous type inference benchmarks (Pradel et al., 2020; Mir et al., 2021; Allamanis et al., 2020) primarily relied on exact match accuracy for evaluation, with Typilus (Allamanis et al., 2020) introducing a relaxed "Match up to Parametric Type" metric that compares only the outermost type constructors. However, these metrics still fall short of capturing full semantic similarity between type annotations. Our benchmark advances this by (1) introducing semantic similarity metrics that better capture the hierarchical and structural relationships between types, and (2) evaluating practical usability through type checking. By requiring predictions in the form of PEP 484 stub files (.pyi), we enable direct validation using production-grade type checkers, providing a more realistic assessment of type inference quality.

2.3 Other Coding Benchmarks

Code-related benchmarks have evolved from isolated to context-dependent evaluations:

Function-level Benchmarks. Traditional benchmarks focused on self-contained programming tasks (Chen et al., 2021; Jain et al., 2024; Zhuo et al., 2024), evaluating specific capabilities like code generation and problem-solving.

Repository-level Benchmarks. Recent work has shifted toward repository-scale assessment, with each benchmark evaluating distinct aspects of code understanding: RepoBench (Liu et al., 2024b) focuses on code completion, SWE-bench (Jimenez et al., 2024) tests bug fixing capabilities, and RepoTransBench (Wang et al., 2024) evaluates cross-language translation. Our work contributes to this ecosystem by examining models' ability to perform consistent type inference across entire repositories, adding another crucial dimension to repository-level model evaluation.

3 Background

This section outlines essential concepts in Python's type system and type inference, which form the foundation of our work.

Gradual Typing in Python. Python supports gradual typing, allowing developers to incrementally add type annotations while maintaining compatibility with untyped code. Introduced in PEP 484, type hints enable specifying types for function parameters, return values, and variables:

```
def greet(name: str) -> str:
    return f"Hello,␣{name}!"
```

These optional annotations do not affect runtime behavior, serving primarily as documentation and enabling static analysis tools to catch type-related errors before execution.

Type Inference. Type inference is the process of automatically deducing appropriate type annotations for variables and expressions in a program. In our context, given a Python repository without type annotations, the goal is to infer types that could have been written by developers:

```
# Original untyped code
def greet(name):
    return f"Hello,␣{name}!"
```

```
# After type inference
def greet(name: str) -> str:
    return f"Hello,␣{name}!"
```

Here, the return type is `str` based on the returned value. The parameter `name` is likely to be `str` based on the semantic information, since the name of the variable is `name` and the function is `greet`.

Type Stub Files. Python uses `.pyi` stub files to separate type information from implementation. These files contain only function signatures and type definitions:

```
# greetings.pyi
def greet(name: str) -> str: ...
```

Stub files enable type checking without modifying source files and are commonly used in library distributions to provide type information.

Static Type Checking. Static type checking verifies type consistency before program execution. Tools like mypy[3] analyze code to detect potential type errors, which include but are not limited to: (1) verifying type compatibility in assignments and function calls, (2) checking subtype relationships (e.g., `List[int]` is a subtype of `Sequence[int]`), (3) ensuring consistent usage of types across modules. In our example, mypy would detect errors like:

```
msg = greet("TypyBench") # Pass, "TypyBench" is str
msg = greet([1, 2, 3]) # Error: Expected str, got list[int]
```

Mypy performs this analysis by constructing a type dependency graph and propagating type constraints through the program, identifying violations of type rules defined in PEP 484 and related specifications.

4 Metric Design

To effectively evaluate type inference systems, we introduce two complementary metrics: TypeSim, which measures type prediction quality, and TypeCheck, which assesses type coherence across codebases.

4.1 Type Similarity

Traditional evaluation methods for type inference rely on exact matching, which fails to capture the nuanced relationships between types. For example, `Sequence[int]` and `Iterable[int]` share most functionality but would receive a score of 0 under exact matching, even though both support iteration operations. Similarly, `int` and `float` would be considered completely different despite sharing most arithmetic operations. To solve this issue, we propose TypeSim (Algorithm 1), a continuous similarity metric that considers both functional similarity and structural relationships.

Algorithm 1 TypeSim

Input: types T, T'
if At least one of T, T' is Union **then**
  Return: SetCompare$(as\_set(T), as\_set(T'))$
end if
$score = s(T.root, T'.root)$
if Both T, T' have arguments **then**
  $score = \frac{1}{2}(score + \textbf{ListCompare}(T.args, T'.args))$
else if One of T, T' has arguments **then**
  $score = \frac{score}{2}$
end if
Return: $score$

4.1.1 Base Type Similarity

For non-generic types, we compute similarity based on their supported operations and methods. Given two types t and t', their similarity is:

[3] https://mypy.readthedocs.io/

$$s(t,t') = \frac{|attrs(t) \cap attrs(t')|}{|attrs(t) \cup attrs(t')|}$$

where $attrs(t)$ represents the set of methods and operations supported by type t, excluding those common to all types that inherit from `object` (like `__str__` or `__init__`). We then use the Jaccard index to measure functional similarity, where two types are similar if they share most of the same methods and operations. This approach captures the intuition that types supporting similar operations should be considered similar.

Consider Python's collection types hierarchy as an example:

- `Iterable` provides `__iter__` for iteration, enabling `for` loops.
- `Sequence` adds `index`, `count`, and length operations to `Iterable`, supporting indexed access.
- `List` adds mutable operations (e.g., `append`, `pop`) to `Sequence`, enabling list modification.

This leads to meaningful similarities: $s(\texttt{Iterable}, \texttt{Sequence}) = 0.92$ as they share core iteration functionality, while $s(\texttt{Sequence}, \texttt{List}) = 0.7$ reflects their additional differences in mutability. Similarly, $s(\texttt{int}, \texttt{float}) = 0.6$ captures their shared arithmetic operations, while $s(\texttt{int}, \texttt{str}) = 0.06$ reflects their fundamental differences. See Appendix B for the TypeSim between builtin types.

4.1.2 Structural Similarity

ListCompare. For generic types (e.g., `List[int]`, `Dict[str, int]`), we compute similarity recursively through their type tree structure, as shown in Figure 2. This allows us to handle nested types of arbitrary depth while considering both the container types and their type arguments.

As shown on the left of Figure 2, for non-union types T and T', their similarity is:

$$S(T,T') = \frac{1}{2}(s(root, root') + S_{list}(args(T), args(T'))),$$

where $s(root, root')$ is the base type similarity, and S_{list} compares type arguments in order (see Algorithm 2 in Appendix E):

$$S_{list}(L, L') = \frac{\sum_{i \le \min(|L|,|L'|)} S(L_i, L'_i)}{\max(|L|, |L'|)}.$$

For example, comparing `List[int]` with `Sequence[float]`:

- Base similarity: $s(\texttt{List}, \texttt{Sequence}) = 0.7$ (shared sequence operations)
- Argument similarity: $S(\texttt{int}, \texttt{float}) = 0.6$ (shared numeric operations)
- Overall: $S = \frac{1}{2}(0.7 + 0.6) = 0.65$

We choose to average the similarity of the root and the arguments instead of multiplying them to give more weight to root types. This design aligns with common development practices where developers often annotate only the root type (e.g., `List`) without specifying arguments, especially during initial typing efforts. For example, when computing $S(\texttt{List}, \texttt{List[int]})$, while the argument similarity is 0 due to the missing argument, the base similarity is 1 for matching root types. Averaging yields 0.5, acknowledging the partial correctness of the annotation, whereas multiplication would give 0, completely penalizing this common and often acceptable practice in gradual typing.

SetCompare. As shown on the right of Figure 2, we treat union types (e.g., `Union[int, str]`) as sets of possible types and compute an optimal matching between their members. This approach accounts for unordered union members and allows partial matches.

Given types T and T' (at least one is a union), $S(T,T') = S_{set}(as\_set(T), as\_set(T'))$, where $as\_set(T)$ converts a type to its member set:

$$as\_set(T) = \begin{cases} \{T\} & \text{if T is not Union.} \\ \{T.args\} & \text{if T is Union.} \end{cases}$$

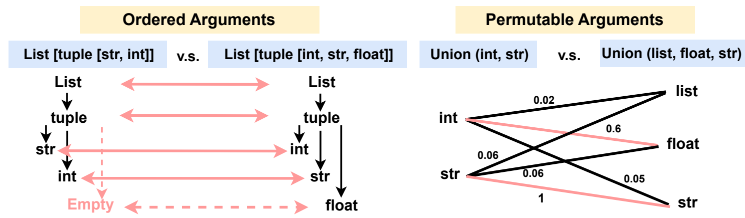


Figure 2: Examples of type similarity computation. Left: List-wise comparison for generic types, where arguments are compared in order. Right: Set-wise comparison for Union types, where an optimal matching is computed between members.

S_{set} finds the optimal matching between members (see Algorithm 3 in Appendix E):

$$S_{set}(A, B) = \frac{\sum_{(i,j)\in M} S(A_i, B_j)}{\max(|A|, |B|)}.$$

4.2 Type Consistency

While TypeSim measures how close the predictions are to human annotations, TypeCheck evaluates whether the predicted types form a coherent system across the codebase. Since type annotations serve not only as documentation but also as a mechanism for early error detection and code maintenance, it is crucial that predicted types work together consistently across the entire codebase.

We use the number of mypy errors as a proxy for type consistency because these errors directly reflect what developers would need to fix before the type annotations become practically useful. For example, if a function is predicted to return `List[int]` but is used in a context expecting `List[str]`, this inconsistency would prevent effective static type checking and IDE support – two key benefits of type annotations. Specifically, we focus on meaningful type errors that affect code correctness, such as incompatible return types and invalid argument types. These errors indicate real issues that would hinder code maintenance and refactoring. A complete list of counted error types is provided in Appendix C.2.

5 Dataset Curation

We curate a benchmark dataset containing 50 popular Python repositories from GitHub and PyPI to evaluate type inference capabilities. The dataset construction involves two key steps: repository selection and cleaning.

Repository Selection. We select candidate repositories from GitHub's trending repositories and frequently downloaded PyPI Packages. In the initial filtering stage, we enforce the constraints of a maximum of 1.5M tokens, a minimum of 30 Python files, at least 50% typed functions, and valid mypy configurations to ensure the suitability of type inference evaluation. We then define a quality score as below and rank the candidate repositories by their score: $S = \alpha S_{\text{coverage}} + \beta S_{\text{popularity}} + \gamma S_{\text{complexity}}$, where S_{coverage} measures the percentage of typed functions, $S_{\text{popularity}}$ is calculated from the number of GitHub stars and PyPI downloads, and $S_{\text{complexity}}$ considers the depth and variety of type annotations. The top 50 repositories with the highest quality scores are selected into the dataset.

Repository Cleaning. To ensure the best quality of type inference evaluation, we remove non-Python files, testing files, and irrelevant files, to only keep the Python files in the source folder containing the main functionality. To ensure the quality of type annotations, we run mypy check on the original repository, and manually resolve errors that stop the check from running.

After initial cleaning, we remove type annotations using scripts while preserving code functionality, creating input–ground truth pairs for evaluation (shown in Figure 1). The type removal algorithm handles function signatures and variable declarations (detailed in Algorithm 4, Appendix E). Note that TypyBench does not include type evaluation on local variables (within functions). To maximize consistency and similarity with the original repository, every single part of each Python program is kept unchanged except for the type annotation. We rebuild each processed repository to ensure that the modified code remains syntactically valid and the runtime behaviors are preserved.

Table 1: The total number of tokens, functions, and variables to be inferred for different splits.

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

Table 2: Average TYPESIM and TYPECHECK scores of all repositories for various models. CLAUDE-3.5-SONNET shows the best TYPECHECK score while top models share similar TYPESIM scores.

MODEL	TYPECHECK ↓	TYPESIM ↑	TYPESIM WO MISSING ↑	MISSING RATE ↓	TYPESIM ON RARE ↑
LLAMA-3-8B	115.2	0.363	0.731	0.508	0.280
LLAMA-3.1-8B	289.9	0.603	0.804	0.261	0.559
QWEN-2.5-7B	196.3	0.604	0.787	0.238	0.540
GPT-4O	130.1	**0.804**	0.893	**0.099**	**0.741**
GPT-4O-MINI	164.9	0.789	0.893	0.116	0.708
CLAUDE-3.5-SONNET	**72.13**	0.788	0.893	0.119	0.740
DEEPSEEK-V2.5	128.7	0.754	**0.907**	0.169	0.700
DEEPSEEK-V3	129.5	0.795	0.897	0.115	0.736
GROK-2	115.2	0.787	0.903	0.129	0.732

Benchmark Statistics. To facilitate the potential evaluation of learning-based methods, we split the 50 repositories into Train/Validation/Test splits, with 20/10/20 repositories correspondingly. The 20 test repositories are selected and further split into two sets with 10 repositories each based on the date created to estimate the level of data contamination.

Table 1 summarizes the number of tokens, functions, and variables to be inferred for different splits. We also depict the diversity of TYPYBENCH in Figure 3 by classifying the repositories into domains spanning Developer Tools, ML/AI, Web/API, and Security. For more comprehensive details about each repository, please refer to Appendix A.

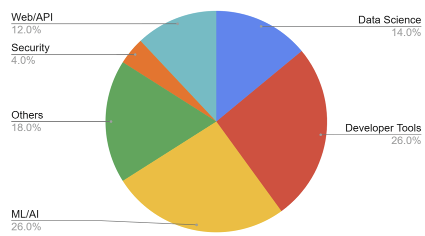


Figure 3: Distribution of repository categories in TYPYBENCH, covering major Python application domains.

6 EXPERIMENTS

In this section, we conduct comprehensive experiments to answer the following key questions:

- **Model Readiness:** How well do current LLMs perform on type inference? Are SOTA models ready for production use on untyped repositories? What are their key limitations? Does a longer context length help mitigate the limitations?
- **Metric Effectiveness:** How do our proposed TYPESIM and TYPECHECK metrics compare to traditional exact matching? What additional insights do they provide?
- **Performance Factors:** How do factors like type complexity and repository age affect model performance? What do these patterns reveal about LLMs' type inference capabilities?

6.1 EXPERIMENTAL SETUP

We evaluate type inference capabilities across a diverse set of SOTA LLMs, including API-accessed models and local-hosted small models. The API-based LLMs include GPT-4O, GPT-4O-MINI (Achiam et al., 2023), CLAUDE-3.5-SONNET (Anthropic, 2024), DEEPSEEK-V2.5, DEEPSEEK-V3 (Liu et al., 2024a), and GROK-2 (XAI, 2024). For local-hosted models, we evaluate popular models including LLAMA-3-8B, LLAMA-3.1-8B (Dubey et al., 2024), and QWEN-2.5-7B (Yang et al., 2024).

While our benchmark provides train/validation/test splits for future learning-based methods, we evaluate pre-trained LLMs on all repositories since fine-tuning is not performed on the training set. We separately analyze potential pre-training data contamination through temporal analysis (Section 6.3). Since most repositories contain more number of tokens than the context length of most LLMs, LLMs are tested in a file-by-file method, where we require LLMs to infer the `.pyi` stub file from the type-removed file with the same prompt (see Figure 7 and Appendix C for more details).

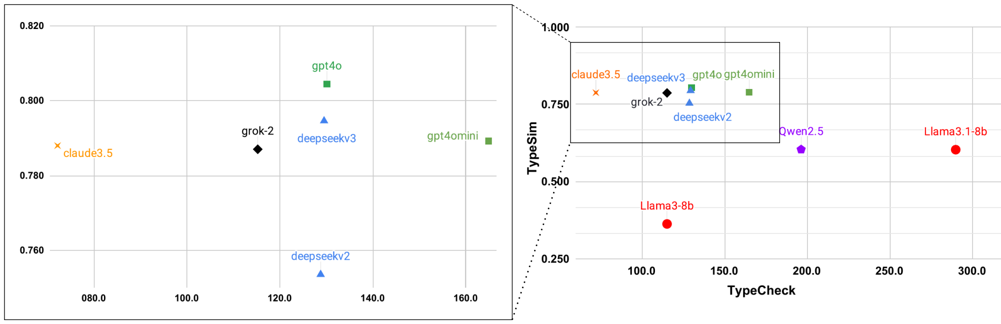


Figure 4: Comparison of TYPESIM and TYPECHECK metrics across models. SOTA models achieve similar high TYPESIM scores while varying in TYPECHECK, showing TYPECHECK's additional discriminative power. Small local-hosted models perform poorly.

Table 3: Compare TYPESIM and exact match by type depth, the difference increases with depth.

	TYPESIM ↑					EXACT MATCH ↑				
MODEL	DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5	DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
LLAMA-3-8B	0.415	0.271	0.235	0.209	0.136	0.406	0.197	0.113	0.056	0.023
LLAMA-3.1-8B	0.635	0.555	0.499	0.478	0.454	0.610	0.427	0.279	0.201	0.118
QWEN-2.5-7B	0.658	0.525	0.475	0.443	0.438	0.640	0.420	0.277	0.191	0.113
GPT-4O	0.822	0.775	0.753	0.733	0.776	0.792	0.653	0.497	0.417	0.350
GPT-4O-MINI	0.813	0.747	0.703	0.721	0.705	0.782	0.616	0.420	0.370	0.209
CLAUDE-3.5-SONNET	0.801	0.768	0.746	0.749	0.746	0.769	0.652	0.517	0.427	0.303
DEEPSEEK-V2.5	0.771	0.722	0.697	0.627	0.676	0.745	0.609	0.462	0.320	0.218
DEEPSEEK-V3	0.809	0.769	0.747	0.702	0.708	0.774	0.644	0.483	0.397	0.295
GROK-2	0.802	0.757	0.737	0.710	0.743	0.771	0.642	0.496	0.387	0.274
AVERAGE	0.725	0.654	0.621	0.597	0.598	0.699	0.540	0.394	0.307	0.211
DIFF						**-0.026**	**-0.114**	**-0.228**	**-0.289**	**-0.387**

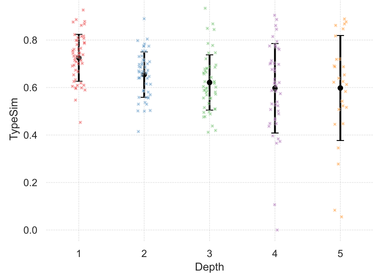
6.2 MAIN EVALUATION ANALYSIS

As shown in Table 2, our analysis reveals both promising advances and concerning limitations in LLM-based type inference. While SOTA models achieve decent TYPESIM scores around 0.80, their non-negligible missing rates suggest systematic limitations in stably generating correct stub files. Moreover, as illustrated in Figure 4, the TYPECHECK metric reveals an interesting pattern in model consistency. While all models struggle with type checking, CLAUDE-3.5-SONNET is the best one to keep the type predictions consistent despite not having the highest TYPESIM. This consistency could be particularly valuable when humans need to fix the type-checking errors manually.

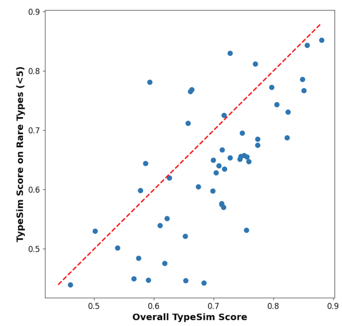
6.3 FACTORS ANALYSIS

Impact of Type Complexity. We first compare TYPESIM with exact match metrics for types with different depths. Table 3 reveals an increasingly widening gap at higher depths. While both metrics show declining trends, exact match scores drop more precipitously – nearly vanishing for types of depth 3 and above. In contrast, TYPESIM still captures semantically valid predictions that would be completely rejected by exact matching. This demonstrates TYPESIM's value in providing more nuanced evaluation, particularly for complex types where multiple valid type annotations may exist. As shown in Figure 5(a), model performance consistently slightly degrades as type complexity increases with increased variance. Even SOTA models struggle with deeper nested types (depth > 2), suggesting that complex type inference remains a significant challenge in some cases.

Impact of Type Frequency. As pointed out in previous work (Allamanis et al., 2020), predicting rare types that are less frequent in the repository is a challenge. As shown in Table 2, we still observe a gap between the TYPESIM scores on all types and the scores on rare types, but the gap is not that large overall. However, when it comes to specific repositories, as shown in Figure 5(b), the drop is significant in many repositories, indicating predicting rare types is still challenging for LLMs. An interesting observation is that TYPESIM on rare types could also be higher than the overall TYPESIM in a minority of repositories.



(a) TYPESIM scores v.s. type complexity



(b) TYPESIM overall v.s. on rare types

Figure 5: (a) Models achieve consistent high scores types with depth 1, but show degraded performance and increased variance for more complex types. (b) Scatter plot of average TYPESIM score on rare types and all types for each repository. A clear drop on the score is observed on 40 out of 50 repositories (under the red line).

Table 4: Comparision between full repo context and single file context with GPT-4O. The TYPECHECK errors decrease significantly with the whole repository context, but the TYPESIM scores also decrease significantly on some repositories.

		FULL REPO		SINGLE FILE	
MODEL	REPO	TYPESIM ↑	TYPECHECK ↓	TYPESIM ↑	TYPECHECK ↓
GPT-4O	GPTME	0.966	15	0.877	107
	PRIVATE-GPT	0.601	0	0.855	13
	SCREENSHOT-TO-CODE	0.696	0	0.915	9

Impact of Repo-Level Context. To reduce the TYPECHECK errors in the predicted types, the most straightforward way is to use the whole repository as the context given to LLMs. However, this approach faces two major challenges, long context length and long output length, due to the size of the whole repository. Nevertheless, we tested this approach with GPT-4O on 3 repositories with total number of tokens smaller than 64k. As shown in Table 4, the TYPECHECK errors do decrease significantly with the whole repository context, with the cost of worse TYPESIM scores, potentially due to harder to exact the correct information from the long context and harder to generate all the predictions in the correct format. This suggests that more context is helpful for enhancing the type consistency, but the long input and output challenges need to be addressed.

Impact of Data Contamination. Data contamination is a potential issue when evaluating the performance of LLMs (Roberts et al., 2023). To verify this, we compare the TYPESIM scores of repositories created in different years. We observe that the TYPESIM scores of repositories created in 2024 and after are lower than those of repositories created before 2024 (see Figure D.1), suggesting that future work should use test sets containing relatively recent repositories when comparing the performance of different models. We include the evaluation results of the test set in Table 7 in Appendix D.

7 CONCLUSION

We present TYPYBENCH, a comprehensive benchmark for evaluating LLMs' Python type inference capabilities. Our evaluation reveals that while SOTA models achieve promising TYPESIM scores, they still face significant challenges: poor handling of complex nested types, and substantial type consistency errors. The proposed TYPESIM and TYPECHECK metrics provide complementary insights, with TYPESIM capturing semantically valid predictions and TYPECHECK revealing critical consistency issues. The experimental finding suggests that the focus of type inference should turn to the repo-level consistency since the similarity is already high. We further find that increased context length improves type consistency but creates challenges in handling long inputs and outputs, suggesting the need for more efficient context handling mechanisms. We hope TYPYBENCH will facilitate progress in LLM-based type inference by providing a standardized evaluation framework and highlighting key areas for improvement.

ACKNOWLEDGMENTS

We thank Jialiang Sun for his valuable contributions to data collection and curation, Shiwen Wu for her insightful discussion, and Chris J. Maddison for his support on this project. We also thank the Microsoft Accelerating Foundation Models Research (AFMR) program for its generous support through Azure credits.

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Allamanis, M., Barr, E. T., Ducousso, S., and Gao, Z. Typilus: Neural type hints. In *Proceedings of the 41st acm sigplan conference on programming language design and implementation*, pp. 91–105, 2020.

Anthropic. Claude 3.5 sonnet, 2024. Available at: https://www.anthropic.com/news/claude-3-5-sonnet.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Cassano, F., Yee, M.-H., Shinn, N., Guha, A., and Holtzen, S. Type prediction with program decomposition and fill-in-the-type training, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Dong, H., Su, Q., Gao, Y., Li, Z., Ruan, Y., Pekhimenko, G., Maddison, C. J., and Si, X. Appl: A prompt programming language for harmonious integration of programs and large language model prompts. *arXiv preprint arXiv:2406.13161*, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Hellendoorn, V. J., Bird, C., Barr, E. T., and Allamanis, M. Deep learning type inference. In *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pp. 152–162, 2018.

Instagram. Monkeytype: A python library that generates static type annotations by collecting runtime types. https://github.com/Instagram/MonkeyType.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Jesse, K., Devanbu, P. T., and Ahmed, T. Learning type annotation: Is big data enough? In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1483–1486, 2021.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Lehtosalo, J., van Rossum, G., Levkivskyi, I., and Sullivan, M. J. mypy. http://mypy-lang.org/. Available at http://mypy-lang.org/.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Liu, T., Xu, C., and McAuley, J. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*, 2024b.

Malik, R. S., Patra, J., and Pradel, M. Nl2type: Inferring javascript function types from natural language information. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 304–315. IEEE, 2019.

Microsoft. Pyright: Static type checker for python. `https://github.com/microsoft/pyright`.

Mir, A. M., Latoškinas, E., and Gousios, G. Manytypes4py: A benchmark python dataset for machine learning-based type inference. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 585–589. IEEE, 2021.

Peng, Y., Wang, C., Wang, W., Gao, C., and Lyu, M. R. Generative type inference for python. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 988–999. IEEE, 2023.

Pradel, M., Gousios, G., Liu, J., and Chandra, S. Typewriter: Neural type prediction with search-based validation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 209–220, 2020.

Raychev, V., Vechev, M., and Krause, A. Predicting program properties from" big code". *ACM SIGPLAN Notices*, 50(1):111–124, 2015.

Roberts, M., Thakur, H., Herlihy, C., White, C., and Dooley, S. Data contamination through the lens of time. *arXiv preprint arXiv:2310.10628*, 2023.

Shivarpatna Venkatesh, A. P., Sabu, S., Wang, J., M. Mir, A., Li, L., and Bodden, E. Typeevalpy: A micro-benchmarking framework for python type inference tools. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pp. 49–53, 2024.

Wang, Y., Wang, Y., Wang, S., Guo, D., Chen, J., Grundy, J., Liu, X., Ma, Y., Mao, M., Zhang, H., et al. Repotransbench: A real-world benchmark for repository-level code translation. *arXiv preprint arXiv:2412.17744*, 2024.

Wei, J., Goyal, M., Durrett, G., and Dillig, I. Lambdanet: Probabilistic type inference using graph neural networks. In *International Conference on Learning Representations*, 2020.

Wei, J., Durrett, G., and Dillig, I. Typet5: Seq2seq type inference using static analysis. In *The Eleventh International Conference on Learning Representations*, 2023.

XAI. Grok-2 beta release, 2024. Available at: `https://x.ai/blog/grok-2`.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

Table 5: The statistics of all repositories in TYPYBENCH, including the total number of tokens, functions, variables to be inferred, the ratio of functions being annotated, the category, and the date created. The test sets are further split into two test1 and test2 based on the date created to test the data contamination issue.

REPOSITORY	# TOKENS	# FUNC	# CASES	FUNC. ANNOT.	AVG DEPTH	CATEGORY	DATE CREATED
			TRAIN				
PRE-COMMIT-HOOKS	16268	105	810	1.00	1.57	DEVELOPER TOOLS	2014/03/13
FLAKE8	34684	209	747	1.00	1.4	DEVELOPER TOOLS	2014/09/13
VNPY	47762	415	909	0.99	1.11	DATA SCIENCE	2015/03/02
GHUNT	63135	284	2185	0.62	1.91	SECURITY	2020/10/02
SPOTIFY-DOWNLOADER	89384	232	2143	0.90	1.32	OTHERS	2016/07/06
DEEPFACE	90831	194	3263	0.57	1.4	OTHERS	2020/02/08
URLLIB3	98963	551	1844	1.00	1.49	DEVELOPER TOOLS	2011/09/18
FASTAPI	137361	307	1983	0.93	1.7	WEB/API	2018/12/08
POETRY	153087	1022	3556	1.00	1.32	DEVELOPER TOOLS	2018/02/28
HAYSTACK	209305	775	2035	0.87	1.57	ML/AI	2019/11/14
RICH	244573	948	518	0.95	1.21	DEVELOPER TOOLS	2019/11/10
PYDANTIC	359486	1780	565	0.98	1.26	DEVELOPER TOOLS	2017/05/03
PHIDATA	552011	1799	2687	0.93	1.23	DEVELOPER TOOLS	2022/05/04
PILLOW	607377	3460	3923	0.99	1.35	DATA SCIENCE	2012/07/24
SPHINX	612272	4647	10285	1.00	1.27	DEVELOPER TOOLS	2015/01/02
FACESWAP	732572	3352	441	0.75	1.46	OTHERS	2017/12/19
STREAMLIT	806972	3236	3991	0.83	1.34	WEB/API	2019/08/24
GRADIO	874060	1845	1400	0.62	1.88	WEB/API	2018/12/19
PIP	1176602	5369	4789	0.62	1.36	DEVELOPER TOOLS	2011/03/06
BLACK	1497055	631	11892	0.99	1.28	DEVELOPER TOOLS	2018/03/14
			VALIDATION				
TYPER	38236	167	1588	0.96	1.16	DEVELOPER TOOLS	2019/12/24
PRE-COMMIT	50380	345	2391	1.00	1.42	DEVELOPER TOOLS	2014/03/13
FLASK	73148	364	834	1.00	1.51	WEB/API	2010/04/06
PDM	172537	1291	532	0.98	1.37	DEVELOPER TOOLS	2019/12/27
MANIM	179218	1733	415	0.82	1.29	DATA SCIENCE	2020/05/19
NICEGUI	206638	1217	6018	0.98	1.09	WEB/API	2021/05/07
OPENAI-PYTHON	274146	1085	263	1.00	1.35	ML/AI	2020/10/25
TAIPY	403846	2356	2064	0.75	1.41	WEB/API	2022/02/18
OPENBB	1290501	3771	3137	0.70	1.8	DATA SCIENCE	2020/12/20
CAPA	1349016	1659	2935	0.64	1.53	SECURITY	2020/06/16
			TEST1				
PRIVATE-GPT	45562	197	257	0.98	1.36	ML/AI	2023/05/02
GPTME	58715	319	512	0.79	1.51	ML/AI	2023/03/24
PAPER-QA	73284	353	764	0.95	1.64	ML/AI	2023/02/05
PANDAS-AI	127754	996	1145	0.66	1.21	ML/AI	2023/04/22
SUPERVISION	150793	505	1101	0.90	1.49	DATA SCIENCE	2022/11/28
GPT4FREE	168395	679	808	0.76	1.11	ML/AI	2023/03/29
AUTOGPT	306046	1797	2235	0.78	1.34	ML/AI	2023/03/16
MLC-LLM	384359	1698	3182	0.83	1.25	ML/AI	2023/04/29
DB-GPT	817402	5329	9732	0.82	1.26	ML/AI	2023/04/13
VLLM	1037766	5271	12064	0.88	1.22	ML/AI	2023/02/09
			TEST2				
SCREENSHOT-TO-CODE	44482	60	102	0.73	1.58	OTHERS	2023/11/14
EXO	69991	406	721	0.61	1.28	OTHERS	2024/06/24
TEN-AGENT	71448	412	1076	0.76	1.07	OTHERS	2024/06/19
GPT-PILOT	94918	516	1101	0.82	1.19	ML/AI	2023/08/16
APPWORLD	156441	1125	2185	0.90	1.42	OTHERS	2024/06/23
AGENTS	156679	966	1956	0.88	1.14	OTHERS	2023/10/19
LEROBOT	183740	612	1068	0.57	1.69	OTHERS	2024/01/26
LLAMA-FACTORY	194043	520	1778	0.88	1.52	ML/AI	2023/05/28
COMPOSIO	345846	1059	1963	0.90	1.44	DATA SCIENCE	2024/02/23
UNSTRACT	495361	2281	2416	0.89	1.13	DATA SCIENCE	2024/02/21

A REPOSITORY INFORMATION

The statistics of all repositories are shown in Table 5. We use the tokenizer `tiktoken_cl100k_base` to count the number of tokens.

A.1 TYPECHECK RESULTS

Table 6 shows the results of running `mypy check` in the original repository from the selected subset of 15 repositories.

Table 6: Repositories chosen to report TypeCheck Scores.

Repo	TypeCheck Baseline	Category	# Tokens
urllib3	0	Train	98963
pre-commit	0	Validation	50380
pre-commit-hooks	0	Train	16268
flake8	0	Train	34684
spotify-downloader	0	Train	89384
black	0	Train	1497055
fastapi	0	Train	137361
typer	0	Validation	38236
gptme	0	Test1	58715
private-gpt	0	Test1	45562
screenshot-to-code	0	Test2	44482
composio	0	Test2	345846
flask	1	Validation	73148
rich	1	Train	244573
pydantic	2	Train	359486

B Basic Type Similarities

We illustrate the TypeSim between builtin types in Figure 6.

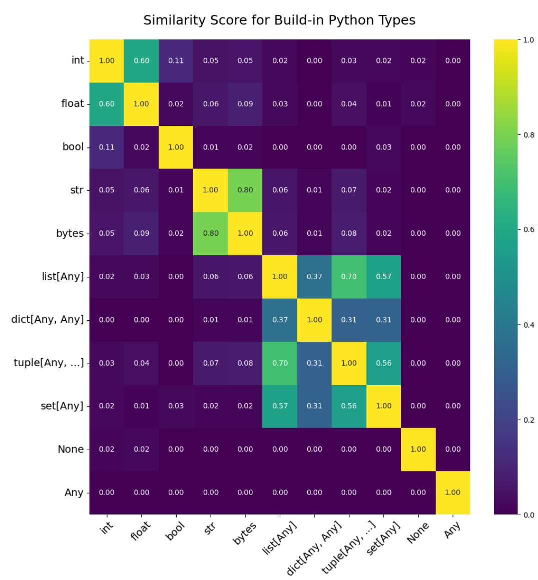


Figure 6: TypeSim scores between builtin types.

C Experimental Settings

We use Python version `3.12` and Mypy version `1.11.1` for all our experiments. For API models with multiple versions, we use GPT-4O-2024-08-06, GPT-4O-MINI-2024-07-18, CLAUDE-3-5-SONNET-20240620, and GROK-2-1212.

We use APPL (Dong et al., 2024) to implement the evaluation for LLMs.

C.1 PROMPT

Figure 7 illustrates the prompt used for single-file type inference in our experiment. The prompt includes a Python file along with its corresponding .pyi file as an example. Additionally, it specifies the expected answer format and emphasizes that the generated output must be free of syntax errors.

User

Given a python file, a stub file is a .pyi file that contains only the typing information.
Here is an example of an input python file that is missing typing information:
{python_example}
Here is the correct stub file generated for the above python file:
{pyi_example}
Your task is to infer the missing typing information and generate a stub file(.pyi) for a newly given python file.
You must generate the stub file in markdown format: ```python ↵ <stub content> ↵```.
The generated content must be a valid python file with no syntax error.
Note that relevent import statements should be included in the stub file.
Here is the python file for your task:
{file_content}

Assistant

. . .

Figure 7: Prompt Template for Single File Context

Figure 8 illustrates the prompt used for full-repo context type inference in our experiment. We first input the structure of the repository and followed by the content of the source code. Additionally, we specifies the expected answer and path format.

User

You are given a repository with repository directory structure and the source files.
Inputs
==== directory structure ====
{director structure}
==== source ====
The contents of the source code are as follows:
{source code}
Task
A stub file is a .pyi file that contains only typing information.
Your task is to generate a stub file(.pyi) for all the pytho files in the repository.
Stub files must be free of syntax error.
Your output should follow the format example below:
stubfilepath: `<relative path>`
``` python ↵ <stub content> ↵```

Assistant

. . .

Figure 8: Prompt Template for Whole Repo Context

C.2 TYPECHECK SCORE CALCULATION

We classify the following Mypy error types as indicators of consistency score: `attr-defined`, `assignment`, `arg-type`, `union-attr`, and `index`.
```

D Supplementary Experimental Results

D.1 Results on Test Set

We present an observation of data contamination in Figure D.1.

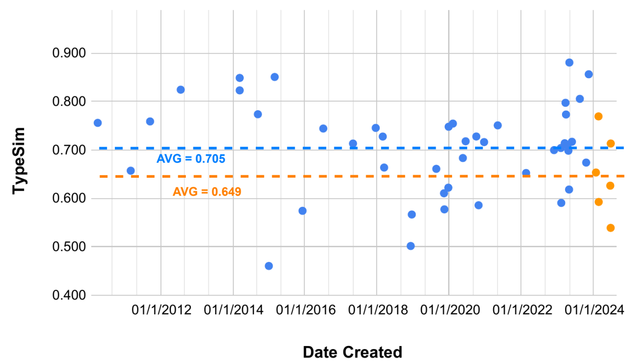


Figure 9: TypeSim on all repos v.s. repository creation dates. Better performance observed on older repositories (pre-2024 avg: 0.705) compared to newer ones (2024+ avg: 0.649), indicating potential data contamination effects.

We further summarize the results for the test set in Table 7.

Table 7: Average TypeSim scores and TypeCheck scores of the repositories in the test set for various models.

Model	TypeCheck ↓	TypeSim ↑	TypeSim wo Mising ↑	Missing rate ↓
llama-3-8b	44.0	0.396	0.747	0.470
llama-3.1-8b	111.5	0.634	0.815	0.225
qwen-2.5-7b	264.0	0.649	0.817	0.207
gpt-4o	312.3	0.787	0.883	0.108
gpt-4o-mini	234.5	0.779	0.882	0.117
claude-3.5-sonnet	205.8	0.790	0.876	0.098
deepseek-v2.5	227.8	0.748	0.893	0.161
deepseek-v3	231.8	0.793	0.889	0.107
grok-2	213.3	0.769	0.885	0.131

D.2 Results for Each Model

We present the detailed results in Table 8- 16 for each model on all repositories.

Table 8: TYPECHECK and TYPESIM scores for GPT-4O on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.774	0.883	0.123	0.766	0.814	0.694	0.000	
APPWORLD	N/A	0.835	0.897	0.068	0.876	0.747	0.729	0.742	1.000
AUTOGPT	N/A	0.755	0.801	0.057	0.746	0.786	0.728	0.794	0.809
BLACK	165	0.807	0.898	0.101	0.822	0.771	0.692	0.888	
CAPA	N/A	0.883	0.902	0.021	0.922	0.786	0.781	0.817	
COMPOSIO	903	0.646	0.665	0.029	0.602	0.722	0.819	0.908	
DB-GPT	N/A	0.803	0.874	0.081	0.808	0.792	0.822	0.790	0.781
DEEPFACE	N/A	0.898	0.963	0.068	0.886	0.940	0.875	0.990	1.000
EXO	N/A	0.746	0.928	0.196	0.729	0.810	0.666	0.875	0.750
FACESWAP	N/A	0.829	0.973	0.149	0.863	0.745	0.595	0.600	0.737
FASTAPI	208	0.810	0.876	0.075	0.893	0.746	0.782	0.917	0.930
FLAKE8	18	0.793	0.936	0.153	0.824	0.720	0.765	1.000	
FLASK	47	0.876	0.931	0.059	0.903	0.863	0.576	0.424	
GHUNT	N/A	0.755	0.860	0.122	0.734	0.732	0.975	0.997	
GPT-PILOT	N/A	0.813	0.885	0.082	0.828	0.812	0.692	0.851	
GPT4FREE	N/A	0.835	0.898	0.071	0.873	0.691	0.695	0.965	
GPTME	107	0.877	0.926	0.053	0.875	0.869	0.963		0.984
GRADIO	N/A	0.655	0.836	0.216	0.715	0.629	0.546	0.457	0.709
HAYSTACK	N/A	0.692	0.910	0.239	0.675	0.696	0.755	0.871	
LEROBOT	N/A	0.785	0.903	0.131	0.778	0.806	0.761	0.730	0.892
LLAMA-FACTORY	N/A	0.820	0.903	0.092	0.858	0.740	0.789	0.783	1.000
MANIM	N/A	0.835	0.943	0.114	0.887	0.691	0.562	0.573	0.000
MLC-LLM	N/A	0.739	0.919	0.196	0.692	0.874	0.858	0.886	0.742
NICEGUI	N/A	0.828	0.863	0.040	0.844	0.786	0.726	0.871	0.403
OPENAI-PYTHON	N/A	0.696	0.783	0.110	0.744	0.633	0.704	0.167	
OPENBB	N/A	0.840	0.857	0.020	0.817	0.853	0.791	0.775	0.963
PANDAS-AI	N/A	0.729	0.787	0.073	0.737	0.731	0.663	0.776	1.000
PAPER-QA	N/A	0.867	0.904	0.041	0.911	0.846	0.726	0.825	0.943
PDM	N/A	0.880	0.932	0.056	0.920	0.773	0.754	0.839	
PHIDATA	N/A	0.883	0.895	0.014	0.908	0.844	0.871	0.770	0.818
PILLOW	N/A	0.715	0.877	0.185	0.754	0.611	0.514	0.530	0.323
PIP	N/A	0.816	0.936	0.128	0.851	0.720	0.686	0.709	0.857
POETRY	69	0.864	0.957	0.097	0.909	0.742	0.663	0.431	0.958
PRE-COMMIT	37	0.900	0.910	0.011	0.910	0.872	0.951		
PRE-COMMIT-HOOKS	1	0.942	0.942	0.000	0.934	0.945	0.973		
PRIVATE-GPT	13	0.855	0.924	0.075	0.846	0.886	0.801		
PYDANTIC	N/A	0.815	0.887	0.081	0.875	0.700	0.647	0.701	0.000
RICH	59	0.785	0.957	0.179	0.794	0.756	0.688	0.969	
SCREENSHOT-TO-CODE	9	0.915	0.915	0.000	1.000	0.847	0.611	1.000	
SPHINX	N/A	0.643	0.836	0.231	0.631	0.708	0.718	0.697	0.588
SPOTIFY-DOWNLOADER	76	0.876	0.911	0.039	0.887	0.837	0.947	0.913	
STREAMLIT	N/A	0.804	0.882	0.089	0.807	0.708	0.941		
SUPERVISION	N/A	0.872	0.971	0.102	0.865	0.876	0.900	0.851	0.917
TAIPY	N/A	0.742	0.883	0.159	0.779	0.671	0.706	0.787	0.875
TEN-AGENT	N/A	0.698	0.833	0.163	0.683	0.776	0.720	0.802	
TYPER	83	0.701	0.937	0.252	0.761	0.625	0.617	0.000	
UNSTRACT	N/A	0.812	0.867	0.064	0.827	0.788	0.779	0.475	
URLLIB3	156	0.819	0.901	0.091	0.877	0.723	0.749	0.962	1.000
VLLM	N/A	0.759	0.885	0.143	0.764	0.758	0.704	0.681	0.972
VNPY	N/A	0.907	0.909	0.002	0.904	0.933	0.976	0.583	

Table 9: TYPECHECK and TYPESIM scores for GPT-4O-MINI on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.756	0.887	0.148	0.751	0.793	0.599	0.000	
APPWORLD	N/A	0.571	0.921	0.380	0.626	0.429	0.512	0.496	1.000
AUTOGPT	N/A	0.786	0.848	0.073	0.802	0.763	0.687	0.679	0.807
BLACK	221	0.731	0.881	0.171	0.759	0.635	0.615	0.913	
CAPA	N/A	0.762	0.806	0.055	0.793	0.732	0.644	0.817	
COMPOSIO	894	0.732	0.779	0.061	0.738	0.724	0.663	0.983	
DB-GPT	N/A	0.815	0.886	0.080	0.819	0.809	0.815	0.803	0.818
DEEPFACE	N/A	0.894	0.949	0.058	0.898	0.898	0.876	0.990	0.688
EXO	N/A	0.730	0.901	0.190	0.713	0.783	0.699	0.859	0.719
FACESWAP	N/A	0.835	0.978	0.146	0.867	0.755	0.632	0.583	0.571
FASTAPI	722	0.777	0.861	0.098	0.872	0.725	0.688	0.824	0.922
FLAKE8	28	0.776	0.923	0.159	0.837	0.656	0.501	1.000	
FLASK	14	0.784	0.922	0.150	0.833	0.704	0.565	0.236	
GHUNT	N/A	0.750	0.854	0.122	0.735	0.717	0.966	0.944	
GPT-PILOT	N/A	0.866	0.907	0.045	0.895	0.849	0.712	0.942	
GPT4FREE	N/A	0.796	0.886	0.102	0.836	0.643	0.655	0.975	
GPTME	29	0.888	0.917	0.031	0.897	0.860	0.888		0.984
GRADIO	N/A	0.629	0.834	0.246	0.684	0.612	0.510	0.397	0.580
HAYSTACK	N/A	0.707	0.915	0.227	0.694	0.699	0.799	0.660	
LEROBOT	N/A	0.752	0.896	0.160	0.767	0.777	0.623	0.577	0.892
LLAMA-FACTORY	N/A	0.802	0.865	0.074	0.834	0.751	0.709	0.738	0.994
MANIM	N/A	0.799	0.926	0.137	0.849	0.670	0.450	0.502	0.000
MLC-LLM	N/A	0.730	0.919	0.207	0.681	0.878	0.817	0.748	0.756
NICEGUI	N/A	0.831	0.898	0.075	0.859	0.741	0.683	0.862	0.482
OPENAI-PYTHON	N/A	0.726	0.807	0.100	0.704	0.760	0.691	0.500	
OPENBB	N/A	0.898	0.914	0.018	0.881	0.914	0.774	0.809	0.946
PANDAS-AI	N/A	0.731	0.804	0.091	0.732	0.760	0.623	0.786	1.000
PAPER-QA	N/A	0.811	0.881	0.080	0.850	0.769	0.741	0.863	0.948
PDM	N/A	0.859	0.909	0.055	0.917	0.711	0.640	0.642	
PHIDATA	N/A	0.899	0.916	0.018	0.950	0.829	0.822	0.729	0.768
PILLOW	N/A	0.720	0.918	0.216	0.772	0.561	0.527	0.492	0.461
PIP	N/A	0.768	0.928	0.173	0.801	0.673	0.624	0.717	0.835
POETRY	58	0.855	0.951	0.101	0.907	0.712	0.561	0.513	0.825
PRE-COMMIT	22	0.879	0.900	0.024	0.913	0.796	0.881		
PRE-COMMIT-HOOKS	12	0.905	0.905	0.000	0.920	0.930	0.808		
PRIVATE-GPT	76	0.819	0.905	0.095	0.829	0.824	0.690		
PYDANTIC	N/A	0.699	0.879	0.204	0.749	0.594	0.620	0.472	0.000
RICH	117	0.742	0.927	0.200	0.760	0.681	0.631	0.719	
SCREENSHOT-TO-CODE	3	0.905	0.905	0.000	0.969	0.855	0.616	1.000	
SPHINX	N/A	0.624	0.808	0.227	0.622	0.645	0.627	0.475	0.525
SPOTIFY-DOWNLOADER	60	0.837	0.902	0.073	0.820	0.860	0.919	0.938	
STREAMLIT	N/A	0.877	0.932	0.060	0.882	0.763	0.973		
SUPERVISION	N/A	0.796	0.966	0.175	0.848	0.727	0.726	0.854	0.301
TAIPY	N/A	0.760	0.878	0.134	0.809	0.667	0.712	0.784	0.813
TEN-AGENT	N/A	0.716	0.838	0.145	0.713	0.743	0.699	0.583	
TYPER	131	0.902	0.932	0.032	0.955	0.830	0.840	0.958	
UNSTRACT	N/A	0.807	0.864	0.066	0.817	0.784	0.811	0.913	
URLLIB3	86	0.820	0.909	0.097	0.876	0.737	0.727	0.655	1.000
VLLM	N/A	0.677	0.874	0.226	0.671	0.710	0.605	0.521	0.394
VNPY	N/A	0.934	0.934	0.000	0.931	0.933	0.973	1.000	

Table 10: TypeCheck and TypeSim scores for claude-3.5-sonnet on each repository.

Repo	Type Check	Type Sim	TypeSim WO Missing	Missing Ratio	TypeSim by Depth				
					Depth1	Depth2	Depth3	Depth4	Depth5
agents	N/A	0.646	0.852	0.242	0.643	0.673	0.514	0.000	
appworld	N/A	0.660	0.927	0.287	0.725	0.502	0.584	0.512	0.000
AutoGPT	N/A	0.769	0.856	0.102	0.749	0.821	0.793	0.752	0.604
black	16	0.767	0.925	0.171	0.785	0.741	0.531	0.888	
capa	N/A	0.887	0.915	0.031	0.898	0.836	0.879	0.821	
composio	805	0.578	0.684	0.155	0.569	0.581	0.634	0.960	
DB-GPT	N/A	0.797	0.870	0.084	0.809	0.777	0.809	0.802	0.806
deepface	N/A	0.930	0.955	0.027	0.934	0.941	0.886	0.990	1.000
exo	N/A	0.825	0.897	0.080	0.810	0.869	0.799	0.984	0.930
faceswap	N/A	0.824	0.965	0.146	0.848	0.770	0.651	0.650	0.571
fastapi	118	0.780	0.887	0.120	0.851	0.732	0.770	0.754	1.000
flake8	5	0.830	0.903	0.080	0.858	0.774	0.709	1.000	
flask	5	0.889	0.948	0.062	0.909	0.858	0.758	0.757	
GHunt	N/A	0.853	0.853	0.000	0.920	0.703	0.975	0.993	
gpt-pilot	N/A	0.848	0.886	0.043	0.856	0.848	0.765	0.973	
gpt4free	N/A	0.835	0.887	0.058	0.877	0.690	0.641	0.960	
gptme	17	0.906	0.915	0.010	0.907	0.891	0.969		0.984
gradio	N/A	0.722	0.846	0.147	0.774	0.701	0.629	0.518	0.700
haystack	N/A	0.726	0.889	0.183	0.708	0.705	0.876	0.788	
lerobot	N/A	0.822	0.916	0.103	0.819	0.833	0.788	0.867	0.892
LLaMA-Factory	N/A	0.746	0.874	0.147	0.771	0.697	0.697	0.797	1.000
manim	N/A	0.752	0.934	0.195	0.779	0.687	0.560	0.423	0.750
mlc-llm	N/A	0.699	0.901	0.224	0.688	0.759	0.576	0.569	0.625
nicegui	N/A	0.806	0.870	0.074	0.822	0.782	0.567	0.843	0.369
openai-python	N/A	0.519	0.804	0.355	0.515	0.518	0.603	0.799	
OpenBB	N/A	0.866	0.875	0.010	0.871	0.871	0.770	0.787	0.967
pandas-ai	N/A	0.759	0.773	0.018	0.800	0.708	0.648	0.804	0.969
paper-qa	N/A	0.830	0.897	0.075	0.860	0.804	0.758	0.907	0.943
pdm	N/A	0.763	0.870	0.122	0.789	0.692	0.701	0.777	
phidata	N/A	0.928	0.939	0.012	0.961	0.887	0.828	0.922	0.827
Pillow	N/A	0.754	0.916	0.178	0.776	0.682	0.680	0.597	0.915
pip	N/A	0.809	0.921	0.122	0.838	0.725	0.661	0.820	0.857
poetry	21	0.863	0.960	0.102	0.892	0.779	0.722	0.734	0.825
pre-commit	4	0.884	0.915	0.035	0.890	0.869	0.883		
pre-commit-hooks	1	0.914	0.914	0.000	0.884	0.964	0.981		
private-gpt	2	0.879	0.946	0.070	0.857	0.934	0.866		
pydantic	N/A	0.684	0.868	0.212	0.680	0.702	0.675	0.324	0.000
rich	10	0.760	0.952	0.202	0.760	0.767	0.669	0.719	
screenshot-to-code	0	0.937	0.937	0.000	0.970	0.924	0.611	1.000	
sphinx	N/A	0.626	0.872	0.282	0.611	0.704	0.725	0.652	0.582
spotify-downloader	16	0.845	0.896	0.057	0.841	0.836	0.957	0.938	
streamlit	N/A	0.564	0.910	0.380	0.530	0.796	0.993		
supervision	N/A	0.884	0.964	0.084	0.895	0.861	0.896	0.926	0.660
taipy	N/A	0.652	0.887	0.265	0.694	0.574	0.589	0.767	0.563
TEN-Agent	N/A	0.799	0.841	0.049	0.801	0.810	0.728	0.531	
typer	32	0.674	0.862	0.218	0.713	0.616	0.669	0.000	
unstract	N/A	0.803	0.833	0.036	0.814	0.788	0.763	0.525	
urllib3	30	0.817	0.927	0.119	0.857	0.734	0.890	0.954	1.000
vllm	N/A	0.743	0.891	0.166	0.733	0.774	0.733	0.637	0.797
vnpy	N/A	0.928	0.928	0.000	0.925	0.930	0.970	1.000	

Table 11: TYPECHECK and TYPESIM scores for DEEPSEEK-V3 on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.728	0.885	0.178	0.719	0.778	0.534	0.000	
APPWORLD	N/A	0.721	0.910	0.208	0.798	0.503	0.755	0.572	0.500
AUTOGPT	N/A	0.788	0.825	0.046	0.781	0.800	0.827	0.764	0.656
BLACK	159	0.899	0.929	0.032	0.914	0.891	0.640	0.975	
CAPA	N/A	0.817	0.869	0.059	0.875	0.740	0.614	0.773	
COMPOSIO	893	0.717	0.766	0.064	0.718	0.721	0.664	0.903	
DB-GPT	N/A	0.816	0.895	0.088	0.828	0.797	0.831	0.761	0.839
DEEPFACE	N/A	0.829	0.951	0.128	0.807	0.907	0.818	0.656	0.500
EXO	N/A	0.739	0.898	0.178	0.720	0.802	0.699	0.859	0.734
FACESWAP	N/A	0.857	0.982	0.127	0.881	0.797	0.694	0.650	0.737
FASTAPI	527	0.433	0.772	0.439	0.424	0.495	0.339	0.073	0.680
FLAKE8	8	0.824	0.931	0.115	0.856	0.751	0.763	1.000	
FLASK	5	0.856	0.927	0.077	0.882	0.829	0.639	0.528	
GHUNT	N/A	0.743	0.846	0.122	0.742	0.678	0.981	1.000	
GPT-PILOT	N/A	0.865	0.908	0.048	0.879	0.855	0.787	0.963	
GPT4FREE	N/A	0.810	0.881	0.080	0.851	0.664	0.650	0.913	
GPTME	32	0.895	0.926	0.033	0.884	0.930	0.888		0.984
GRADIO	N/A	0.681	0.842	0.190	0.723	0.673	0.567	0.503	0.691
HAYSTACK	N/A	0.738	0.935	0.210	0.727	0.726	0.827	0.881	
LEROBOT	N/A	0.796	0.916	0.131	0.785	0.821	0.795	0.676	0.892
LLAMA-FACTORY	N/A	0.844	0.907	0.069	0.862	0.813	0.815	0.712	1.000
MANIM	N/A	0.767	0.944	0.187	0.815	0.636	0.491	0.563	0.000
MLC-LLM	N/A	0.749	0.930	0.195	0.700	0.896	0.811	0.840	0.693
NICEGUI	N/A	0.837	0.907	0.078	0.851	0.801	0.710	0.884	0.643
OPENAI-PYTHON	N/A	0.665	0.822	0.190	0.627	0.716	0.668	0.799	
OPENBB	N/A	0.883	0.929	0.050	0.912	0.876	0.789	0.913	0.973
PANDAS-AI	N/A	0.718	0.792	0.093	0.721	0.722	0.652	0.846	0.969
PAPER-QA	N/A	0.867	0.915	0.052	0.913	0.817	0.799	0.896	0.958
PDM	N/A	0.858	0.920	0.067	0.906	0.731	0.702	0.642	
PHIDATA	N/A	0.931	0.948	0.018	0.976	0.866	0.860	0.844	0.853
PILLOW	N/A	0.828	0.924	0.104	0.865	0.709	0.721	0.672	0.756
PIP	N/A	0.803	0.936	0.142	0.836	0.736	0.686	0.254	0.839
POETRY	13	0.876	0.966	0.094	0.916	0.770	0.670	0.458	0.825
PRE-COMMIT	37	0.897	0.908	0.012	0.911	0.855	0.954		
PRE-COMMIT-HOOKS	0	0.912	0.923	0.012	0.898	0.910	0.977		
PRIVATE-GPT	13	0.825	0.910	0.093	0.826	0.830	0.801		
PYDANTIC	N/A	0.701	0.861	0.187	0.711	0.667	0.739	0.757	0.000
RICH	78	0.754	0.939	0.197	0.754	0.762	0.662	0.719	
SCREENSHOT-TO-CODE	2	0.927	0.927	0.000	1.000	0.872	0.611	1.000	
SPHINX	N/A	0.621	0.780	0.203	0.611	0.668	0.724	0.761	0.527
SPOTIFY-DOWNLOADER	37	0.865	0.921	0.060	0.854	0.882	0.916	0.950	
STREAMLIT	N/A	0.882	0.934	0.056	0.886	0.769	0.993		
SUPERVISION	N/A	0.826	0.967	0.146	0.866	0.762	0.822	0.807	0.301
TAIPY	N/A	0.775	0.879	0.119	0.799	0.722	0.784	0.815	0.625
TEN-AGENT	N/A	0.631	0.832	0.242	0.628	0.651	0.670	0.406	
TYPER	101	0.675	0.815	0.171	0.737	0.579	0.675	0.000	
UNSTRACT	N/A	0.803	0.861	0.067	0.810	0.782	0.843	0.875	
URLLIB3	26	0.846	0.922	0.082	0.878	0.805	0.767	0.641	1.000
VLLM	N/A	0.719	0.908	0.209	0.698	0.767	0.735	0.702	0.947
VNPY	N/A	0.897	0.911	0.015	0.896	0.918	0.970	0.375	

Table 12: TYPECHECK and TYPESIM scores for DEEPSEEK-V2.5 on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.715	0.894	0.200	0.720	0.720	0.550	0.000	
APPWORLD	N/A	0.542	0.918	0.409	0.608	0.385	0.438	0.440	0.000
AUTOGPT	N/A	0.764	0.796	0.040	0.763	0.774	0.735	0.791	0.754
BLACK	54	0.749	0.937	0.201	0.788	0.612	0.614	0.879	
CAPA	N/A	0.823	0.886	0.071	0.841	0.750	0.798	0.894	
COMPOSIO	830	0.671	0.757	0.114	0.683	0.659	0.569	0.813	
DB-GPT	N/A	0.776	0.899	0.137	0.778	0.774	0.768	0.730	0.837
DEEPFACE	N/A	0.932	0.965	0.034	0.925	0.972	0.906	0.741	1.000
EXO	N/A	0.811	0.935	0.133	0.816	0.814	0.649	0.833	0.710
FACESWAP	N/A	0.802	0.969	0.172	0.838	0.716	0.561	0.467	0.737
FASTAPI	292	0.533	0.961	0.445	0.498	0.650	0.336	0.073	0.680
FLAKE8	28	0.832	0.950	0.125	0.870	0.748	0.749	1.000	
FLASK	24	0.799	0.940	0.150	0.843	0.735	0.545	0.257	
GHUNT	N/A	0.766	0.880	0.129	0.742	0.751	0.974	1.000	
GPT-PILOT	N/A	0.871	0.911	0.045	0.894	0.856	0.754	0.947	
GPT4FREE	N/A	0.759	0.898	0.155	0.787	0.654	0.649	0.903	
GPTME	68	0.898	0.924	0.027	0.898	0.900	0.883		0.984
GRADIO	N/A	0.574	0.855	0.328	0.603	0.574	0.448	0.481	0.772
HAYSTACK	N/A	0.708	0.949	0.254	0.700	0.708	0.736	0.875	
LEROBOT	N/A	0.682	0.925	0.262	0.646	0.729	0.757	0.493	0.892
LLAMA-FACTORY	N/A	0.803	0.927	0.133	0.828	0.748	0.811	0.644	0.987
MANIM	N/A	0.635	0.870	0.270	0.687	0.481	0.398	0.493	0.000
MLC-LLM	N/A	0.575	0.951	0.396	0.537	0.696	0.574	0.698	0.601
NICEGUI	N/A	0.801	0.901	0.111	0.797	0.822	0.767	0.851	0.623
OPENAI-PYTHON	N/A	0.571	0.819	0.302	0.568	0.573	0.606	0.625	
OPENBB	N/A	0.790	0.927	0.148	0.836	0.778	0.707	0.671	0.758
PANDAS-AI	N/A	0.746	0.817	0.087	0.737	0.789	0.658	0.803	1.000
PAPER-QA	N/A	0.641	0.885	0.275	0.653	0.631	0.634	0.506	0.943
PDM	N/A	0.765	0.865	0.116	0.814	0.638	0.592	0.637	
PHIDATA	N/A	0.883	0.939	0.059	0.914	0.848	0.810	0.823	0.443
PILLOW	N/A	0.630	0.857	0.265	0.666	0.530	0.471	0.387	0.484
PIP	N/A	0.788	0.925	0.148	0.820	0.724	0.664	0.253	0.741
POETRY	32	0.876	0.970	0.097	0.914	0.773	0.682	0.463	0.958
PRE-COMMIT	42	0.943	0.953	0.011	0.965	0.887	0.960		
PRE-COMMIT-HOOKS	7	0.952	0.960	0.008	0.983	0.907	0.871		
PRIVATE-GPT	41	0.751	0.923	0.186	0.764	0.736	0.686		
PYDANTIC	N/A	0.667	0.912	0.268	0.714	0.562	0.631	0.431	0.000
RICH	328	0.618	0.934	0.339	0.631	0.563	0.659	0.719	
SCREENSHOT-TO-CODE	6	0.887	0.887	0.000	0.980	0.813	0.611	0.833	
SPHINX	N/A	0.498	0.840	0.407	0.476	0.625	0.610	0.513	0.590
SPOTIFY-DOWNLOADER	30	0.861	0.910	0.053	0.856	0.858	0.973	0.900	
STREAMLIT	N/A	0.910	0.980	0.071	0.916	0.782	1.000		
SUPERVISION	N/A	0.803	0.969	0.171	0.847	0.747	0.756	0.762	0.315
TAIPY	N/A	0.790	0.897	0.120	0.810	0.750	0.775	0.783	0.875
TEN-AGENT	N/A	0.590	0.869	0.322	0.584	0.621	0.631	0.469	
TYPER	102	0.676	0.903	0.252	0.736	0.581	0.678	0.000	
UNSTRACT	N/A	0.812	0.872	0.068	0.823	0.793	0.788	0.738	
URLLIB3	58	0.848	0.933	0.091	0.896	0.776	0.790	0.641	1.000
VLLM	N/A	0.661	0.874	0.244	0.660	0.669	0.646	0.573	0.576
VNPY	N/A	0.898	0.914	0.018	0.895	0.940	0.976	0.375	

Table 13: TYPECHECK and TYPESIM scores for GROK-2 on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.700	0.900	0.222	0.695	0.742	0.493	0.000	
APPWORLD	N/A	0.535	0.931	0.426	0.596	0.395	0.413	0.491	0.000
AUTOGPT	N/A	0.755	0.803	0.060	0.756	0.752	0.762	0.773	0.750
BLACK	26	0.825	0.925	0.107	0.847	0.807	0.489	0.958	
CAPA	N/A	0.905	0.924	0.021	0.936	0.794	0.851	0.852	
COMPOSIO	828	0.482	0.593	0.188	0.454	0.563	0.445	0.897	
DB-GPT	N/A	0.820	0.898	0.086	0.834	0.805	0.789	0.764	0.841
DEEPFACE	N/A	0.868	0.961	0.096	0.862	0.915	0.808	0.990	1.000
EXO	N/A	0.778	0.928	0.161	0.763	0.838	0.696	0.875	0.750
FACESWAP	N/A	0.885	0.973	0.091	0.904	0.833	0.784	0.817	0.737
FASTAPI	559	0.659	0.940	0.299	0.751	0.646	0.468	0.577	0.979
FLAKE8	5	0.917	0.938	0.023	0.930	0.884	0.903	1.000	
FLASK	8	0.900	0.935	0.037	0.933	0.847	0.692	0.861	
GHUNT	N/A	0.751	0.894	0.160	0.732	0.735	0.921	1.000	
GPT-PILOT	N/A	0.839	0.891	0.059	0.843	0.841	0.777	0.948	
GPT4FREE	N/A	0.839	0.883	0.050	0.879	0.690	0.705	0.917	
GPTME	22	0.908	0.937	0.031	0.907	0.912	0.894		0.984
GRADIO	N/A	0.695	0.861	0.192	0.739	0.677	0.612	0.569	0.630
HAYSTACK	N/A	0.747	0.933	0.199	0.712	0.773	0.837	0.867	
LEROBOT	N/A	0.851	0.922	0.077	0.866	0.860	0.764	0.704	0.892
LLAMA-FACTORY	N/A	0.837	0.889	0.059	0.864	0.785	0.797	0.775	0.987
MANIM	N/A	0.650	0.872	0.255	0.667	0.619	0.461	0.641	0.000
MLC-LLM	N/A	0.734	0.946	0.224	0.703	0.819	0.837	0.790	0.905
NICEGUI	N/A	0.772	0.871	0.114	0.765	0.815	0.688	0.853	0.576
OPENAI-PYTHON	N/A	0.693	0.814	0.149	0.690	0.691	0.776	0.713	
OPENBB	N/A	0.864	0.928	0.070	0.854	0.875	0.764	0.738	0.967
PANDAS-AI	N/A	0.741	0.817	0.093	0.745	0.759	0.642	0.833	1.000
PAPER-QA	N/A	0.857	0.903	0.051	0.904	0.808	0.791	0.828	0.838
PDM	N/A	0.870	0.920	0.055	0.919	0.741	0.701	0.688	
PHIDATA	N/A	0.899	0.943	0.047	0.945	0.844	0.798	0.682	0.443
PILLOW	N/A	0.818	0.923	0.114	0.863	0.688	0.629	0.533	0.864
PIP	N/A	0.814	0.936	0.130	0.842	0.765	0.704	0.179	0.848
POETRY	27	0.867	0.961	0.098	0.909	0.752	0.652	0.519	0.825
PRE-COMMIT	2	0.886	0.946	0.063	0.895	0.858	0.966		
PRE-COMMIT-HOOKS	2	0.905	0.905	0.000	0.888	0.924	0.952		
PRIVATE-GPT	2	0.645	0.939	0.313	0.609	0.712	0.724		
PYDANTIC	N/A	0.816	0.920	0.113	0.811	0.827	0.837	0.551	0.500
RICH	16	0.628	0.856	0.266	0.628	0.630	0.601	0.719	
SCREENSHOT-TO-CODE	1	0.936	0.936	0.000	0.990	0.904	0.616	0.833	
SPHINX	N/A	0.638	0.803	0.205	0.633	0.679	0.639	0.531	0.594
SPOTIFY-DOWNLOADER	21	0.868	0.920	0.057	0.864	0.867	0.969	0.788	
STREAMLIT	N/A	0.954	0.979	0.025	0.967	0.769	0.993		
SUPERVISION	N/A	0.852	0.967	0.119	0.904	0.795	0.779	0.708	0.301
TAIPY	N/A	0.737	0.875	0.158	0.763	0.687	0.712	0.797	0.875
TEN-AGENT	N/A	0.468	0.912	0.487	0.475	0.425	0.561	0.323	
TYPER	179	0.690	0.923	0.252	0.744	0.608	0.689	0.000	
UNSTRACT	N/A	0.809	0.877	0.078	0.818	0.791	0.807	0.738	
URLLIB3	30	0.854	0.910	0.062	0.902	0.774	0.862	0.608	1.000
VLLM	N/A	0.732	0.869	0.158	0.717	0.754	0.809	0.731	0.972
VNPY	N/A	0.860	0.913	0.057	0.871	0.600	0.976	1.000	

Table 14: TypeCheck and TypeSim scores for llama-3-8b on each repository.

Repo	Type Check	Type Sim	TypeSim WO Missing	Missing Ratio	TypeSim by Depth				
					Depth1	Depth2	Depth3	Depth4	Depth5
agents	N/A	0.415	0.809	0.487	0.426	0.408	0.154	0.000	
appworld	N/A	0.177	0.901	0.804	0.194	0.131	0.176	0.168	0.000
AutoGPT	N/A	0.451	0.749	0.398	0.500	0.370	0.180	0.227	0.000
black	18	0.186	0.805	0.769	0.201	0.133	0.142	0.388	
capa	N/A	0.235	0.606	0.612	0.246	0.147	0.258	0.219	
composio	116	0.382	0.573	0.333	0.458	0.218	0.198	0.068	
DB-GPT	N/A	0.317	0.604	0.476	0.381	0.228	0.266	0.289	0.216
deepface	N/A	0.231	0.630	0.634	0.298	0.141	0.107	0.000	0.000
exo	N/A	0.377	0.782	0.519	0.390	0.343	0.341	0.208	0.458
faceswap	N/A	0.326	0.867	0.624	0.374	0.180	0.092	0.250	0.323
fastapi	1769	0.054	0.754	0.928	0.089	0.032	0.062	0.000	0.000
flake8	28	0.573	0.815	0.297	0.649	0.441	0.181	0.250	
flask	12	0.346	0.764	0.547	0.392	0.242	0.259	0.257	
GHunt	N/A	0.481	0.849	0.434	0.543	0.338	0.690	0.005	
gpt-pilot	N/A	0.574	0.768	0.253	0.620	0.523	0.438	0.828	
gpt4free	N/A	0.543	0.809	0.329	0.579	0.418	0.343	0.892	
gptme	36	0.567	0.827	0.315	0.613	0.491	0.196		0.000
gradio	N/A	0.227	0.576	0.605	0.326	0.169	0.084	0.018	0.150
haystack	N/A	0.308	0.558	0.448	0.463	0.114	0.135	0.188	
lerobot	N/A	0.216	0.754	0.714	0.309	0.096	0.147	0.079	0.000
LLaMA-Factory	N/A	0.366	0.730	0.498	0.401	0.325	0.225	0.106	0.828
manim	N/A	0.299	0.697	0.570	0.335	0.198	0.103	0.445	0.000
mlc-llm	N/A	0.243	0.738	0.670	0.260	0.200	0.180	0.243	0.000
nicegui	N/A	0.514	0.811	0.367	0.527	0.506	0.269	0.424	0.100
openai-python	N/A	0.317	0.668	0.525	0.358	0.270	0.213	0.354	
OpenBB	N/A	0.298	0.589	0.495	0.473	0.237	0.129	0.194	0.350
pandas-ai	N/A	0.471	0.585	0.196	0.558	0.396	0.205	0.250	0.000
paper-qa	N/A	0.286	0.715	0.601	0.352	0.201	0.255	0.197	0.000
pdm	N/A	0.413	0.781	0.472	0.477	0.246	0.187	0.116	
phidata	N/A	0.443	0.629	0.296	0.593	0.236	0.173	0.117	0.000
Pillow	N/A	0.382	0.885	0.569	0.427	0.254	0.181	0.142	0.053
pip	N/A	0.440	0.861	0.489	0.478	0.356	0.245	0.058	0.134
poetry	82	0.600	0.873	0.312	0.662	0.433	0.224	0.344	0.492
pre-commit	65	0.584	0.851	0.314	0.634	0.447	0.761		
pre-commit-hooks	19	0.649	0.887	0.269	0.657	0.737	0.500		
private-gpt	11	0.410	0.814	0.497	0.386	0.440	0.513		
pydantic	N/A	0.113	0.652	0.826	0.140	0.051	0.091	0.111	0.000
rich	42	0.170	0.664	0.744	0.191	0.091	0.173	0.000	
screenshot-to-code	5	0.532	0.952	0.441	0.633	0.456	0.306	0.000	
sphinx	N/A	0.097	0.620	0.844	0.097	0.096	0.083	0.103	0.207
spotify-downloader	15	0.395	0.631	0.374	0.459	0.255	0.294	0.654	
streamlit	N/A	0.079	0.232	0.658	0.077	0.111	0.059		
supervision	N/A	0.203	0.644	0.685	0.260	0.114	0.189	0.142	0.000
taipy	N/A	0.357	0.589	0.394	0.501	0.129	0.088	0.029	0.188
TEN-Agent	N/A	0.393	0.827	0.525	0.412	0.307	0.287	0.139	
typer	74	0.300	0.917	0.673	0.314	0.265	0.369	0.000	
unstract	N/A	0.573	0.759	0.245	0.638	0.435	0.518	0.763	
urllib3	88	0.462	0.837	0.448	0.537	0.362	0.281	0.079	0.000
vllm	N/A	0.185	0.531	0.651	0.214	0.132	0.112	0.072	0.169
vnpy	N/A	0.590	0.772	0.235	0.652	0.117	0.079	0.000	

Table 15: TYPECHECK and TYPESIM scores for LLAMA-3.1-8B on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.636	0.809	0.214	0.634	0.669	0.410	0.000	
APPWORLD	N/A	0.347	0.789	0.560	0.381	0.258	0.326	0.392	0.000
AUTOGPT	N/A	0.676	0.783	0.137	0.698	0.629	0.604	0.679	0.500
BLACK	193	0.518	0.782	0.338	0.554	0.398	0.345	0.763	
CAPA	N/A	0.574	0.816	0.297	0.588	0.575	0.498	0.906	
COMPOSIO	310	0.580	0.693	0.163	0.626	0.495	0.377	0.847	
DB-GPT	N/A	0.642	0.769	0.165	0.685	0.586	0.595	0.571	0.605
DEEPFACE	N/A	0.716	0.923	0.224	0.753	0.680	0.612	0.616	1.000
EXO	N/A	0.734	0.938	0.218	0.708	0.812	0.732	0.943	0.714
FACESWAP	N/A	0.710	0.908	0.219	0.745	0.629	0.427	0.533	0.722
FASTAPI	2232	0.275	0.573	0.519	0.231	0.359	0.175	0.008	0.000
FLAKE8	42	0.757	0.882	0.142	0.813	0.653	0.488	0.750	
FLASK	67	0.639	0.772	0.173	0.687	0.556	0.363	0.444	
GHUNT	N/A	0.755	0.863	0.126	0.749	0.717	0.964	0.672	
GPT-PILOT	N/A	0.799	0.854	0.065	0.814	0.789	0.706	0.951	
GPT4FREE	N/A	0.747	0.885	0.156	0.782	0.621	0.588	0.957	
GPTME	109	0.586	0.850	0.311	0.607	0.531	0.504		0.984
GRADIO	N/A	0.456	0.593	0.231	0.552	0.404	0.299	0.249	0.296
HAYSTACK	N/A	0.243	0.721	0.663	0.274	0.212	0.183	0.288	
LEROBOT	N/A	0.537	0.830	0.352	0.694	0.337	0.446	0.150	0.250
LLAMA-FACTORY	N/A	0.628	0.794	0.209	0.650	0.597	0.574	0.389	0.961
MANIM	N/A	0.681	0.833	0.183	0.738	0.524	0.360	0.440	0.000
MLC-LLM	N/A	0.569	0.856	0.336	0.543	0.661	0.553	0.448	0.423
NICEGUI	N/A	0.579	0.824	0.297	0.573	0.612	0.557	0.652	0.507
OPENAI-PYTHON	N/A	0.539	0.702	0.233	0.572	0.504	0.405	0.608	
OPENBB	N/A	0.599	0.839	0.286	0.607	0.607	0.479	0.339	0.200
PANDAS-AI	N/A	0.670	0.779	0.141	0.673	0.723	0.507	0.601	0.875
PAPER-QA	N/A	0.595	0.742	0.198	0.675	0.486	0.592	0.376	0.948
PDM	N/A	0.601	0.800	0.249	0.671	0.427	0.306	0.160	
PHIDATA	N/A	0.751	0.868	0.135	0.800	0.693	0.656	0.542	0.136
PILLOW	N/A	0.495	0.784	0.369	0.550	0.329	0.275	0.294	0.296
PIP	N/A	0.594	0.831	0.285	0.631	0.505	0.475	0.201	0.272
POETRY	197	0.754	0.922	0.182	0.799	0.633	0.499	0.371	0.958
PRE-COMMIT	58	0.813	0.878	0.074	0.895	0.640	0.535		
PRE-COMMIT-HOOKS	19	0.827	0.885	0.066	0.884	0.800	0.605		
PRIVATE-GPT	49	0.750	0.809	0.073	0.852	0.559	0.542		
PYDANTIC	N/A	0.290	0.565	0.486	0.358	0.142	0.198	0.111	0.000
RICH	299	0.458	0.705	0.350	0.464	0.433	0.521	0.250	
SCREENSHOT-TO-CODE	8	0.912	0.930	0.020	0.960	0.883	0.616	0.833	
SPHINX	N/A	0.022	0.537	0.959	0.021	0.030	0.016	0.029	0.000
SPOTIFY-DOWNLOADER	172	0.722	0.891	0.189	0.714	0.745	0.712	0.600	
STREAMLIT	N/A	0.593	0.885	0.330	0.569	0.722	0.980		
SUPERVISION	N/A	0.508	0.875	0.420	0.465	0.562	0.559	0.653	0.000
TAIPY	N/A	0.571	0.769	0.256	0.623	0.495	0.405	0.663	0.438
TEN-AGENT	N/A	0.655	0.778	0.158	0.652	0.690	0.551	0.486	
TYPER	438	0.508	0.889	0.429	0.555	0.446	0.460	0.000	
UNSTRACT	N/A	0.709	0.821	0.137	0.743	0.647	0.633	0.650	
URLLIB3	155	0.702	0.812	0.136	0.761	0.621	0.535	0.515	1.000
VLLM	N/A	0.326	0.634	0.485	0.337	0.313	0.273	0.186	0.175
VNPY	N/A	0.827	0.924	0.105	0.828	0.788	0.939	0.375	

Table 16: TYPECHECK and TYPESIM scores for QWEN-2.5-7B on each repository.

REPO	TYPE CHECK	TYPE SIM	TYPESIM WO MISSING	MISSING RATIO	TYPESIM BY DEPTH				
					DEPTH1	DEPTH2	DEPTH3	DEPTH4	DEPTH5
AGENTS	N/A	0.695	0.865	0.196	0.706	0.683	0.507	0.000	
APPWORLD	N/A	0.462	0.921	0.498	0.502	0.380	0.350	0.368	0.000
AUTOGPT	N/A	0.680	0.784	0.132	0.703	0.633	0.596	0.626	0.750
BLACK	112	0.491	0.651	0.247	0.543	0.292	0.411	0.325	
CAPA	N/A	0.576	0.757	0.240	0.589	0.402	0.656	0.793	
COMPOSIO	916	0.546	0.655	0.167	0.639	0.358	0.211	0.636	
DB-GPT	N/A	0.596	0.693	0.141	0.702	0.460	0.440	0.508	0.481
DEEPFACE	N/A	0.492	0.840	0.415	0.528	0.582	0.214	0.323	0.000
EXO	N/A	0.682	0.881	0.226	0.669	0.713	0.747	0.708	0.714
FACESWAP	N/A	0.643	0.844	0.238	0.748	0.329	0.151	0.229	0.380
FASTAPI	604	0.191	0.484	0.605	0.315	0.118	0.155	0.065	0.690
FLAKE8	42	0.662	0.887	0.253	0.715	0.546	0.571	0.750	
FLASK	73	0.715	0.836	0.145	0.797	0.560	0.400	0.167	
GHUNT	N/A	0.696	0.851	0.183	0.701	0.613	0.959	0.998	
GPT-PILOT	N/A	0.776	0.872	0.110	0.778	0.805	0.623	0.739	
GPT4FREE	N/A	0.795	0.900	0.116	0.821	0.702	0.701	0.500	
GPTME	110	0.652	0.847	0.231	0.680	0.583	0.529		1.000
GRADIO	N/A	0.459	0.613	0.251	0.580	0.394	0.248	0.260	0.177
HAYSTACK	N/A	0.326	0.528	0.383	0.493	0.136	0.078	0.296	
LEROBOT	N/A	0.438	0.752	0.417	0.581	0.267	0.283	0.289	0.000
LLAMA-FACTORY	N/A	0.608	0.789	0.230	0.644	0.555	0.537	0.226	0.000
MANIM	N/A	0.732	0.877	0.166	0.806	0.522	0.319	0.778	0.000
MLC-LLM	N/A	0.527	0.777	0.322	0.585	0.388	0.232	0.358	0.137
NICEGUI	N/A	0.790	0.882	0.105	0.811	0.733	0.644	0.712	0.325
OPENAI-PYTHON	N/A	0.544	0.654	0.168	0.617	0.462	0.319	0.187	
OPENBB	N/A	0.409	0.453	0.097	0.776	0.260	0.352	0.359	0.346
PANDAS-AI	N/A	0.721	0.793	0.090	0.720	0.791	0.518	0.719	1.000
PAPER-QA	N/A	0.583	0.776	0.249	0.588	0.597	0.490	0.704	0.833
PDM	N/A	0.723	0.891	0.188	0.776	0.580	0.566	0.765	
PHIDATA	N/A	0.804	0.894	0.100	0.856	0.741	0.671	0.850	0.361
PILLOW	N/A	0.574	0.841	0.318	0.634	0.402	0.300	0.390	0.417
PIP	N/A	0.717	0.902	0.206	0.761	0.615	0.550	0.166	0.804
POETRY	126	0.854	0.932	0.084	0.903	0.708	0.619	0.676	0.958
PRE-COMMIT	105	0.854	0.916	0.068	0.869	0.828	0.752		
PRE-COMMIT-HOOKS	11	0.921	0.958	0.039	0.927	0.890	0.934		
PRIVATE-GPT	38	0.485	0.873	0.444	0.459	0.534	0.546		
PYDANTIC	N/A	0.383	0.789	0.514	0.445	0.269	0.201	0.111	0.000
RICH	311	0.579	0.834	0.305	0.587	0.545	0.667	0.750	
SCREENSHOT-TO-CODE	19	0.756	0.918	0.177	0.837	0.683	0.616	0.833	
SPHINX	N/A	0.373	0.669	0.443	0.377	0.373	0.255	0.274	0.393
SPOTIFY-DOWNLOADER	25	0.429	0.637	0.327	0.497	0.286	0.386	0.167	
STREAMLIT	N/A	0.287	0.426	0.326	0.246	0.528	0.882		
SUPERVISION	N/A	0.554	0.827	0.331	0.645	0.392	0.562	0.612	0.315
TAIPY	N/A	0.487	0.704	0.308	0.600	0.328	0.225	0.046	0.000
TEN-AGENT	N/A	0.686	0.808	0.152	0.683	0.726	0.532	0.472	
TYPER	314	0.472	0.891	0.470	0.493	0.454	0.417	0.000	
UNSTRACT	N/A	0.796	0.859	0.073	0.812	0.767	0.779	0.650	
URLLIB3	139	0.662	0.794	0.167	0.755	0.524	0.522	0.231	1.000
VLLM	N/A	0.513	0.673	0.238	0.569	0.400	0.389	0.299	0.750
VNPY	N/A	0.816	0.841	0.030	0.857	0.808	0.140	0.000	

D.3 Type Inference Examples

We a code snippet from the repo DB-GPT to demonstrate the type inference accuracy of each LLM. We have conducted necessary code formatting for better visualization.

Source Code

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: str, candidates: List[str]) -> List[float]:
        ...

    async def apredict(self, query: str, candidates: List[str]) -> List[float]:
        ...


class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

GPT-4o (Score: 0.874)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...

    async def apredict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...


class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

GPT-4O-MINI (SCORE: 0.886)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...

    async def apredict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...

class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

CLAUDE-3.5-SONNET (SCORE: 0.870)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...

    async def apredict(self, query: Any, candidates: List[Any]) -> List[float]:
        ...

class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

DEEPSEEK-V3 (SCORE: 0.898)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: str, candidates: List[str]) -> List[float]:
        ...

    async def apredict(self, query: str, candidates: List[str]) -> Awaitable[List[float]]:
        ...


class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> Awaitable[List[List[float]]]:
        ...

    async def aembed_query(self, text: str) -> Awaitable[List[float]]:
        ...
```

DEEPSEEK-V2.5 (SCORE: 0.899)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: str, candidates: List[str]) -> List[float]:
        ...

    async def apredict(self, query: str, candidates: List[str]) -> List[float]:
        ...


class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

GROK-2 (SCORE: 0.898)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: str, candidates: List[str]) -> List[float]:
        ...

    def apredict(self, query: str, candidates: List[str]) -> asyncio.Future[List[float]]:
        ...

class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    def aembed_documents(self, texts: List[str]) -> asyncio.Future[List[List[float]]]:
        ...

    def aembed_query(self, text: str) -> asyncio.Future[List[float]]:
        ...
```

LLAMA-3-8B (SCORE: 0.604)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: object, candidates: List[object]) -> object:
        ...

    async def apredict(self, query: object, candidates: List[object]) -> Awaitable[object]:
        ...

class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[object]) -> List[object]:
        ...

    @abstractmethod
    def embed_query(self, text: object) -> object:
        ...

    async def aembed_documents(self, texts: List[object]) -> Awaitable[List[object]]:
        ...

    async def aembed_query(self, text: object) -> Awaitable[object]:
        ...
```

LLAMA-3.1-8B (SCORE: 0.769)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: Any, candidates: List[Any]) -> Any:
        ...

    async def apredict(self, query: Any, candidates: List[Any]) -> Awaitable[Any]:
        ...

class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[Any]) -> List[Any]:
        ...

    @abstractmethod
    def embed_query(self, text: Any) -> Any:
        ...

    async def aembed_documents(self, texts: List[Any]) -> Awaitable[List[Any]]:
        ...

    async def aembed_query(self, text: Any) -> Awaitable[Any]:
        ...
```

QWEN-2.5-7B (SCORE: 0.693)

```
class RerankEmbeddings(ABC):
    @abstractmethod
    def predict(self, query: str, candidates: List[str]) -> List[float]:
        ...

    async def apredict(self, query: str, candidates: List[str]) -> List[float]:
        ...


class Embeddings(ABC):
    @abstractmethod
    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    @abstractmethod
    def embed_query(self, text: str) -> List[float]:
        ...

    async def aembed_documents(self, texts: List[str]) -> List[List[float]]:
        ...

    async def aembed_query(self, text: str) -> List[float]:
        ...
```

E IMPLEMENTATION DETAILS

E.1 COMPUTING TYPESIM

Algorithm 2 outlines the implementation of argument comparison in the case of ordered lists.

Algorithm 3 outlines the implementation of argument comparison in the case of unordered sets (i.e., comparison between two union types).

E.2 TYPE REMOVAL

Algorithm 4 outlines the implementation of how types are removed from the datasets collected.

Algorithm 2 ListCompare

Input: type lists L, L'
// List-wise comparison for generic type arguments
$S = 0$
for $i = 1$ **to** $\min(|L|, |L'|)$ **do**
    $S = S +$ **GetTypeSimilarity**(L_i, L'_i)
end for
Return: $\frac{S}{\max(|L|,|L'|)}$

Algorithm 3 SetCompare

Input: type sets A, B
// Set-wise comparison for Union types
for $i = 1$ **to** $|A|$ **do**
    for $j = 1$ **to** $|B|$ **do**
        $c_{ij} =$ **GetTypeSimilarity**(A_i, B_j)
    end for
end for
Find optimal matching M using cost matrix $C = [c_{ij}]$
$S = \sum_{(i,j)\in M} c_{ij}$
Return: $\frac{S}{\max(|A|,|B|)}$

Algorithm 4 Type Removal Process

Input: Repository R with typed Python files
Output: Repository R' with types removed
for each Python file $f \in R$ **do**
    Parse AST of f to locate type annotations
    for each function/variable declaration d **do**
        if d has type annotation t **then**
            Remove t from d in f {e.g., def foo(x: int) $\rightarrow$ def foo(x)}
        end if
        if the docstring c contains type hint for the function arguments or returns (or variable annotations) **then**
            Remove the type hint
        end if
    end for
    Save modified file to R'
end for
Return: R'