

Benchmarking AI Models in Software Engineering: A Review, Search Tool, and Unified Approach for Elevating Benchmark Quality

ROHAM KOOHESTANI, Delft University of Technology, The Netherlands

PHILIPPE DE BEKKER, Delft University of Technology, The Netherlands

BEGÜM KOÇ, Delft University of Technology, The Netherlands

MALIHEH IZADI, Delft University of Technology, The Netherlands

Benchmarks are fundamental for ensuring unified evaluation and reproducibility. The rapid adoption of Artificial Intelligence in Software Engineering (AI4SE) has led to numerous benchmarks for tasks such as code generation and bug repair. However, this proliferation has introduced major challenges: (1) fragmented knowledge of benchmarks across tasks, (2) difficulty in selecting contextually relevant benchmarks, (3) lack of standardization in benchmark creation and enhancement, and finally (4) flaws that limit the utility of benchmarks. Addressing these challenges requires a dual approach: first, systematically mapping the current benchmark ecosystem to enable informed selection, and second, establishing unified guidelines for developing robust and adaptable benchmarks. In this work, we first provide a comprehensive review of 247 studies from which we identify **273 benchmarks** for evaluating AI4SE tasks since 2014. We categorize these benchmarks and analyze their limitations to highlight gaps in current benchmarking practices. Drawing on the insights derived from the review, we built an extensible semantic search tool, **BenchScout**, to locate suitable benchmarks. We utilized an automated clustering technique that incorporates embedded contexts of a given benchmark's associated studies.

We then applied dimensionality reduction and performed a parameter search to automatically optimize the configuration. To evaluate BenchScout's *usability*, *effectiveness*, and *intuitiveness*, we conducted a user study with 22 participants, yielding average scores of 4.5, 4.0, and 4.1 out of 5, respectively.

Next, to improve benchmarking standards, we propose **BenchFrame**, a unified approach to elevate benchmarks' quality. As a case study, we applied BenchFrame to the widely-used HumanEval benchmark to identify and rectify its key shortcomings. This resulted in **HumanEvalNext** which includes (1) corrected errors, (2) better language conversion, (3) increased test coverage, and (4) increased difficulty. We then assessed 10 state-of-the-art code language models on HumanEval, HumanEvalPlus, and HumanEvalNext. Compared to HumanEval and HumanEvalPlus, models' performance on HumanEvalNext drops by an average of 31.22% and 19.94% in the pass@1 scores, respectively. This significant drop illustrates the values of continuous improvement for higher quality benchmarks. Moreover, we tackle issues related to the scalability of BenchFrame through a case study involving an agentic pipeline that delivers similar results. Similarly, we confront questions of generalizability by employing BenchFrame on the MBPP dataset, which equally impacts models' performance comparably.

Lastly, we publicly release the material of our review, user study, and the enhanced benchmark. ¹

Additional Key Words and Phrases: Benchmark, Code Generation, AI4SE, Large Language Models, Search

¹<https://github.com/AISE-TUDELFT/AI4SE-benchmarks>

Authors' Contact Information: Roham Koohestani, Delft University of Technology, Delft, The Netherlands, rkoolestani@tudelft.nl; Philippe de Bekker, Delft University of Technology, Delft, The Netherlands, research@philippedebecker.com; Begüm Koç, Delft University of Technology, Delft, The Netherlands, bkoc@tudelft.nl; Maliheh Izadi, Delft University of Technology, Delft, The Netherlands, m.izadi@tudelft.nl.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

ACM Reference Format:

Roham Koohestani, Philippe de Bekker, Begüm Koç, and Maliheh Izadi. 2018. Benchmarking AI Models in Software Engineering: A Review, Search Tool, and Unified Approach for Elevating Benchmark Quality . 1, 1 (October 2018), 39 pages. <https://doi.org/XXXXXXX.XXXXXX>

1 Introduction

Benchmarks are essential for assessing artificial intelligence-driven software engineering (AI4SE) techniques. They provide standardized performance metrics, facilitate reproducibility, and guide innovation. However, the exponential growth in benchmark development has introduced significant challenges: researchers must navigate an increasingly fragmented landscape to identify benchmarks that align well with their specific objectives. This complexity often incentivizes reliance on popular or widely-adopted benchmarks without scrutinizing their applicability, inherent limitations, or potential flaws. Such practices risk propagating biases, overestimating technical progress, and misdirecting research priorities.

A notable example of benchmark limitations in code generation evaluation is HumanEval [26], a widely-used dataset that has been used for assessing large language models (LLMs) such as Codex [26], Gemini [46], and GPT-4 [136]. HumanEval was downloaded 82 thousand times in July 2025 on a single platform (Hugging Face) ², which supports the strong and sustained interest in this benchmark. Despite its broad adoption, HumanEval contains numerous flaws and inconsistencies [179]. For instance, Task 47, which requires computing the median of a numerical list, incorrectly states the median of the list $[-10, 4, 6, 1000, 10, 20]$ is 15. When queried on this task, ChatGPT-3.5 Turbo reproduced the same incorrect result (see Figure 1). This suggests potential data contamination and benchmark overfitting which can artificially inflate performance scores.

As HumanEval remains a widely-used benchmark in both AI and software engineering communities, several efforts have sought to expand its language support [9, 23, 131, 242] or improve test coverage [36, 118]. However, these extensions often build upon the original dataset without addressing its fundamental deficiencies, allowing inherent issues—such as

²https://huggingface.co/datasets/openai/openai_humanEval

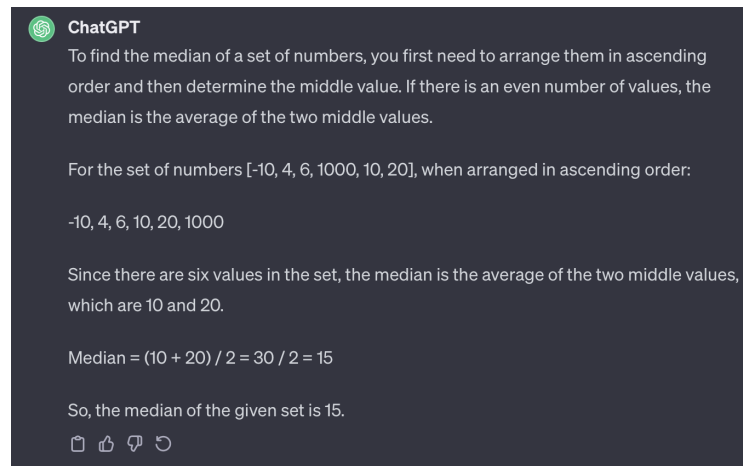


Fig. 1. ChatGPT repeating the same error from HumanEval (Screenshot captured in Dec 2023)

flawed canonical solutions, vague problem definitions, incorrect tests, and insufficient coverage—to persist. Moreover, LLM-augmented improvements, such as automatic translation, often lack rigorous quality control as well. Lastly, as models have advanced, HumanEval and similar popular benchmarks have become increasingly saturated, with close to 100% scores for recent models.³ This necessitates continuous elevation of the complexity of programs to better reflect the capabilities of contemporary models.

Our central objective is to **address the systemic challenges within AI4SE benchmarking**. we aim to address four key challenges in benchmarking: (1) the fragmentation of benchmark knowledge across tasks, (2) the difficulty of selecting contextually relevant benchmarks, (3) the lack of standardized approaches for benchmark creation and refinement, and (4) existing inherent flaws that limit benchmark utility. To address this, we use a three-stage approach where each stage builds upon the findings of the previous one.

More specifically, we first conducted a systematic review of benchmarking efforts in AI4SE from 2014 onward and identified **273 benchmarks** from 247 studies. We then extracted a given benchmark’s key metadata, including (1) objectives, (2) category, (3) programming language, (4) natural language, (5) relevant tasks, (6) extent to which tests are present, (7) scale of the dataset, (8) dataset source, (9) language specificity, (10) maintenance adequacy, (11) whether the dataset is reviewed or not, (12) whether it is frequently used or not, (13) the licensing, and (14) how it was created, to structure the AI4SE benchmark landscape. The findings from our review confirm a fragmented and difficult-to-navigate ecosystem, which motivated us to use these data to develop tooling to easily navigate the landscape of AI4SE benchmarking. **BenchScout**⁴ is an *extensible semantic* search tool that enables users to efficiently identify relevant benchmarks for specific software engineering tasks. To build BenchScout, we applied clustering techniques to contextual embeddings derived from related studies and benchmark documentation, along with our manually-extracted metadata. Additionally, we employed dimensionality reduction techniques to visualize the AI4SE benchmark landscape. We conducted a user study with 22 participants from both industry and academia to gauge the *usability*, *effectiveness*, and *intuitiveness* of BenchScout. It achieved average scores of 4.5, 4.0, and 4.1 out of 5, respectively.

While BenchScout aids navigation, our analysis confirms that many foundational benchmarks are flawed, which means a systematic solution is required to deal with them. Based on the identified gaps and limitations in current benchmarks, we introduce **BenchFrame**, a unified approach to enhance the quality of both existing and new benchmarks. To demonstrate its efficacy, we conduct a case study on HumanEval and present *HumanEvalNext* as an enhanced version.

When evaluating performance using HumanEval (original) and HumanEvalNext (our improved version based on the BenchFrame), we observe a substantial decline in pass@1 accuracy. Across ten state-of-the-art open-source code models, the average pass@1 score decreases by 31.2%, with a median drop of 26.0%. Performance remains significantly lower even on HumanEvalPlus [118], an enhanced version of HumanEval, with an average decline of 19.94% in pass@1 scores. These results highlight the importance of continuously refining benchmarks to better guide future research and provide more realistic assessments of model performance.

In summary, our contributions are as follows.

- We conducted a comprehensive review of 247 studies, identified 273 AI4SE benchmarks, and analyzed their limitations and gaps (Section 3),
- We developed and released **BenchScout**, an extensible semantic search tool to facilitate locating appropriate AI4SE benchmarks. Our user study with 22 participants demonstrated its effectiveness (Section 4),

³<https://evalplus.github.io/leaderboard.html>

⁴Accessible through <https://evalpro.online/>

- We propose a unified approach, **BenchFrame**, to improve the quality and reliability of benchmarks. A case study on HumanEval resulted in a refined, peer-reviewed benchmark, HumanEvalNext (Section 5). We assessed ten recent LLMs on this benchmark and verified significant drops in performance compared to the original HumanEval[26] and the enhanced version, HumanEvalPlus [118],
- We publicly share our data and details of the user study in our replication package.⁵

2 Research Questions

To systematically address the challenges outlined in the introduction, we structure our study around our central research objective: **to diagnose the fragmentation and quality issues within the AI4SE benchmark ecosystem, provide practical tools to navigate this complexity, and establish a rigorous methodology for future quality improvement.** We deconstruct this objective into three guiding Research Questions (RQs), which are designed to be addressed sequentially.

- **RQ1: Diagnosing the problem:** To what extent is the AI4SE benchmarking landscape fragmented, and what are its primary, systemic limitations?
- **RQ2: Providing a Navigational Aid:** How can a semantic search and visualization tool improve the discoverability and selection of relevant AI4SE benchmarks?
- **RQ3: Proposing a Systemic Solution:** What constitutes a rigorous and scalable methodology for improving benchmark quality, and what is its measurable impact on model evaluation?

These research questions directly map to the three-stage approach of our study. **RQ1** fulfills the diagnostic part of our objective by mapping the landscape and identifying its core problems. The findings from RQ1 directly motivate **RQ2**, which provides a practical tool to help the community navigate the benchmark fragmentation we uncovered. Finally, **RQ3** addresses the root cause of the quality issues found in RQ1 by proposing and validating a systematic framework for improvement, thereby fulfilling the final part of our research objective.

3 Existing Benchmarks, a Review

3.1 Search Criteria and Quality Assessment

We employed a systematic method to search for, identify, and classify AI4SE benchmarks. This procedure involved three primary stages: conducting structured searches, verifying credibility, and developing a taxonomy. We searched on two platforms, namely Google Scholar and Semantic Scholar using the following keywords: “Benchmarks”, “Software Engineering”, “Large Language Models”, “Evaluation”, and “AI4SE”. We additionally search for benchmarks present in the PapersWithCode datasets collection due to the popularity and wide usage of the platform⁶.

We selected these keywords based on a preliminary assessment of highly-cited benchmark research. Three authors worked together to revise the search criteria to ensure they were both relevant and complete. Our selection criteria targeted primary studies published in English from 2014 to 2025. After the initial paper collection process, duplicates were removed. Two authors reviewed the relevance of the identified pieces of literature, during which the originality (that is, the status as primary study), reproducibility, and accessibility of each study was assessed.

More specifically, for originality, we assessed whether a study was a primary study that introduced a new benchmark or a significant extension of an existing one. This filter was set in place to distinguish between benchmark-proposing

⁵<https://github.com/AISE-TUDELFT/AI4SE-benchmarks>

⁶This platform has since been discontinued and no longer accessible. The previously used url was <https://paperswithcode.com/datasets>

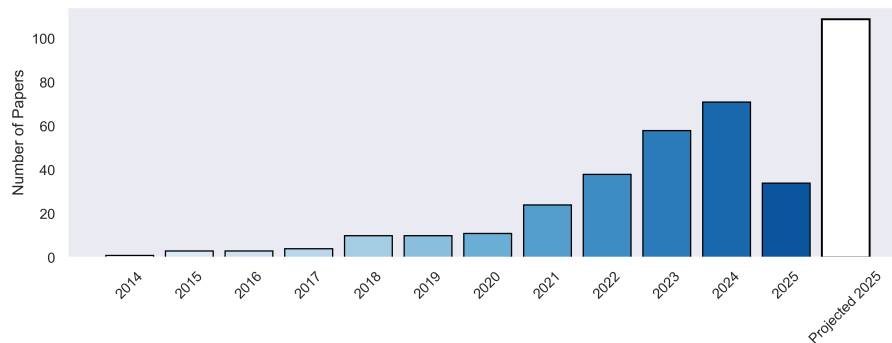


Fig. 2. Number of published benchmark papers in the last decade. (Jan 2014, Aug 2025)

papers from studies that merely applied them with a certain level preprocessing. In terms of Reproducibility and Accessibility, we evaluated whether the benchmark and its associated resources were made publicly available. We looked for the presence of a public repository (e.g., GitHub, HuggingFace, etc.) which would allow the community to access and use the benchmark. After this phase, we performed forward and backward snowballing. A total of 247 papers were found through this process.

After the papers were collected, the authors worked together to develop and consistently improve a taxonomy to efficiently categorize the benchmarks and extract metadata. Through continuous discussions, we identified initial essential categories while systematically gathering additional details for each study, including DOI and publication date. An iterative strategy was used to design the taxonomy and categorize SE tasks, starting with overarching categories such as reasoning, synthesis, and debugging. When the categories became too broad or lacked cohesion, the authors refined them into more detailed subcategories, establishing a multi-tiered hierarchy.

In our evaluation, we not only considered benchmark datasets from academic sources, but also included those suggested by industry and subsequently adopted by scholars (for example, IBM CODAIT and Aider [30]). Our replication package also serves as a dynamic repository, allowing researchers to contribute additional benchmarks and related papers by submitting a pull request that includes the paper's DOI.

After finalizing the taxonomy, and the list of benchmarks, we extracted (1) objectives, (2) categories, (3) programming language, (4) natural languages, (5) relevant tasks, (6) extent to which tests are present, (7) scale of the dataset, (8) dataset source, (9) language specificity, (10) maintenance adequacy, (11) whether the dataset is reviewed or not, (12) whether it is frequently used or not, (13) the licensing, and (14) how it was created, for each of the benchmarks.

3.2 Results of the Review

Our review revealed a significant increase in the number of benchmarks published over time (see Figure 2). We identified 273 benchmarks in total, with 71 published in 2024 alone. Using an exponential projection, we estimate that this number will reach 109 for the year 2025. This highlights the growing impact of AI4SE benchmarking and the need for a comprehensive overview of the existing literature.

Additionally, upon analyzing the distribution of the tasks included in the review, we find that a great portion of benchmarks are related to *Code Generation*, more specifically, 31.8% of all the benchmarks included. Along with Project-Level Code Generation/Analysis and Code Understanding (with respectively 18.7% and 12.0%), these make up

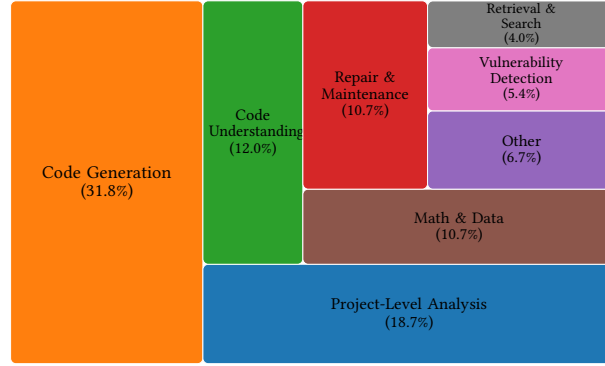


Fig. 3. The distribution of the high-level categories of the benchmarks included in the study.

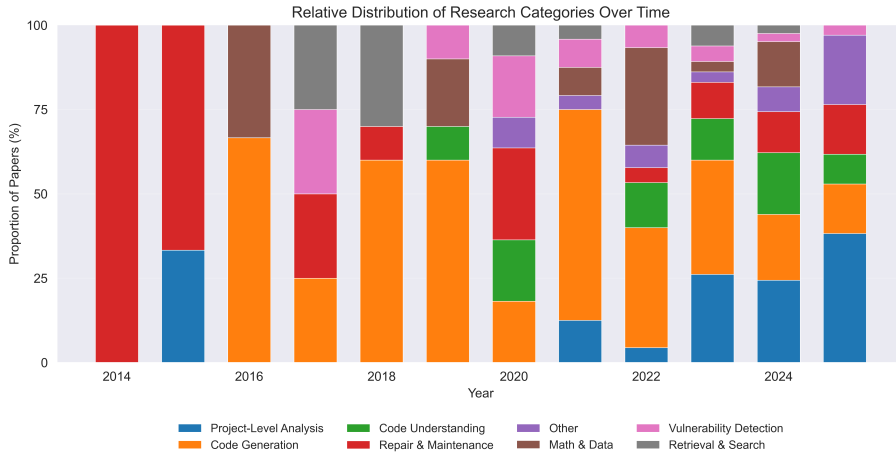


Fig. 4. The progression of task distribution for the benchmarks included in the study;

more than 60% of all benchmarks included. This, in itself, underscores a lack of attention to tasks such as vulnerability detection and code retrieval/search (cf. Figure 3).

The data reveal a notable change over time in the distribution of research categories throughout the observed period (cf. Figure 4). A key insight is the shift from code repair and maintenance to automated synthesis, marked by two opposing trends. Initially dominant, the Repair & Maintenance category gradually experiences a notable decline in its share. In contrast, starting around 2020, the Code Generation category gains prominence and reaches its peak in 2022, which coincides with the advancement of large-scale language models. In the recent years presented, there is an emerging trend of thematic diversification; as the focus on Code Generation recedes from its high, categories like Code Understanding and Retrieval & Search show a relative resurgence. This suggests a community-wide transition towards tackling additional challenges related to verification, integration, and the understanding of algorithmically generated outputs.

In the following, we summarize key insights from our review.

Table 1. Overview of AI4SE benchmarks stemming from HumanEval [26].

Category	Name	Language(s)	# Tests
Original	HumanEval [26]	Python	Avg. 7.7
Improved Language Support	MultiPL-HumanEval [23]	18 programming languages	Avg. 7.7
	HumanEval-Fix [131]	6 programming languages	Avg. 7.7
	HumanEval-Explain [131]	6 programming languages	Avg. 7.7
	HumanEval-Synthesize [131]	6 programming languages	Avg. 7.7
	HumanEval-X [242]	5 programming languages	Avg. 7.7
	Multi-HumanEval [9]	12 programming languages	Avg. 7.7
	HumanEvalXL [143]	12PLs, 23NLs	Avg. 8.33
	mHumanEval [148]	Python, 204NLs	Avg. 7.7
Improved Testing	HumanEval+ [118]	Python	Scaled $\times 80$
	HumanEval-MINI [118]	Python	Scaled $\times 47$
	HE-Eval [36]	Python	Scaled $\times 14$
Instruction-based	InstructHumanEval ⁷	Python	Avg. 7.7
Extended	EvoEval [201]	Python	Multiple categories, scaled with EVALPLUS

HumanEval Benchmark family: Currently, one of the most popular AI4SE benchmarks is HumanEval [26], used to evaluate the performance of many notable code-aware models (e.g., Codex [26], Gemini [46], and GPT-4 [136]). This benchmark is used mainly for code synthesis, though there also exist some variations for code repair and code explanation [131]. Table 1 presents the family of HumanEval benchmarks. After an in-depth analysis of these benchmarks, we identified the following issues: (1) incorrect tests, (2) lack of proper test coverage, (3) incorrect canonical solutions, and (4) imprecise problem definitions. While there are versions that have improved the language support [9, 23, 131, 242] and test coverage [36, 118], there is no version that contains all the improvements combined nor fixed the original issues. The issues for enhancing the original dataset can be generalized as follows:

- Variants that cover multiple languages have duplicated the original issues.
- Variants that added tests used the original incorrect solutions to generate the output.
- Variants based on human corrections or translations are inconsistent.

Furthermore, production systems like ChatGPT-3.5 tend to replicate errors from the original HumanEval benchmark. This indicates potential contamination from the benchmark data, *with the systems not only producing incorrect answers but also seemingly optimizing to match the flawed responses from the widely used benchmark*. Given the widespread use and ongoing popularity of the HumanEval benchmark in the research community, it is crucial to address its inherent flaws to prevent the perpetuation of these issues.

MBPP Benchmark Family: Another AI4SE benchmark, highly similar in style and popularity compared to HumanEval, is MBPP [10]: Mostly Basic Python Problems. It contains nearly a thousand crowdsourced problems, where almost half of it is sanitized and separately released. Furthermore, several enhancements have been published for MBPP (Table 2). Upon a more in-depth analysis of MBPP and its family of benchmarks, there are many signs suggesting deficient quality. One notable problem is the lack of proper testing, as MBPP originally only has three (rather trivial) tests

⁷<https://huggingface.co/datasets/codeparrot/instructhumaneval>

Table 2. Overview of AI4SE benchmarks derived from MBPP [10].

Category	Name	Language(s)	# Problems
Original	MBPP [10]	Python	974
Improved Language Support	MultiPL-MBPP [23]	18 programming languages	354-397 per language
	MBXP [9]	13 programming languages	848-974 per language
Improved Testing	MBPP+ [118]	Python	427
	MBPP-Eval [36]	Python	974

Table 3. Overview of competitive programming, code complexity, and code efficiency benchmarks.

Category	Name	Language(s)	# Tests
Competitive Programming	CodeContests [101]	12 programming languages	Avg. 203.7
	APPS [59]	Python	Avg. 13.2
	LiveCodeBench [76]	Python	Avg. 17.23
	LeetCode [172]	Python	Avg. 135
	CodeElo [147]	N/A	408 problems
	FVAPPS [37]	Python	4715 problems
	KodCode [206]	12 programming languages	Avg. 7.52
	FC2Code [123]	Python	320 flowcharts
	CodeContests+ [193]	Python	11,690 problems
	LiveCodeBenchPro [244]	Python	584 problems
Code Complexity	CoRCoD [162]	Java	932
	GeeksForGeeks (GFG) [130]	C++, Python	±1,400 per lang.&categ
	CODAIT ⁸	Python	4,000,000
	CodeComplex [14]	Java, Python	4,900 per language
	PythonSaga [208]	Python	185
Code Efficiency	EffiBench [68]	Python	Self-defined, avg. 100
	CODAL [196]	Python	3 ref. / problem
	PIE [159]	C++	82.5(median, train)
	COFFE [145]	Python	756

per problem – which are all revealed in the prompt as well. With such a test suite in place, evaluation metrics become unstable and insignificant for proper comparison. The strength of the written tests and solutions themselves is not only troublesome in the original data but also the sanitized data features many flaws (even in *corrected* variants [118]). From negligible observations such as poor syntax (e.g., too many spaces, Python method names starting with a capital – this is a common convention to only use for classes) to uncaught bugs and edge cases that break the implementation. While there are enhancements that improve the language support and extend the test cases, they are all built upon inadequate foundations, which renders any MBPP benchmark suboptimal for proper assessment.

Other Existing Benchmarks: Besides HumanEval and MBPP, the standardized benchmarks for code synthesis evaluation, there are many more considerable benchmarks for assessing various categories of SE tasks. We share several additional categorized tables as a guide for finding specific AI4SE benchmarks and highlight the most notable benchmarks for each in detail.

Table 4. Overview of data science & domain-specific benchmarks. (JN refers to Jupyter Notebooks.)

Name	Language(s)	# Tests	Comment
DS-1000 [87]	Python	Avg. 1.6	7 DS/ML libraries
NumpyEval [224]	Python	Avg. 20 functions	NumPy (101 problems) (Avg. 1 variable)
PandasEval [224]	Python	Avg. 20 functions	Pandas (101 problems) (Avg. 1 variable)
JulCe [2]	Python, JN	N/A	Cell completion (1.5M/3.7K train test)
DSP [24]	Python, JN	Available	Cell completion (1,119 problems)
ExeDS [69]	Python, JN	Execution Based	Cell generation (ground truth), 534 tasks
DSEval [240]	Python	custom approach	Models Evaluated via the DSEval Approach from the Paper
TorchDataEval [223]	Python	50	Private PyTorch data library
MonkeyEval [223]	Python	101	Private Pandas library fork
BeatNumEval [223]	Python	101	Private NumPy variant
Bio-Coder [171]	Python Java	1,026 1,243	Identify and import necessary classes for given task
Bio-Coder-Rosalind [171]	Python	253 golden solution	Generate code for question
WebApp1k [32]	React	Available	evaluates whether a model can generate React web-app

Table 5. Overview of Mathematical Reasoning Benchmarks.

Name	Language(s)	# Problems
MATH [60]	English	12,500
MATH500 [108]	English	500
MathQA [8]	English	37,297
MathQA-Python [10]	Python	23,914
MathQA-X [9]	Python, Java, JS	1,883 per language
LiLA[129]	Python	133,815 questions, 358,769 programs
MultiArith [152]	English	600
GSM8K [45]	English	1,320
GSM-HARD [45]	English	1,320
TheoremQA [28]	English	800
PECC [54]	Python	1,006
BRIGHT [167]	English	395
AMC12 ⁹	English	82

Table 3 features benchmarks with competitive programming as their root, i.e., those used for understanding code complexity and efficiency. Table 4 features a set of benchmarks specifically designed to evaluate the performance of

⁸CODAIT-2021 <https://ibm.co/4emPBla>

¹⁰<https://github.com/Aider-AI/aider/blob/main/benchmark/README.md>

Table 6. Overview of Natural Language Benchmarks.

Category	Name	Language(s)	No. of Problems
Text2Code	CoNaLa [215]	English → Python	2,879
	MCoNaLa [191]	{Spanish, Japanese, Russian} → Python	896
	CoNaLa-SO [137]	English → Python	10,000
	APPS [59]	English → Python	10,000
	APPS-Eval [36]	English → Python	10,000
	AixBench [55]	English, Chinese → Java	175
	Natural2Code [46]	English → Python	Unknown
	CoSQA [70]	English → Python	20,604
	WebQueryTest [124]	English → Python	1,046
	AdvTest [124]	English → Python	280,634
	CONCODE [75]	English → Java	104,000
	MTPB [132]	English → Python	115
	CAASD [238]	English → Python	72
	Shellcode_IA32 [110]	English → IA32/Shell	3200
	Odex [110]	{Spanish, Japanese, Russian, English} → Python	945 {90, 164, 252, 439} 1707 test total
	PSB2 [58]	English → {Clojure, Python}	25 question-answer pairs
	TACO [98]	English → Python	1,539,152 on 26,433 distinct tasks
	Turbulence [61]	English → Python	60 (with 420 total test cases)
	Aider ¹⁰	English → {C++, GO, Java, JS, Python, Rust}	225 problmes
	NL2ML-lib [158]	English → Python (ML libraries)	11,000
	RMCBench [25]	English → 9 Languages	473 malicious prompts
	Evil [109]	English → {Python, IA_32}	19255
	Exec-CSN [203]	English → Python	1,931
	CodeIF [209]	English → {Java, Python, Go, C++}	1,200 tasks
	CodeIF-Bench [184]	English → Python	122 tasks (42 repositories)
	ARCADE [216]	English → Python	1082
	StackEval [155]	English → Python	925
	SwiftEval [146]	English → Swift	28
	CoSQA+ [47]	English → Python	412,080 pairs

models on Data Science-related tasks along with some other domain-specific SE tasks. A notable example in this table is the Bio-Coder series of benchmarks specifically designed for bioinformatics tasks. To assess the mathematical reasoning capabilities of AI4SE models, see Table 5. Besides numbers and code, natural language is also a key component in AI4SE. From supporting instruction-tuned AI4SE models, which align more with the human brain [11], that aim to accomplish question and answering (QA) similar to the widely recognized platform StackOverflow, to summarizing code and generating tags, Table 6 and Table 7 feature AI4SE benchmarks including natural language: text-to-code, code-to-text and text-to-text (code related). We have intentionally isolated all SQL-related benchmarks due to their abundance and to facilitate locating the correct benchmark; these are presented in Table 8.

Table 7. Overview of Natural Language Benchmarks (Continued).

Category	Name	Language(s)	No. of Problems
Text2Text (about code)	InfCoder-Eval [74]	English → English	270
	BRIGHT [167]	English → English	1,398
Code2Text	DeepCom [64]	Java → English	588K
	Hybrid-DeepCom [65]	Java → English	466k
	BinSum [82]	Binary functions → English	557K
	Code Attention [6]	Java → English	11 projects
	Funcom [88]	Java → English	2.1M problems
	CodeSum [66]	Java → English	410,630
	CoDesc [56]	Java → English	4.21M datapoints
	Parallel [17]	Python → English	150k function/doc pairs
	CoDocBench [140]	Python → English	4573 code/doc pairs
	PoorCodeSum [63]	{Java, Python, Go} → English	{10,955 , 14,918 , 8,122}
	P-CodeSum [221]	{Python, Java, Go, JS, PHP, Ruby} → English	1,500 pairs

Table 8. Overview of SQL-related Benchmarks.

Category	Name	Language(s)	No. of Problems
Text2Code	BIRD [94]	English → SQL	12,751
	KaggleDBQA [89]	English → SQL	272, paired with golden solutions
	StacQc [213]	English → {Python/SQL}	{147,546 / 119,519} question-answer pairs
	Spider(V2 ¹¹) [91]	English → SQL	632 queries
	Spider-Syn [43]	English → SQL	(7000 / 1034)
	Spider-Real [33]	English → SQL	508
	Spider-DK [44]	English → SQL	535 pairs
	Spider-CN [127]	Chinese → SQL	9691 queries
	SParC [220]	English → SQL	4,298 question sequences
	Lyra [106]	{English, Chinese} → {python, SQL}	2000
	DuSQL [183]	Chinese → SQL	23,797 question/SQL pairs
	CoSQL [218]	English → SQL	3,007 Question Sequences
	SynSQL-2.5M [93]	English → SQL	2,544,390
	PAUQ [15]	Russian → SQL	9,691
	Ar-Spider [7]	Arabic → SQL	9,691
	Tur2SQL [84]	Turkish → SQL	10,809 question/SQL pairs

While translating natural language is more trivial nowadays, translating code remains challenging due to various reasons (e.g. versioning, semantics, dependencies). With the lack of diversity in language support for AI4SE benchmarks and also benefiting numerous other SE tasks, Table 14 features an overview of resources that can support the ongoing development of code translation.

However, the workflow of a developer is not merely an exercise of writing snippets of code for given descriptions, but rather having a general overview of a project and how one can implement functionality such that it fits well into a

Table 9. Overview of Selected Real-to-Life SE Benchmarks. (Note: X/Y/Z denotes Train/Dev/Test)

Category	Benchmark	Language(s)	No. of Problems
Software Development & Agent Benchmarks	DevBench [92]	Python, C/C++, Java, JavaScript	22 repositories
	DevEval [96]	Python	1,874
	CoderUJB [228]	Java	2,239
	CODAL [196]	Python	500
	ToolQA [252]	Python, Math, English	800(Easy)/730(Hard)
	MIT [188]	Python, English	586 Problems
	SAFIM [48]	Python, Java, C++, C#	17720
	AgentBench [122]	N/A	1360 prompts
	CSR-Bench [202]	Python	100 repositories
Class Level	ClassEval [38]	Python	100
	CONCODE [75]	English, Java	104,000
	BigCodeBench [253]	Python	1,140
	OOP-Bench [185]	Python	431
	CodeSense [151]	Python, C, Java	2125 (Python), 876 (C)
	ClassEval-T [207]	Java, C++	875 (Java) 1,243
Project & Cross-file	SWE-bench [81]	Python	19,008 (Train), 225 (Dev), 2,294 (Test) 144 (Small)
	CrossCodeEval [35]	C#, TypeScript, Java, Python	2,665 (Python), 2,139 (Java), 3,356 (TypeScript), 1,768 (C#)
	CoderEval [217]	Java, Python	230
	DotPrompts [3]	Java	105538 problems (1420 methods)
	BigCloneBench [168]	Java	25,000 Java Systms
	DI-Bench [235]	Python, C#, Rust, JS	581 repositories (w/ dependencies)
	DyPyBench [20]	Python	50 repositories
	Multi-SWE	Java, TypeScript, JavaScript	500 (Python), 128 (Java),
	-bench [225]	Go, Rust, C, C++	224 (TS), 356 (JS), 428 (Go), 239 (Rust) 128 (C), 129 (C++)
	KernelBench [139]	Python	250
	CodeMEnv [29]	Python, Java	587 (Python), 335 (Java)
	CodeEditorBench [52]	Python, Java, C++	7,961
	ProjectEval [119]	Python	284

collective code base. For this, Table 9 and Table 10 provide a collection of benchmarks that examine the capabilities of models to generate code on a larger scale.

The utilization of APIs plays a significant role in AI4SE benchmarks, specifically for models with Retrieval Augmented Generation (RAG) capabilities. In Table 11, prominent benchmarks focusing on leveraging the power of APIs are denoted. Furthermore, Table 12 lists benchmarks related to pseudocode, followed by an overview of notable crowd-sourced AI4SE resources in Table 13.

Table 10. Overview of Selected Real-to-Life SE Benchmarks. (Continued - Note: X/Y/Z denotes Train/Dev/Test)

Category	Benchmark	Language(s)	No. of Problems
Repository Level	RepoBench [121]	Python, Java	Cross-file: 8,033, In-file: 7,910
	RepoEval [232]	Python	1,600 (line), 1,600 (API), 373 (function)
	EvoCodeBench [95]	Python	275
	SketchEval [226]	Python	19 repositories (5 easy, 8 medium, 6 hard)
	Stack Repo [226]	Python	(435,890 / 220,615 / 159,822) answer pairs
	ML-BENCH [170]	Python & Bash	9641 problems
	CodeGen4Libs [120]	Java	403,780 prompts
	SWE-rebench [13]	Python	294 tasks, 169 repositories
	SWE-Polybench [150]	Java, JavaScript TypeScript, Python	165 (Java), 1017 (JavaScript), 729 (TypeScript), 199 (Python)
	HumanEvo [241]	Python, Java	200 (Python), 200 (Java)
	REPOCOD [107]	Python	980 functions (11 projects)
	FEA-Bench [99]	Python	83 repositories
	JavaBench [22]	Java	389 methods (106 classes)
	SWE-bench live [234]	Java	1,319 tasks (93 repositories)
	SWE-Lancer [128]	N/A	1,488 tasks)

Table 11. Overview of Selected API and Retrieval Benchmarks by Category.

Category	Benchmark	Sources/API(s)	No. of Problems
API Prediction	RestBench [165]	Spotify, TMDb	57, 100
	APIBench-Q [144]	StackOverflow, Tutorial Websites	6,563 (Java), 4,309 (Python)
	BIKER [71]	StackOverflow	33,000
	Gorilla APIBench [141]	HuggingFace, TensorHub, TorchHub	925, 696, 94
	Gorilla APIZoo [141]	Open submissions (Google, YouTube, Zoom, etc.)	-
Retrieval & Planning	API-Bank [97]	73 commonly used APIs	753
	CodeRAG-Bench [194]	Competition solutions, tutorials, documentation, StackOverflow, GitHub	25,859
	Search4Code [149]	Bing	6596(java)/4974(c#)
	CoIR [100]	GitHub, StackOverflow, and Various Benchmarks	2.38M (corpus) 3.37(queries)
Memorization	SATML-ext [5]	GitHub	1,000 samples
API Misuse Detection	ExampleCheck [239]	StackOverflow	100 (Java)
	ROBUSTAPI [245]	StackOverflow	1208 (18 Java APIs)
	APIMU4C [51]	Juliet Test Suite, ITC, OpenSSL, Curl, Httpd	2272 (C)

Table 12. Overview of AI4SE Benchmarks Related to Pseudocode.

Category	Benchmark	Language(s)	No. of Problems	Crowdsourced
Pseudocode to Code	SPoC [85]	C++	18,356	Yes
	NAPS [227]	Java/UAST	17,477	No
	PseudoEval [198]	Python, C++, Rust	1,060	No
Code to Pseudocode	Django [135]	Python, English & Japanese	18,805 (Train), 1,000 (Dev), 1,805 (Test)	No

Table 13. Overview of Selected Crowd-sourced Benchmarks.

Benchmark	Language(s)	No. of Problems	Source
WikiSQL [246]	Natural language → SQL query	80,654	Amazon MTurk (deprecated - 2017)
Spider [219]	Natural language → SQL query	10,181	11 Yale students (2018)
NL2Bash [114]	Natural language → Bash	9,305	Upwork (2018)
NAPS [227]	Java/UAST → Pseudocode	17,477	Self-hosted crowdsourcing, comp. programming community (2018)
SPoC [85]	C++	18,356	comp. programming websites (2019)
MBPP [10]	Python	974	Google Research, internal crowdworkers (2021)

Table 14. Overview of Programming Language Translation Benchmarks (Note: X/Y/Z denotes Train/Dev/Test).

Category	Name	Language(s)	No. of Samples
Programming Languages	CodeTrans [124]	C#, Java	11,800
	TransCoder-ST [153]	C++, Java, Python	437,030
	CoST [250]	7 programming languages	16,738
	AVATAR [4]	Java, Python	7,133 / 476 / 1,906
	Multilingual-Trans [211]	8 programming languages	30,419 total
	NicheTrans [211]	Various niche languages	236,468 total
	LLMTrans [211]	8 programming languages	350
	G-TransEva [80]	5 programming languages	400 total
	CODEEDITOR [233]	C# & Java	6613
	RustRepoTrans [138]	C++, Java, Python → Rust	375
	AVATAR-TC [77]	Java, Python	55,179 / 443 / 1,746
	RustRepoTrans [138]	C++, Java, Python → Rust	375
	RepoTransBench [189]	Python → Java	100 repositories
Libraries	DLTrans [211]	PyTorch, TensorFlow, MXNet, Paddle	408 total
Intermediate Representation	SLTrans [142]	14 Languages → LLVM-IR	4M
Language Conversion Frameworks	MultiPL-E [23]	19 programming languages	-
	MultiEval [9]	13 programming languages	-

Table 15. Overview of Automated Program Repair, Fault Localization, and Vulnerability Detection Benchmarks.

Category	Benchmark	Language(s)	No. of Samples
Automated Program Repair & Fault Localization	Defects4J [83]	Java	835
	GitBug-Java [164]	Java	199
	EvalGPTFix [237]	Java	4530
	TutorCode [212]	C++	1239
	GHRB [90]	Java	107
	IntroClass [49]	C	998
	ManyBugs [49]	7 Languages	185
	DebugBench [173]	C++, Java, Python	1,438 & 1,401 & 1,414
	QuixBugs [112]	Java	40 (locations of bugs)
	RES-Q [86]	Python, JS	100 hand-crafted questions + test scripts
	StudentEval [12]	Python	1,749 buggy programs (48 tasks) (3 test cases per problem)
	Re-Factory [67]	Python	1783(buggy)/2442(correct)
	ConDefects [200]	Python, Java	526(Python), Java(477)
	Cerberus [156]	C, C++, Java	2242 (across 4 task types)
	RepairBench [163]	Java	574
	MaRV [134]	Java	693
	BugsinPy [197]	Python	493 bugs
Vulnerability Detection	CVEFixes [18]	Various Languages	5,365
	LLMSECEval [176]	C	150 (on 25 distinct vulnerabilities)
	SecurityEval [161]	6 languages	130 (on 75 common vulnerabilities)
	Vul4J [21]	Java	79 vulnerabilities
	FormAI [175]	C	112,000 labeled instances
	VJBench [199]	Java	42 vulnerabilities
	SmartBugs [39]	Solidity	69 Vulnerable Smart Contracts
	Devign [247]	C	4 large-scale software repositories
	D2A [243]	C/C++	6 OSS Programs
	BigVul [41]	C/C++	348 Projects
	SARD ¹²	Java, C, C++, C#, PHP	32k ¹³
	Juliet 1.3 ¹⁴	C/C++	64k ¹⁵
	NVD ¹⁶	Various Languages	265k ¹⁷
	ARVO [126]	C, C++	1,001 vulnerabilities
	VADER [116]	15 languages	174 vulnerabilities
	ManyVuls4J [111]	Java	103 vulnerabilities
Software Testing	CoverageEval [177]	Python	1160
	ATLAS [195]	Java	9,275 projects
	HITS [192]	Java	10 projects
	MeMo [16]	Java	9 projects
	MLAPIs [181]	Python	63 applications
	CoderUJB [229]	Java	2,239
	TestBench [236]	Java	108
	TestEval [187]	Python	210
	TARBENCH [214]	Java	45,373 (59 projects)
	ProjectTest [190]	Python, Java, Javascript	20 per language
	CLOVER [204]	Python	845

Table 16. Overview of Selected SE-Workflow Benchmarks.

Category	Benchmark	Language(s)	No. of Samples
Code Synthesis & Understanding	Methods2Test [178]	Java	780,944
	CRUXEval [50]	Python	800
	CRQBench [34]	C++	100
	CriticBench [115]	Python	3,825(across 5 tasks)
	CodeScope [210]	8 PLs	13,390 (across 8 tasks)
	CodeCriticBench [230]	Various Languages	1,517 (Easy), 1,084 (Medium), 1,699 (Hard)
	CRUXEval-X [205]	19 PLs	19K
Merge Conflict Repair	ConflictBench [157]	Java	180
Type Inference	TypeEvalPy [180]	Python	845 (annotated labels)
	TypeEvalPy AutoGen [180]	Python	78373 (annotated labels)
Automatic Code Quality Review	CodeReview [104]	8 languages	7.9M pull requests
	Software Maintainability [154]	Java	519 projects (evaluations of quality)
	BenMark [186]	Java	1,299,186 methods
	CodeReviewer [105]	9 languages	13,100
	CodeReview-New [53]	16 languages	14,600
	CodeReviewQA [113]	9 languages	900
Hallucination Detection	HALLUCODE [117]	Python	5,663
	CodeHaluEval [174]	Python	699
	Collu-Bench [78]	N/A	13,234
	LMDefects [42]	Java	60 (Easy), 53 (Medium)
	Codemirage [1]	Python	1,137

With AI4SE models mainly being utilized for program synthesis, it remains relatively questionable how effective these models are in generating tests and repairing bugs, as it is unclear whether these models *truly* understand code. For example, Siddiq et al. [160] observed Codex [26] being able to get above 80% coverage for HumanEval [26], yet many test smells were discovered and for another dataset, no higher than 2% coverage was attained. This reveals the importance of benchmarking AI4SE models' capabilities in test generation, bug repair, and understanding. In Table 15, several benchmarks are listed that make an effort to assess the aforementioned. Additionally, Table 16 table features benchmarks that have been designed to evaluate a model's capabilities in dealing with everyday tasks of a software engineer (e.g., merge-conflict repair, Code Reviews, etc.).

Additionally, we collect a set of generic benchmarks in Table 17. These benchmarks are not only single-task benchmarks like others previously seen in this section but are rather a collection of tasks spanning a wide range of languages and task types. A notable, and widely known benchmark in this category is the BigBench Benchmark [166] which consists of 167 tasks (not all relevant to AI4SE).

¹³<https://samate.nist.gov/SARD/>

¹⁴<https://samate.nist.gov/SARD/test-suites/112>

¹⁵As of 4th Feb 2025

¹⁶<https://nvd.nist.gov/developers/data-sources>

¹⁷As of 4th Feb 2025

¹⁸<https://github.com/MicrosoftDocs/>

Table 17. Overview of Multi-Category Benchmarks, Covering Various Tasks.

Name	Language(s)	Tasks	Information
Big-Bench [166]	Python, Numeric, JSON, English	Functions over numbers, Mathematical Reasoning, Text2Code, Code2Text, Code explanation, Debugging, Turing Complete Concept Learning, amongst other tasks	250, several per category, 42, 60, 66, 34, 6,390
XLCoST [249]	C, C++, C#, Java, JavaScript, Kotlin, PHP, Python, Ruby, Rust	Text2Code (program synthesis, code search), Code Summarization, Code Translation	567K (509k, 58k), 567K, 122K
CrossCodeBench [133]	Java, C#, Python, C++, JS, PHP, Go, Ruby, TS, C, Bash, Shell	Classification, In-Filling, Translation, Generation, Summarization, Type Prediction, Question Answering	6.6M, 13.4M, 2.4M, 19.5M, 11.2M, 773K, 190K
Long Code Arena [19]	English, Python, Java, Kotlin	Commit Message Generation, Module Summarization, Library-Based Code Generation, Project-Level Code Completion, Bug Localization, CI Builds Repair	163, 216, 150, 908 (varying sizes), 14.96K, 78
CodeXGLUE [19] MicrosoftDocs ¹⁸ CodeSearchNet [73]	English, Chinese, Norwegian, Danish, Latvian Go, Java, JavaScript, PHP, Python, Ruby	Code Documentation Translation, Code Documentation (Code Summarization, Comment Generation)	(CN: 52K, NOR: 26K, DK: 45K, LT: 21K), 621870
DomainEval [251]	Python	Computation, Network, Basic operation, System, Visualization, Cryptography	5,892 cases total
CodeApex [123]	C++, English, Chinese	Programming comprehension Code generation, Code correction	250, 476, 1330

Table 18. Overview of Selected Benchmarks for Log Statement Generation and Parsing.

Category	Benchmark	Language(s)	No. of Problems
Log Statement Generation	LANCE [125]	Java	76,421
	LogBench [102]	Java	6,849
	SCLogger [103]	Java	31,170
	AL-Bench [169]	Java	39,600
Log Parsing	LogBase [231]	Various Languages	85,300
	LogHub [248]	Various Languages	32,000
	LogHub-2.0 [79]	Various Languages	3.6M
	LogPM [57]	Various Languages	10,821,589
	LogEval [31]	English, Chinese	4,000 log entries across 4 tasks

Table 19. Common limitations in AI4SE benchmarks

Limitation	Description	Representative Examples
Language Specific	Focuses only on one programming language	HumanEval, MBPP, APPS
No Leaderboard	No official leaderboard; hinders fair and consistent comparisons	SPoC, RepoEval, CONCODE
Poor Maintenance / Data Quality	Dataset contains outdated problems, lacks documentation, or is not updated regularly	ToolQA, ExampleCheck
Not Peer Reviewed	Not been published in a peer-reviewed venue	CodeSearchNet, Methods2Test
Infrequently Used	Rarely cited in academic work	JavaBench, CoIR
Dataset Creation Method	How problems were obtained: <i>mined</i> from real-world sources, <i>created</i> by humans, or <i>LLM-generated</i>	AVATAR (mined), Spider (created), CRUXEval (generated)

Finally, we have identified a rather large count of papers that have to do with the matter of Log Parsing and Log Statement generation. we have presented these benchmarks in Table 18.

3.3 Limitations of Existing Benchmarks

Across our review, we observed several trends in the existing AI4SE benchmarks. The majority are language-specific, most commonly targeting Python or Java. Although these languages are popular in both education and industry, this narrow scope potentially limits their generalization across domains and programming environments. Many benchmarks suffer from poor maintenance, with little to no updates or active support following their initial release. Official and dynamic leaderboards are often missing, which makes it difficult to fairly compare model performance across time or publications, particularly as newer models continue to emerge. Notably, benchmark popularity does not appear to strongly correlate with its maintenance quality or whether it was peer reviewed, suggesting other factors such as visibility or convenience may drive adoption.

There is also variation in how benchmarks are constructed. Manually created datasets are often significantly smaller, whereas newer benchmarks increasingly leverage LLMs to generate large volumes of tasks. Mined datasets which are mostly sourced from GitHub, can be large but are frequently under-specified, with limited transparency into the repositories used or the sampling heuristics applied.

Table 19 summarizes these limitations and provides representative benchmarks for each. The full list of evaluated benchmarks and their metadata can be found in the replication package of the study.

In Conclusion, based on our in-depth inspection of HumanEval and MBPP and combined with the inspection of other benchmarks in our review, we obtained an overview of the limitations in current AI4SE benchmarks which we use as a guide in shaping our methodology proposed in Section 5.

4 BenchScout- Locating AI4SE Benchmarks

Due to the abundance of AI4SE benchmarks, identifying the most suitable one for a specific software engineering task can be challenging. As a result, many default to evaluating their models on popular benchmarks like HumanEval [26] which has its own flaws.

To address this gap, we developed BENCHSCOUT¹⁹, a tool to systematically and semantically search the existing benchmarks and their corresponding use cases. We additionally provide an interface to visually evaluate the closeness and similarity of a group of datasets, along with capabilities to find relations between citing bodies for identifying patterns relevant to different use cases.

4.1 Context Extraction and Visualization

4.1.1 Overview of the Semi-Automated Pipeline. To effectively contextualize and visualize the growing corpus of AI4SE benchmarks, we developed a semi-automated pipeline that extracts, embeds, clusters, and visualizes benchmark metadata. As illustrated in Figure 5, the system is organized into four modular layers: data collection, metadata enrichment, clustering and labeling, and front-end interaction.

The pipeline begins with benchmark sources collected from a curated spreadsheet, user-submitted suggestions, and an automated paper discovery script using the Semantic Scholar API to make suggestions. All inputs are aggregated into a processing queue, where metadata is enriched using both automated API calls and basic scraping techniques (e.g., fetching GitHub or Hugging Face README files). These metadata are transformed into dense vector representations using OpenAI’s `text-embedding-3-small` model.

To support user navigation and cluster discovery, the high-dimensional embeddings are reduced via UMAP and grouped using HDBSCAN. A key component of the pipeline is the integration of GPT-based cluster labeling, which generates short, descriptive names for each cluster based on the titles and descriptions of the papers it contains. This step is fully automated but subject to human validation when new clusters emerge; meaning, these clusters *can* be modified by humans if needed.

4.1.2 Balancing Automation and Oversight. Unlike other prior approaches that rely on manual curation or entirely unsupervised grouping, our pipeline adopts a semi-automated architecture designed to scale with minimal human intervention while retaining curatorial oversight. Specifically, the system regularly ingests new suggestions, both from automated Semantic Scholar queries and from users interacting with the front-end interface, and proposes candidate entries for including in the pool of benchmarks. Although metadata extraction and embedding are fully automated, curators retain the ability to approve or reject new data sources, review GPT-generated labels, and fine-tune cluster boundaries as needed.

4.1.3 Interactive Visualization. The final layer of the pipeline provides an interactive 2D interface that enables users to explore benchmarks based on their semantic similarity. The visualization supports fuzzy search, local citation graphs, and interactive paper details. Importantly, user interactions on the frontend—such as suggesting new benchmarks—are routed back into the data collection layer (cf. Figure 5).

¹⁹<https://evalpro.online/search.html>

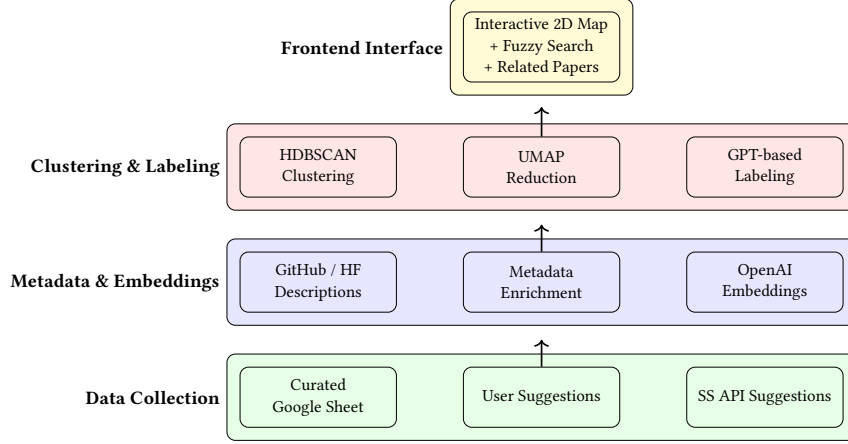


Fig. 5. Pipeline architecture of BenchScout. This includes automated metadata extraction, semantic embedding, clustering, and interactive visualization.

4.2 Additional Features

To improve the search and exploration experience, we incorporated several key functionalities. First, a **text-based search interface** allows users to find articles by title and abstract using **fuzzy search** via the Fuse.js library²⁰. Users can also issue **advanced field-specific queries** (e.g., `language:python`) or **exact phrase queries** (e.g., `"code generation"`), and benefit from **autocomplete suggestions** for common terms, programming languages, tasks, and datasets.

A dynamic **2-D UMAP visualization** displays papers embedded in semantic space. Hovering reveals abstracts via a **Paper Content Tooltip**, while double-clicking redirects to the paper's DOI page. Paper nodes are sized based on **normalized citation count**, colored by cluster, and annotated with metadata such as year, tasks, and dataset names.

Clicking a point activates the **Related Papers feature**, which shows (1) The top-5 most similar papers based on cosine similarity, (2) A **detailed paper overview** with title, abstract, authors, venue/journal, and publication type, (3) A list of **citing papers** with sortable tables and filters by year and venue, and (4) An interactive **Paper Citations Graph**, which visualizes inter-citation relationships among citing papers.

Users can navigate between **chart**, **list**, and **grid** views, paginate results, and apply **advanced filters** by programming/natural language, dataset, task, cluster, year, and citation count. A **search history** and **keyboard shortcuts** improve usability. The entire interface supports responsive interaction with detailed feedback and error handling.

4.3 User Study

With all the mentioned features, we aim to create a platform that can be extended and used by both academics and practitioners alike to find the appropriate dataset/benchmark for their use case with more ease. To evaluate how effective and usable this new tool is for the end-users, we conducted a user study on 22 people from both industry (9) and academia (13). In selecting demographics for the BenchScout user study, we aimed to assess the tool's effectiveness across a diverse group of users with varying degrees of expertise. This approach seeks to determine the tool's applicability for individuals at either end of the spectrum—whether they are beginners or seasoned experts, from academia or industry. The ultimate objective of the tool is to facilitate the selection of the appropriate benchmark, making it more accessible

²⁰<https://www.fusejs.io/>

Table 20. Questionnaire Design Overview

Section	Questions	Scale	Additional Info
Participant Background	What is your professional background? What is your role? How familiar are you with AI4SE benchmarks? How many years of experience do you have in this field?	5-point Likert Scale (1: Not familiar, 5: Very familiar)	Experience question (<1, 1–3, 3–5, 5+ years)
Search Functionality	How easy was it to navigate the interface? How intuitive was the search functionality? How effective was the tool in finding benchmarks? Was the visual evaluation of datasets useful?	5-point Likert Scale (1: Not useful, 5: Extremely useful)	N/A
Cross-referencing Feature	How useful was the cross-referencing feature? Did the tool help in understanding relationships between benchmarks? Was the visual interface for benchmark similarity useful?	5-point Likert Scale, Open-ended	Includes qualitative feedback option
User Experience & Feedback	How would you rate the overall user experience? How likely are you to use the tool in your work? Did you experience any issues or challenges? What other tools do you use for searching benchmarks? How does this tool compare to others? How well does the tool meet the needs of professionals in AI4SE?	Likert Scale, Open-ended	Open-ended questions for in-depth feedback

irrespective of prior knowledge. For this, we had each participant interact with the tool for however long they saw fit and asked them to fill out a questionnaire consisting of eleven 5-point Likert Scale questions and three open questions.

4.3.1 Questionnaire Design. The questionnaire designed for the study was divided into four key sections to gather feedback about the tool, namely the participant’s background, the quality of the search functionality, the quality of the cross-referencing feature, and the overall user experience. Table 20 is an overview of the posed questions.

4.3.2 Results and Analysis. We analyzed the data collected through the questionnaire both quantitatively (Likert scale responses, scaling between 1-5) and qualitatively (open-ended questions). The detailed results are provided in the replication package.²¹ Below, we present an overview.

²¹<https://github.com/AISE-TUdelft/AI4SE-benchmarks>

The respondents of the questionnaire were generally familiar with AI4SE, with an average familiarity rating of 3.8. There were varying levels of experience in the field, with the range between 1-3 years being most common (eight people). More concretely, in our pool of participants, the demographic distribution was as follows:

- (1) **Roles:** The participants held diverse roles, including 6 Researchers, 5 PhD Candidates, 5 Students, 4 Software/Research Engineers, and 2 Lead Researchers
- (2) **Experience Level:** 8 participants had 1-3 years, 6 had 3-5 years, 3 had 5+ years, and 5 had less than 1 year of experience
- (3) **Familiarity with AI4SE:** On a scale of 1 (Not familiar) to 5 (Very familiar), 8 participants rated themselves a "5" and 7 rated themselves a "4," indicating a strongly informed participant pool

In terms of search functionality, the tool scored high on usability with an average rating of 4.50, showing that users found it easy to navigate. The intuitiveness of the search interface received a solid 4.1, while its effectiveness in helping users find benchmarks was rated 4.0. The visual evaluation feature received a rating of 3.8, indicating some room for improvement in how visual elements assist in the search process.

When evaluating the cross-referencing features, respondents found the overall usefulness to be 4.1. However, the tool's ability to help users understand connections between different benchmarks was at 3.7. The visual interface for exploring these connections was rated 3.9, suggesting that while users found it generally helpful, enhancements could improve its utility.

Regarding the overall user experience, participants gave an average rating of 4.2, with a score of 4.0 on the likelihood of using the tool in their own research. Several issues were highlighted, particularly around the dimensionality reduction and how the scatter plot is organized and presented. Users also noted that the citation network feature becomes less effective with larger papers and called for improved clustering by topic and additional features to explain and control visualizations.

The participants' responses confirmed our findings and highlighted the lack of a specific tool dedicated to locating AI4SE benchmarks. Instead, respondents commonly rely on generic platforms like Huggingface, Semantic Scholar, Google, and ConnectedPapers. When compared to these tools, BenchScout received an average score of 4.2 out of 5, with 5 indicating a much better experience. One participant mentioned using their personal network to find benchmarks, which limits broader access, further supporting the need for the proposed tool.

The tool's ability to meet the professional needs of users was rated 4.2 which affirmed its usefulness in the AI4SE domain. However, respondents suggested several additional features that could enhance its functionality, such as pagination for citations, incorporating metadata and additional information about the papers in the search process, improved clustering and filtering options, and sorting citations based on specific criteria. Additional requests included dark mode support, better overall search functionality, and clearer explanations and control over the chart visualizations. Based on the users' feedback and components' scores, we prioritized these features and incrementally added them to the platform.

In conclusion, while the tool is largely perceived as useful and user-friendly, there were several areas for improvement, particularly around visualization, citation handling, and filtering options, which could enhance the overall user experience and the tool's effectiveness in AI4SE research. Given the time constraints, we prioritized and implemented key features, leaving some for the future.

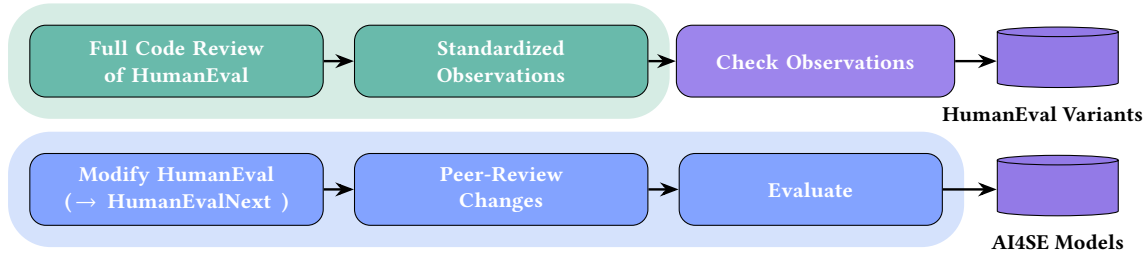


Fig. 6. Outline of the general approach behind BENCHFRAME through a case-study of HUMANEVAL.

5 BenchFrame

In this section, we propose BENCHFRAME, a detailed and peer-review-oriented framework for improving the quality of existing benchmarks. We explain our approach through a case study in which we propose HUMANEVALNEXT, a corrected foundation based on the HumanEval benchmark.

The HumanEval benchmark has long been the de facto standard for evaluating the code-generation capabilities of AI4SE models. It has recently been used to evaluate the latest and greatest LLMs from large companies such as Google Gemini [46] and OpenAI GPT4 [136]. Despite the great fame, however, our review in Section 3 points towards the existence of numerous notable problems with this foundational benchmark, namely, the existence of incorrect tests, suboptimal canonical solutions, and imprecise problem definitions amongst many others.

5.1 Approach

To improve the quality of the given benchmark, we pursue the following approach – also illustrated in Figure 6. First, we initiate a comprehensive code review, leading to standardized observations (Section 5.1.1). Then, we address these identified issues through a series of modifications (Section 5.1.2), followed by a peer review (Section 5.1.3) to ensure accuracy and reliability. Finally, we experiment with the revised benchmark to evaluate and discuss the results. Below, more details are provided for specific steps in this approach.

5.1.1 Standardized Observations in Current HumanEval Benchmarks. Upon examining and manually reconstructing canonical solutions and experimenting with various HumanEval benchmark test suites, we identified several recurring issues. The system frequently produces incorrect and sub-optimal code, as canonical solutions are inefficient and fail to address critical assumptions outlined in the problem descriptions.

Additionally, these solutions often lack type annotations, further complicating the evaluation process. Another significant problem is the absence of quality testing. Test suites tend to overlap with example tests from the prompt, allowing incorrect canonical solutions to pass. Moreover, there are instances where the expected outputs in the test suites do not align with the canonical solutions’ actual performance.

Compounding these issues is the poor quality of the problem descriptions, which contain grammatical errors, ambiguous instructions, and inconsistent formatting, particularly in the test examples. Furthermore, the system’s support for language conversion frameworks, such as MultiPL-E, is inadequate. MultiPL-E, while the most comprehensive framework available, only supports equality assertions, which proves incompatible with the setup of many problems, further hindering the system’s effectiveness.

Table 21. Comparison of test statistics between HumanEval (based on <https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz> human-eval-v2-20210705.json) and HUMANEVALNEXT.

Metric	HumanEval (Original)	HUMANEVALNEXT	Δ
Total number of asserts	1325	2551	$\times 1.92$
Average number of asserts	8	16	$\times 2$
Median number of asserts	7	11	$\times 1.57$
Min. number of asserts	1	4	+3
Problems with < 5 asserts	34	2	-94%

5.1.2 Modifications in HUMANEVALNEXT. In HUMANEVALNEXT, we address all the above issues by manually modifying all problems in the original HumanEval benchmark. We decide to improve the original HumanEval as (1) flaws in the original version persist even in improved versions such as HumanEvalPlus, and (2) the fact that HumanEval is still widely used in new literature. While in our we have gone for the original benchmark, due to the adaptable nature of BenchFrame one could opt to have an improved version of the dataset as the starting point. To summarize, these are the general changes made in HUMANEVALNEXT and their benefits: In this work, several key improvements have been made to address the shortcomings of previous benchmarks. First, **all suboptimal and incorrect canonical solutions have been fixed**, which were previously missed due to insufficient testing and a lack of comprehensive quality review. Furthermore, **type annotations have been added** to all problems, offering valuable context and simplifying the translation to other programming languages. The original HumanEval benchmark only included type annotations for the first 30 problems, representing merely 18% of the total set of 164 problems. In addition, **better support for language conversion frameworks** has been incorporated. For instance, HUMANEVALNEXT now features improved compatibility with frameworks like MultiPL-E [23], which supports translation to 18 additional programming languages by standardizing all tests to equality assertions where feasible. This change has reduced the number of incompatible problems by a factor of ten.

Besides these adjustments, challenging scenarios (such as negative values, zero instances, empty inputs, and non-alphanumeric symbols) are incorporated into each task to guarantee that only top-quality AI4SE models, adept at addressing diverse situations, succeed. Accordingly, **assertions are implemented** within the code wherever constraints are detailed in the problem description. This measure prevents models from ignoring crucial details, thus improving the benchmark’s worth. In particular, we employ specification-based testing to assess functions; using boundary analysis, we explore combinations of within, on, and outside points. Furthermore, the **test examples in the problem descriptions have been refined**, which significantly affects model performance [10]. Problems with excessive test examples now feature a reduced set, distributing the evaluation workload more fairly. Lastly, **spelling errors have been corrected**, descriptions have been consistently formatted, and **problem descriptions have been aligned with the implementations**, while still leaving room for models to demonstrate intuitive problem-solving skills expected of high-quality AI4SE models. The difficulty level has also been raised by incorporating **more edge cases and modifying various problems**, aiming to better reflect real-world challenges faced by engineers and to mitigate issues related to data leakage and saturated performance on the HumanEval leaderboards.

To illustrate the modifications of the tests in HUMANEVALNEXT, consider Table 21.

5.1.3 Peer Review Process of HUMANEVALNEXT. To ensure the accuracy and reliability of the initial version of HUMANEVALNEXT, an independent reviewer verified all changes. This thorough review involved verifying the clarity and completeness of the problem docstrings, checking for consistency between the problem descriptions and the

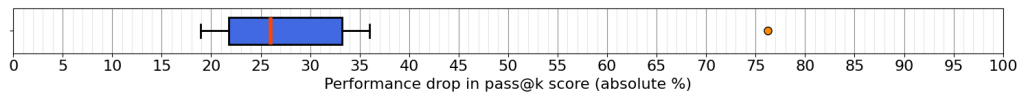


Fig. 7. Boxplot depicting the distribution of absolute drops in pass@1 score between HumanEval and the newly introduced HUMANEVALNEXT benchmark, based on 10 LLMs (Table 22).

canonical solutions, and ensuring that both the solutions and test cases were correct and efficient. Where inefficiencies were identified, suggestions for optimization were provided. The review also scrutinized the test cases to ensure comprehensive coverage, identifying any gaps that could allow incorrect solutions to pass. While the initial creation of the benchmark took over 100 hours, the independent peer-review process required an additional 16 hours. As a result of this review, 1% of the problems were redesigned due to structural issues, 9% received additional test cases, and 15% underwent minor grammar or clarity improvements. All suggested changes were documented and reviewed by the original author, with every recommendation either implemented or refined further upon discussion. Since the peer-review involved modifications beyond the test suites, completions for all models were re-run. Despite these changes, the results remained largely consistent with the original, with 40% of models showing no change in their pass@1 scores, 50% showing a 1-2% absolute change, and only 10%—previously top performers—experiencing a 5% drop. This demonstrated that the peer-review process upheld the benchmark’s robustness while refining its quality.

5.2 Experimental Setup

To assess the impact of the modifications applied to the benchmark, we examine the PASS@1 performance of ten state-of-the-art open-source software code models using the original HumanEval alongside two enhanced variants, HUMANEVALNEXT and EVALPLUS [118]. We selected the top-performing models from the big code LLM leaderboards at the evaluation’s start: “NTQAI/Nxcode-CQ-7B-orpo”, “Qwen/CodeQwen1.5-7B”, “deepseek-ai/deepseek-coder-6.7b-instruct”, “TechxGenus/starcoder2-15b-instruct”, “ise-uiuc/Magicoder-S-DS-6.7B”, “Artigenz/Artigenz-Coder-DS-6.7B”, “HuggingFaceH4/starchat2-15b-v0.1”, “google/codegemma-7b-it”, “codeLlama/CodeLlama-13b-Instruct-hf”, and “Stabilityai/stable-code-3b”. For each task in the benchmark (164 total), the LLM is prompted using an instructional preamble asking the model to finish the implementation of the function requested in addition to providing the imports, function header, and function description with each request. We run the inference for the models on a cluster with one NVIDIA A100 80GB GPU and 32 CPU cores. During test execution, a timeout limit of 15 seconds is utilized per function call to disregard completions that are potentially looping forever or are considered overly inefficient with regard to the canonical solutions. Each test is executed in an evaluation suite using the precautions deployed by OpenAI.

5.3 Results

Upon conducting the experiments outlined in Section 5.2, a key observation is an **average decrease of 31.22%** and a **median decrease of 26.02%** (both absolute percentages) in pass@1 results of HumanEval when contrasted with the newly introduced HUMANEVALNEXT benchmark, using 10 different LLMs. Specific model outcomes are detailed in Table 22, with an overall depiction of the performance decreases displayed in Figure 7. While there are still significant performance declines when comparing HumanEvalPlus with the original HumanEval (11.28 mean and 7.05 median), the declines are substantially larger with HumanEvalNext than with EvalPlus. Overall, the findings highlight a marked reduction in model performance when measured against HUMANEVALNEXT as opposed to the initial

HumanEval benchmark. This pattern emphasizes the enhanced difficulty and refined evaluation precision offered by HUMANEVALNEXT, which incorporates more resilient environments featuring type annotations and clearer instructions.

A closer look reveals that the top-performing models from the original HumanEval benchmark do not maintain their standings in HUMANEVALNEXT. For example, while HUMANEVALNEXT consistently ranks deepseek-ai's deepseek-coder-6.7b-instruct as the top performer, previous leaders like NTQAI's Nxcode-CQ-7B-orpo and Qwen's CodeQwen1.5-7B show a significant drop in their rankings. Specifically, NTQAI's Nxcode-CQ-7B-orpo falls from an impressive 87.23% pass@1 in HumanEval to 51.22% in HUMANEVALNEXT, and Qwen's CodeQwen1.5-7B plummets from 87.2% to 10.98%. This sharp decline suggests that certain models may have benefited from data leakage or other issues in the original HumanEval benchmark, indicating the necessity for a more rigorous benchmark like HUMANEVALNEXT to accurately assess model performance.

Especially the resilience of models, e.g., deepseek-ai/deepseek-coder-6.7b-instruct, can be confirmed by evaluating the model on the new challenges presented in HUMANEVALNEXT. When models also score relatively well in this benchmark, it highlights the models' ability to adapt and perform competently under more demanding conditions, making HUMANEVALNEXT a great benchmark to reveal the reliability of the capabilities of models. This finding also emphasizes the need for regular updates to benchmarks, as reliance on outdated benchmarks can misrepresent model capabilities over time, even when models claim not to train on the test data.

Furthermore, an interesting observation emerges when analyzing model size and performance: bigger is not always better. Larger models such as TechxGenus/starcoder2-15b-instruct and HuggingFaceH4/starchat2-15b-v0.1 do not consistently outperform smaller models. This observation suggests that model size alone is not a definitive predictor of success in complex, edge-case-inclusive benchmarks like HUMANEVALNEXT. For instance, looking at pass@1 scores, Techx-Genus/starcoder2-15b-instruct scores 77.4% on HumanEval but drops to 43.29% on HUMANEVALNEXT, while ise-uiuc/Magicoder-S-DS-6.7B scores 76.8% on HumanEval (lower) but only drops to 53.66% on HUMANEVALNEXT (higher), highlighting that increased model size does not necessarily equate to better performance in more challenging assessments.

Lastly, ranking problems based on their difficulty using pass metrics per problem, confirms that HUMANEVALNEXT presents a well-distributed range of complexity over the complete set of challenges. It also shows the increased difficulty of the benchmark, where even the easiest problems are not universally passed, with roughly 30% of the models still failing to solve them. Altogether, this distribution highlights HUMANEVALNEXT's effectiveness in providing a thorough assessment environment, making it an excellent tool for evaluating straightforward coding capabilities of LLMs in a lightweight manner - the main reason behind the popularity of the original HumanEval benchmark and its variants.

6 Discussion

In this section, we discuss the broader implications of our study; here, we attempt to tie our findings back to the RQs that guided our study. We, specifically, touch on what our results mean for researchers navigating the field, address the practical salability of our proposed framework, and outline threats to validity and opportunities for future work.

6.1 Implications

6.1.1 Finding the Right Benchmark. In our analysis, and specifically RQ1 we find that AI4SE benchmarking is highly fragmented, with many benchmarks suffering from poor maintenance and lack of discoverability. With this finding, we underscore the fact a major challenge in AI4SE is the difficulty of selecting a contextually relevant benchmark.

Table 22. Performance Comparison on HumanEval Benchmarks (pass@1)

Model	HumanEval	HumanEvalPlus (Δ)	HumanEvalNext (Δ)
stable-code-3b	30.72	25.60 (−5.12)	1.83 (−28.89)
CodeLlama-13b-instruct-hf	50.60	34.10 (−16.50)	29.88 (−20.72)
codegemma-7b-it	60.40	51.80 (−8.60)	41.46 (−18.94)
starchat2-15b-v0.1	73.80	71.30 (−2.50)	43.29 (−30.51)
Artigenz-Coder-DS-6.7B	75.60	72.60 (−3.00)	53.66 (−21.94)
Magocoder-S-DS-6.7B	76.80	71.30 (−5.50)	53.66 (−23.14)
starcode2-15b-instruct	77.40	60.00 (−17.40)	43.29 (−34.11)
deepseek-coder-6.7b-instruct	80.22	71.30 (−8.92)	58.54 (−21.68)
CodeQwen1.5-7B	87.20	45.70 (−41.50)	10.98 (−76.22)
Nxcode-CQ-7B-orpo	87.23	83.50 (−3.73)	51.22 (−36.01)

BenchScout was our direct answer to RQ2. With BenchScout, users can select more relevant benchmarks and gain deeper insights into model performance for their specific needs.

6.1.2 Benchmarking the Benchmark. Our answer to RQ3, BenchFrame, can have an impact on how we measure progress in AI4SE. Our results show the significant impact of integrating the BENCHFRAME framework and highlight the value of peer-reviewed, validated benchmarks. With many state-of-the-art models tested on similar benchmarks, both industry and academia should adapt their evaluation methods to ensure robust results. The refinement of benchmarks across various AI4SE tasks will be critical for guiding future research and ensuring that these models can perform effectively. Although this study, specifically in RQ1, highlights the issue of data leakage in current benchmarks, it remains true that HumanEvalNext would also be affected by this issue. However, we argue that benchmarks should not be used indefinitely and should evolve to pose increasing challenges in alignment with model improvements.

6.2 A Case Study on the agentification of BenchFrame

A potential criticism of the framework proposed in RQ3 is the significant manual effort it demands compared to a more automated system. To address these concerns about the generalizability and manual labor intensity of BenchFrame, we developed an agentic pipeline designed to replicate and automate the benchmark improvement process with minimal human intervention. Our baseline for this effort was the HumanEvalNext dataset, which had previously been improved by manual human supervision. We aimed to determine whether an agentic approach could yield comparable improvements without the extensive human labor (i.e., to show noninferiority).

The agent pipeline was structured into three distinct phases, which mirrors the manual process described in Section 5.1.2. In the first phase, the agent improved the problem description, which includes the function signature and the docstring. Using this refined description, the agent then generated an improved canonical solution. In the final phase, the agent created and set up the test cases to validate the solution. This process was applied across all the problems in the dataset, and our hypothesis was that this automated approach would be noninferior to the human-improved dataset.

To validate this, we conducted a paired evaluation comparing the human- and agent-improved versions of each benchmark problem in the HumanEvalNext dataset, specifically HumanEvalNext versus HumanEvalNext-Agentic. Two independent reviewers rated each pair using a five-point ordinal scale ranging from -2 (strongly preferring the human version) to +2 (strongly preferring the agentic version), with 0 indicating no preference. We tested whether the agentic pipeline was noninferior to the human-improved process, using a conservative noninferiority margin of $\delta = -0.5$. That

is, we assessed whether the average rating for the agentic outputs was not more than 0.5 points worse than the human outputs.

For each reviewer, as well as for the combined ratings (averaged across reviewers), we performed both a one-sided one-sample t-test and a Wilcoxon signed-rank test to assess noninferiority. The results were consistent and conclusive: the mean ratings were 0.16 (SD = 0.67) for Reviewer 1, 0.53 (SD = 0.51) for Reviewer 2, and 0.35 (SD = 0.49) for the combined average. The corresponding t-statistics were 12.68, 25.75, and 22.10, with one-sided p-values all equal to 1.000 (indicating extremely strong evidence in favor of noninferiority given the direction of the test). The non-parametric Wilcoxon signed-rank tests further confirmed these results, with statistics of 12,155.5 ($p = 3.46 \times 10^{-20}$), 13,198.0 ($p = 7.20 \times 10^{-28}$), and 12,698.0 ($p = 5.94 \times 10^{-27}$) respectively.

These results indicate that the agentic pipeline is noninferior to the human-improved process—indeed, with mean and median ratings above zero, the agentic output was often preferred. This supports the feasibility of using an agentic approach to automate benchmark refinement while significantly reducing manual effort.

While noninferior, we noted several standardized observations, in terms of pitfalls, during this evaluation, such as (1) instances where the agent’s docstring occasionally revealed aspects of the solution, or (2) where canonical solutions made unintended assumptions, or (3) where the test cases did not adhere to the required, single-line assert, format. These observations were later accounted for in later stages of the feasibility study to make adjustments in the pipeline, and to ensure that it remained flexible and effective.

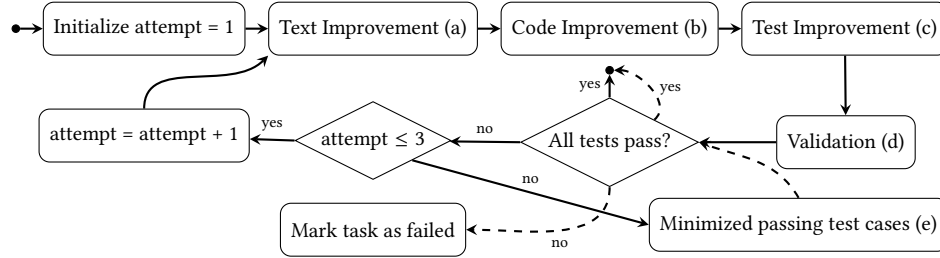


Fig. 8. A flow-chart indicating the pipeline used to improve MBPP.

For the sake of completeness, we present the refined pipeline used in the process of benchmark refinement. The pipeline begins by initializing an attempt counter, which tracks the number of iterations taken to complete a given benchmark improvement. The process then proceeds through three core phases: (a) Text Improvement, in which the agent refines the problem description and adds a type-annotated function signature; (b) Code Improvement, where the canonical implementation is modified according to Python best practices and based on the improved task description; and (c) Test Improvement, which involves generating assert-based test cases that include edge cases, boundary conditions, and common error scenarios. These three components are then assembled into a complete Python program and passed to the Validation phase (d), which runs the implementation against the test suite in a sandboxed subprocess.

If all tests pass on the first try, the process terminates successfully. Otherwise, the pipeline evaluates whether the number of attempts is below a fixed threshold (three, in our case). If so, the pipeline either increments the attempt counter and retries the full process or, alternatively, in the case that more tests are passing than were present in the original test suite, the pipeline identifies a minimal set of passing tests (e) to be used as the new test suite. This loop ensures that even partially successful test runs can be used to improve the benchmark iteratively. After three failed attempts, the task is marked as unsuccessful.

Table 23. Pass@1 results for HumanEval, HumanEvalNext, and HUMANEVALNEXT-AGENTIC.

Model	HumanEval	HumanEvalNext-Agentic (Δ)	HumanEvalNext (Δ)
stable-code-3b	30.72	2.44(−28.28)	1.83 (−28.89)
CodeLlama-13b-instruct-hf	50.60	18.90(−31.70)	29.88 (−20.72)
codegemma-7b-it	60.40	22.56 (−37.84)	41.46 (−18.94)
starchat2-15b-v0.1	73.80	26.83 (−46.97)	43.29 (−30.51)
Artigenz-Coder-DS-6.7B	75.60	31.70 (−43.90)	53.66 (−21.94)
Magocoder-S-DS-6.7B	76.80	31.70 (−45.10)	53.66 (−23.14)
starcode2-15b-instruct	77.40	14.02 (−63.38)	43.29 (−34.11)
deepseek-coder-6.7b-instruct	80.22	35.37 (−44.85)	58.54 (−21.68)
CodeQwen1.5-7B	87.20	1.22 (−86.00)	10.98 (−76.22)
Nxcode-CQ-7B-orpo	87.23	31.70 (−55.53)	51.22 (−36.01)

For our experiments, we used the OpenAI API ²², specifically, the o3-mini-2025-01-31 as the underlying model for the agent. We evaluated this new dataset on the same set of models, and the results (as presented in Table 23) indicated comparable (and even better) performance patterns. These findings provide adequate justification for how an agentic pipeline can effectively generalize and provide a semi-automated solution for benchmark improvement.

The results presented indicate, that while a certain level of human intervention, specifically in the review phase, is still required to ensure quality of the improvements, it is but a fraction of the total time required to manually improve the benchmark. From a cost perspective, we can also report that the total incurred cost from calling the apis for the models was \$5.28²³. This is but a small portion of the labor costs required to improve the benchmark manually. The introduction of this agentic layer significantly reduced the manual effort required. We claim that, with this approach, one can extend it to other datasets with similar success.

6.3 Evaluating the Generalizability of BenchFrame

An additional critique that can be raised regarding our approach is its generalizability. To tackle this issue, we adopted a strategy akin to HumanEval. Using our systematic observations of MBPP from the review, we apply BenchFrame to 100 problems selected from a pool of 500 MBPP test samples. The rationale behind this selection is twofold: firstly, it aims to assess a new task, specifically program synthesis, and secondly, it considers the benchmark’s popularity. During this process, we subject the function descriptions, implementations (including variable typing), and significantly extend test cases; more specifically, we go from a fixed 3 tests per problem to an average of 12.35, with a median of 11.5. We finally go through several rounds of peer review to ensure the quality of the improvements. We refer to the resulting benchmark as MBPPNext. The evaluation results of this dataset in comparison to the baseline are detailed in Table 24. We observe a comparable decline in performance similar to that noted for HumanEval, specifically, an average decrease of 13.4 percentage points across the methods evaluated. These findings further underscore the necessity for thorough peer review and verification of benchmarks, as discussed in previous sections of this paper.

6.4 Future Work

Future work on BENCHFRAME will focus on expanding to additional programming languages, which it already has been optimized for, yet these variants have not been produced or evaluated. Although evaluating ten LLMs provides

²²<https://platform.openai.com>

²³At the time of running these experiments, on the 31st of July 2025

Table 24. Pass@1 Results for MBPP and MBPPNext. Based on <https://github.com/google-research/google-research/blob/master/mbpp/mbpp.jsonl> accessed on the 14th of July 2025

Model	MBPP	MBPPNext (Δ)
stable-code-3b	34.0	28.0 (-6%)
CodeLlama-13b-instruct-hf	40.0	31.0 (-9%)
codegemma-7b-it	43.0	33.0 (-10%)
starchat2-15b-v0.1	59.0	42.0 (-17%)
Artigenz-Coder-DS-6.7B	60.0	47.0 (-13%)
Magocoder-S-DS-6.7B	63.0	45.0 (-18%)
starcode2-15b-instruct	44.0	36.0 (-8%)
deepseek-coder-6.7b-instruct	68.0	50.0 (-18%)
CodeQwen1.5-7B	53.0	40.0 (-13%)
Nxcode-CQ-7B-orpo	74.0	52.0 (-22%)

valuable insight, it remains unclear how larger, top-performing models behind paywalls, such as GPT and Gemini, would perform; Future research could evaluate the performance differences of such models when comparing the base and the pro version. Furthermore, future research should focus on applying the underlying ideas of BenchFrame to mutli-file and project-level benchmarks like Defects4J.

6.5 Threats to the Validity

Construct Validity: To reduce bias and errors in the literature review, two authors followed a structured protocol for selecting and filtering sources. A threat to construct validity comes from the subjectivity in defining and applying the inclusion/exclusion criteria. To address this, we pre-defined clear criteria and peer-reviewed the selection process for consistency. Another potential validity threat stems from the design of our user study, as it may not fully capture the tool’s overall functionality, strengths, or weaknesses. To mitigate this, we evaluated diverse aspects including usability, functionality, usefulness, and intuitiveness. Lastly, to minimize biases during HumanEvalNext’s development, we applied consistent refinement criteria and peer-reviews, although subjective interpretation remains a residual risk.

Internal Validity: A potential threat in the user study is selection bias. To mitigate this, we included participants from both industry and academia, ensuring a range of skills and experience. To reduce the real-world impact arising from any study biases, we have created a continuous feedback mechanism in BenchScout to receive continuous feedback from users to be able to meet their needs. All participants received the same tool, guidance, and instructions. To reduce bias in the peer review or HumanEvalNext, the reviewer was not informed of specific changes made by the first author.

External Validity: A threat here is the generalizability of the user study’s results. We mitigated this by including 22 participants, but the sample size may still limit the generalizability of the results. Although we evaluated the effects of the BENCHFRAME framework on the performance of ten models with one of the most widely used benchmarks, this could not be considered sufficient to prove the generalizability of the results; adding more models and application to more benchmarks could further confirm our results. BenchFrame may be criticized for its limited applicability due to the significant manual effort it demands compared to more automated systems like EvalPlus. We demonstrate that the process of improving benchmarks such as HumanEval and MBPP can be (semi-)automated through the use of agentic pipelines.

7 Related Work

The rapid integration of LLMs into software engineering has led to a corresponding proliferation of benchmarks designed to evaluate them. This has spurred research to systematically map this new terrain, assess the quality of evaluation resources, and build tools for navigating them. Our work is situated within these three emerging areas.

AI4SE Surveys and Taxonomies. Initial research in this area included broad surveys of LLMs in software engineering, which established evaluation as a central research challenge [40, 62, 222, 237]. More recently, meta-analyses have focused on the benchmarks themselves. For instance, Wang et al. created a taxonomy based on the Software Development Life Cycle, revealing a research gap with a strong focus on code implementation and a negligence towards design and requirements engineering [182]. Our work contributes to this area by providing a comprehensive review of 273 benchmarks from 247 studies (**RQ1**). We go beyond categorization by analyzing each benchmark across a 14-point metadata schema to identify systemic limitations, such as poor maintenance, language specificity, and a lack of peer review.

Tools for Benchmark Discovery and Navigation. While platforms like Hugging Face and the now-discontinued Papers With Code serve as valuable repositories for hosting datasets [72], they often lack the specialized search and visualization capabilities needed by AI4SE researchers. BenchScout addresses this gap (**RQ2**) by providing an extensible semantic search tool specifically designed for the AI4SE community. Unlike general-purpose repositories, it combines semantic embeddings with structured metadata filters and an interactive visualization of the benchmark landscape to improve the discoverability and selection of relevant evaluation tools.

Methodologies for Improving Benchmark Quality. There is a growing agreement that many static benchmarks are prone to saturation and data contamination, which can compromise the validity of evaluation results [76]. In response, two movements have arisen. Efforts like LiveCodeBench fall under the "build anew" category; these strive to develop dynamic, "live" benchmarks that continuously gather new challenges from real-world environments to avoid overfitting.

Our method, BenchFrame, belongs to the "repair and refine" category and focuses on improving the quality of existing benchmarks (**RQ3**). A method within this category is DyCodeEval, which uses seed contexts to frame problems in specific scenarios (e.g., banking, healthcare, education). This approach is designed to produce more reliable results by minimizing data contamination and memorization effects [27]. Our approach sets itself apart by incorporating a peer-review-oriented process that includes standardized observations, precise modifications, and independent validation. Like DyCodeEval, we show its practical scalability through an agentic pipeline that automates the improvement process and tackles the main challenge, namely, the manual efforts required in BenchFrame.

8 Conclusion

The findings of our study highlight the importance of reliable and consistent benchmarking in AI4SE to drive the development of more robust models. Through the creation of BenchFrame and the enhancement of the HumanEval benchmark, we have demonstrated that higher-quality benchmarks reveal substantial performance gaps, as shown by the 31.2% average reduction in pass@1 scores across ten state-of-the-art models. This significant decline highlights the impact of more stringent evaluations. BenchScout further enhances this process by facilitating the discovery of relevant benchmarks and reduces the overhead associated with selecting the appropriate tools for evaluation.

8.1 Data Availability

We publicly release the results of our literature review, user study, and 50% of the manually refined benchmark.²⁴ Upon acceptance, the complete benchmark will be made available on both GitHub and HuggingFace.

References

- [1] Agarwal, et al. 2024. Codemirage: Hallucinations in code generated by large language models. *arXiv preprint arXiv:2408.08333* (2024).
- [2] Agashe, et al. 2019. JulCe: A Large Scale Distantly Supervised Dataset for Open Domain Context-based Code Generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, China, 5436–5446. <https://doi.org/10.18653/v1/D19-1546>
- [3] Agrawal, et al. 2023. Guiding Language Models of Code with Global Context using Monitors. <https://doi.org/10.48550/arXiv.2306.10763>
- [4] Ahmad, et al. 2023. AVATAR: A Parallel Corpus for Java-Python Program Translation. In *Findings of the Association for Computational Linguistics*. Association for Computational Linguistics, Toronto, Canada, 2268–2281. <https://doi.org/10.18653/v1/2023.findings-acl.143>
- [5] Al-Kaswan, et al. 2024. Traces of Memorisation in Large Language Models for Code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–12. <https://doi.org/10.1145/3597503.3639133>
- [6] Allamanis, et al. 2016. A Convolutional Attention Network for Extreme Summarization of Source Code. *arXiv: Learning* (2016). <https://doi.org/10.48550/arXiv.1602.03001>
- [7] Almohaimeed, et al. 2024. Ar-Spider: Text-to-SQL in Arabic. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24)*. Association for Computing Machinery, New York, NY, USA, 1024–1030. <https://doi.org/10.1145/3605098.3636065>
- [8] Amini, et al. 2019. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. <https://arxiv.org/abs/1905.13319>
- [9] Athiwaratkun, et al. 2022. Multi-lingual evaluation of code generation models. <https://arxiv.org/abs/2210.14868>
- [10] Austin, et al. 2021. Program Synthesis with Large Language Models. <https://arxiv.org/abs/2108.07732>
- [11] Aw, et al. 2023. Instruction-tuning Aligns LLMs to the Human Brain. <https://arxiv.org/abs/2312.00575>
- [12] Babe, et al. 2023. StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code. *arXiv.org* (2023). <https://doi.org/10.48550/arXiv.2306.04556>
- [13] Badertdinov, et al. 2025. SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents. *arXiv preprint arXiv:2505.20411* (2025).
- [14] Baik, et al. 2024. CodeComplex: A Time-Complexity Dataset for Bilingual Source Codes. <https://arxiv.org/abs/2401.08719>
- [15] Bakshandaeva, et al. 2022. PAUQ: Text-to-SQL in Russian. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2355–2376. <https://doi.org/10.18653/v1/2022.findings-emnlp.175>
- [16] Bareiß, et al. 2022. Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language Models on Code. <https://doi.org/10.48550/arXiv.2206.01335>
- [17] Barone et al. [n. d.]. A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Documentation and Code Generation. ([n. d.]).
- [18] Bhandari, et al. 2021. CVEfixes: Automated Collection of Vulnerabilities and Their Fixes from Open-Source Software. *arXiv: Software Engineering* (2021). <https://doi.org/10.1145/3475960.3475985>
- [19] Bogomolov, et al. 2024. Long Code Arena: a Set of Benchmarks for Long-Context Code Models. <https://arxiv.org/abs/2406.11612>
- [20] Bouzenia, et al. 2024. DyPyBench: A Benchmark of Executable Python Software. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 338–358. <https://doi.org/10.1145/3643742>
- [21] Bui, et al. 2022. Vul4J: a dataset of reproducible Java vulnerabilities geared towards the study of program repair techniques. In *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, Pittsburgh Pennsylvania, 464–468. <https://doi.org/10.1145/3524842.3528482>
- [22] Cao, et al. 2024. JavaBench: A Benchmark of Object-Oriented Code Generation for Evaluating Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 870–882.
- [23] Cassano, et al. 2022. MULTIPL-E: a scalable and extensible approach to benchmarking neural code generation. <https://arxiv.org/abs/2208.08227>
- [24] Chandel, et al. 2022. Training and Evaluating a Jupyter Notebook Data Science Assistant. <https://arxiv.org/abs/2201.12901>
- [25] Chen, et al. 2024. RMCBench: Benchmarking Large Language Models’ Resistance to Malicious Code. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 995–1006. <https://doi.org/10.1145/3691620.3695480>
- [26] Chen, et al. 2021. Evaluating large language models trained on code. <https://arxiv.org/abs/2107.03374>
- [27] Chen, et al. 2025. Dynamic Benchmarking of Reasoning Capabilities in Code Large Language Models Under Data Contamination. [arXiv:cs.SE/2503.04149](https://arxiv.org/abs/2503.04149) <https://arxiv.org/abs/2503.04149>
- [28] Chen, et al. 2023. TheoremQA: A Theorem-driven Question Answering Dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 7889–7901. <https://doi.org/10.18653/v1/2023.emnlp-main.489>

²⁴<https://github.com/AISE-TUdelft/AI4SE-benchmarks>

- [29] Cheng, et al. 2025. Codemenv: Benchmarking large language models on code migration. *arXiv preprint arXiv:2506.00894* (2025).
- [30] CODAIT. 2021. AI for Code: Predict Code complexity using IBM's CodeNet Dataset. <https://community.ibm.com/community/user/datascience/blogs/sepideh-seifzadeh1/2021/10/05/ai-for-code-predict-code-complexity-using-ibms-cod>
- [31] Cui, et al. 2024. LogEval: A Comprehensive Benchmark Suite for Large Language Models In Log Analysis. *arXiv:cs.CL/2407.01896* <https://arxiv.org/abs/2407.01896>
- [32] Cui. 2024. WebApp1K: A Practical Code-Generation Benchmark for Web App Development. *arXiv:cs.SE/2408.00019* <https://arxiv.org/abs/2408.00019>
- [33] Deng, et al. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1337–1350. <https://doi.org/10.18653/v1/2021.naacl-main.105>
- [34] Dinella, et al. 2024. CRQBench: A Benchmark of Code Reasoning Questions. (2024). <https://doi.org/10.48550/arXiv.2408.08453>
- [35] Ding, et al. 2023. CrossCodeEval: a diverse and multilingual benchmark for Cross-File code completion. <https://arxiv.org/abs/2310.11248>
- [36] Dong, et al. 2023. CodeScore: Evaluating code generation by learning code execution. <https://arxiv.org/abs/2301.09043>
- [37] Dougherty et al. 2025. Proving the Coding Interview: A Benchmark for Formally Verified Code Generation. *arXiv preprint arXiv:2502.05714* (2025).
- [38] Du, et al. 2023. ClassEval: a Manually-Crafted benchmark for evaluating LLMs on class-level code generation. <https://arxiv.org/abs/2308.01861>
- [39] Durieux, et al. 2020. Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 530–541. <https://doi.org/10.1145/3377811.3380364>
- [40] Fan, et al. 2023. Large Language Models for Software Engineering: Survey and Open Problems. *arXiv:cs.SE/2310.03533* <https://arxiv.org/abs/2310.03533>
- [41] Fan, et al. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20)*. Association for Computing Machinery, New York, NY, USA, 508–512. <https://doi.org/10.1145/3379597.3387501>
- [42] Fan, et al. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.
- [43] Gan, et al. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. <https://doi.org/10.48550/arXiv.2106.01065>
- [44] Gan, et al. 2021. Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization. <https://doi.org/10.48550/arXiv.2109.05157>
- [45] Gao, et al. 2022. PAL: Program-aided Language Models. <https://arxiv.org/abs/2211.10435>
- [46] Gemini Team. 2023. Gemini: A Family of Highly Capable Multimodal Models. https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf
- [47] Gong, et al. 2025. CoSQA+: Pioneering the Multi-Choice Code Search Benchmark with Test-Driven Agents. *arXiv:cs.SE/2406.11589* <https://arxiv.org/abs/2406.11589>
- [48] Gong, et al. 2024. Evaluation of LLMs on Syntax-Aware Code Fill-in-the-Middle Tasks. <https://doi.org/10.48550/arXiv.2403.04814>
- [49] Goues, et al. 2015. The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs. *IEEE Transactions on Software Engineering* 41, 12 (2015), 1236–1256. <https://doi.org/10.1109/tse.2015.2454513>
- [50] Gu, et al. 2024. CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution. <https://arxiv.org/abs/2401.03065>
- [51] Gu, et al. 2019. An empirical study on api-misuse bugs in open-source c programs. In *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, Vol. 1. IEEE, 11–20.
- [52] Guo, et al. 2025. CodeEditorBench: Evaluating code editing capability of LLMs. In *ICLR 2025 Third Workshop on Deep Learning for Code*.
- [53] Guo, et al. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
- [54] Haller, et al. 2024. PECC: Problem Extraction and Coding Challenges. *International Conference on Language Resources and Evaluation* (2024). <https://doi.org/10.48550/arxiv.2404.18766>
- [55] Hao, et al. 2022. AixBench: a Code Generation Benchmark Dataset. <https://arxiv.org/abs/2206.13179>
- [56] Hasan, et al. 2021. CoDesc: A Large Code-Description Parallel Dataset. <https://doi.org/10.48550/arXiv.2105.14220>
- [57] Hashemi, et al. 2024. LogPM: Character-Based Log Parser Benchmark. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 705–710. <https://doi.org/10.1109/SANER60148.2024.00077>
- [58] Helmuth, et al. 2021. PSB2: the second program synthesis benchmark suite. *Annual Conference on Genetic and Evolutionary Computation* (2021), 785–794. <https://doi.org/10.1145/3449639.3459285>
- [59] Hendrycks, et al. 2021. Measuring coding challenge competence with APPS. <https://arxiv.org/abs/2105.09938v3>
- [60] Hendrycks, et al. 2021. Measuring Coding Challenge Competence With APPS. *NeurIPS Datasets and Benchmarks* (2021). <https://doi.org/10.48550/arXiv.2105.09938>
- [61] Honarvar, et al. 2023. Turbulence: Systematically and Automatically Testing Instruction-Tuned Large Language Models for Code. *arXiv.org* (2023). <https://doi.org/10.48550/arxiv.2312.14856>
- [62] Hou, et al. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. <https://arxiv.org/abs/2308.10620>
- [63] Hu, et al. 2024. How Effectively Do Code Language Models Understand Poor-Readability Code?. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 795–806.
- [64] Hu, et al. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension (ICPC '18)*. Association for Computing Machinery, NY, USA, 200–210. <https://doi.org/10.1145/3196321.3196334>

- [65] Hu, et al. 2020. Deep code comment generation with hybrid lexical and syntactical information. *Empirical Software Engineering* 25, 3 (May 2020), 2179–2217. <https://doi.org/10.1007/s10664-019-09730-9>
- [66] Hu, et al. [n. d.]. Summarizing Source Code with Transferred API Knowledge. ([n. d.]).
- [67] Hu, et al. 2019. Re-Factoring Based Program Repair Applied to Programming Assignments. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 388–398. <https://doi.org/10.1109/ASE.2019.00044>
- [68] Huang, et al. 2024. EffiBench: Benchmarking the Efficiency of Automatically Generated Code. <https://arxiv.org/abs/2402.02037v2>
- [69] Huang, et al. 2022. Execution-based Evaluation for Data Science Code Generation Models. *Cornell University - arXiv* (2022). <https://doi.org/10.48550/arxiv.2211.09374>
- [70] Huang, et al. 2021. CoSQA: 20,000+ Web Queries for Code Search and Question Answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. ACL, Online, 5690–5700. <https://doi.org/10.18653/v1/2021.acl-long.442>
- [71] Huang, et al. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. Association for Computing Machinery, NY, USA, 293–304. <https://doi.org/10.1145/3238147.3238191>
- [72] Hugging Face. 2016. Hugging Face. <https://huggingface.co>
- [73] Husain, et al. 2019. CodeSearchNet Challenge: Evaluating the state of semantic code Search. <https://arxiv.org/abs/1909.09436>
- [74] InfiCoder. 2023. InfiCoder-Eval: Systematically Evaluating Question-Answering for Code Large Language Models.
- [75] Iyer, et al. 2018. Mapping Language to Code in Programmatic Context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <https://doi.org/10.18653/v1/d18-1192>
- [76] Jain, et al. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. <https://doi.org/10.48550/arXiv.2403.07974>
- [77] Jana, et al. 2024. Cotran: An llm-based code translator using reinforcement learning with feedback from compiler and symbolic execution. In *ECAI 2024*. IOS Press, 4011–4018.
- [78] Jiang, et al. 2024. Collu-Bench: A Benchmark for Predicting Language Model Hallucinations in Code. *arXiv preprint arXiv:2410.09997* (2024).
- [79] Jiang, et al. 2024. A large-scale evaluation for log parsing techniques: How far are we?. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 223–234.
- [80] Jiao, et al. 2023. On the Evaluation of Neural Code Translation: Taxonomy and Benchmark. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Luxembourg, Luxembourg, 1529–1541. <https://doi.org/10.1109/ASE56229.2023.00114>
- [81] Jimenez, et al. 2023. SWE-Bench: Can language models resolve Real-World GitHub Issues? *arXiv (Cornell University)* (2023). <https://doi.org/10.48550/arxiv.2310.06770>
- [82] Jin, et al. 2023. Binary Code Summarization: Benchmarking ChatGPT/GPT-4 and Other Large Language Models. <https://arxiv.org/abs/2312.09601>
- [83] Just, et al. 2014. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. Association for Computing Machinery, NY, USA, 437–440. <https://doi.org/10.1145/2610384.2628055>
- [84] Kanburoğlu et al. 2023. TUR2SQL: A Cross-Domain Turkish Dataset For Text-to-SQL. In *2023 8th International Conference on Computer Science and Engineering (UBMK)*. 206–211. <https://doi.org/10.1109/UBMK59864.2023.10286686>
- [85] Kulal, et al. 2019. SPoC: Search-based Pseudocode to Code. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/7298332f04ac004a0ca44cc69ecf6fb-Paper.pdf
- [86] Labash, et al. 2024. RES-Q: Evaluating Code-Editing Large Language Model Systems at the Repository Scale. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2406.16801>
- [87] Lai, et al. 2022. DS-1000: A natural and reliable benchmark for data science code Generation. *arXiv (Cornell University)* (2022). <https://doi.org/10.48550/arxiv.2211.11501>
- [88] LeClair, et al. 2019. A neural model for generating natural language summaries of program subroutines. *International Conference on Software Engineering* (2019), 795–806. <https://doi.org/10.1109/icse.2019.00087>
- [89] Lee, et al. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. *Annual Meeting of the Association for Computational Linguistics* (2021). <https://doi.org/10.18653/v1/2021.acl-long.176>
- [90] Lee, et al. 2023. The GitHub Recent Bugs Dataset for Evaluating LLM-based Debugging Applications. *arXiv.org* (2023). <https://doi.org/10.48550/arxiv.2310.13229>
- [91] Lei, et al. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *arXiv:cs.CL/2411.07763* <https://arxiv.org/abs/2411.07763>
- [92] Li, et al. 2024. DevBench: A Comprehensive Benchmark for Software Development. <https://arxiv.org/abs/2403.08604>
- [93] Li, et al. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240* (2025).
- [94] Li, et al. 2023. Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. *Neural Information Processing Systems* (2023). <https://doi.org/10.48550/arxiv.2305.03111>
- [95] Li, et al. 2024. EvoCodeBench: An Evolving Code Generation Benchmark Aligned with Real-World Code Repositories. <https://arxiv.org/abs/2404.00599>

- [96] Li, et al. 2024. DevEval: A Manually-Annotated Code Generation Benchmark Aligned with Real-World Code Repositories. <https://arxiv.org/abs/2405.19856>
- [97] Li, et al. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. <https://arxiv.org/abs/2304.08244>
- [98] Li, et al. 2023. TACO: Topics in Algorithmic CODE generation dataset. *arXiv.org* (2023). <https://doi.org/10.48550/arxiv.2312.14852>
- [99] Li, et al. 2025. Fea-bench: A benchmark for evaluating repository-level code generation for feature implementation. *arXiv preprint arXiv:2503.06680* (2025).
- [100] Li, et al. 2024. CoIR: A Comprehensive Benchmark for Code Information Retrieval Models. <https://doi.org/10.48550/arXiv.2407.02883>
- [101] Li, et al. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. <https://doi.org/10.1126/science.abq1158>
- [102] Li, et al. 2023. Exploring the effectiveness of llms in automated logging generation: An empirical study. *arXiv preprint arXiv:2307.05950* (2023).
- [103] Li, et al. 2024. Go static: Contextualized logging statement generation. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 609–630.
- [104] Li, et al. 2022. Automating code review activities by large-scale pre-training. *ESEC/SIGSOFT FSE* (2022). <https://doi.org/10.1145/3540250.3549081>
- [105] Li, et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1035–1047.
- [106] Liang, et al. 2022. Lyra: A Benchmark for Turducken-Style Code Generation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 4238–4244. <https://doi.org/10.24963/ijcai.2022/588>
- [107] Liang, et al. 2024. Can Language Models Replace Programmers? REPOCOD Says ‘Not Yet’. *arXiv preprint arXiv:2410.21647* (2024).
- [108] Lightman, et al. 2023. Let’s Verify Step by Step. <https://doi.org/10.48550/arXiv.2305.20050>
- [109] Liguori, et al. 2021. EVIL: Exploiting Software via Natural Language. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 321–332. <https://doi.org/10.1109/ISSRE52982.2021.00042>
- [110] Liguori, et al. 2022. Can we generate shellcodes via natural language? An empirical study. *Automated software engineering* 29, 1 (2022). <https://doi.org/10.1007/s10515-022-00331-3>
- [111] Lin, et al. 2024. There are More Fish in the Sea: Automated Vulnerability Repair via Binary Templates. *arXiv preprint arXiv:2411.18088* (2024).
- [112] Lin, et al. 2017. QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge. *ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity* (2017), 55–56. <https://doi.org/10.1145/3135932.3135941>
- [113] Lin, et al. 2025. CodeReviewQA: The Code Review Comprehension Assessment for Large Language Models. *arXiv preprint arXiv:2503.16167* (2025).
- [114] Lin, et al. 2018. NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System. *arXiv: Computation and Language* (2018). <https://doi.org/arXiv.1802.08979>
- [115] Lin, et al. 2024. CriticBench: Benchmarking LLMs for Critique-Correct Reasoning. <https://doi.org/10.48550/arXiv.2402.14809>
- [116] Liu, et al. 2025. VADER: A Human-Evaluated Benchmark for Vulnerability Assessment, Detection, Explanation, and Remediation. *arXiv preprint arXiv:2505.19395* (2025).
- [117] Liu, et al. 2024. Exploring and Evaluating Hallucinations in LLM-Powered Code Generation. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2404.00971>
- [118] Liu, et al. 2023. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. <https://arxiv.org/abs/2305.01210>
- [119] Liu, et al. 2025. ProjectEval: A Benchmark for Programming Agents Automated Evaluation on Project-Level Code Generation. *arXiv preprint arXiv:2503.07010* (2025).
- [120] Liu, et al. 2023. CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Luxembourg, Luxembourg, 434–445. <https://doi.org/10.1109/ASE56229.2023.00159>
- [121] Liu, et al. 2023. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems. <https://arxiv.org/abs/2306.03091>
- [122] Liu, et al. 2023. AgentBench: Evaluating LLMs as Agents. <https://doi.org/10.48550/arXiv.2308.03688>
- [123] Liu, et al. 2022. Code generation from flowcharts with texts: A benchmark dataset and an approach. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. 6069–6077.
- [124] Lu, et al. 2021. CodeXGLUE: a machine learning benchmark dataset for code understanding and generation. <http://arxiv.org/abs/2102.04664>
- [125] Mastropaolo, et al. 2022. Using Deep Learning to Generate Complete Log Statements. *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)(2022)*, 2279–2290.
- [126] Mei, et al. 2024. ARVO: Atlas of Reproducible Vulnerabilities for Open Source Software. *arXiv preprint arXiv:2408.02153* (2024).
- [127] Min, et al. 2019. A Pilot Study for Chinese SQL Semantic Parsing. <https://doi.org/10.48550/arXiv.1909.13293>
- [128] Miserendino, et al. 2025. SWE-Lancer: Can Frontier LLMs Earn \$1 Million from Real-World Freelance Software Engineering? *arXiv preprint arXiv:2502.12115* (2025).
- [129] Mishra, et al. 2022. Lila: A Unified Benchmark for Mathematical Reasoning. *Cornell University - arXiv* (2022). <https://doi.org/10.48550/arxiv.2210.17517>
- [130] Moudgalya, et al. 2023. TASTY: A Transformer based Approach to Space and Time complexity. <https://arxiv.org/abs/2305.05379>
- [131] Muennighoff, et al. 2023. OctoPack: Instruction Tuning Code Large Language Models. <https://arxiv.org/abs/2308.07124>
- [132] Nijkamp, et al. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. <https://arxiv.org/abs/2203.13474>
- [133] Niu, et al. 2023. CrossCodeBench: Benchmarking Cross-Task Generalization of Source Code Models. *International Conference on Software Engineering* (2023). <https://doi.org/10.1109/icse48619.2023.00055>

- [134] Nunes, et al. 2025. MaRV: A Manually Validated Refactoring Dataset. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*. IEEE, 141–145.
- [135] Oda, et al. 2015. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 574–584. <https://doi.org/10.1109/ASE.2015.36>
- [136] OpenAI. 2023. GPT-4 Technical Report. <https://arxiv.org/abs/2303.08774>
- [137] Orlanski et al. 2021. Reading StackOverflow Encourages Cheating: Adding Question Text Improves Extractive Code Generation. <https://doi.org/10.48550/arXiv.2106.04447>
- [138] Ou, et al. 2024. Repository-level code translation benchmark targeting rust. *arXiv preprint arXiv:2411.13990* (2024).
- [139] Ouyang, et al. 2025. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517* (2025).
- [140] Pai, et al. 2025. CoDocBench: A Dataset for Code-Documentation Alignment in Software Maintenance. *arXiv:cs.SE/2502.00519* <https://arxiv.org/abs/2502.00519>
- [141] Patil, et al. 2023. Gorilla: Large Language Model Connected with Massive APIs. <https://arxiv.org/abs/2305.15334>
- [142] Paul, et al. 2024. IRCoder: Intermediate Representations Make Language Models Robust Multilingual Code Generators. <https://doi.org/10.48550/arXiv.2403.03894>
- [143] Peng, et al. 2024. HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization. <https://doi.org/10.48550/arXiv.2402.16694>
- [144] Peng, et al. 2021. Revisiting, Benchmarking and Exploring API Recommendation: How Far Are We? <https://arxiv.org/abs/2112.12653>
- [145] Peng, et al. 2025. Coffe: A code efficiency benchmark for code generation. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 242–265.
- [146] Petrukha, et al. 2025. SwiftEval: Developing a Language-Specific Benchmark for LLM-generated Code Evaluation. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*. IEEE, 73–77.
- [147] Quan, et al. 2025. CodeElo: Benchmarking Competition-level Code Generation of LLMs with Human-comparable Elo Ratings. <https://doi.org/10.48550/arXiv.2501.01257>
- [148] Raihan, et al. 2024. mHumanEval–A Multilingual Benchmark to Evaluate Large Language Models for Code Generation. *arXiv preprint arXiv:2410.15037* (2024).
- [149] Rao, et al. 2021. Search4Code: Code Search Intent Classification Using Weak Supervision. <https://doi.org/10.48550/arXiv.2011.11950>
- [150] Rashid, et al. 2025. SWE-PolyBench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703* (2025).
- [151] Roy, et al. 2025. CodeSense: a Real-World Benchmark and Dataset for Code Semantic Reasoning. *arXiv:cs.SE/2506.00750* <https://arxiv.org/abs/2506.00750>
- [152] Roy et al. 2015. Solving General Arithmetic Word Problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, 1743–1752. <https://doi.org/10.18653/v1/D15-1202>
- [153] Roziere, et al. 2022. Leveraging Automated Unit Tests for Unsupervised Code Translation. <https://arxiv.org/abs/2110.06773>
- [154] Schnappinger, et al. 2020. Defining a Software Maintainability Dataset: Collecting, Aggregating and Analysing Expert Evaluations of Software Maintainability. *IEEE International Conference on Software Maintenance and Evolution* (2020), 278–289. <https://doi.org/10.1109/icsme46990.2020.00035>
- [155] Shah, et al. 2024. Stackeval: Benchmarking llms in coding assistance. *Advances in Neural Information Processing Systems* 37 (2024), 36976–36994.
- [156] Shariffdeen, et al. 2023. Cerberus: a Program Repair Framework. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, Melbourne, Australia, 73–77. <https://doi.org/10.1109/ICSE-Companion58688.2023.00028>
- [157] Shen et al. 2024. ConflictBench: A benchmark to evaluate software merge tools. *Journal of Systems and Software* 214 (2024), 112084. <https://doi.org/10.1016/j.jss.2024.112084>
- [158] Shin, et al. 2024. The Good, the Bad, and the Missing: Neural Code Generation for Machine Learning Tasks. *ACM Transactions on Software Engineering and Methodology* 33, 2 (Feb. 2024), 1–24. <https://doi.org/10.1145/3630009>
- [159] Shypula, et al. 2024. Learning Performance-Improving Code Edits. <https://doi.org/10.48550/arXiv.2302.07867>
- [160] Siddiq, et al. 2024. Using Large Language Models to Generate JUnit Tests: An Empirical Study. <https://arxiv.org/abs/2305.00418>
- [161] Siddiq, et al. 2022. SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. (2022). <https://doi.org/10.1145/3549035.3561184>
- [162] Sikka, et al. 2019. Learning based Methods for Code Runtime Complexity Prediction. <https://arxiv.org/abs/1911.01155>
- [163] Silva et al. 2025. Repairbench: Leaderboard of frontier models for program repair. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*. IEEE, 9–16.
- [164] Silva, et al. 2024. GitBug-Java: A Reproducible Benchmark of Recent Java Bugs. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. 118–122. <https://doi.org/10.48550/arXiv.2402.02961>
- [165] Song, et al. 2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. <https://arxiv.org/abs/2306.06624>
- [166] Srivastava, et al. 2022. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. <https://arxiv.org/abs/2206.04615>
- [167] Su, et al. 2024. BRIGHT: A Realistic and Challenging Benchmark for Reasoning-Intensive Retrieval. (2024). <https://doi.org/10.48550/arXiv.2407.12883>
- [168] Svajlenko et al. 2015. Evaluating clone detection tools with BigCloneBench. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSM)*. IEEE, Bremen, Germany, 131–140. <https://doi.org/10.1109/ICSM.2015.7332459>

- [169] Tan, et al. 2025. AL-Bench: A Benchmark for Automatic Logging. *arXiv:cs.SE/2502.03160* <https://arxiv.org/abs/2502.03160>
- [170] Tang, et al. 2024. ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code. <https://doi.org/10.48550/arXiv.2311.09835>
- [171] Tang, et al. 2024. BioCoder: a benchmark for bioinformatics code generation with large language models. *Bioinformatics* (2024). <https://doi.org/10.1093/bioinformatics/btae230>
- [172] Tian, et al. 2023. Is ChatGPT the Ultimate Programming Assistant – How far is it? <https://doi.org/10.48550/arXiv.2304.11938>
- [173] Tian, et al. 2024. DebugBench: Evaluating Debugging Capability of Large Language Models. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2401.04621>
- [174] Tian, et al. 2025. Codehalu: Investigating code hallucinations in llms via execution-based verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 25300–25308.
- [175] Tihanyi, et al. 2023. The FormAI Dataset: Generative AI in Software Security through the Lens of Formal Verification. In *Proceedings of the 19th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, San Francisco CA USA, 33–43. <https://doi.org/10.1145/3617555.3617874>
- [176] Tony, et al. 2023. LLMSecEval: A Dataset of Natural Language Prompts for Security Evaluations. *IEEE Working Conference on Mining Software Repositories* (2023). <https://doi.org/10.1109/msr59073.2023.00084>
- [177] Tufano, et al. 2023. Predicting Code Coverage without Execution. (2023). <https://doi.org/10.48550/arxiv.2307.13383>
- [178] Tufano, et al. 2020. Unit Test Case Generation with Transformers and Focal Context. <https://arxiv.org/abs/2009.05617>
- [179] Dam, et al. 2024. Investigating the Performance of Language Models for Completing Code in Functional Programming Languages: a Haskell Case Study. <https://arxiv.org/abs/2403.15185>
- [180] Venkatesh, et al. 2023. TypeEvalPy: A Micro-Benchmarking Framework for Python Type Inference Tools. *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (2023). <https://doi.org/10.1145/3639478.3640033>
- [181] Wan, et al. 2022. Automated testing of software that uses machine learning APIs. In *Proceedings of the 44th International Conference on Software Engineering*. ACM, Pittsburgh Pennsylvania, 212–224. <https://doi.org/10.1145/3510003.3510068>
- [182] Wang, et al. 2025. Software Development Life Cycle Perspective: A Survey of Benchmarks for Code Large Language Models and Agents. *arXiv:cs.SE/2505.05283* <https://arxiv.org/abs/2505.05283>
- [183] Wang, et al. 2020. DuSQL: A Large-Scale and Pragmatic Chinese Text-to-SQL Dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6923–6935. <https://doi.org/10.18653/v1/2020.emnlp-main.562>
- [184] Wang, et al. 2025. Codeif-bench: Evaluating instruction-following capabilities of large language models in interactive code generation. *arXiv preprint arXiv:2503.22688* (2025).
- [185] Wang, et al. 2024. OOP: Object-Oriented Programming Evaluation Benchmark for Large Language Models. In *Findings of the Association for Computational Linguistics ACL 2024*. 13619–13639.
- [186] Wang, et al. 2025. Deep learning based identification of inconsistent method names: How far are we? *Empirical Software Engineering* 30, 1 (2025), 31.
- [187] Wang, et al. 2024. Testeval: Benchmarking large language models for test case generation. *arXiv preprint arXiv:2406.04531* (2024).
- [188] Wang, et al. 2024. MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback. <https://doi.org/10.48550/arXiv.2309.10691>
- [189] Wang, et al. 2024. RepoTransBench: A Real-World Benchmark for Repository-Level Code Translation. *arXiv preprint arXiv:2412.17744* (2024).
- [190] Wang, et al. 2025. ProjectTest: A Project-level Unit Test Generation Benchmark and Impact of Error Fixing Mechanisms. *arXiv preprint arXiv:2502.06556* (2025).
- [191] Wang, et al. 2022. MCoNaLa: A Benchmark for Code Generation from Multiple Natural Languages. <https://arxiv.org/abs/2203.08388>
- [192] Wang, et al. 2024. HITS: High-coverage LLM-based Unit Test Generation via Method Slicing. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. ACM, Sacramento CA USA, 1258–1268. <https://doi.org/10.1145/3691620.3695501>
- [193] Wang, et al. 2025. CodeContests+: High-Quality Test Case Generation for Competitive Programming. *arXiv preprint arXiv:2506.05817* (2025).
- [194] Wang, et al. 2024. CodeRAG-Bench: Can Retrieval Augment Code Generation? <https://arxiv.org/abs/2406.14497>
- [195] Watson, et al. 2020. On Learning Meaningful Assert Statements for Unit Test Cases. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1398–1409. <https://doi.org/10.1145/3377811.3380429>
- [196] Weyssow, et al. 2024. CodeUltraFeedback: An LLM-as-a-Judge Dataset for Aligning Large Language Models to Coding Preferences. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2403.09032>
- [197] Widyasari, et al. 2020. Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 1556–1560.
- [198] Wu, et al. 2025. Isolating Language-Coding from Problem-Solving: Benchmarking LLMs with PseudoEval. *arXiv:cs.SE/2502.19149* <https://arxiv.org/abs/2502.19149>
- [199] Wu, et al. 2023. How Effective Are Neural Networks for Fixing Security Vulnerabilities. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1282–1294. <https://doi.org/10.1145/3597926.3598135>
- [200] Wu, et al. 2023. ConDefects: A New Dataset to Address the Data Leakage Concern for LLM-based Fault Localization and Program Repair. <https://doi.org/10.48550/arXiv.2310.16253>

- [201] Xia, et al. 2024. Top Leaderboard Ranking = Top Coding Proficiency, Always? EvoEval: Evolving Coding Benchmarks via LLM. <https://arxiv.org/abs/2403.19114>
- [202] Xiao, et al. 2025. CSR-Bench: Benchmarking LLM Agents in Deployment of Computer Science Research Repositories. *arXiv preprint arXiv:2502.06111* (2025).
- [203] Xie, et al. 2024. Codebenchgen: Creating scalable execution-based code generation benchmarks. *arXiv preprint arXiv:2404.00566* (2024).
- [204] Xu, et al. 2025. CLOVER: A Test Case Generation Benchmark with Coverage, Long-Context, and Verification. *arXiv:cs.SE/2502.08806* <https://arxiv.org/abs/2502.08806>
- [205] Xu, et al. 2025. CRUXEval-X: A Benchmark for Multilingual Code Reasoning, Understanding and Execution. *arXiv:cs.AI/2408.13001* <https://arxiv.org/abs/2408.13001>
- [206] Xu, et al. 2025. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951* (2025).
- [207] Xue, et al. 2025. ClassEval-T: Evaluating Large Language Models in Class-Level Code Translation. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA063 (June 2025), 24 pages. <https://doi.org/10.1145/3728940>
- [208] Yadav et al. 2024. PythonSaga: Redefining the Benchmark to Evaluate Code Generating LLM. <https://arxiv.org/abs/2401.03855v2>
- [209] Yan, et al. 2025. Codeif: Benchmarking the instruction-following capabilities of large language models for code generation. *arXiv preprint arXiv:2502.19166* (2025).
- [210] Yan, et al. 2024. CodeScope: An Execution-based Multilingual Multitask Multidimensional Benchmark for Evaluating LLMs on Code Understanding and Generation. <https://doi.org/10.48550/arXiv.2311.08588>
- [211] Yan, et al. 2023. CodeTransOcean: A Comprehensive Multilingual Benchmark for Code Translation. <https://arxiv.org/abs/2310.04951>
- [212] Yang, et al. 2024. CREF: An LLM-based Conversational Software Repair Framework for Programming Tutors. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2406.13972>
- [213] Yao, et al. 2018. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow. *The Web Conference* (2018). <https://doi.org/10.1145/3178876.3186081>
- [214] Yaraghi, et al. 2025. Automated test case repair using language models. *IEEE Transactions on Software Engineering* (2025).
- [215] Yin, et al. 2018. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. Association for Computing Machinery, NY, USA, 476–486. <https://doi.org/10.1145/3196398.3196408>
- [216] Yin, et al. 2023. Natural Language to Code Generation in Interactive Data Science Notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 126–173.
- [217] Yu, et al. 2023. CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. *arXiv (Cornell University)* (2023). <https://doi.org/10.1145/3597503.3623322>
- [218] Yu, et al. 2019. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. <https://doi.org/10.48550/arXiv.1909.05378>
- [219] Yu, et al. 2018. Spider: a Large-Scale Human-Labeled dataset for complex and Cross-Domain semantic parsing and Text-to-SQL task. <https://arxiv.org/abs/1809.08887>
- [220] Yu, et al. 2019. SPaRC: Cross-Domain Semantic Parsing in Context. <https://doi.org/10.48550/arXiv.1906.02285>
- [221] Yun, et al. 2024. Project-specific code summarization with in-context learning. *Journal of Systems and Software* 216 (2024), 112149.
- [222] Zakeri-Nasrabadi, et al. 2023. A systematic literature review on source code similarity measurement and clone detection: techniques, applications, and challenges. <https://arxiv.org/abs/2306.16171>
- [223] Zan, et al. 2022. When Language Model Meets Private Library. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. 277–288.
- [224] Zan, et al. 2022. CERT: Continual Pre-Training on Sketches for Library-Oriented Code Generation. <https://arxiv.org/abs/2206.06888>
- [225] Zan, et al. 2025. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605* (2025).
- [226] Zan, et al. 2024. CodeS: Natural Language to Code Repository via Multi-Layer Sketch. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2403.16443>
- [227] Zavershynskiy, et al. 2018. NAPS: Natural Program Synthesis Dataset. <https://arxiv.org/abs/1807.03168>
- [228] Zeng, et al. 2024. CoderUJB: An Executable and Unified Java Benchmark for Practical Programming Scenarios. <https://arxiv.org/abs/2403.19287>
- [229] Zeng, et al. 2024. Coderujb: An executable and unified java benchmark for practical programming scenarios. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 124–136.
- [230] Zhang, et al. 2025. Codecriticbench: A holistic code critique benchmark for large language models. *arXiv preprint arXiv:2502.16614* (2025).
- [231] Zhang, et al. 2025. LogBase: A Large-Scale Benchmark for Semantic Log Parsing. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 2091–2112.
- [232] Zhang, et al. 2023. RepoCoder: Repository-Level Code completion through iterative retrieval and generation. <https://arxiv.org/abs/2303.12570>
- [233] Zhang, et al. 2023. Multilingual Code Co-evolution using Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, San Francisco CA USA, 695–707. <https://doi.org/10.1145/3611643.3616350>
- [234] Zhang, et al. 2025. SWE-bench Goes Live! *arXiv preprint arXiv:2505.23419* (2025).
- [235] Zhang, et al. 2025. DI-BENCH: Benchmarking Large Language Models on Dependency Inference with Testable Repositories at Scale. *arXiv:cs.CL/2501.13699* <https://arxiv.org/abs/2501.13699>
- [236] Zhang, et al. 2024. TestBench: Evaluating Class-Level Test Case Generation Capability of Large Language Models. *arXiv preprint arXiv:2409.17561*

- (2024).
- [237] Zhang, et al. 2023. A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair. *arXiv.org* (2023). <https://doi.org/10.48550/arxiv.2310.08879>
 - [238] Zhang, et al. 2024. Experimenting a New Programming Practice with LLMs. *arXiv.org* (2024). <https://doi.org/10.48550/arxiv.2401.01062>
 - [239] Zhang, et al. 2018. Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. *IEEE, New York, United States* (2018), 886–896.
 - [240] Zhang, et al. 2024. Benchmarking Data Science Agents. <https://doi.org/10.48550/arXiv.2402.17168>
 - [241] Zheng, et al. 2025. Humanevo: An evolution-aware benchmark for more realistic evaluation of repository-level code generation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. *IEEE Computer Society*, 764–764.
 - [242] Zheng, et al. 2023. CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X. <http://arxiv.org/abs/2303.17568>
 - [243] Zheng, et al. 2021. D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. <https://doi.org/10.48550/arXiv.2102.07995>
 - [244] Zheng, et al. 2025. LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming? *arXiv preprint arXiv:2506.11928* (2025).
 - [245] Zhong et al. 2024. Can llm replace stack overflow? a study on robustness and reliability of large language model code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 21841–21849.
 - [246] Zhong, et al. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. <https://arxiv.org/abs/1709.00103>
 - [247] Zhou, et al. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. <https://doi.org/10.48550/arXiv.1909.03496>
 - [248] Zhu, et al. 2023. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. *IEEE*, 355–366.
 - [249] Zhu, et al. 2022. XLCoST: A Benchmark Dataset for Cross-lingual Code Intelligence. <https://arxiv.org/abs/2206.08474>
 - [250] Zhu, et al. 2022. Multilingual Code Snippets Training for Program Translation. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 10 (2022), 11783–11790. <https://doi.org/10.1609/aaai.v36i10.21434>
 - [251] Zhu, et al. 2025. Domaineval: An auto-constructed benchmark for multi-domain code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 26148–26156.
 - [252] Zhuang, et al. 2023. ToolQA: A Dataset for LLM Question Answering with External Tools. <https://doi.org/10.48550/arXiv.2306.13304>
 - [253] Zhuo, et al. 2024. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. <https://arxiv.org/abs/2406.15877>

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009